Oniga Ștefan

Circuite digitale I

Note de curs

UTPRESS
Cluj-Napoca, 2020
ISBN 978-606-737-482-7

Editura U.T.PRESS
Str.Observatorului nr. 34
C.P.42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: Ing. Călin D. Câmpean

Recenzia: Șl. dr. ing. Ioan Orha
Șl. dr. ing. Claudiu Lung

ISBN 978-606-737-482-7

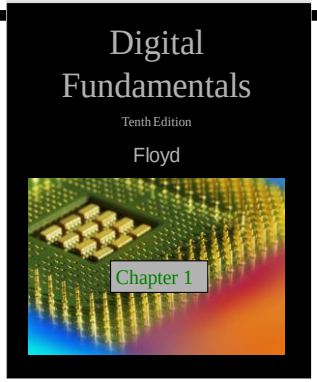
Cuprins

Resurse de învățare

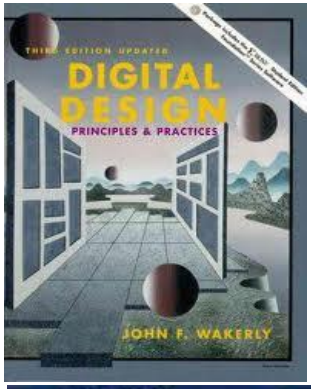
- Site curs: http://ece.ubm.ro/ea/cursuri/CID/cid.html
- Ştefan Oniga, Circuite digitale, Editura Risoprint Cluj Napoca, 2002
- Dan Nicula - Electronica digitala. Carte de invățătură - Editura Universității Transilvania din Brașov, 2012
- John F. Wakerly - Circuite digitale. Principiile și practicile folosite în proiectare, Editura Teora, București, 2002. (Original English language title: Digital Design: Principles and Practices, Third Edition, Prentice Hall, 2000)
- Gheorghe Ștefan - Circuite și sisteme digitale, Editura Tehnica, București, 2000
- Gheorghe Toacșe, Dan Nicula - Electronică digitală. Dispozitive, circuite, proiectare, vol. 1, Editura Tehnică, 2005
- Gheorghe Toacșe, Dan Nicula - Electronică digitală. Verilog HDL, vol. 2, Editura Tehnică, 2005

Cărți limba engleză

- Thomas L. Floyd, Digital fundamentals - editia 10, Prentice Hall, 2009, ISBN-10: 0132359235



- John F. Wakerly, Digital Design, Prentice Hall, 2001, ISBN 0-13-089896-1



- M. Morris Mano, Charles R. Kime, Logic and Computer Design Fundamentals - ediţia 2, Prentice Hall PTR, 1997.

- Edward Karalis, Digital Design principles and Computer Arhitecture, Prentice Hall PTR, 1999.

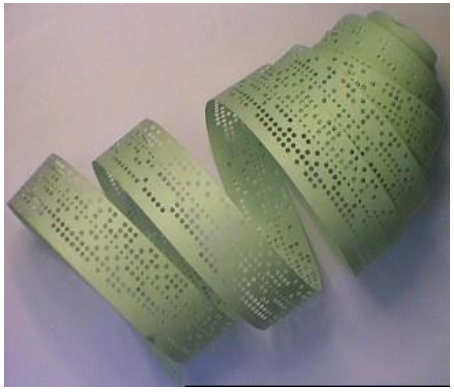
Istoric

Preistoria

1642. B. Pascal realizează o mașină de calculat (+,-).

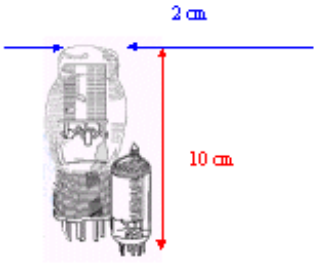
1694. von Leibniz construiește o mașină de +,- şi x,:

1823. Ch.Babbage proiectează primul calculator cu execuție automata a programului.



Calculatoare mecanice [1]

Prima generație



•1904 : John Fleming inventează tubul cu vid

•1945 : John Von Neumann definește arhitectura cunoscută sub numele de Von Neumann utilizată şi în zilele noastre.

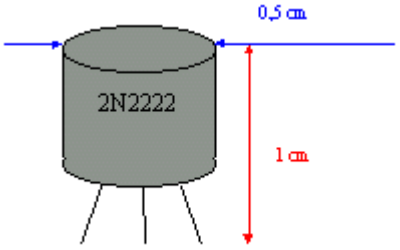
•1937 - 1945. Mașini electromecanice de calculat, bazate pe relee electromagnetice (Mark I), cu program cablat .

•1946 : ENIAC 30 Tonnes / 72 m^2/140 K watts/ 19000 tuburi, 1500 relee. 350 x /s 5000 + /s. Programare prin fire.

Computer history museum
[https://computerhistory.org/]

A doua generație

1947 : Inventarea tranzistorului, în anul 1947, de către John Bardeen, Walter Brattain și William Shockley în 1948 la Bell Labs.

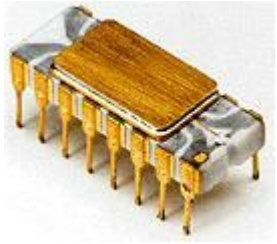


1965 : Mini calculatorul PDP 8 1MHz, 790 W, 1m^2
Memorie de date de 4096.

Mini calculatorul PDP [1]

A treia generaţie

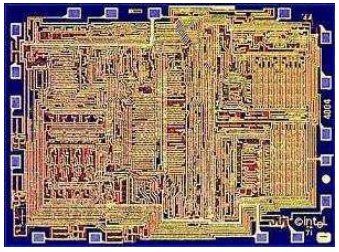
1961 Fairchild Semiconducteur, circuite integrate



1965 : Legea lui Moore:

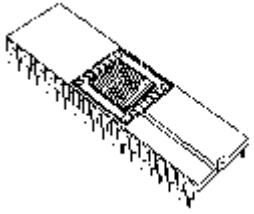
Complexitatea circuitelor x 2 / 1,5 ani

1971 : Intel a lansat primul microprocesor i4004 (60000 instrucțiuni /s , frecvența 108 KHz, 2300 tranzistoare.



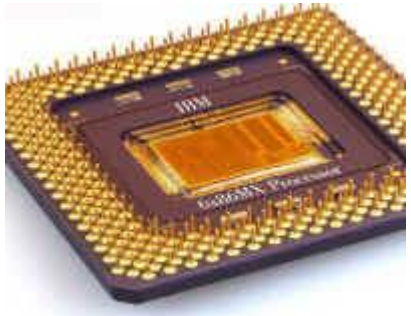
1978 : Intel a lansat microprocesorul 8086 330 000 instrucțiuni /s

1981 : IBM PC



A patra generație

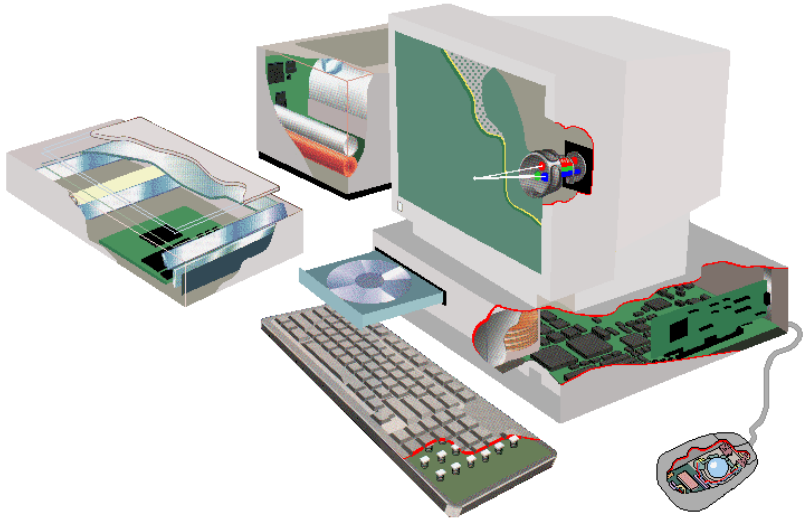
1985 : Intel 80386, 1 milion de instrucțiuni/s, 200.000 transistoare



1986 : Primele mașini paralele

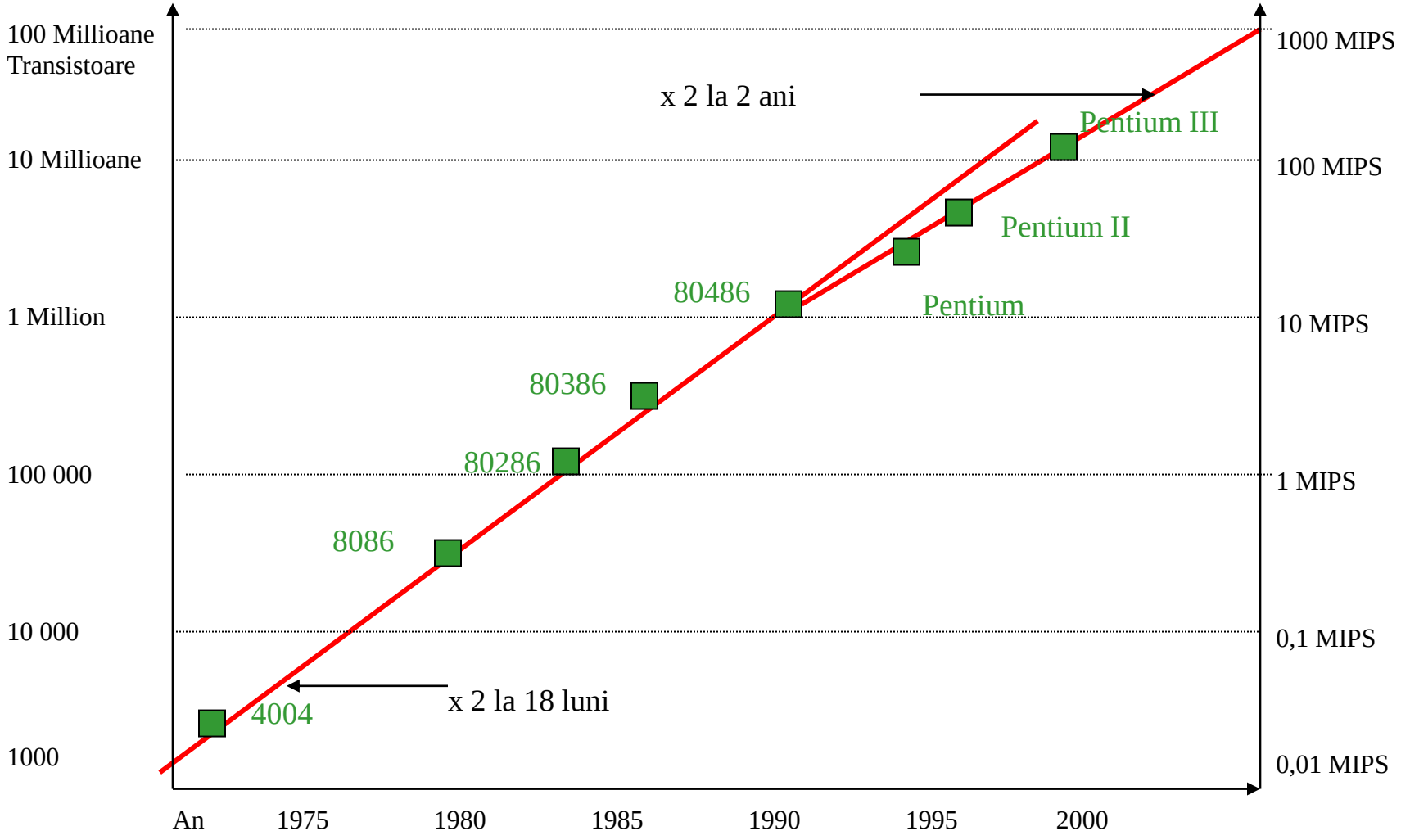
1990 : Calculatoare multimedia

2000 : Intel a lansat Pentium IV



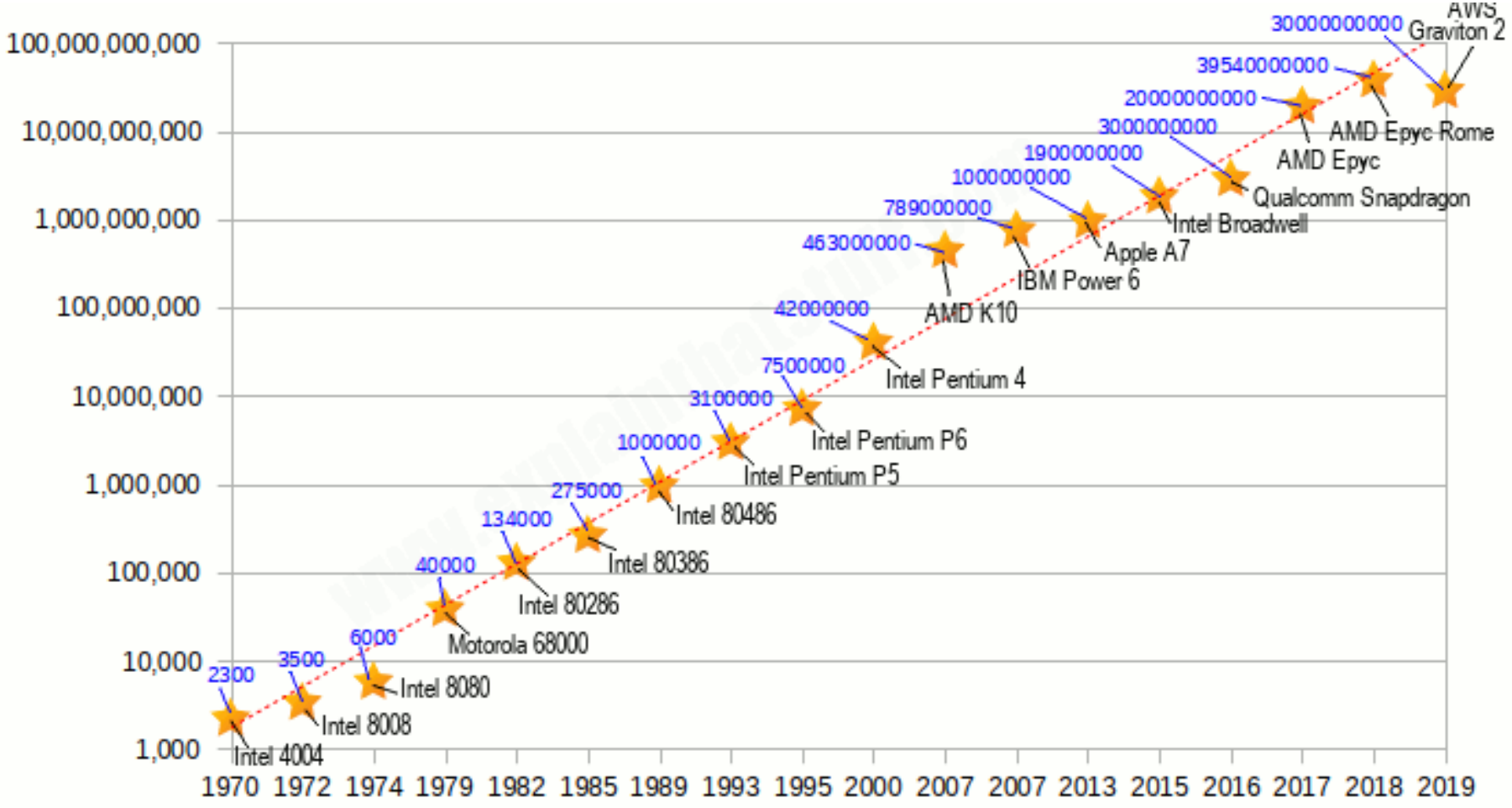
Calculatoare cu microprocesoare [1]

Legea lui MOORE



Dublarea numărului de tranzistoare [1]

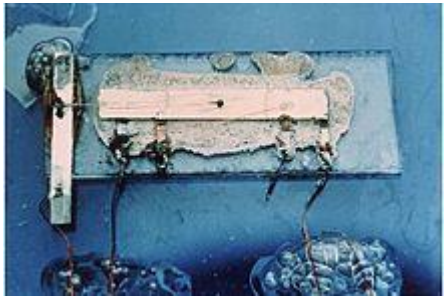
Legea lui Moore – 50 ani



www.explainthatsuff.com

Tehnologia CI

- Primul CI Jack Kilby, inginer la Texas Instruments 1958.
- Elementul de bază tipic al unui CI este tranzistorul.
- Procedeele tehnologice specifice sunt depunerea, fotolitografia, corodarea, difuzia și implantarea ionică



https://www.edn.com/

10 µm – 1971
6 µm – 1974
3 µm – 1977
1.5 µm – 1981
1 µm – 1984
800 nm – 1987
600 nm – 1990
350 nm – 1993
250 nm – 1996
180 nm – 1999
130 nm – 2001
90 nm – 2003
65 nm – 2005
45 nm – 2007
32 nm – 2009
22 nm – 2012
14 nm – 2014
10 nm – 2016
7 nm – 2018
5 nm – 2020

Future
3 nm – ~2022
2 nm – ~2023<

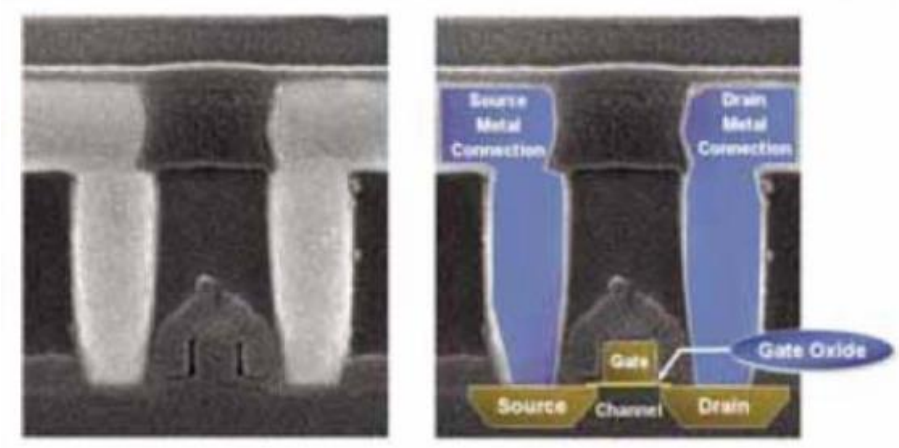
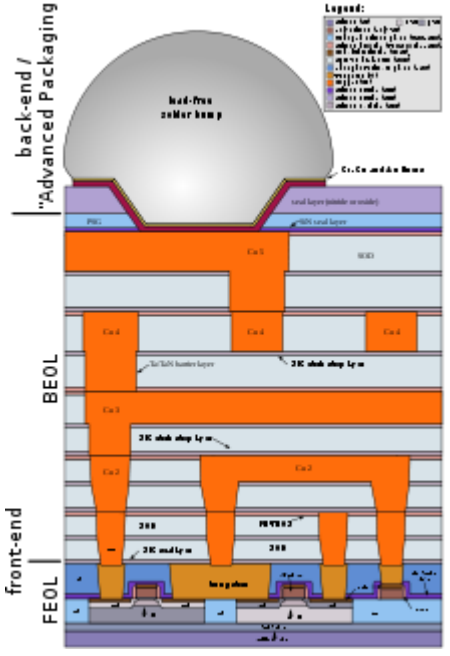


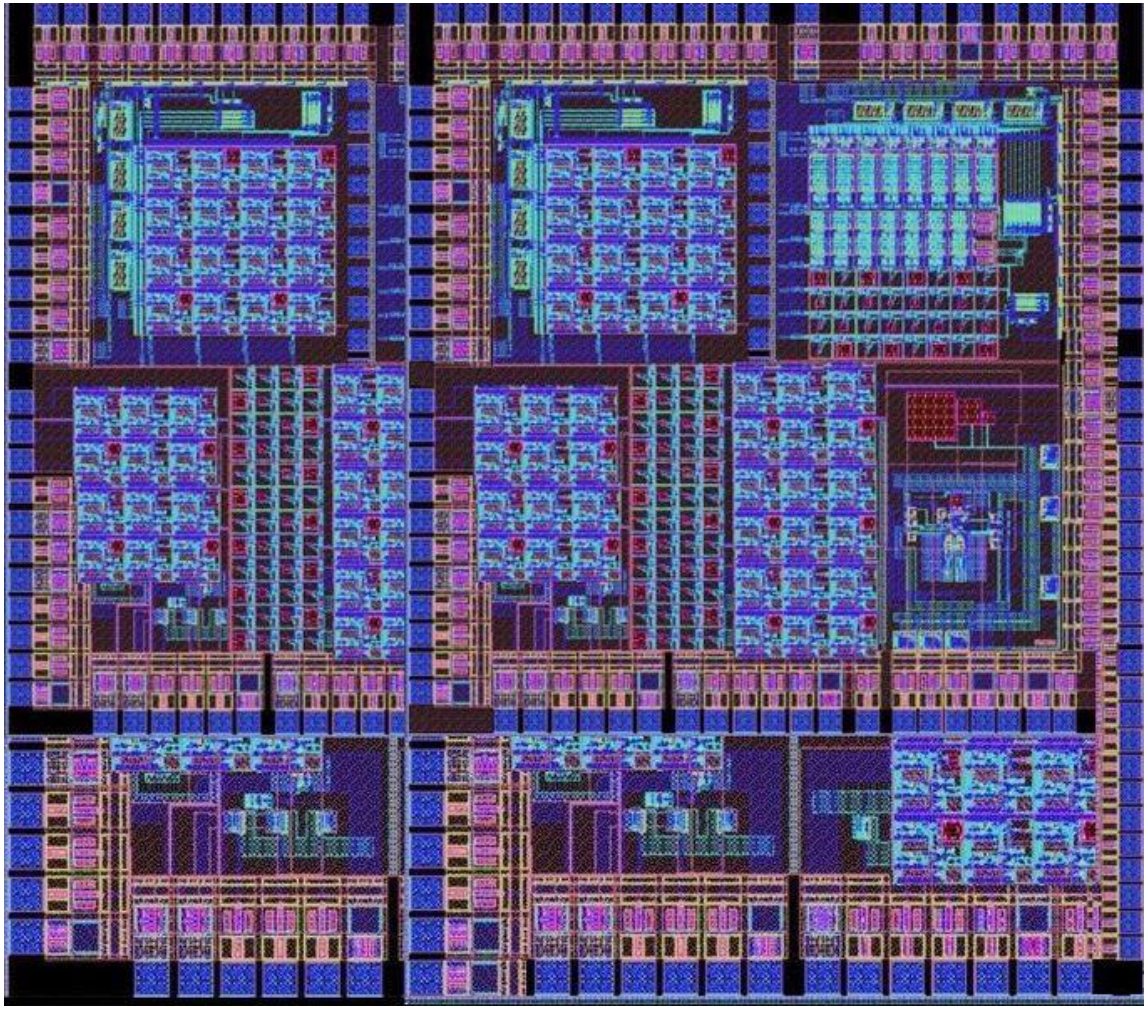
Figure 3a, 3b – Middle oxide thickness Virtex-4 transistor used in triple-oxide process and with highlighted portions of the transistors

[Xcell Journal Issue 52, 2005]



https://en.wikipedia.org/wiki/Back_end_of_line

Imaginea mărită CI



Wikimedia Commons, Picture by Angeloleithold

Introducere în conceptele digitale

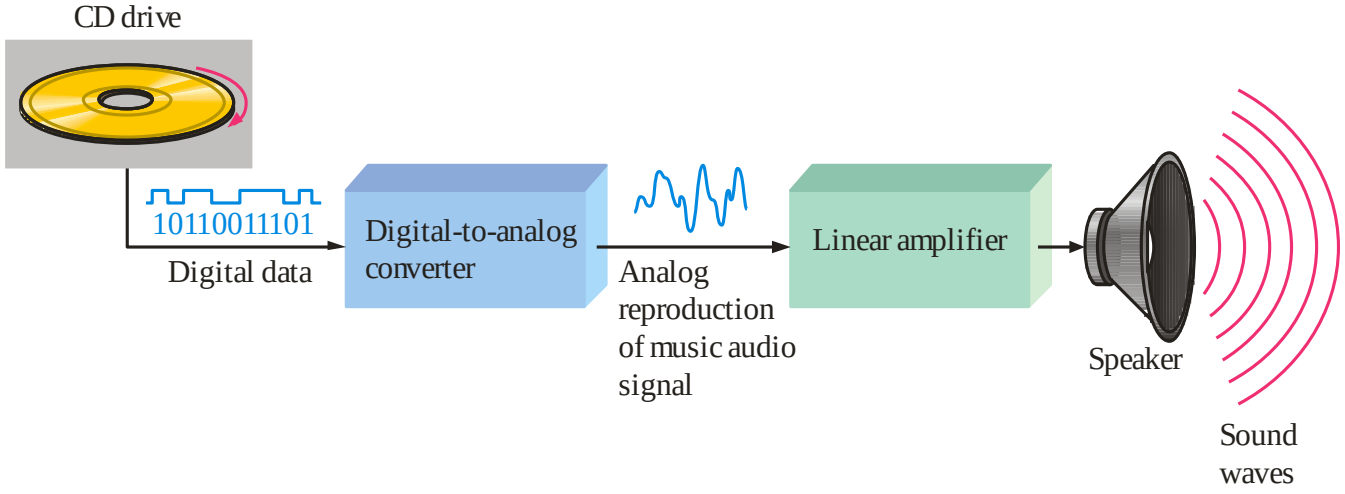
STRUCTURI ELEMENTARE INTEGRATE

- circuite integrate pe scară mică, **SSI** ce includ până la 12 porți logice;
- circuite integrate pe scară medie, **MSI** cu o capacitate între 12 și 100 de porți;
- circuite integrate pe scară largă, **LSI** ce cuprind între 100 și 100.000 de porți logice;
- circuite integrate pe scară foarte largă, **VLSI** ce conțin peste 100.000 porți.

Sisteme analogice și digitale

• Multe sisteme folosesc circuite mixte analogice și digitale.

• Un CD player acceptă la intrare semnalul digital și îl convertește în semnal analogic pe care îl amplifică.



Arhitectura unui sistem analog-digital [2]

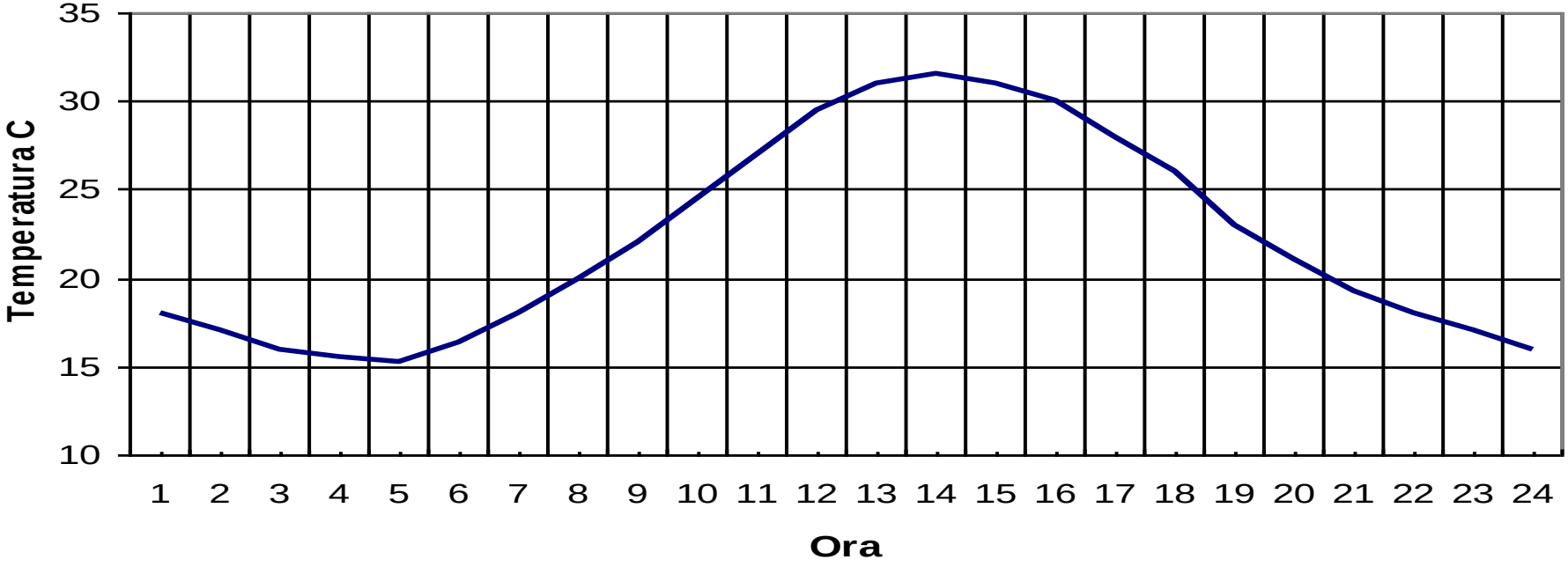
Mărimi analogice

•Circuitele electronice pot fi clasificate în:

circuite digitale – mărimi discrete

circuite analogice - mărimi continue.

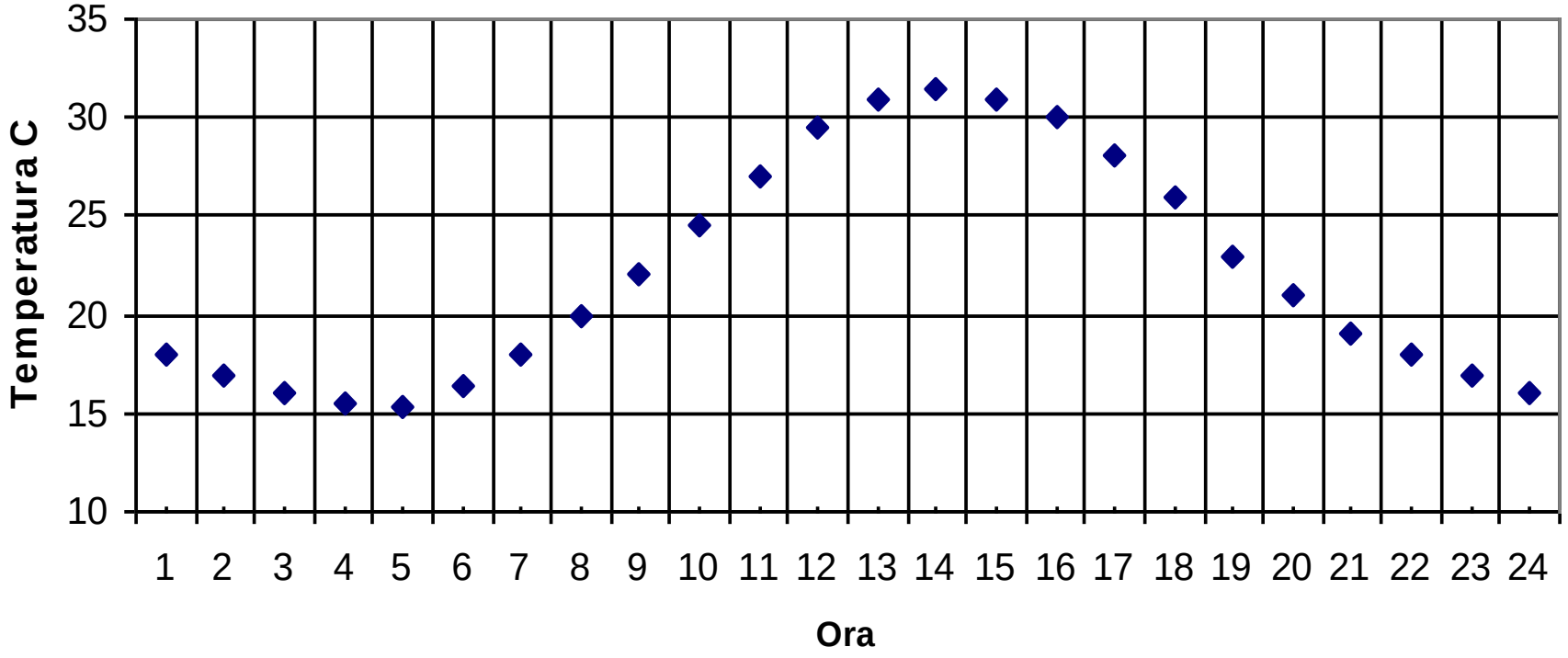
Majoritatea mărimilor din natură sunt mărimi analogice, lumea reală este preponderent analogică.



De exemplu temperatura aerului se modifică într-un domeniu continuu de valori => curba este continuă

Mărimi discrete

În loc să reprezentăm temperatura în mod continuu să presupunem că înregistrăm temperatura odată pe oră. Rezultă o nouă curbă cu valori eșantionate care reprezintă temperatura în momente discrete de timp (la fiecare oră) într-un interval de 24 ore după cum se poate observa în figură



Avantajele tehnicii digitale

- Datele digitale pot fi procesate şi transmise mai ușor și mai exact.
- Circuitele digitale sunt mai ușor de proiectat.
- Imunitatea la perturbații este considerabil mai mare decât cea a circuitelor analogice.
- Stocarea informației este mai ușoară.
  - De exemplu muzica sub formă digitală poate fi memorată mai compact şi reprodusă cu acuratețe mai mare.
- Siguranță mare în funcționare.
  - Circuitele digitale funcționează corect chiar în condițiile în care parametrii electrici ai elementelor componente variază considerabil.
- Exactitate. În sistemele numerice lipsesc punctele de ajustare sau circuitele de compensare.
- Afișare digitală.
- Operațiile pot fi programate.
- Multe circuite digitale se pot fabrica sub formă integrată.

Dezavantajele tehnicii digitale

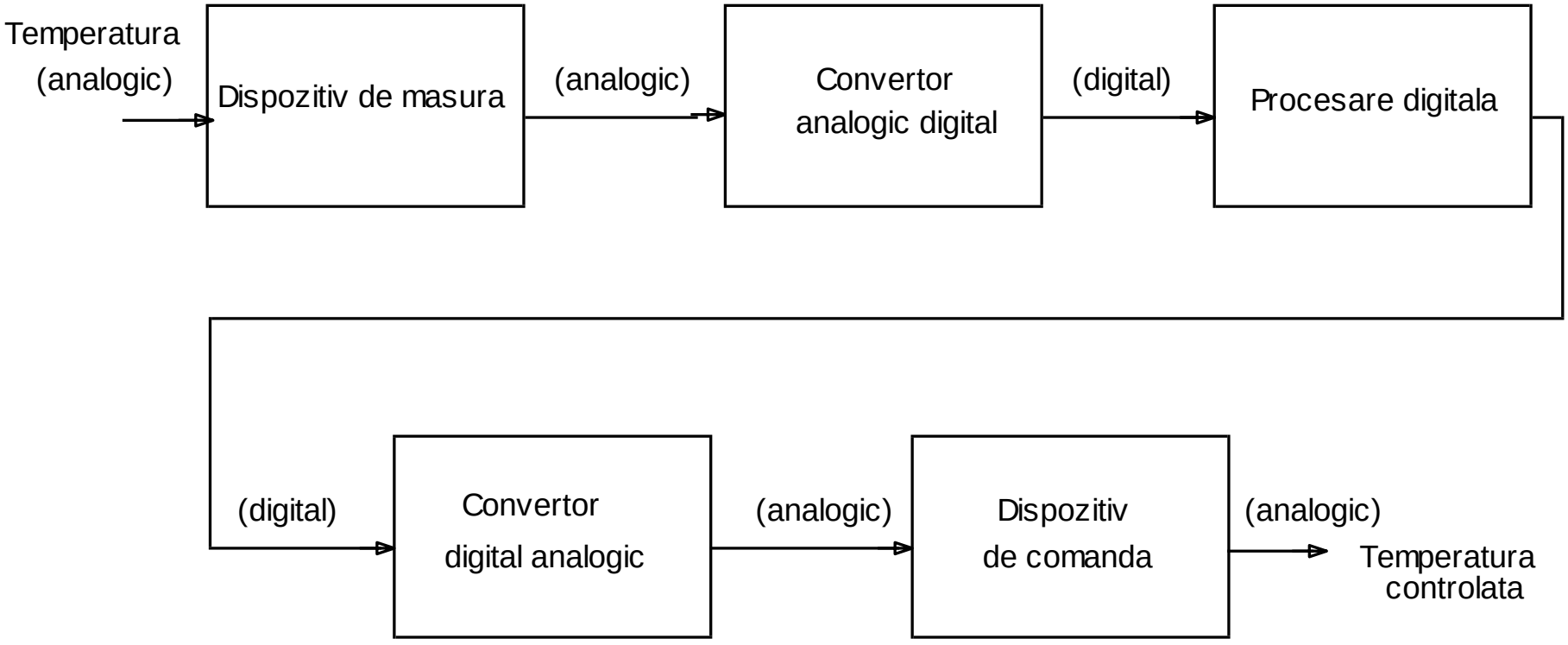
Există un singur dezavantaj major al tehnicii digitale

Lumea reală este preponderent analogică.

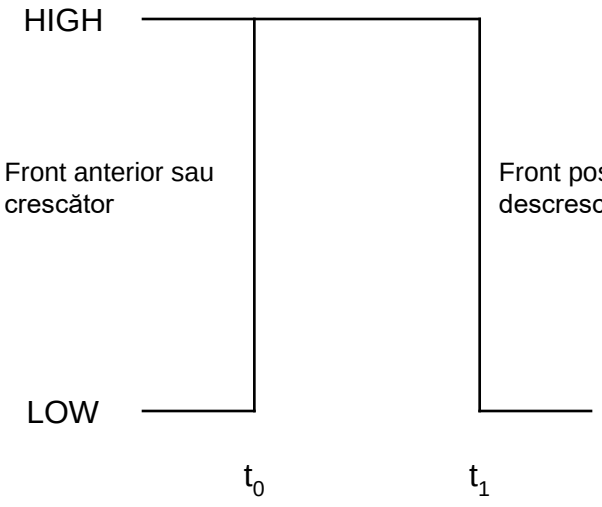
Conversia A-D, D-A

- Pentru a putea beneficia de avantajele tehnicii digitale atunci când lucrăm cu semnale de intrare și de ieșire analogice trebuie să parcurgem trei pași:
- Convertirea semnalelor din lumea reală analogice în formă digitală Această operație se numește conversie analog numerică (sau analog digitală).
- Prelucrarea datelor sub formă digitală
- Convertirea semnalului de ieșire spre lumea reală din formă digitală în formă analogică. Acest procedeu se numește conversie numeric (digital) analogică

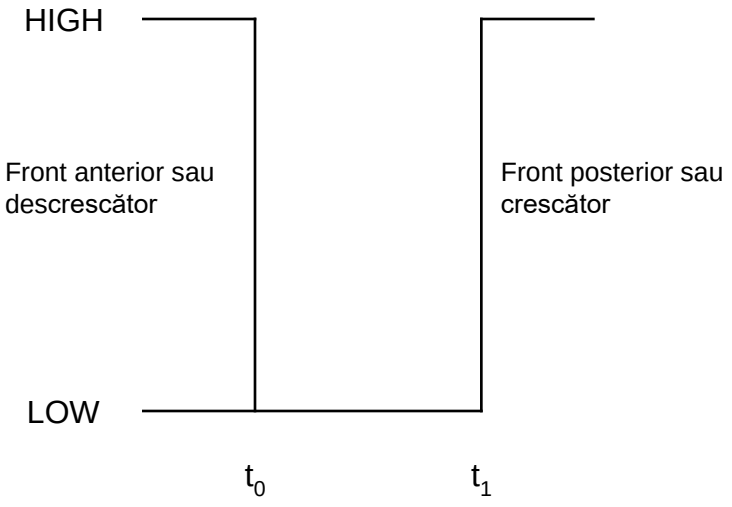
Sistem de control digital al temperaturii



Impulsuri ideale

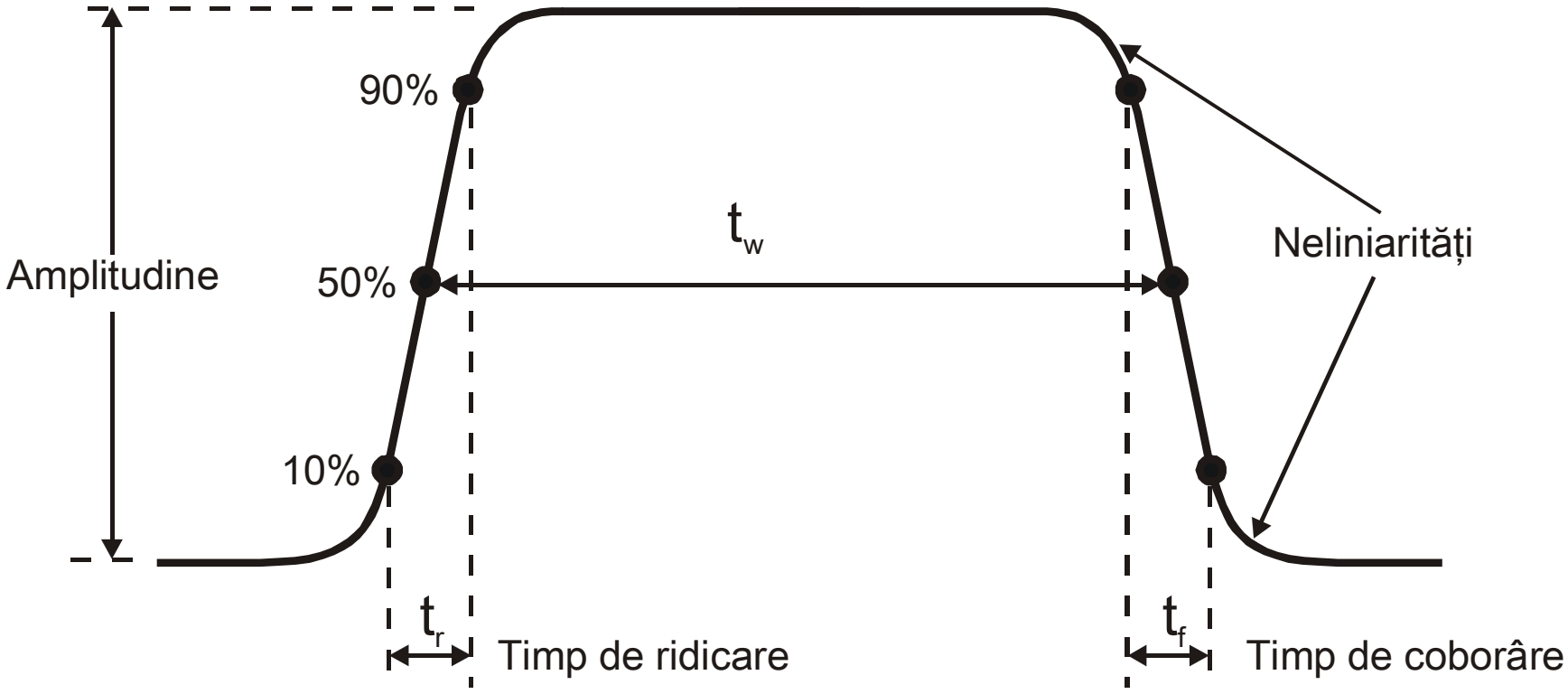


a) Impuls pozitiv

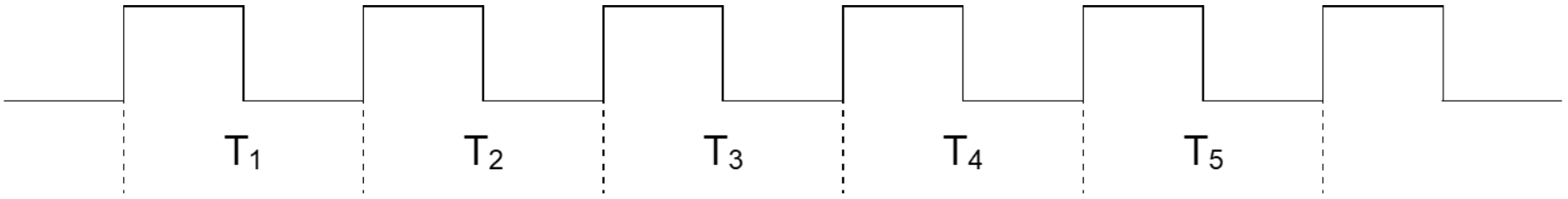


b) Impuls negativ

Impulsuri reale

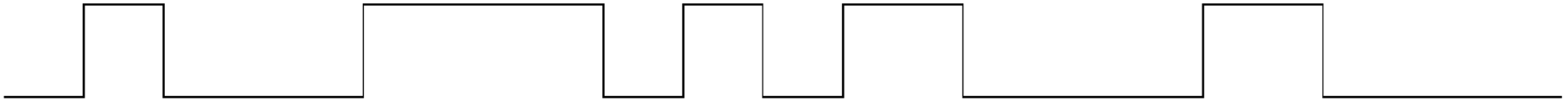


Exemple de semnale digitale



Perioada $T_1 = T_2 = T_3 = T_4 = T_5$

a) Semnal digital periodic (Impulsuri dreptunghiulare)



b) Semnal digital neperiodic

Perioada și factorul de umplere

$$f = \frac{1}{T} \text{ respectiv } T = \frac{1}{f} .$$

O caracteristică importantă a unui semnal digital periodic este factorul de umplere definit ca raportul dintre lăţimea impulsului (tw) şi perioada (T) şi care poate fi exprimată procentual:

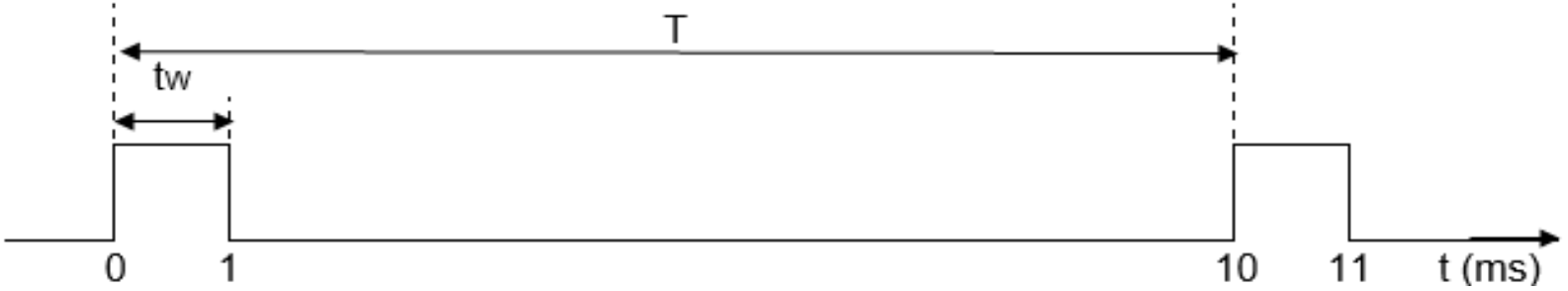
$$\text{Factor de umplere} = \frac{t_w}{T} 100\%.$$

De exemplu pentru semnalul digital prezentat în figură, perioada T este egală cu 10 ms. Frecvenţa acestui semnal este:

$$f = \frac{1}{T} = \frac{1}{10ms} = 100Hz .$$

Factorul de umplere este:

$$\text{Factor de umplere} = \frac{t_w}{T} 100\% = = \frac{1ms}{10ms} 100\% = 10\%.$$



Reprezentarea informației

Sisteme de numerație

- Cele mai utilizate sisteme de numerație:

	baza	simboluri
• Zecimal	B = 10	0, 1, …8, 9
• Binar	B = 2	0, 1
• Octal	B = 8	0, 1,…6, 7
• Hexazecimal	B = 16	0, 1, …. 9, A, B, C, D, E, F

- Cele mai utilizate sisteme sunt cel zecimal, binar, și hexazecimal.

Reprezentarea numerelor în diferite sisteme de numerație

Numărul N reprezentat în baza R:

$$N_R = \pm \sum_{k=-h}^{n-1} A_k \cdot R^k$$

Aici

$$N_{\hat{i}ntreg} = A_{n-1}R^{n-1} + \ldots + A_1 R + A_0$$

Reprezintă partea întreagă, și

$$N_{fractionar} = A_{-1}R^{-1} + \ldots + A_{-h+1}R^{-h+1} + A_{-h}R^{-h}$$

Partea fracționară.

$$N_R = A_{n-1}\ldots A_1 A_0, A_{-1}\ldots A_{-h-1}A_{-h}(R)$$

Exemple

Reprezentarea unui număr zecimal:

- $N = 6543_{10} = 6 \cdot 10^3 + 5 \cdot 10^2 + 4 \cdot 10^1 + 3 \cdot 10^0 = 6000+500+40+3 = 6543$
- $N = 65,43_{10} = 6 \cdot 10^1 + 5 \cdot 10^0 + 4 \cdot 10^{-1} + 3 \cdot 10^{-2} = 60+5+0,4+0,03 = 65,43$

Reprezentarea unui număr binar:

- $10101101_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
- $1011,1001_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}$

Simboluri utilizate de diferite sisteme de numerație

Zecimal	Binar	Octal	Hexazecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	10000	20	10

Sistemul binar

- **În tehnica digitală, informația este reprezentată în cod binar.**
- Unitatea de informație este bitul, **Bi**nary Digi**t.** Simbol **b** (b mic)
- Valorile posibile ale unui bit sunt 0 sau 1
- **Poate reprezenta două stări:**
- Ex..
  - Adevărat sau Fals,
  - Da sau Nu,
- Nu există informație mai mică decât 1 bit.

$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	Echivalent zecimal
0	0	0	0	0
0	0	0	**1**	1
0	0	**1**	0	2
0	0	**1**	**1**	3
0	**1**	0	0	4
0	**1**	0	**1**	5
0	**1**	**1**	0	6
0	**1**	**1**	**1**	7
1	0	0	0	8
1	0	0	**1**	9
1	0	**1**	0	10
1	0	**1**	**1**	11
1	**1**	0	0	12
1	**1**	0	**1**	13
1	**1**	**1**	0	14
1	**1**	**1**	**1**	15

……..	2^3	2^2	2^1	2^0	,	2^{-1}	2^{-2}	2^{-3}	……..
	8	4	2	1		1/2	1/4	1/8	
Bitul cel mai semnificativ					Virgulă binară				Bitul cel mai puțin semnificativ

Conversia din diverse sisteme de numerație în zecimal

Conversia binar – zecimal:

$N = 110001_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32+16+1 = 49$

$N = 100{,}011_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 4+0{,}25+0{,}125 = 4{,}375$

Conversia octal– zecimal :

$N = 24{,}6_8 = 2 \times (8^1) + 4 \times (8^0) + 6 \times (8^{-1}) = 2 \times 8 + 4 \times 1 + 6 \times 0{,}125 = 20{,}7510$

Conversia hexazecimal – zecimal:

$2AF_{16} = 2 \times (16^2) + 10 \times (16^1) + 15 \times (16^0) = 2 \times 256 + 10 \times 16 + 15 \times 1 = 687_{10}$

Exerciții:

$110101{,}011_{(2)} = ?_{(10)}$

$101010{,}011_{(2)} = ?_{(10)}$

$1CD{,}A2_{(16)} = ?_{(10)}$

$467{,}57_{(8)} = ?_{(10)}$

Conversia din zecimal în alte sisteme de numerație

1. Conversia părții întregi

Conversia părții întregi se face prin împărțiri repetate cu baza în care se trece, până ce rezultatul devine =0

Sensul de citire a rezultatului este de la ultimul rest spre primul

Conversia unui număr zecimal întreg în binar

exemplu: $84=?_{(2)}$

84	:2
42	0
21	0
10	1
5	0
2	1
1	0
0	1

Sensul de citirie

Deci: $84=1010100_{(2)}$

Conversia din zecimal în alte sisteme de numerație (2)

Conversia unui număr zecimal întreg în octal

Exemplu: $177=?_{(8)}$

177/8	= 22+ rest **1**	(Bitul cel mai puțin semnificativ - LSB)
22/ 8	= 2 + rest 6	
2 / 8	= 0 + rest **2**	(Bitul cel mai semnificativ - MSB)

Sensul de citire

Deci: $177_{10}=261_{(8)}$

Conversia unui număr zecimal întreg în hexazecimal

Exemplu: $378=?_{(16)}$

378/16	= 23+ rest **10**	**A** (Bitul cel mai puțin semnificativ - LSB)
23/ 16	= 1 + rest 7	**7**
1 / 16	= 0 + rest **1**	***1*** (Bitul cel mai semnificativ - MSB)

Sensul de citire

Deci: $378_{10}=17A_{(16)}$

Conversia din zecimal în alte sisteme de numerație (3)

2. Conversia părții fracționare:

Partea fracționară se înmulțește repetat cu 2, până ce rezultatul înmulțirii devine 1,0. Înmulțirea se face numai cu partea fracționară.

Conversia unui număr zecimal fracționar în binar

Exemplu: $0,3125=?_{(2)}$

	2·	0,3125
	0	,625
	1	,25
Sensul de citire ↓	0	,5
	1	,0

Deci: $0,3125=0,0101_{(2)}$

Exerciții:

$97,375_{(10)}=?_{(2)}$

$475,8\ 35\ 937\ 510_{(10)}=?_{(2)}$

Conversia din binar în hexazecimal și invers

Se înlocuiește fiecare simbol hexazecimal cu grupul de 4 biți echivalenți.

$5A,8_{16} = (0101)\ (1010)\ ,\ (1000)_2 = 1011010,1_2$

Se împarte numărul binar în grupe de 4 caractere începând cu bitul cel mai din dreapta, și se înlocuiește grupul de 4 biți cu echivalentul lui hexazecimal

Grupul din stânga respectiv din dreapta se poate completa cu zerouri pentru a forma un grup de 4 biți.

$101\ 1010\ ,11_2 = (\mathbf{0}101)\ (1010)\ ,\ (11\mathbf{00})_2 = 5A,C_{16}$

Hex	Nible
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Conversia octal-binar și binar-octal

Se înlocuiește fiecare simbol octal cu grupul de 3 biți echivalenți.

$53,6_8 = (101)\ (011)\ ,\ (110)_2 = 101011,11_{2/}$

Se împarte numărul binar în grupe de 3 caractere începând cu bitul cel mai din dreapta, și se înlocuiește grupul de 3 biți cu echivalentul lui octal

Grupul din stânga respectiv din dreapta se poate completa cu zerouri pentru a forma un grup de 3 biți.

$1110,1_2 = (001)\ (110)\ ,\ (100)_2 = 16,4_8$

Octal	
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Exerciții:

$1101001101,01101_2 = ?_8 = ?_{16}$,

$E37,1A_{16} = ?_2 = ?_8$

EXEMPLU

Să se scrie numărul zecimal 725,9375 în binar trecând prin sistemul octal:

Conversia în octal:

a. Conversia părții întregi:

725	:8
90	5
11	2
1	3
0	1

Sensul de citire

b. Conversia părții zecimale:

8·	0,9375
7	,5000
4	,0000

Formatul octal: $1325,74_{(8)}$

Conversia în binar are loc prin înlocuirea cifrelor octale cu echivalentul lor binar:

$1325,74_{(8)}$=001 011 010 101,111 $100_{(2)}$ = =$1011010101,1111_{(2)}$

Codul BCD

BCD = Binary Coded Decimal (zecimal codificat în binar) – *se utilizează în tehnica de calcul pentru reprezentarea numerelor zecimale*

•Reprezentarea fiecărui număr se face prin reprezentarea fiecărei cifre în binar pe 4 biți

•Ex. 1956 nu se reprezintă în binar adică sub forma (11110100100)

• Fiecare cifră a numărului 1, 9, 5, și 6 este înlocuită cu echivalentul binar pe 4 biți:
0001, 1001, 0101, 0110.

• În formatul BCD compact se poate reprezenta pe doi octeți:
00011001 și 01010110

	BCD			
	8	4	2	1
0	***0***	***0***	***0***	***0***
1	***0***	***0***	***0***	***1***
2	***0***	***0***	***1***	***0***
3	***0***	***0***	***1***	***1***
4	***0***	***1***	***0***	***0***
5	***0***	***1***	***0***	***1***
6	***0***	***1***	***1***	***0***
7	***0***	***1***	***1***	***1***
8	***1***	***0***	***0***	***0***
9	***1***	***0***	***0***	***1***
neutilizat	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Reprezentarea numerelor cu semn

– ***Bitul de semn*** este bitul aflat în poziția cea mai din stânga într-un număr binar cu semn și indică dacă numărul este pozitiv sau negativ.

0 pentru numere pozitive și 1 pentru numere negative

– Cele mai utilizate moduri de reprezentare a numerelor cu semn sunt:

a) *semn - mărime*,
b) în *complement față de 1*
c) în *complement față de 2*.

a) **Reprezentarea numerelor în semn - mărime**

Semn	Mărime
1 bit	n biţi

= primul bit reprezintă semnul, următoarele valoare absolută

Ex.: $+85_{10}=01010101_2$ $-85_{10}=11010101_2$

Reprezentarea numerelor reale cu semn:

S	Mărime partea întreagă	,	Mărime partea fracţionară
1	n1	1	n2

Exemplu: $r = 45{,}34375_{10} = 0\ 101101{,}01011$

0	101101	,	01011

Reprezentarea numerelor cu semn

b) **Reprezentarea numerelor cu semn în complement față de 1**

Pentru numere pozitive reprezentarea în complement față de 1 este identică cu reprezentarea în semn - mărime. În cazul numerelor negative acestea se reprezintă prin complementul față de 1 a numărului pozitiv corespondent.

În complementul fată de 1 un număr negativ se reprezintă prin complementul faţă de 1 a numărului pozitiv corespondent.

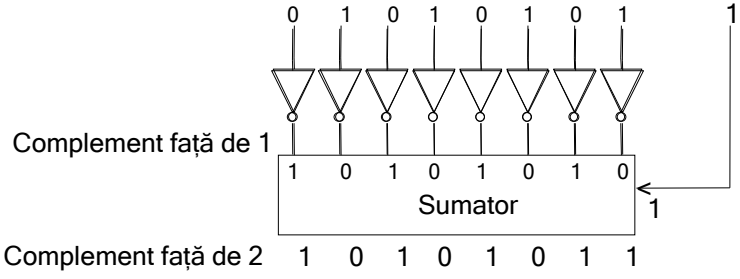
Ex.: $+85_{10}=01010101_2$ $-85_{10}=10101010_2$

c) **Reprezentarea numerelor cu semn în complement față de 2** =

Pentru numere pozitive reprezentarea în complement față de 2 este identică cu reprezentarea în semn - mărime şi cu reprezentarea în complement față de 1. Numerele negative se reprezintă prin complementul față de 2 a numărului pozitiv corespondent.

În complementul față de 2 un număr negativ se reprezintă prin complementul față de 2 a numărului pozitiv corespondent.

Ex.: $+85_{10}=01010101_2$ $-85_{10}=10101011_2$



Valoarea zecimală a numerelor cu semn

Valoarea zecimală a numerelor pozitive și negative în reprezentate în complement față de 2 se determină prin însumarea ponderilor biților care au valoarea 1, și ignorând pe cei ce sunt 0. Ponderea bitului de semn pentru un număr negativ se ia cu valoare negativă.

Ex. Fie numărul: $01111010_2 = +122_{10}$

Complementul față de 2: 10000110 = -122

Valoarea zecimală este:

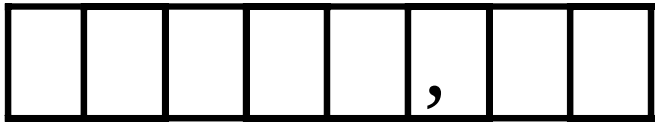
Ponderea coloanei : -128 64 32 16 8 4 2 1

1 0 0 0 0 1 1 0

-128 +4 +2 = -122

Reprezentarea numerelor binare

- **Reprezentarea în virgulă fixă**



Folosind 8 biți se pot reprezenta 256 numere diferite. Cu ajutorul a doi octeți, deci 16 biți, se pot reprezenta 65536 numere diferite. Numărul total de combinații diferite cu *n* biți este 2^n.

La reprezentarea numerelor binare cu semn în complement fată de 2, domeniul valorilor posibile pentru un număr reprezentat pe n biți este:

$$-(2^{n-1}) \text{ la } + (2^{n-1} - 1)$$

unde 1 bit este de semn şi n-1 biți reprezintă mărimea.

Ex. - cu 4 biți se pot reprezenta numere între $-(2^3) = -8$ la $(2^3) - 1 = 7$.

- cu 8 biți se pot reprezenta numere între -128 şi +127, ş.a.m.d.

Reprezentarea în virgulă mobilă

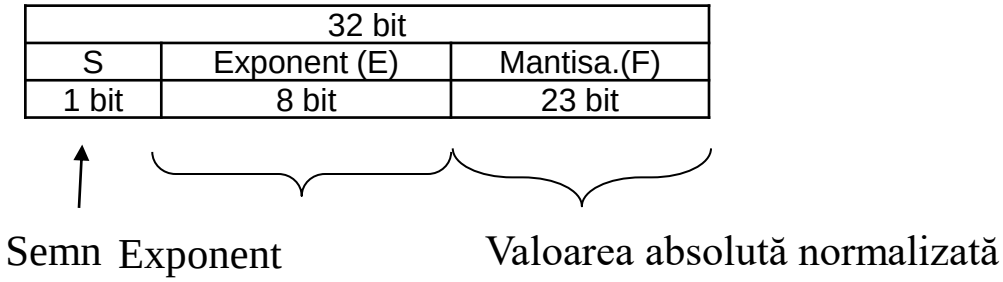
Standardul ANSI/IEEE 754-1985 în trei forme:

–*simplă-precizie(32 biți)*,

–*dublă-precizie (64 biți)*,

–*precizie extinsă(80 biți)*.

Reprezentarea numerelor binare în virgulă flotantă simplă precizie



Ex.: 1011010010001

Exemplu de reprezentare în virgulă mobilă

Ex. : $1011010010001 = 1{,}011010010001 \times 2^{12}$

Exponentul, =12, se exprimă în forma deplasată prin adunarea lui 127

Mantisa este partea fracționară a numărului binar, .011010010001.

Numărul se exprimă pe 24 de biți din care de fapt 23 reprezintă partea de după virgulă

E = 12+127=139 = 10001011 Mantisa : ,011010010001

S	E	F
0	10001011	01101001000100000000000

Evaluare a unui număr binar exprimat în virgulă flotantă

$$N = (-1)^S(1+F)(2^{E-127})$$

Pl. :

1	10010001	10001110001000000000000

$N = (-1)^1(1{,}10001110001)(2^{145-127}) = (-1)^1(1{,}10001110001)(2^{18}) = -110001110001000000 = 407.680_{10}$

Coduri detectoare și corectoare de erori

Metoda parității pentru detecția erorilor:

- *paritate pară*
- *paritate impară*

Paritate pară		Paritate impară	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	0100	0	0100
0	1001	1	1001

Dezavantaje:
- Nu permite detectarea poziției eronate
- Nu permite corecția erorii
- paritate permite detecția unei erori care apare la un singur bit (sau la un număr impar de erori):

- Coduri corectoare de erori:
  - Codul Hamminng
  - Codul Red Solomon

Coduri alfanumerice

- ASCII (American Code for Information Interchange)
  - 128 de caractere şi simboluri reprezentate de un cod pe 7 biți, (8 biţi având bitul cel mai semnificativ întotdeauna zero de la 00 la 7F.
  - 32 de caractere sunt caractere nongrafice care nu se afișează şi nu se tipăresc niciodată şi reprezintă comenzi. Ex.: „rând nou”, „început de text”, „escape”.
  - caractere sunt caractere grafice conțin literele alfabetului englezesc (litere mari şi mici), cele 10 cifre zecimale, semne de punctuație, şi alte simboluri uzuale.

ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX
NULL	0	00	(SP)	32	20	@	64	40	`	96	60
SOH	1	01	!	33	21	A	65	41	a	97	61
STX	2	02	"	34	22	B	66	42	b	98	62
ETX	3	03	#	35	23	C	67	43	c	99	63
EOT	4	04	$	36	24	D	68	44	d	100	64
ENQ	5	05	%	37	25	E	69	45	e	101	65
ACK	6	06	&	38	26	F	70	46	f	102	66
BEL	7	07	'	39	27	G	71	47	g	103	67
BS	8	08	(	40	28	H	72	48	h	104	68
HT	9	09	)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	[	91	5B	{	123	7B
FS	28	1C	<	60	3C	\	92	5C	|	124	7C
GS	29	1D	=	61	3D	]	93	5D	}	125	7D
RS	30	1E	>	62	3E	^	94	5E	~	126	7E
US	31	1F	?	63	3F	_	95	5F	(sp)	127	7F

Codul ASCII extins

- 128 de caractere adiționale - cod de 8 biți de la 80 la FF hexazecimal.
- Codul ASCII extins conțin următoarele categorii:
  - Caractere ale altor alfabete (în afară de cel englez)
  - Simboluri ale monedelor străine
  - Literele grecești
  - Simboluri matematice
  - Caractere grafice pentru desen

128	€	144	□	160		176	°	193	Á	209	Ñ	225	á	241	ñ
129	□	145	‘	161	¡	177	±	194	Â	210	Ò	226	â	242	ò
130	,	146	’	162	¢	178	²	195	Ã	211	Ó	227	ã	243	ó
131	*ƒ*	147	“	163	£	179	³	196	Ä	212	Ô	228	ä	244	ô
132	„	148	”	164	¤	180	´	197	Å	213	Õ	229	;	245	õ
133	…	149	•	165	¥	181	µ	198	Æ	214	Ö	230	æ	246	ö
134	†	150	–	166	¦	182	¶	199	Ç	215	×	231	ç	247	÷
135	‡	151	—	167	§	183	·	200	È	216	Ø	232	è	248	ø
136	ˆ	152	˜	168	¨	184	¸	201	Ì	217	Ù	233	é	249	ù
137	‰	153	™	169	©	185	¹	202	Ê	218	Ú	234	ê	250	ú
138	Š	154	š	170	ª	186	º	203	Ë	219	Û	235	ë	251	û
139	‹	156	œ	171	«	187	»	204	Ì	220	Ü	236	ì	252	ü
140	Œ	157	□	172	¬	188	¼	205	Í	221	Ý	237	í	253	ý
141	□	158	ž	173		189	½	206	Î	222	Þ	238	î	254	þ
142	Ž	159	Ÿ	174	®	190	¾	207	ï	223	ß	239	ï	255	ÿ
143	□	192	À	175	¯	191	¿	208	Ð	224	à	240	ð		

Elemente de algebra Booleană

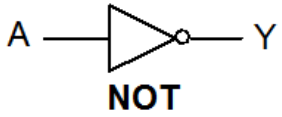
Algebra Booleană

- A primit numele după matematicianul englez George Boole (1815-1864)
- Operează cu propoziții despre care are sens să afirmăm că sunt adevărate sau false
- Convenim ca
  - unei propoziții adevărate să-i atribuim valoarea binară “1”,
  - iar falsitatea acesteia să o notăm cu valoarea binară “0”.
- Variabilele sunt reprezentate prin litere, valoarea lor putând fi 0 sau 1
- Propoziția compusă a cărei valoare depinde de valorile propozițiilor simple putând avea tot două valori, se numește **funcție logică** sau **funcție binară**

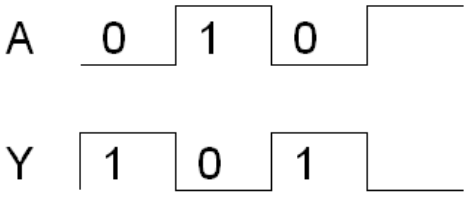
Funcții logice I

- **Funcția logică NU (NOT, negație logică):** $Y = \overline{A} =$
  - 1 dacă A = 0
  - 0 dacă A = 1

$Y = \overline{A}$ *(non A)*

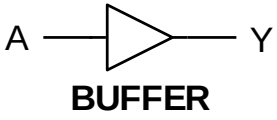


A	NU ($\overline{A}$)
0	1
1	0

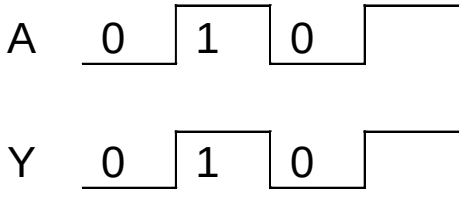


- **Buffer neinversor (tampon):** $Y = A =$
  - 0 dacă A = 0
  - 1 dacă A = 1

$Y = A$



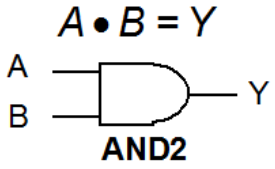
A	Y=A
0	0
1	1



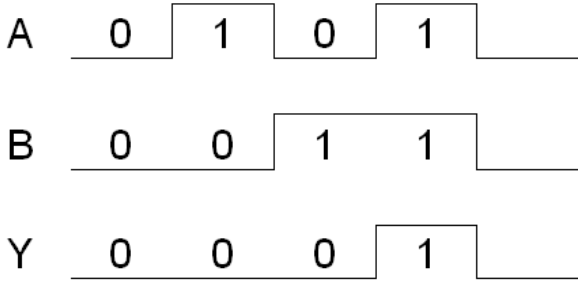
Ieșirea bufferului are aceeași valoare logică ca și intrarea lui dar are capabilitate de curent mai mare

Funcții logice II

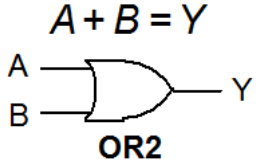
- **ȘI logic (AND) (produs logic, conjuncția):** Y = AB =
  - 1 dacă A = 1 ȘI B = 1
  - 0 în rest



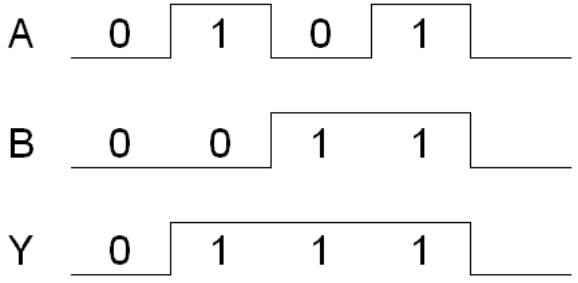
A	B	ŞI
0	0	0
0	1	0
1	0	0
1	1	1



- **SAU logic(OR) (suma logică, disjuncția):** Y = A+B =
  - 0 dacă A = 0 ȘI B = 0
  - 1 în rest

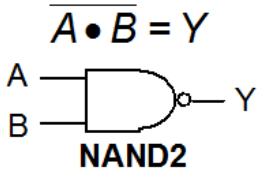


A	B	SAU
0	0	0
0	1	1
1	0	1
1	1	1

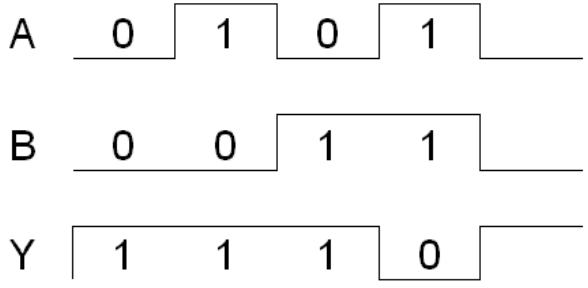


Funcții logice III

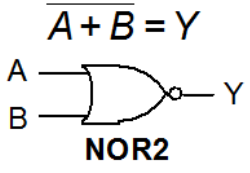
- **ȘI-NU logic (NAND) :** $Y = \overline{AB} =$
  - 0 dacă A = 1 ȘI B = 1
  - 1 în rest



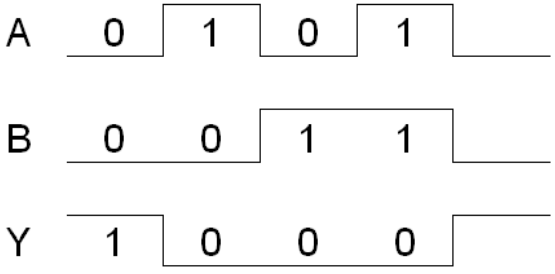
A	B	ŞI-NU
0	0	1
0	1	1
1	0	1
1	1	0



- **SAU-NU (NOR):** $Y = \overline{A+B} =$
  - 1 dacă A = 0 ȘI B = 0
  - 0 în rest

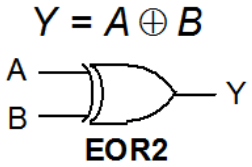


A	B	SAU-NU
0	0	1
0	1	0
1	0	0
1	1	0

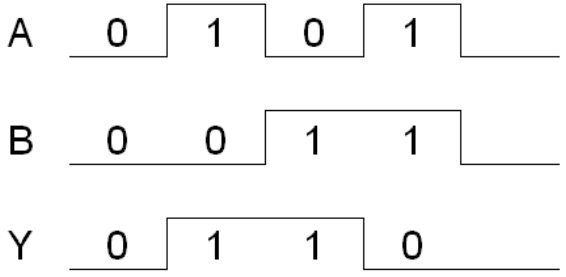


Funcții logice IV

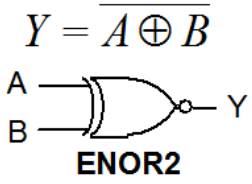
- **SAU-EXCLUSIV (XOR, antivalența) :** $Y = A \oplus B =$
  - 1 dacă A ≠ B
  - 0 în rest



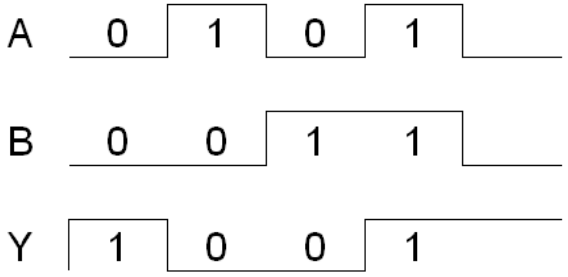
A	B	SAU EXCLUSIV
0	0	0
0	1	1
1	0	1
1	1	0



- ***SAU-EXCLUSIV-NU (XNOR, echivalența)*** **:** $Y = \overline{A \oplus B} =$
  - 1 dacă A = B
  - 0 în rest



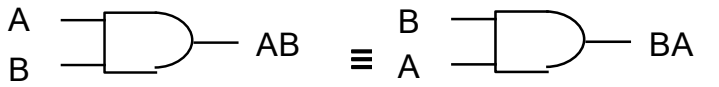
A	B	SAU-EXCLUSIV NU
0	0	1
0	1	0
1	0	0
1	1	1



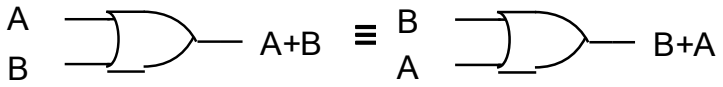
Proprietățile operațiilor logice I

Comutativitatea (interschimbabilitatea variabilelor)

$$A \bullet B = B \bullet A$$



$$A + B = B + A$$



Asociativitatea (posibilitatea grupării variabilelor)

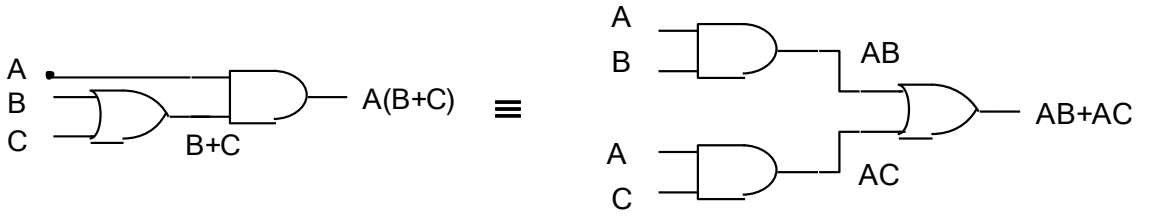
$$A \bullet (B \bullet C) = (A \bullet B) \bullet C$$

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

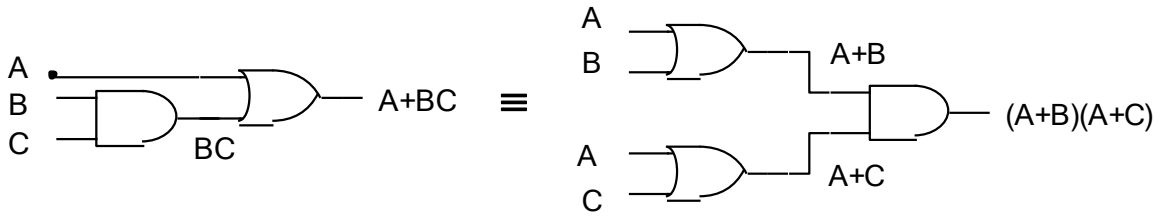
Proprietățile operațiilor logice II

Distributivitatea

$$A \bullet (B + C) = A \bullet B + A \bullet C$$

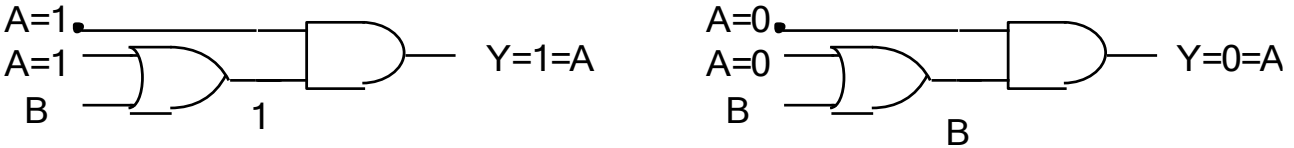


$$A + B \bullet C = (A + B) \bullet (A + C)$$

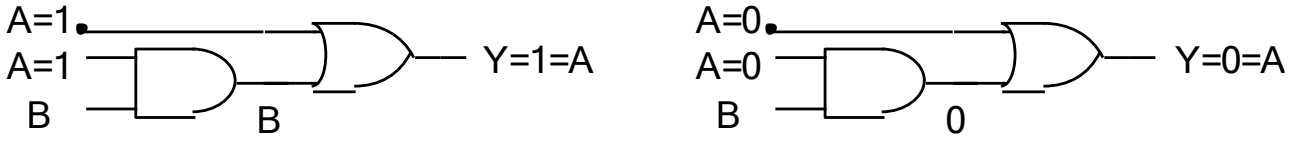


Legile de absorbție

$$A \bullet (A + B) = A$$



$$A + A \bullet B = A$$

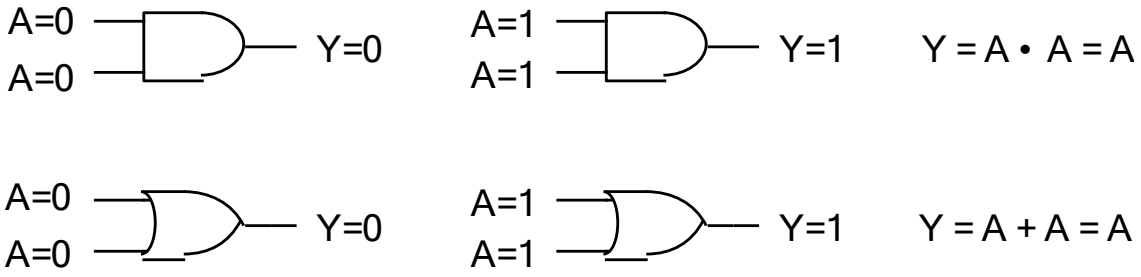


Proprietățile operațiilor logice III

Legile de idempotență:

$A \bullet A = A$

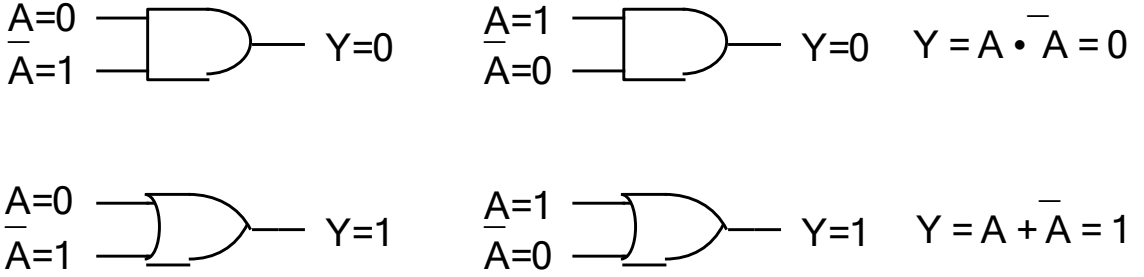
$A + A = A$



Legile de complementaritate:

$A \bullet \overline{A} = 0$

$A + \overline{A} = 1$

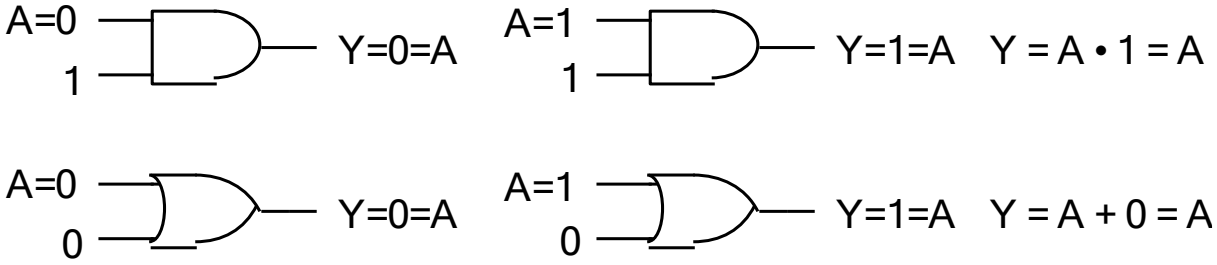


Proprietățile operațiilor logice IV

Legile elementului nul:

$A \bullet 1 = A$

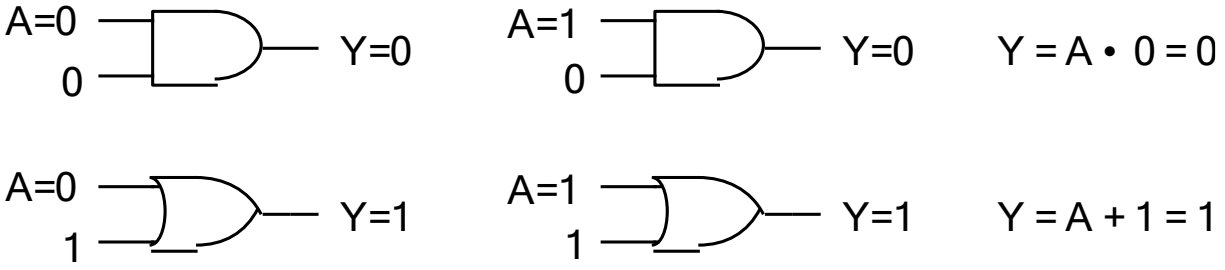
$A + 0 = A$



Legile de identitate:

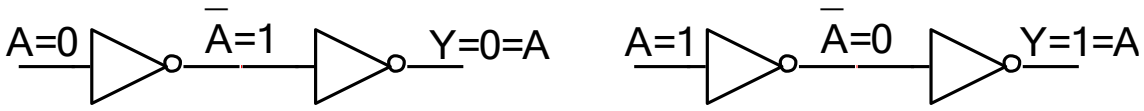
$A \bullet 0 = 0$

$A + 1 = 1$



Principiul dublei negații:

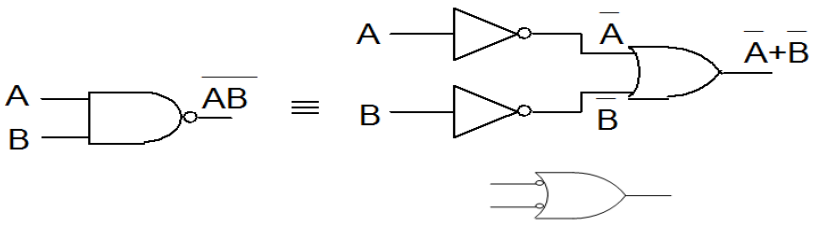
$\overline{\overline{A}} = A$



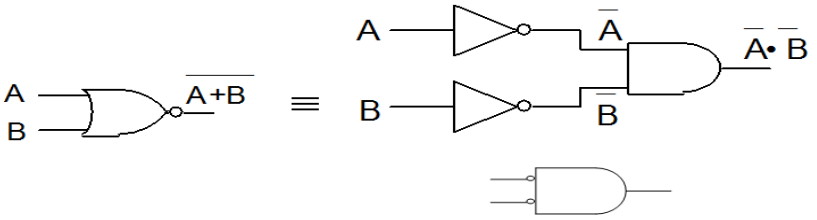
Teoremele lui De Morgan

$$\overline{A \bullet B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \bullet \overline{B}$$

$$\overline{A \bullet B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \bullet \overline{B}$$



Intrări		Ieşiri	
A	B	$\overline{AB}$	$\overline{A}+\overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0



Intrări		Ieşiri	
A	B	$\overline{A+B}$	$\overline{AB}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Teoremele lui De Morgan pentru mai multe variabile respectiv mai mulți termeni:

$$\overline{A \bullet B \bullet C} = \overline{A} + \overline{B} + \overline{C}$$
$$\overline{A + B + C} = \overline{A} \bullet \overline{B} \bullet \overline{C}$$

$$X = \overline{A \bullet B + A \bullet \overline{C} + ABC} = \overline{A \bullet B} \bullet \overline{A \bullet \overline{C}} \bullet \overline{ABC}$$

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

Exprimarea funcțiilor logice

- Funcțiile logice pot fi exprimate cu ajutorul
  - Tabelului de adevăr,
  - Formă algebrică,
  - Formă matematică,
  - Mod grafic,
  - Diagrame de timp.
- Toate formele de mai sus sunt echivalente și pot fi transcrise dintr-o formă în alta!

- În tabelul de adevăr apar toate combinațiile posibile ale variabilelor funcției în ordinea numărării binare. O funcție de n variabile poate avea 2^n combinații diferite.

Exprimarea termenilor

- Exprimarea canonică a termenilor funcției:
  - Sub formă de produse (ȘI), între toate variabilele independente sub formă directă sau negată
  - Sub formă de sume (SAU) între toate variabilele independente sub formă directă sau negată
- O funcție de n variabile poate avea 2^n combinații diferite de tip produs sau sumă.

Minterm	Maxterm
$\overline{C}\overline{B}\overline{A}$	$C+B+A$
$\overline{C}\overline{B}A$	$C+B+\overline{A}$
$\overline{C}B\overline{A}$	$C+\overline{B}+A$
$\overline{C}BA$	$C+\overline{B}+\overline{A}$
$C\overline{B}\overline{A}$	$\overline{C}+B+A$
$C\overline{B}A$	$\overline{C}+B+\overline{A}$
$CB\overline{A}$	$\overline{C}+\overline{B}+A$
CBA	$\overline{C}+\overline{B}+\overline{A}$

$m_0^2=\overline{B}\overline{A}$	$M_3^2=B+A$
$m_1^2=\overline{B}A$	$M_2^2=B+\overline{A}$
$m_2^2=B\overline{A}$	$M_1^2=\overline{B}+A$
$m_3^2=BA$	$M_0^2=\overline{B}+\overline{A}$

Exprimarea algebrică a funcțiilor logice

Două forme canonice:

- forma canonică disjunctivă: este formată din disjuncția termenilor de tip produs P_i.
  - **Sum of Products (SOP)**

$$X = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + ABC$$

  - Ex.

- forma canonică conjunctivă: este formată din conjuncția termenilor de tip sumă S_i.
  - **Product of Sums (POS)**

  - Ex.

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

Exprimarea matematică simplificată

- Exprimare ponderată: C – 2^2, B – 2^1, A – 2^0,

$P_0 = \overline{B}\,\overline{A}$	$S_0 = B + A$
$P_1 = \overline{B}A$	$S_1 = B + \overline{A}$
$P_2 = B\overline{A}$	$S_2 = \overline{B} + A$
$P_3 = BA$	$S_3 = \overline{B} + \overline{A}$

$P_0 = \overline{C}\,\overline{B}\,\overline{A}$	$S_0 = C + B + A$
$P_1 = \overline{C}\,\overline{B}A$	$S_1 = C + B + \overline{A}$
$P_2 = \overline{C}B\overline{A}$	$S_2 = C + \overline{B} + A$
$P_3 = \overline{C}BA$	$S_3 = C + \overline{B} + \overline{A}$
$P_4 = C\overline{B}\,\overline{A}$	$S_4 = \overline{C} + B + A$
$P_5 = C\overline{B}A$	$S_5 = \overline{C} + B + \overline{A}$
$P_6 = CB\overline{A}$	$S_6 = \overline{C} + \overline{B} + A$
$P_7 = CBA$	$S_7 = \overline{C} + \overline{B} + \overline{A}$

$$X = \overline{C}\,\overline{B}A + \overline{C}B\overline{A} + CBA = P_1 + P_0 + P_7$$

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

$$= S_1 \bullet S_0 \bullet S_7$$

Exprimarea grafică

Diagrama Veitch-Karnaugh

- Exprimare sub forma unui tabel. Fiecare celulă reprezintă un termen al funcției.
- In diagrama Karnaugh termenul care pentru care funcția are valoarea 1 se trece un 1 în celula corespunzătoare din tabel

Diagrama Veitch-Karnaugh pentru două variabile

B \ A	0	1
0	P_0	P_1
1	P_2	P_3

Diagrama Veitch-Karnaugh pentru trei variabile

- Pentru o funcție de 3 variabile rezultă $2^3 = 8$ combinații posibile, diagrama Karnaugh va avea 8 celule

C \ BA	00	01	11	10
0	P_0	P_1	P_3	P_2
1	P_4	P_5	P_7	P_6

C \ A	000	001	011	010
100	101	111	110	

B

Diagrama Veitch-Karnaugh pentru patru variabile

Patru variabile => 2^4 = 16 stări diferite, diagrama Karnaugh = tabel cu 16 celule

DC \ BA	00	01	11	10
00	P_0	P_1	P_3	P_2
01	P_4	P_5	P_7	P_6
11	P_{12}	P_{13}	P_{15}	P_{14}
10	P_8	P_9	P_{11}	P_{10}

D \ DC \ BA	00	01	11 (B)	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

C

A

Minimizarea funcțiilor logice

- Scop: ***Mai puține operații (termeni), și/sau mai puține variabile.***
- Criteriile de minimizare care s-au impus sunt:

  - reducerea numărului de variabile;
  - reducerea numărului de termeni;
  - reducerea pe ansamblu a variabilelor şi termenilor astfel ca suma lor să fie minimă.

- Metode de minimizare pot fi grupate în:

  - metode algebrice;
  - metode grafice, - Metoda diagramei Veitch-Karnaugh.

  – Regulile metodei de minimizare:

  - Se marchează prin linii continue sau întrerupte respectiv se delimitează suprafețele care conțin celule care conțin „1” (pentru forma SOP), pe principiul compartimentelor adiacente.
  - Numărul celulelor din grupare trebuie să fie 2^n celule vecine
  - Se caută formarea de grupe cât mai mari
  - Celulele de la marginea tabelului sun considerate vecine
  - Gruparea se poate face doar pe orizontală si verticală dar oblic nu
  - Fiecare 1 trebuie prins în cel puțin o grupare
  - Din expresia algebrică a grupării se vor elimina n variabile (unde 2^n este numărul celulelor din grupare).

Minimizarea funcțiilor cu termeni redundanți

- Daca funcția nu este complet definită există cel puțin un termen pentru care funcția are valoare nedeterminată (numite şi stări indiferente, arbitrare sau redundante).
- Variabilei X i se atribuie valoarea “1” sau “0”.
- Dacă este avantajos d.p.d.v al minimizării (contribuie la formarea unor grupe mai mari) poate fi considerat ***1,*** dacă nu, atunci va fi considerat ***0***.

Ex.

$$F = \overline{D} \bullet \overline{C} \bullet \overline{B} \bullet \overline{A} + \overline{D} \bullet \overline{C} \bullet B \bullet \overline{A} + \overline{D} \bullet \overline{C} \bullet B \bullet A +$$
$$+ \overline{D} \bullet C \bullet \overline{B} \bullet A + D \bullet \overline{C} \bullet B \bullet \overline{A} + D \bullet C \bullet B \bullet A$$

- Termeni redundanți:

$$\overline{D} \bullet C \bullet B \bullet A \qquad D \bullet \overline{C} \bullet \overline{B} \bullet \overline{A} \qquad D \bullet \overline{C} \bullet B \bullet A$$

$F = P_0 + P_2 + P_3 + P_5 + P_{10} + P_{15}$

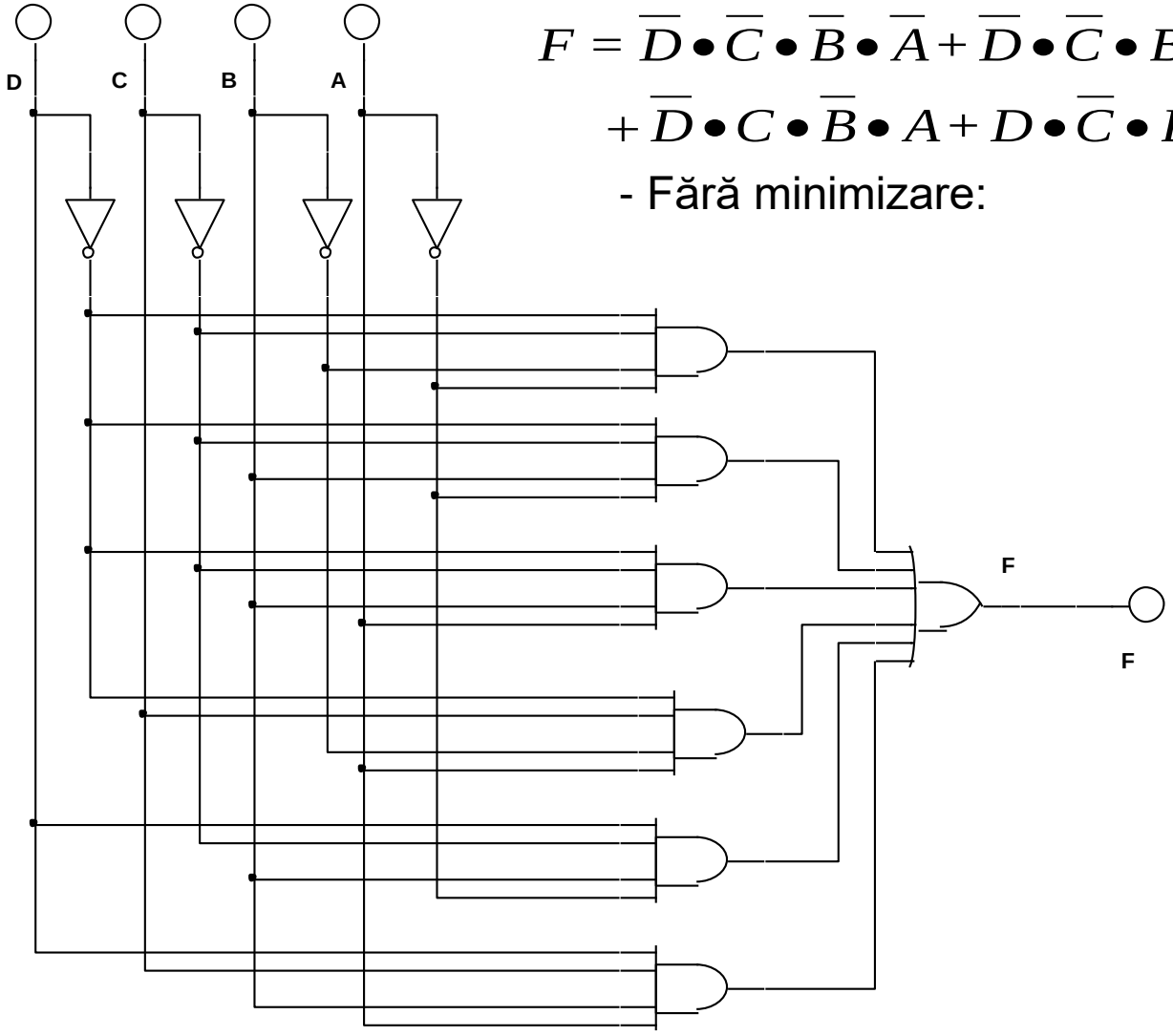
Termeni redundanți: P_7, P_8 şi P_{11}

D	C	B	A	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	X
1	0	0	0	X
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Implementarea funcției cu porți logice

$$F = \overline{D} \bullet \overline{C} \bullet \overline{B} \bullet \overline{A} + \overline{D} \bullet \overline{C} \bullet B \bullet \overline{A} + \overline{D} \bullet \overline{C} \bullet B \bullet A +$$
$$+ \overline{D} \bullet C \bullet \overline{B} \bullet A + D \bullet \overline{C} \bullet B \bullet \overline{A} + D \bullet C \bullet B \bullet A$$

- Fără minimizare:



Minimizarea funcției

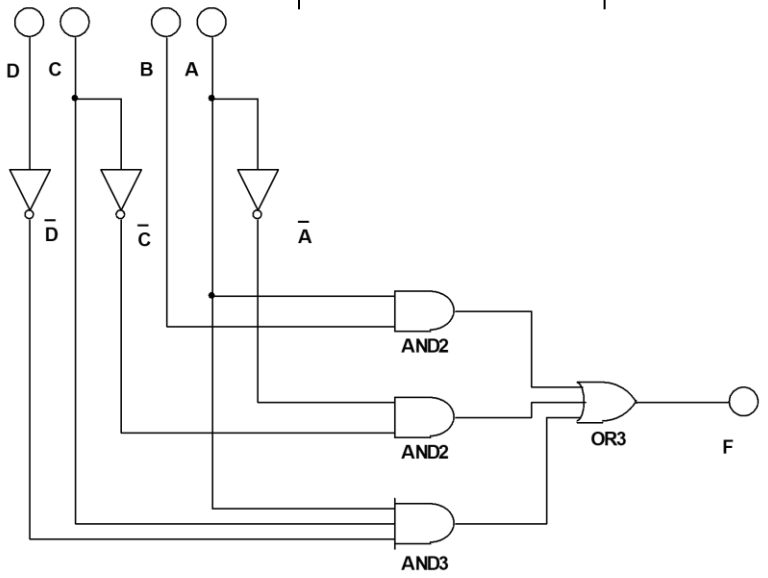
$$F = \overline{D} \bullet \overline{C} \bullet \overline{B} \bullet \overline{A} + \overline{D} \bullet \overline{C} \bullet B \bullet \overline{A} + \overline{D} \bullet \overline{C} \bullet B \bullet A +$$
$$+ \overline{D} \bullet C \bullet \overline{B} \bullet A + D \bullet \overline{C} \bullet B \bullet \overline{A} + D \bullet C \bullet B \bullet A$$

DC \ BA	00	01	11	10
00	1	0	1	1
01	0	1	X	0
11	0	0	1	0
10	X	0	X	1

- Realizarea diagramei Karnaugh
- Gruparea celulelor
- Scrierea formei simplificate

$$F = A \bullet B + \overline{A} \bullet \overline{C} + A \bullet C \bullet \overline{D}$$

- Implementarea funcției minimizate cu porți logice

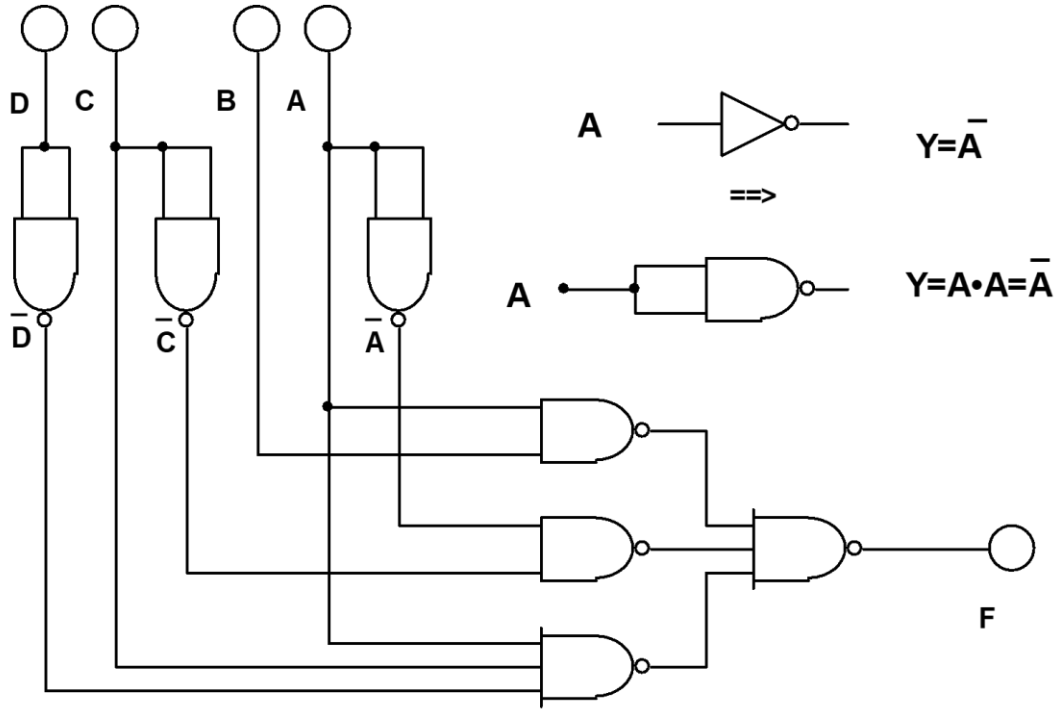


Realizarea funcției folosind porți NAND

$$F = A \bullet B + \overline{A} \bullet \overline{C} + A \bullet C \bullet \overline{D}$$

- Funcția anterioară poate fi rescrisă folosind teoremele lui De Morgan

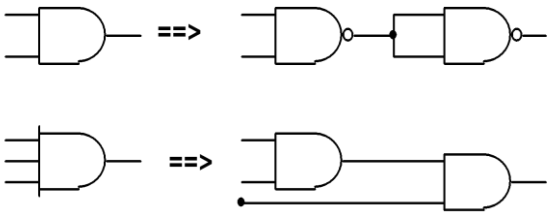
$$F = \overline{\overline{F}} = \overline{\overline{A \bullet B + \overline{A} \bullet \overline{C} + A \bullet C \bullet \overline{D}}} = \overline{\overline{A \bullet B} \bullet \overline{\overline{A} \bullet \overline{C}} \bullet \overline{A \bullet C \bullet \overline{D}}}$$



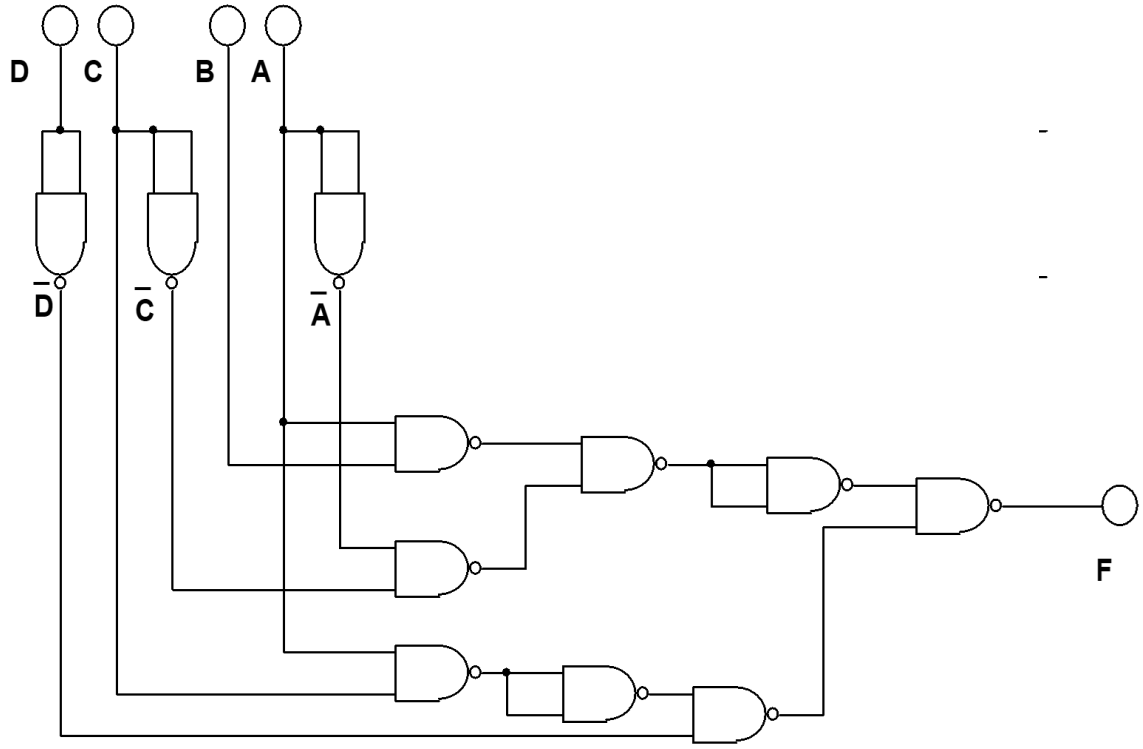
Realizarea funcției cu porți NAND cu 2 intrări

- Mai puține capsule de CI: utilizarea unui singur tip
- Funcția anterioară poate fi rescrisă folosind proprietățile de asociativitate

$$F = \overline{\overline{A \bullet B} \bullet \overline{\overline{A} \bullet \overline{C}} \bullet \overline{A \bullet C} \bullet \overline{D}} = \overline{(\overline{A \bullet B} \bullet \overline{\overline{A} \bullet \overline{C}}) \bullet (A \bullet C) \bullet \overline{D}}$$

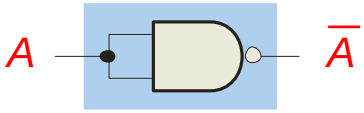


$$F = \overline{\overline{(\overline{A \bullet B} \bullet \overline{\overline{A} \bullet \overline{C}})} \bullet \overline{\overline{(A \bullet C)}} \bullet \overline{D}}$$

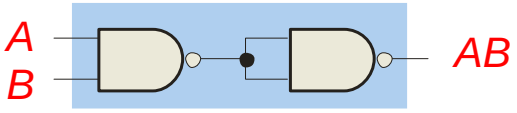


Porți universale

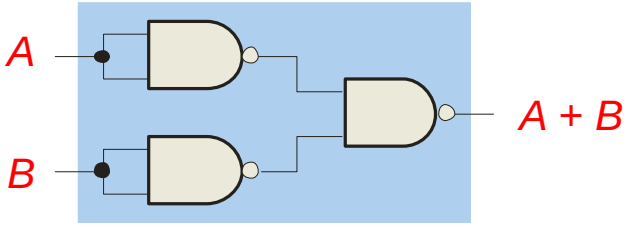
Porțile ȘI-NU (NAND) se numesc **universale** deoarece pot fi folosite pentru a genera celelalte funcții logice.



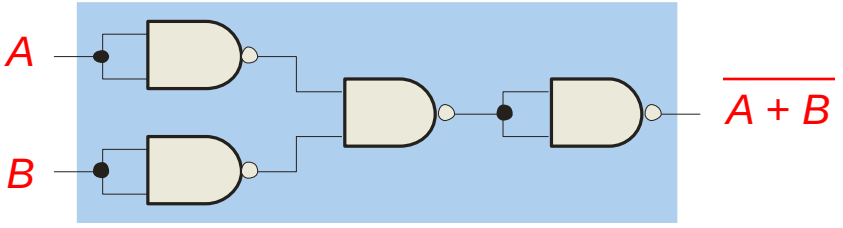
Inversor



Poarta ȘI



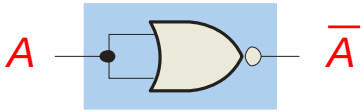
Poarta SAU



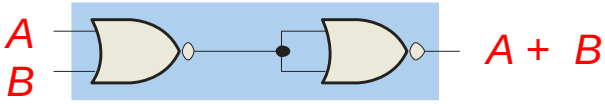
Poarta SAU-NU

Porți universale

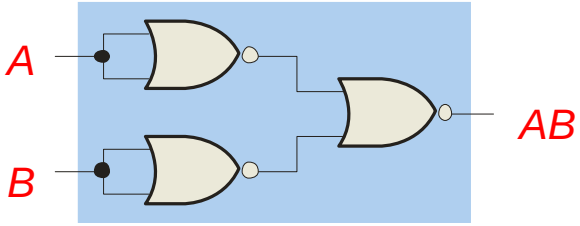
Porțile SAU-NU (NOR) se numesc **universale** deoarece pot fi folosite pentru a genera celelalte funcții logice.



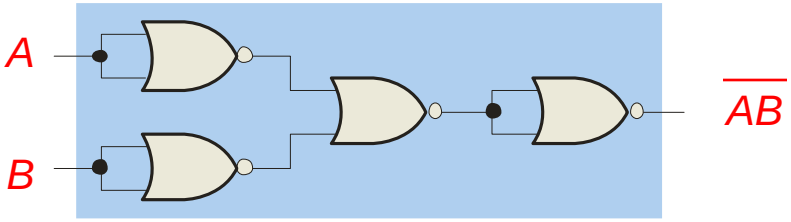
Inversor



Poartă SAU



Poartă ȘI

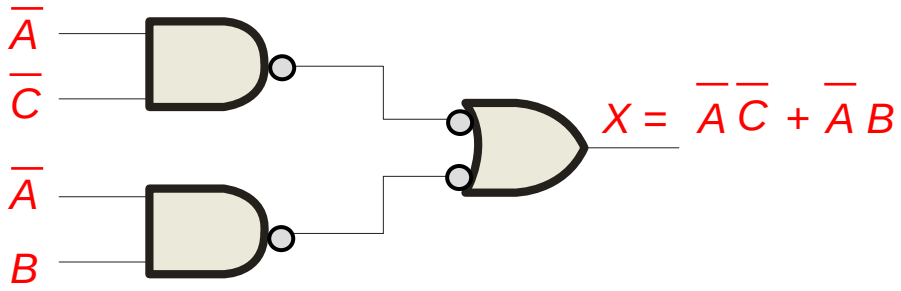


Poartă ȘI-NU

NAND Logic

Din teoremele lui DeMorgan: $\overline{AB} = \overline{A} + \overline{B}$.

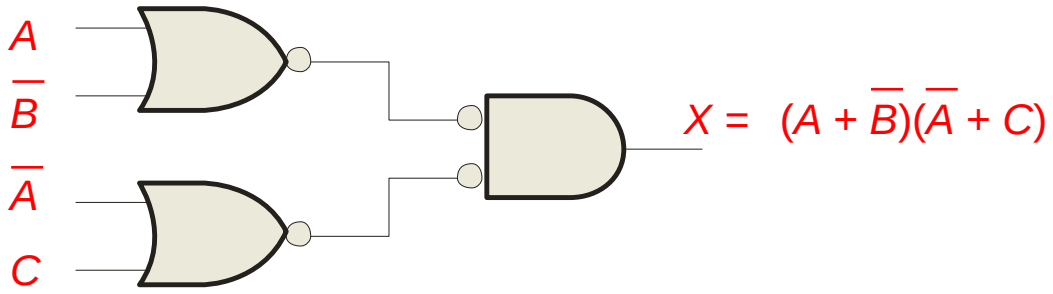
Utilizând simboluri echivalente se poate deduce forma SOP:



NOR Logic

De asemenea tot din teoremele lui DeMorgan

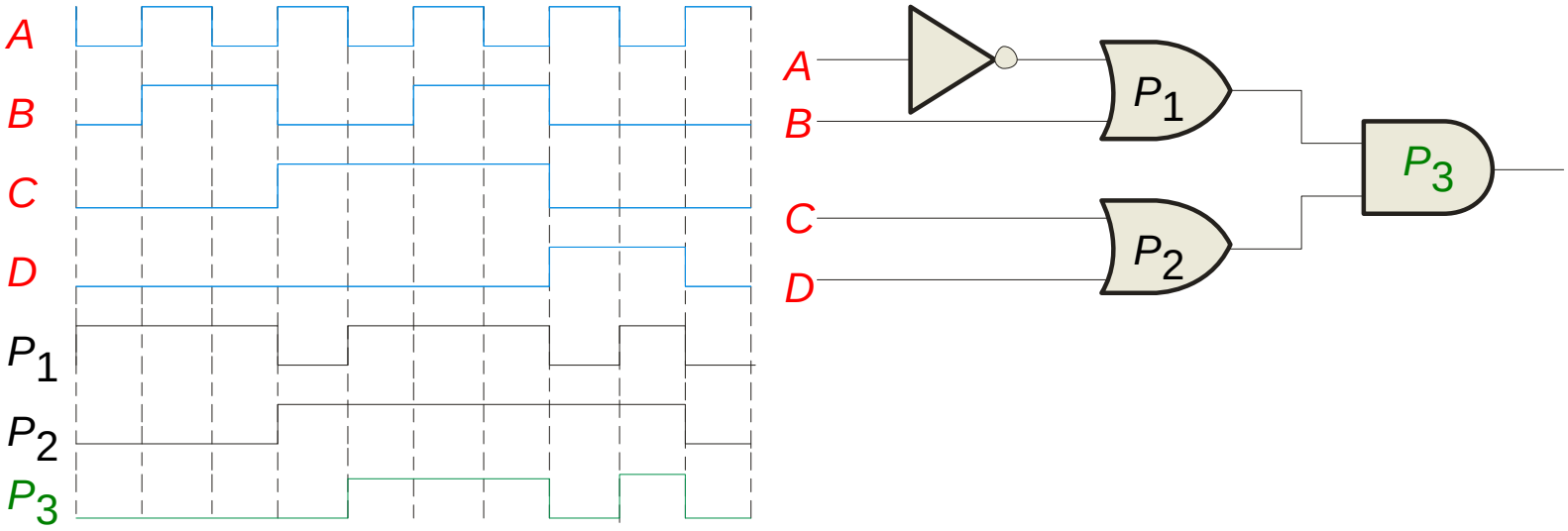
$\overline{A + B} = \overline{A}\,\overline{B}$.



Forme de undă

Exemplu:

Să se analizeze formele de undă pentru circuitul din figură:



Porți logice I

Două tehnologii principale:

- TTL (Transistor-Transistor Logic)
- CMOS (Complementary Metal-Oxide-Semiconductor)

Porțile TTL

- 5 V
- Prefix 74 sau 54

- Principalele tipuri:
  - 74 –TTL normal (fără litere)
  - 74S – Schottky TTL
  - 74LS – Low-power Schottky TTL (TTL Schottky de consum redus)
  - 74AS – Advanced Schottky TTL (TTL Schottky de viteză mare)
  - 74ALS – Advanced Low-power Schottky TTL (TTL Schottky de viteză mare și consum redus)
  - 74F – Fast TTL (TTL rapide)

Porți logice II

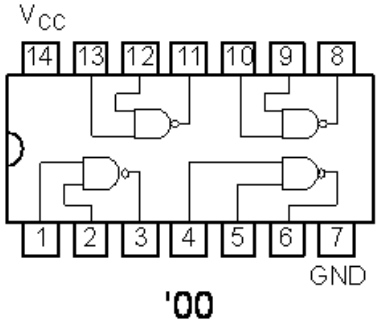
Porți CMOS

- 5V, 3,3V, 2,5V şi 1,8V
- prefix74 (sau 54) + literă(litere)

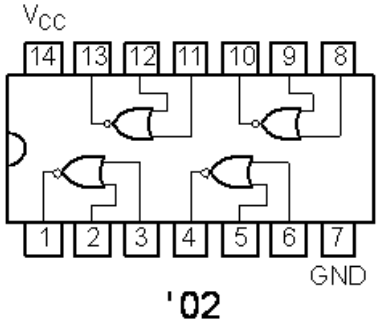
- Principalele tipuri de 5 V:
  - 74HC și 74HCT – High-speed CMOS (T : compatibil TTL)
  - 74AC şi 74 ACT – Advanced CMOS (CMOS de consum redus)
  - 74AHC şi 74AHCT - Advanced High-speed CMOS (CMOS de viteză mare și consum redus)

- Principalele tipuri de 3,3 V:
  - 74LC – Low-voltage CMOS
  - 74LVC – Low-voltage CMOS
  - 74ALVC – Advanced Low-voltage CMOS

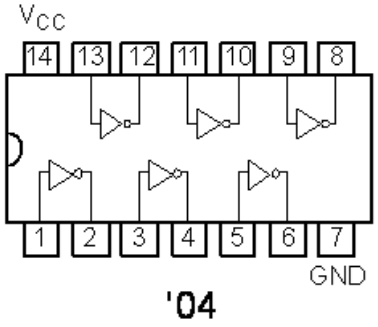
Porți logice III



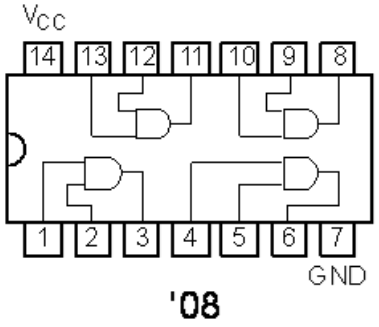
'00



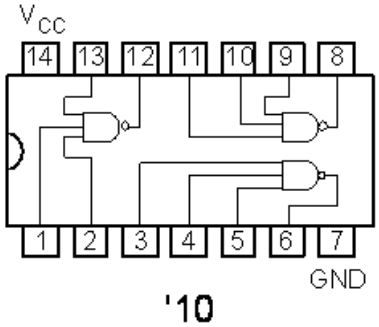
'02



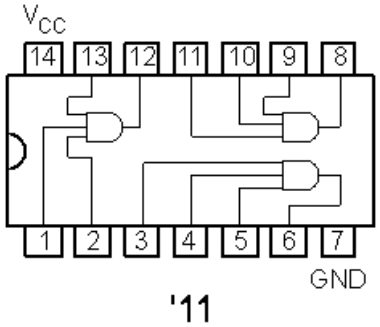
'04



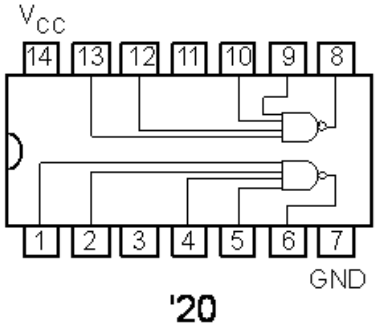
'08



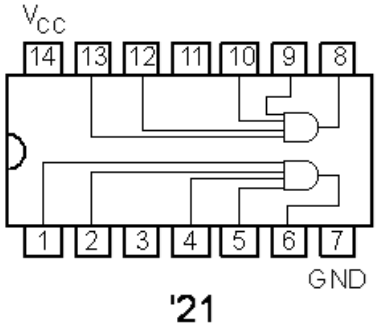
'10



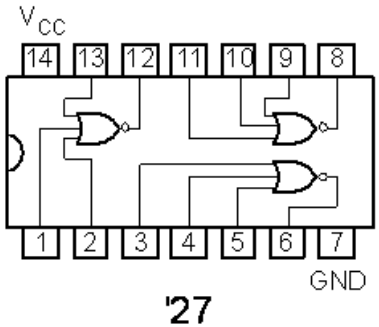
'11



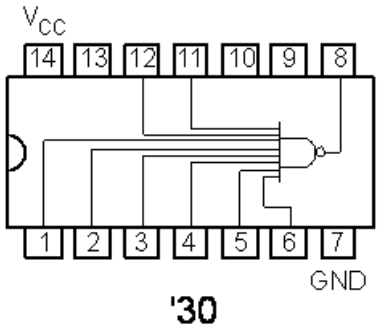
'20



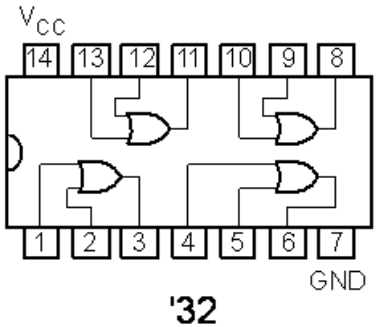
'21



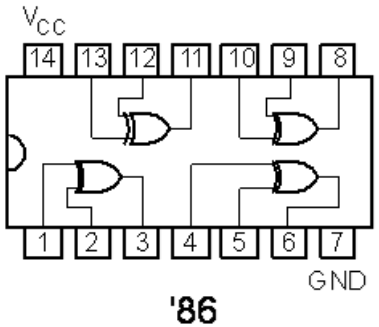
'27



'30



'32

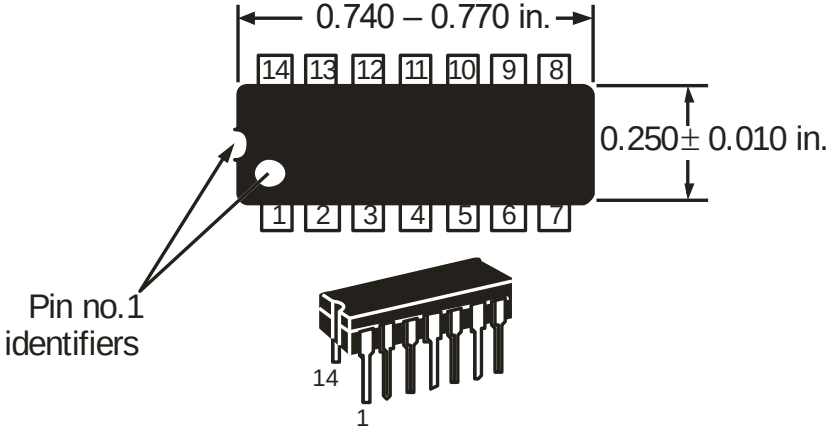


'86

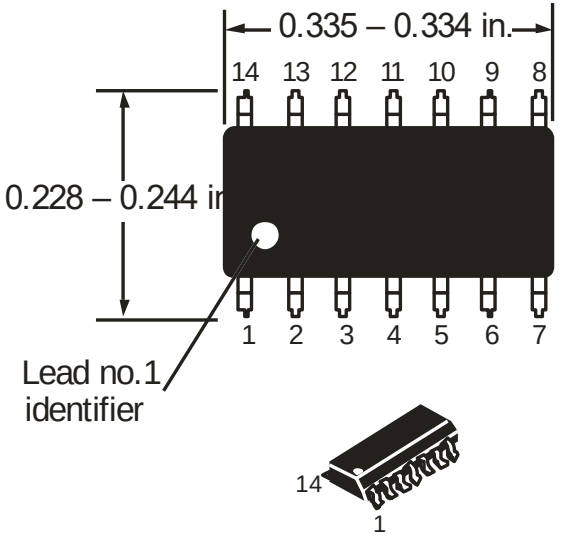
Porți logice IV

Capsule:

DIP (Dual Inline Package)
SOIC (Small Outline Integrated Circuit)



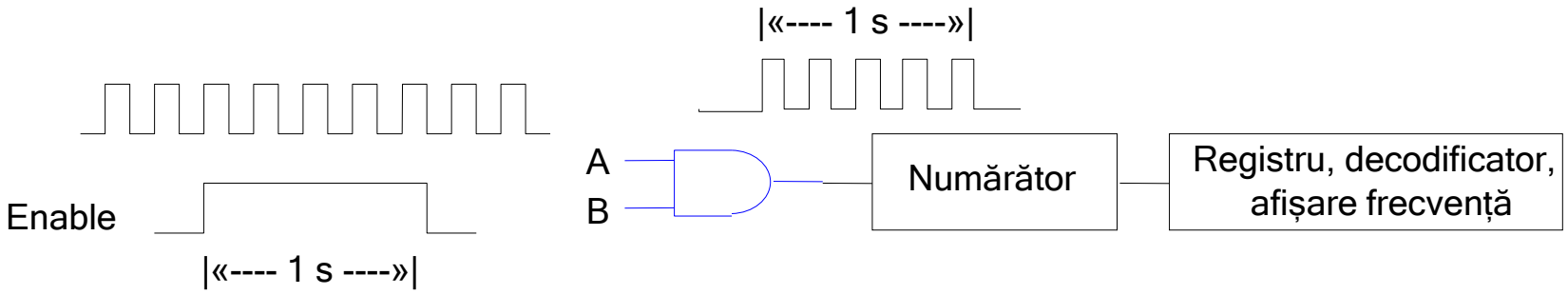
DIP package



SOIC package

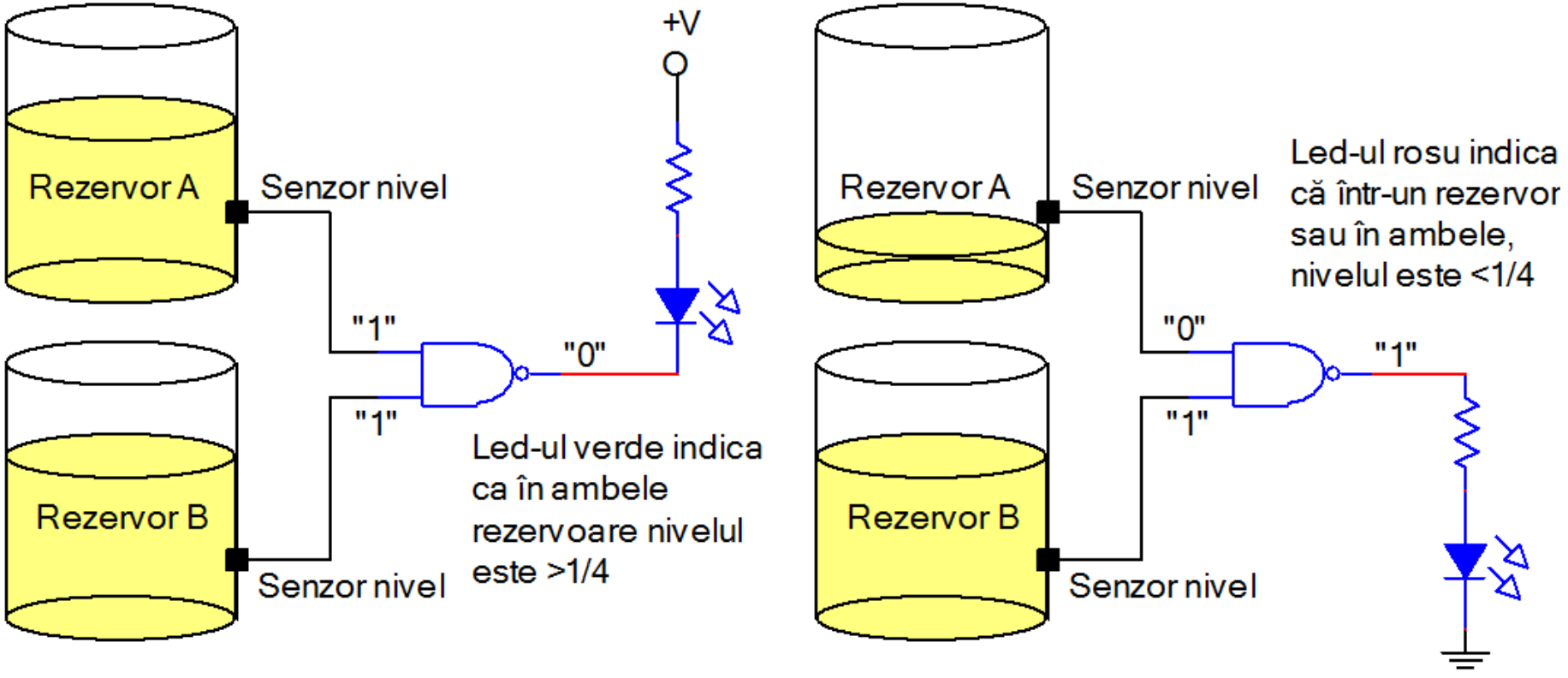
Aplicații ale porților logice I

Utilizarea porților ȘI ca poartă de autorizare (Enable)



Aplicații ale porților logice II

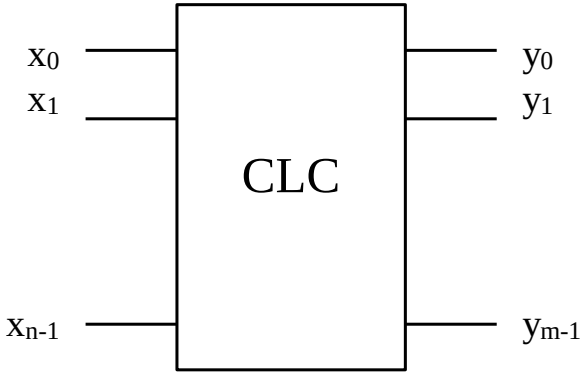
Utilizarea porților ȘI-NU (NAND) pentru monitorizarea nivelului lichidelor



Circuite logice combinaționale I

Circuite logice combinaționale

- Circuitele logice se pot clasifica în:
  - circuite logice combinaționale
  - circuite logice secvențiale.



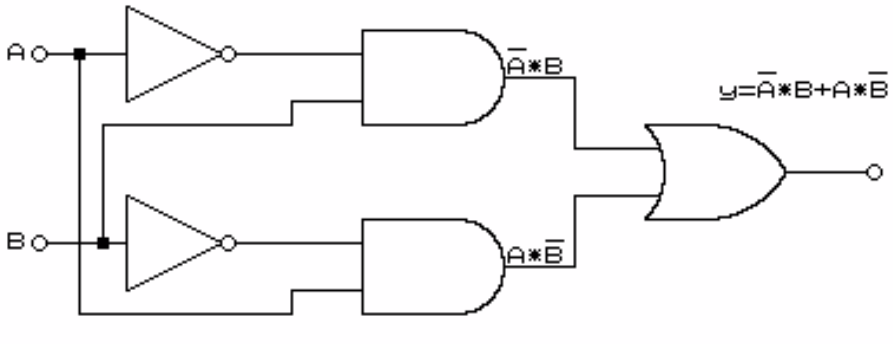
- **Circuitele logice combinaționale** :
  - semnalul ce apare pe o ieșire y_k este o combinație a semnalelor de la intrare $y_k=f_k(x_0,x_1,\ldots x_{n-1})$ (variabilele de la ieșire sunt independente de timp și de starea inițială a circuitului)
  - sunt circuite fără memorie (semnalele de pe ieșiri există doar în prezența semnalelor aplicate pe intrări).

Hazard combinațional

- Independența față de timp ⇔ la modificarea simultană a variabilelor de intrare ale circuitului combinațional are loc modificarea instantanee și simultană a variabilelor de ieșire. **FALS**
- Pe durata procesului tranzitoriu de stabilire a variabilelor de ieșire, acestea pot lua valori intermediare diferite de cele finale = **hazard combinațional**.
- Măsurile care se iau pentru evitarea efectelor hazardului combinațional:
  - uniformizarea întârzierilor pentru toate canalele de ieșire
  - utilizarea unei memorii intermediare.

Analiza circuitelor logice combinaționale

- În cadrul *analizei* se pleacă de la cunoașterea schemei circuitului și se urmărește stabilirea funcționării acestuia, concretizată prin tabelul de funcționare sau prin scrierea expresiei semnalelor de ieșire în funcție de cele de intrare.



A	B	$\bar{A} \bullet B$	$A \bullet \bar{B}$	Y
L	L	L	L	L
L	H	H	L	H
H	L	L	H	H
H	H	L	L	L

Sinteza circuitelor logice combinaționale I

- În cadrul *sintezei* se cunoaște funcția pe care trebuie s-o îndeplinească circuitul și se caută să se găsească structura acesteia.
- Etapele sintezei sunt următoarele:
  - definirea funcției (sau a funcțiilor),
  - minimizarea funcției,
  - desenarea schemei circuitului.

Sinteza circuitelor logice combinaționale II

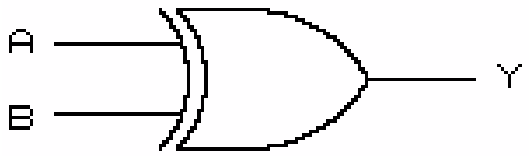
- Circuit poate fi realizat în mai multe variante (după modul cum a fost scrisă funcția) de exemplu:
  - cu circuite **ŞI, SAU, NU**, sau sumă de produse (**SOP** = sum of products);
  - cu circuite **SAU, ŞI, NU** sau produs de sume (**POS** = products of sums);
  - cu circuite **SAU-NU (NOR);**
  - cu circuite **ŞI-NU (NAND);** etc.

Sinteza circuitelor logice combinaționale III

Exemplu:

- **Realizarea circuitului de anticoincidență *(XOR)***

A	B	Y=A ⊕ B
L	L	L
L	H	H
H	L	H
H	L	H
H	H	L

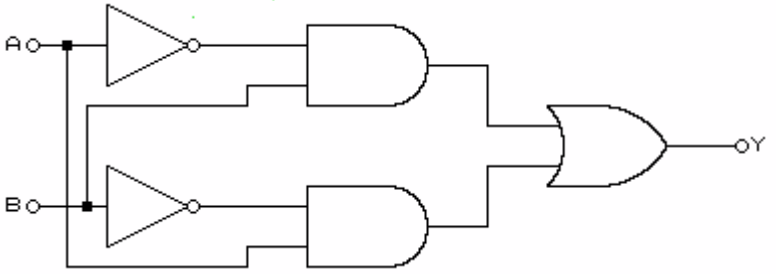


B \ A	**0**	**1**
0	0	1
1	1	0

Utilizarea porților NOT, AND, OR

1. Sumă de produse (NOT, AND, OR):
   - Diagrama Karnaugh

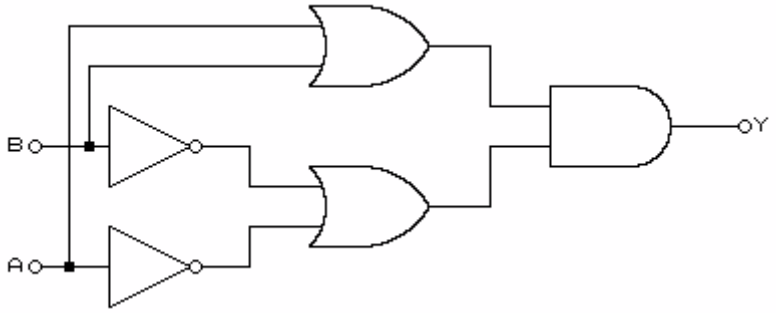
B \ A	0	1
0	0	(1)
1	(1)	0



$$Y = (\overline{A} \bullet B) + (A \bullet \overline{B})$$

2. Produs de sume (NOT, OR, AND)
   - Diagrama Karnaugh

B \ A	0	1
0	(0)	1
1	1	(0)



$$Y = (A + B) \bullet (\overline{A} + \overline{B})$$

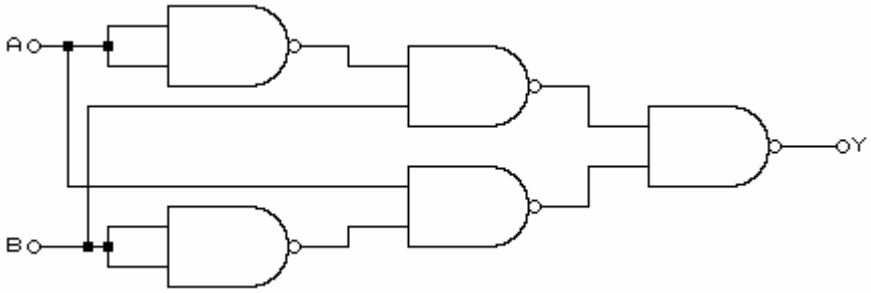
Implementarea XOR cu porți NAND sau NOR

Utilizând teoremele lui De Morgan asupra ecuațiilor anterioare, obținem:

3. Folosind porți NAND:

$$Y = \overline{\overline{Y}} = \overline{\overline{\overline{A} \bullet B + A \bullet \overline{B}}} =$$

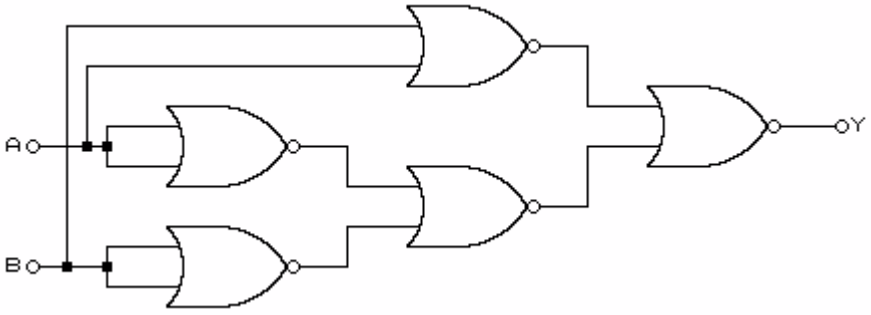
$$= \overline{\overline{\overline{A} \bullet B} \bullet \overline{A \bullet \overline{B}}}$$



4. Folosind porți NOR:

$$Y = \overline{\overline{Y}} = \overline{(A + B) \bullet (\overline{A} + \overline{B})} =$$

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



Exemple de circuite logice combinaționale

În continuare vor fi prezentate următoarele circuite logice combinaționale:

- codificator,
- decodificator,
- convertor de cod,
- multiplexor,
- demultiplexor,
- comparator numeric,
- detector și generator de paritate,
- sumator.

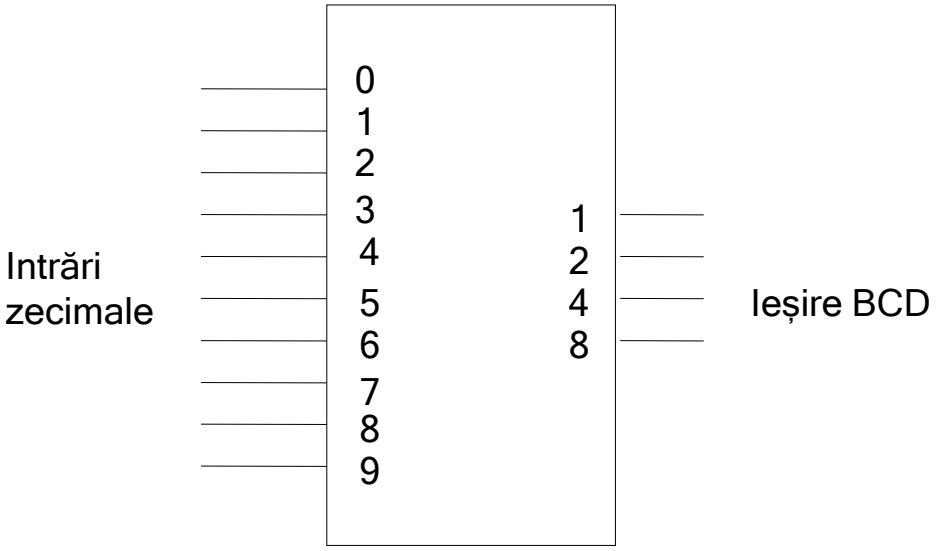
Codificatorul

- Codificatorul este un circuit logic combinațional care furnizează la ieșire un cuvânt binar de *k* biți atunci când numai una dintre cele *m* intrări ale sale este activă.

$$Y_r = \sum_{0}^{m-1} a_n I_n \qquad r = 0,1,\ldots,(k-1),$$

- Zecimal – zecimal codificat binar (BCD)

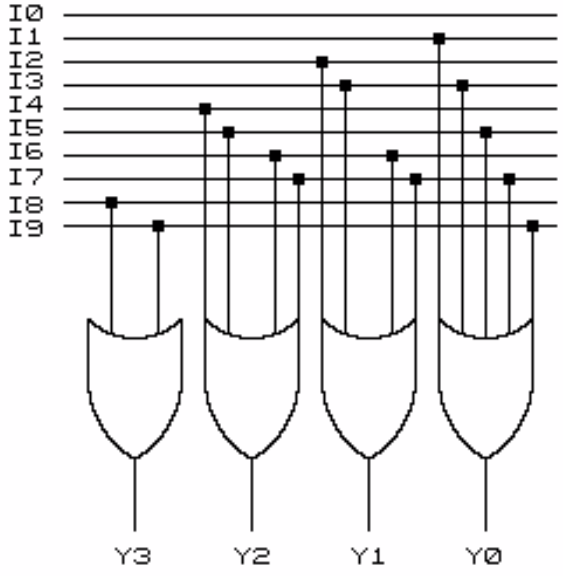
I	Y_3	Y_2	Y_1	Y_0
I_0	0	0	0	0
I_1	0	0	0	1
I_2	0	0	1	0
I_3	0	0	1	1
I_4	0	1	0	0
I_5	0	1	0	1
I_6	0	1	1	0
I_7	0	1	1	1
I_8	1	0	0	0
I_9	1	0	0	1



Codificatorul zecimal-BCD

- În conformitate cu tabelul se pot scrie relațiile:
  - $Y_0=I_1+I_3+I_5+I_7+I_9$
  - $Y_1=I_2+I_3+I_6+I_7$
  - $Y_2=I_4+I_5+I_6+I_7$
  - $Y_3=I_8+I_9$

I	Y_3	Y_2	Y_1	Y_0
I_0	0	0	0	0
I_1	0	0	0	1
I_2	0	0	1	0
I_3	0	0	1	1
I_4	0	1	0	0
I_5	0	1	0	1
I_6	0	1	1	0
I_7	0	1	1	1
I_8	1	0	0	0
I_9	1	0	0	1

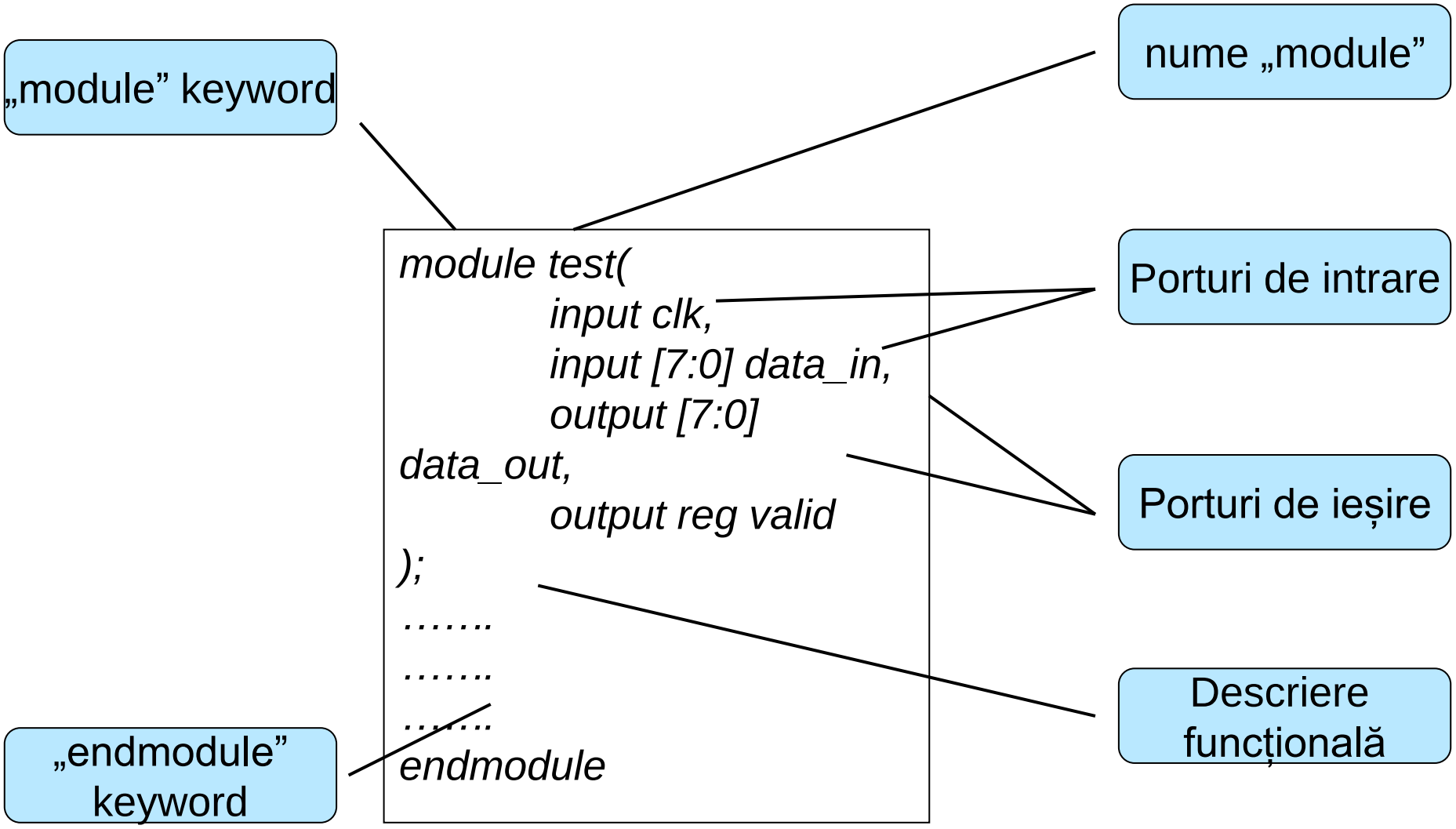


Codificator prioritar I

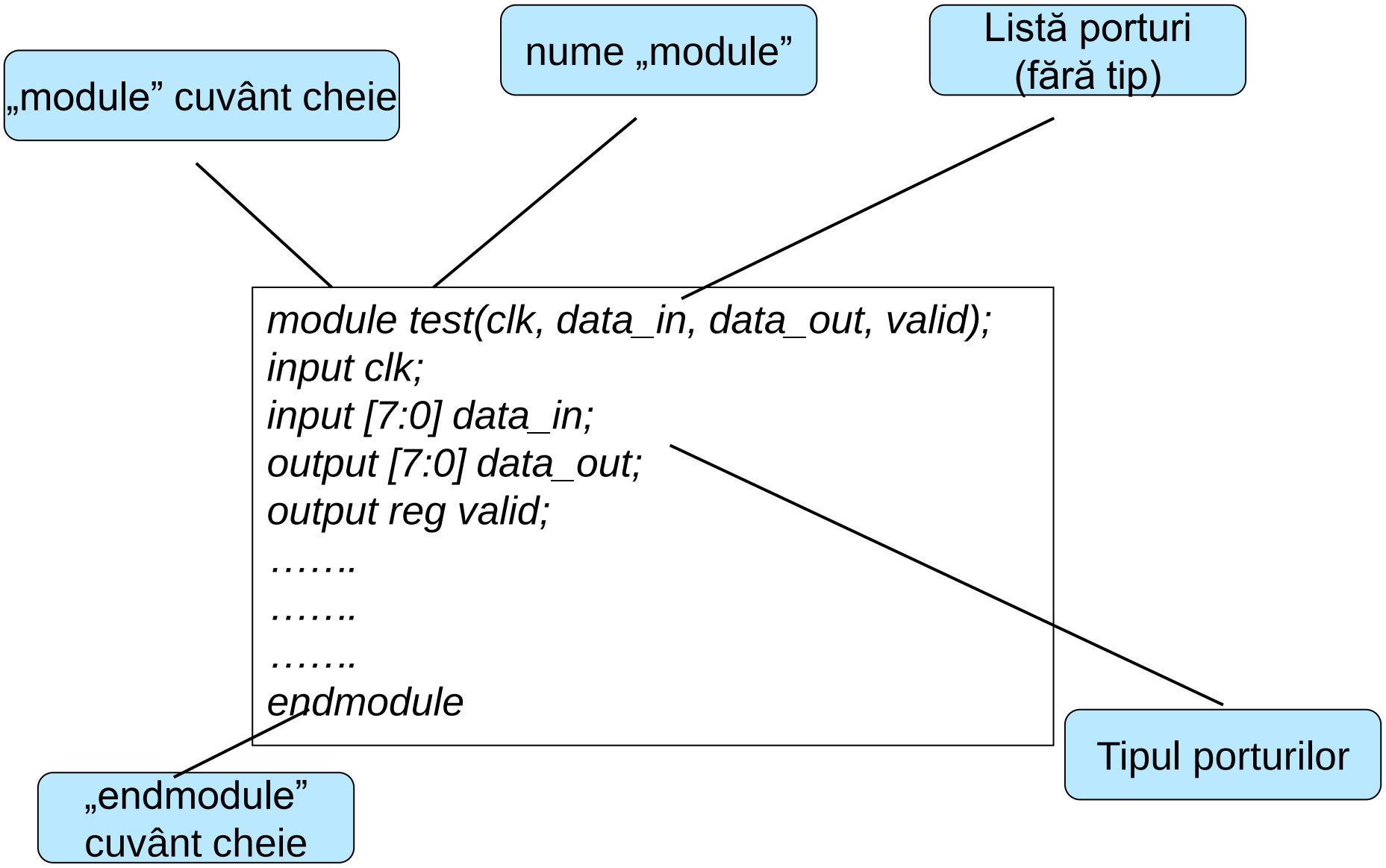
- Dezavantajul circuitului anterior este că atunci când se activează simultan mai multe intrări, codul citit la ieșire este eronat
- În cazul în care nu se poate evita acționarea simultană a mai multor intrări se folosesc circuite de **codificare prioritare**.
- în cazul acționării simultane a mai multor intrări la ieșire apare codul corespunzător intrării cu prioritatea cea mai mare

D3	D2	D1	D0	Y2	Y1	Y0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	0
0	1	x	x	0	1	1
1	x	x	x	1	0	0

Verilog: module (2001)



Verilog: module (1995)



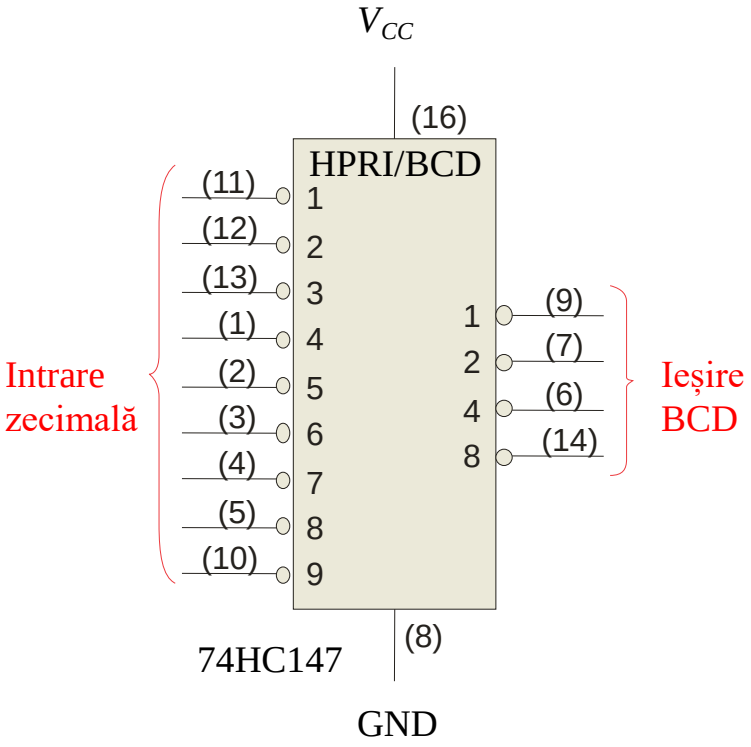
Codificator prioritar - Verilog

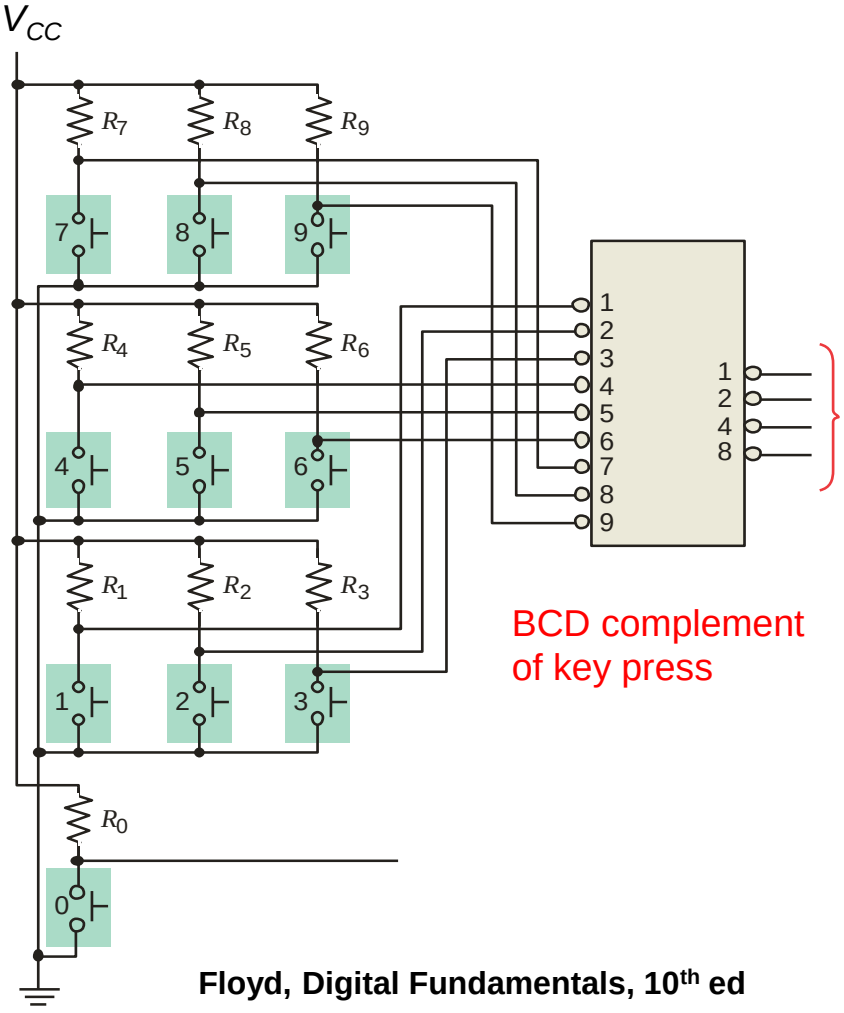
```
1  //
2  // 3-Bit 1-of-9 Priority Encoder
3  //
4
5  module v_priority_encoder_1 (sel, code);
6      input  [7:0] sel;
7      output [2:0] code;
8      reg    [2:0] code;
9
10     always @(sel)
11     begin
12         if      (sel[0]) code = 3'b000;
13         else if (sel[1]) code = 3'b001;
14         else if (sel[2]) code = 3'b010;
15         else if (sel[3]) code = 3'b011;
16         else if (sel[4]) code = 3'b100;
17         else if (sel[5]) code = 3'b101;
18         else if (sel[6]) code = 3'b110;
19         else if (sel[7]) code = 3'b111;
20         else             code = 3'bxxx;
21     end
22
23 endmodule
24
```

Aplicație codificator prioritar

Exemplu de utilizare: codificator de tastatură

Pl. 74HC147

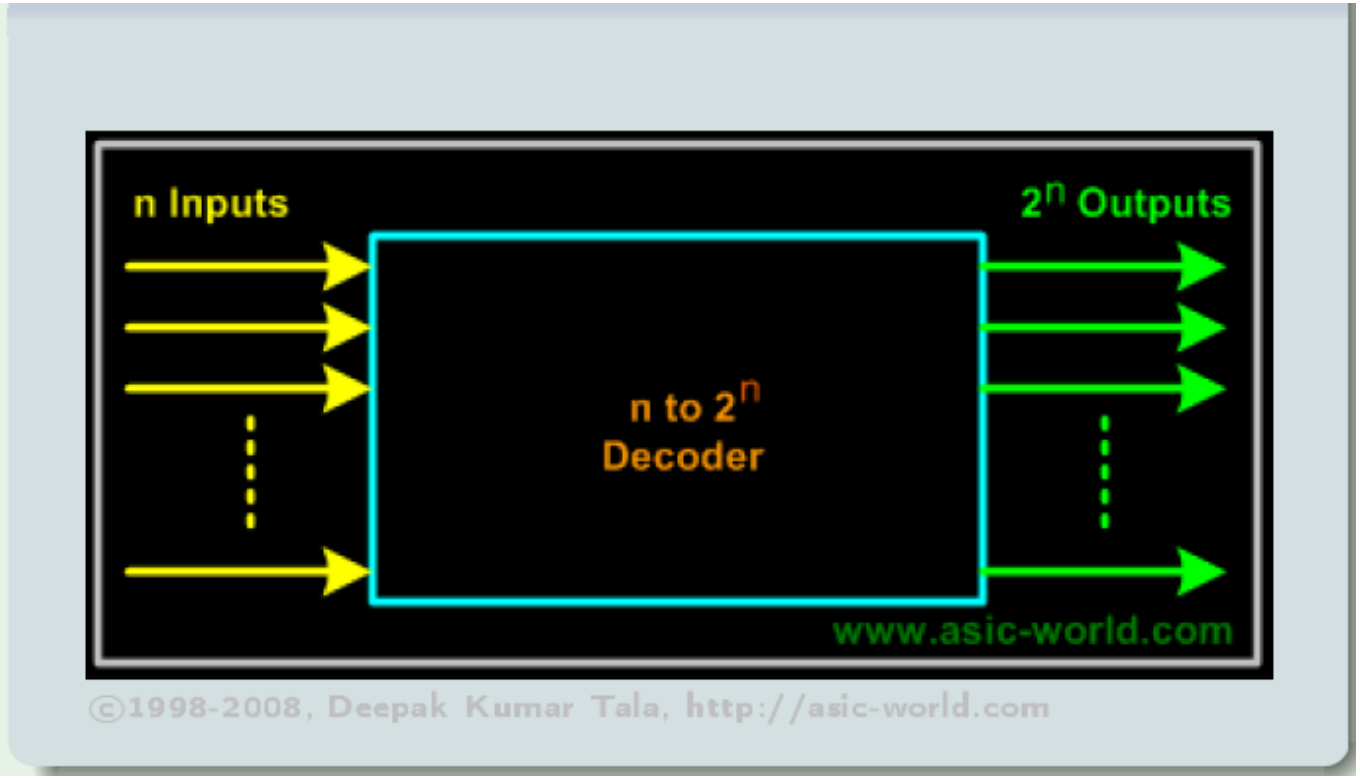




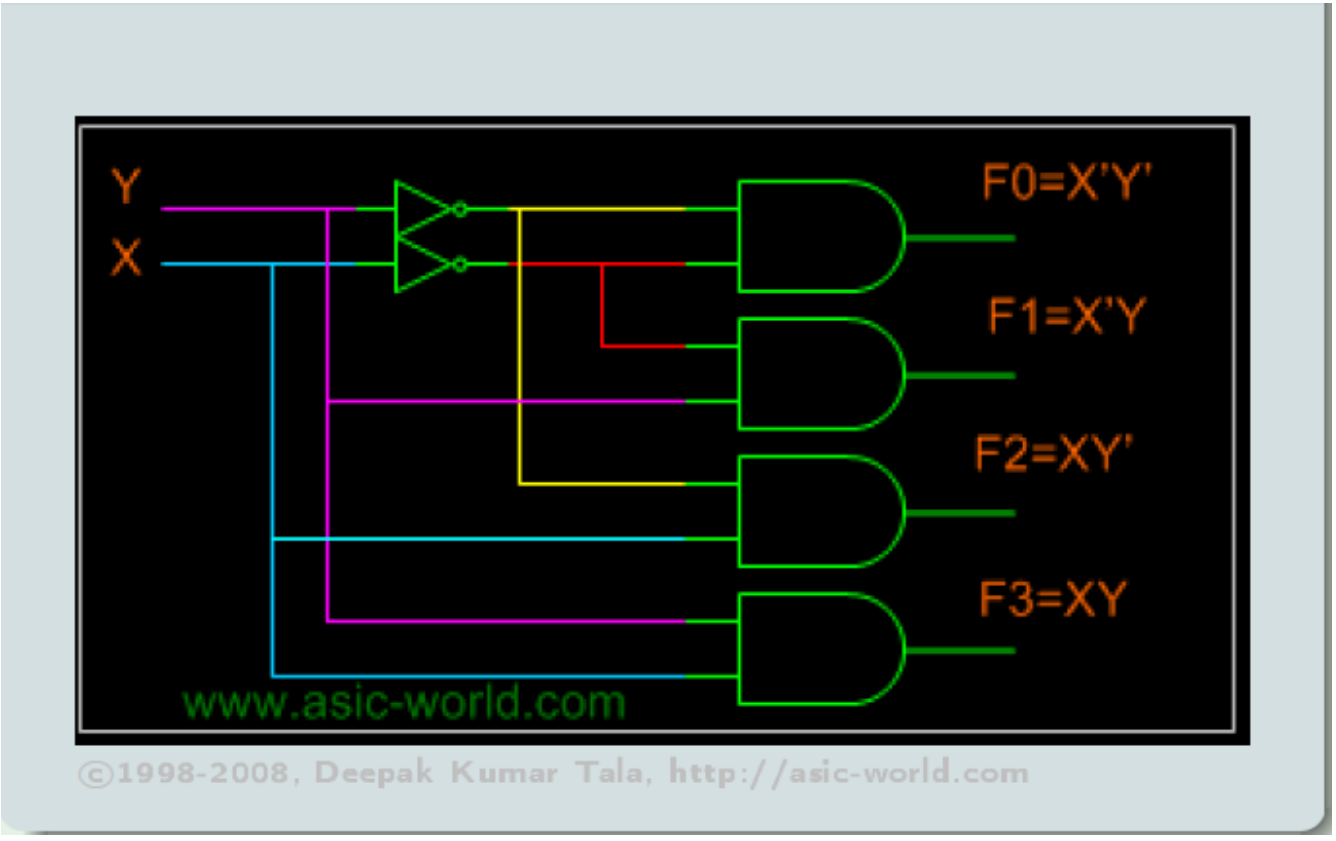
Floyd, Digital Fundamentals, 10th ed

Decodificatoare

- Decodificatorul este un CLC care activează o singură linie de ieșire în funcție de cuvântul de cod aplicat la intrare
- Numărul maxim de linii de ieșire *m* corespunde numărului de combinații posibile ale celor *n* variabile binare de intrare ($m = 2^n$).



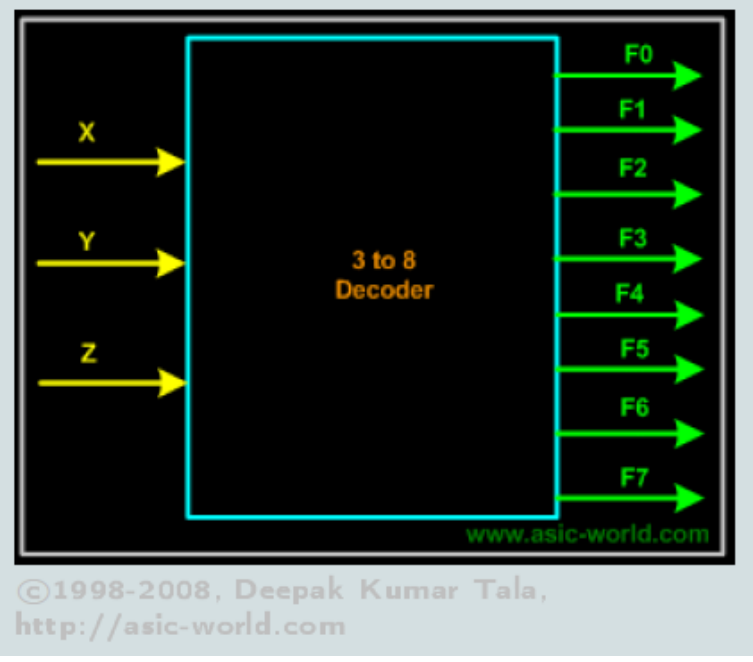
Decodificator binar 2:4

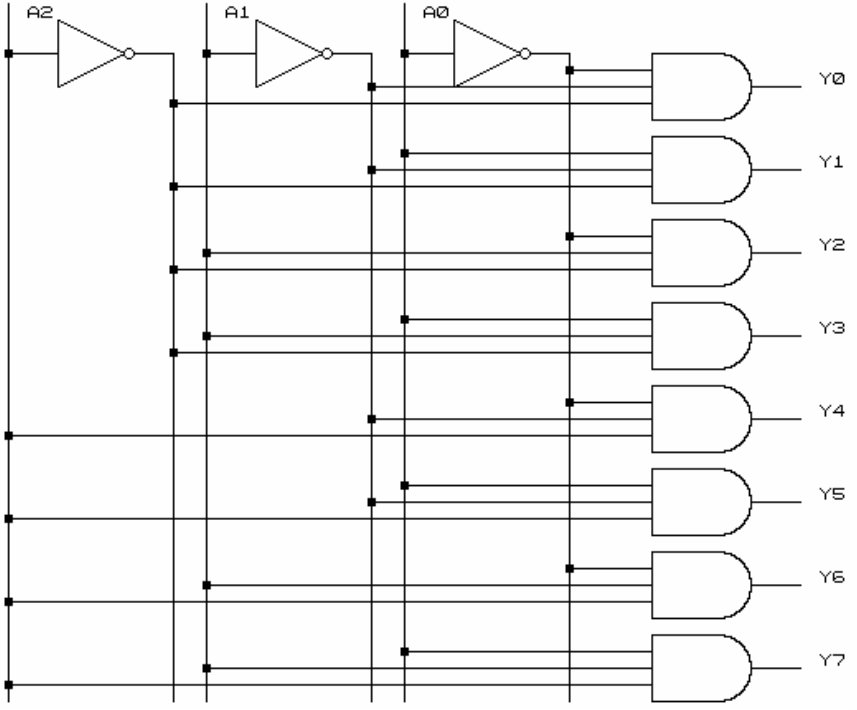


Decodificatorul binar 3:8

Schema bloc generală a decodificatorului complet cu trei intrări (n=3) și 8 ieșiri (m=8).

A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1





Decodificatorul binar 3:8 - VHDL

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

entity decoders_1 is
    port (sel: in std_logic_vector (2 downto 0);
          res: out std_logic_vector (7 downto 0));
    end decoders_1;

architecture archi of decoders_1 is
begin
    res <= "00000001" when sel = "000" else
           "00000010" when sel = "001" else
           "00000100" when sel = "010" else
           "00001000" when sel = "011" else
           "00010000" when sel = "100" else
           "00100000" when sel = "101" else
           "01000000" when sel = "110" else
           "10000000";
end archi;
```

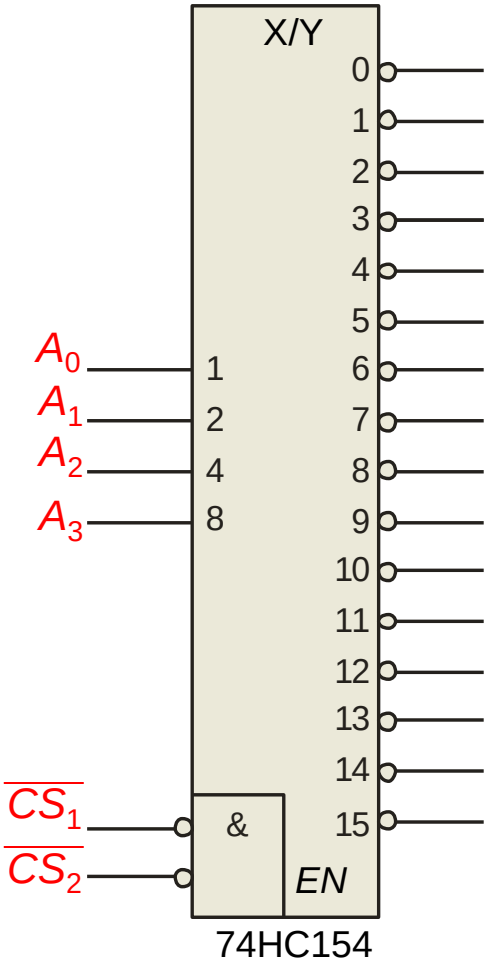
Decodificatorul binar 3:8 - Verilog

```
5   module v_decoders_1 (sel, res);
6       input [2:0] sel;
7       output [7:0] res;
8       reg [7:0] res;
9
10      always @(sel or res)
11      begin
12          case (sel)
13              3'b000 : res = 8'b00000001;
14              3'b001 : res = 8'b00000010;
15              3'b010 : res = 8'b00000100;
16              3'b011 : res = 8'b00001000;
17              3'b100 : res = 8'b00010000;
18              3'b101 : res = 8'b00100000;
19              3'b110 : res = 8'b01000000;
20              default : res = 8'b10000000;
21          endcase
22      end
23  endmodule
```

Decodificatorul complet 4:16

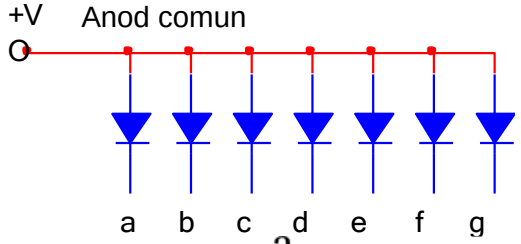
CI 74HC154 decodificatorul 4:16:

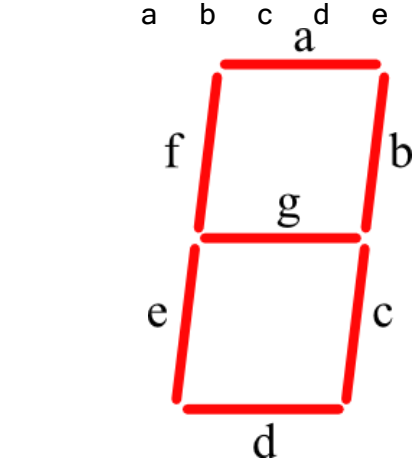
- Două intrări de autorizare CS conectate într-un ȘI
- Intrarea este autorizată dacă ambele intrări au nivel logic jos
- Ieșirile sunt active pe zero.

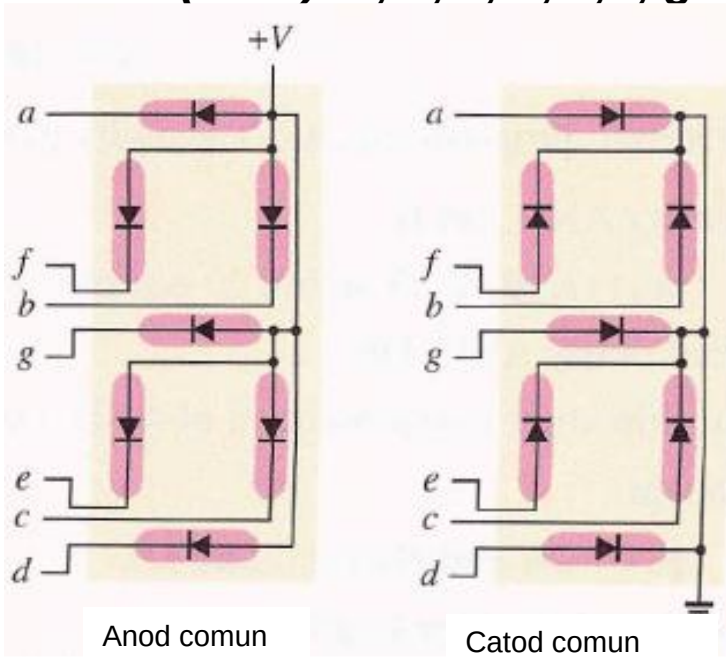


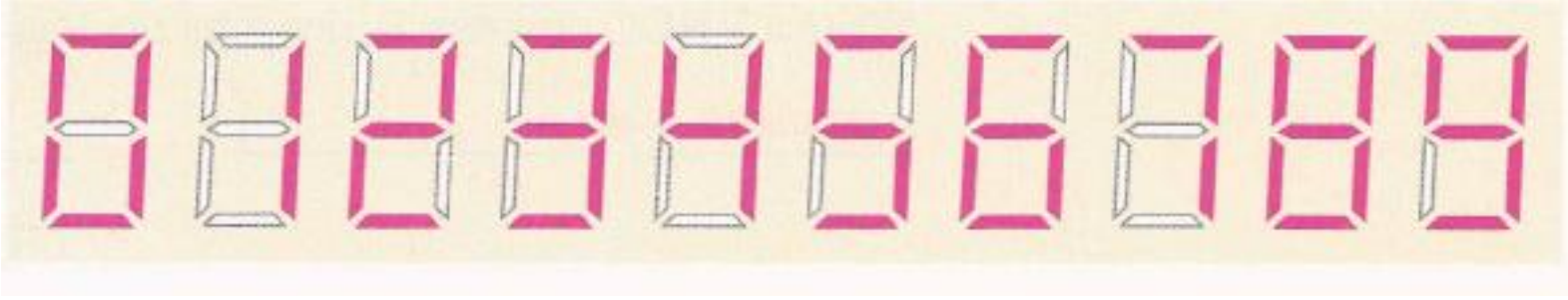
Decodificatorul BCD – 7 segmente

Decodificatorul generează semnalele necesare afișajul cu 7 segmente pornind de la reprezentarea în cod ***BCD (8421):*** **a, b, c, d, e, f, g.**







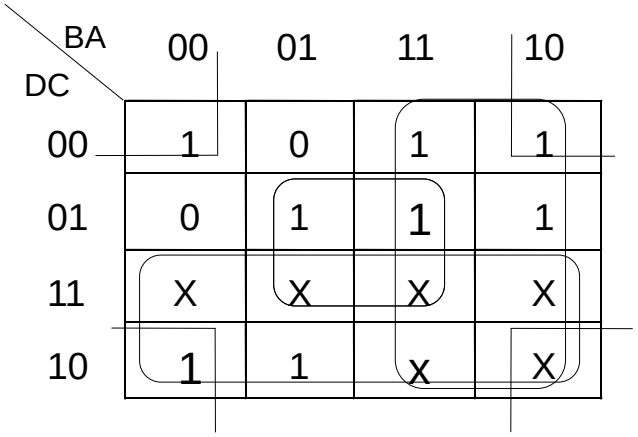


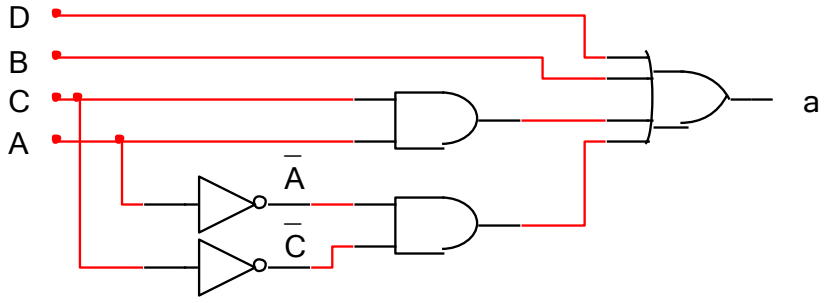
Decodificatorul BCD – 7 segmente II

Pornind de la tabelul de adevăr se poate scrie funcția pentru segmentul a:

$$a = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}B\,\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\,\overline{B}A +$$
$$+ \overline{D}CB\,\overline{A} + \overline{D}CBA + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$$

Zecimal	D	C	B	A	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	1	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	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X





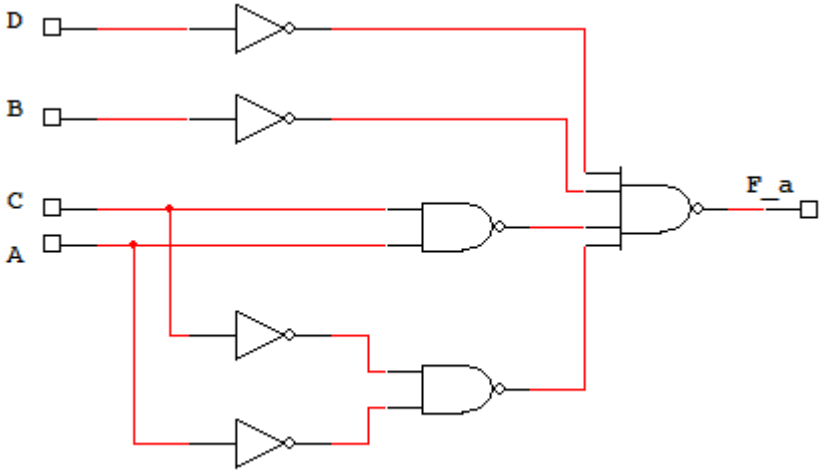
$$a = D + B + CA + \overline{C}\,\overline{A}$$

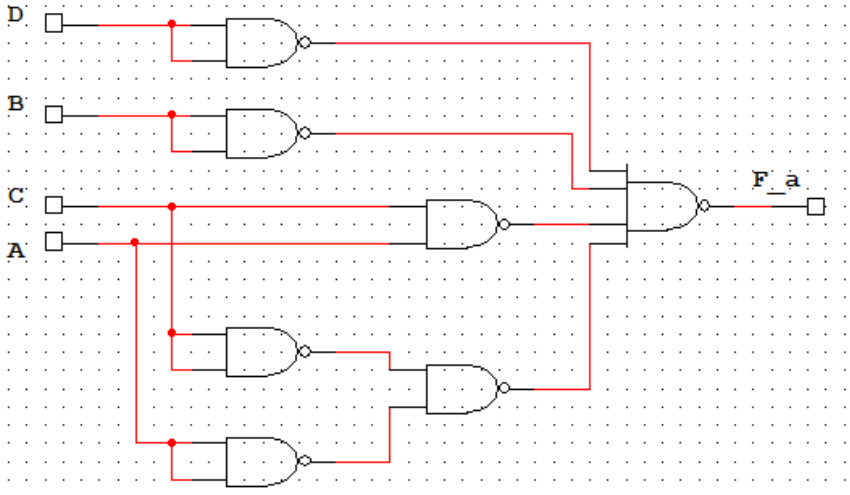
Decodificatorul BCD – 7 segmente III

Realizarea cu porți NAND

- Transformăm funcția cu ajutorul teoremelor lui De Morgan

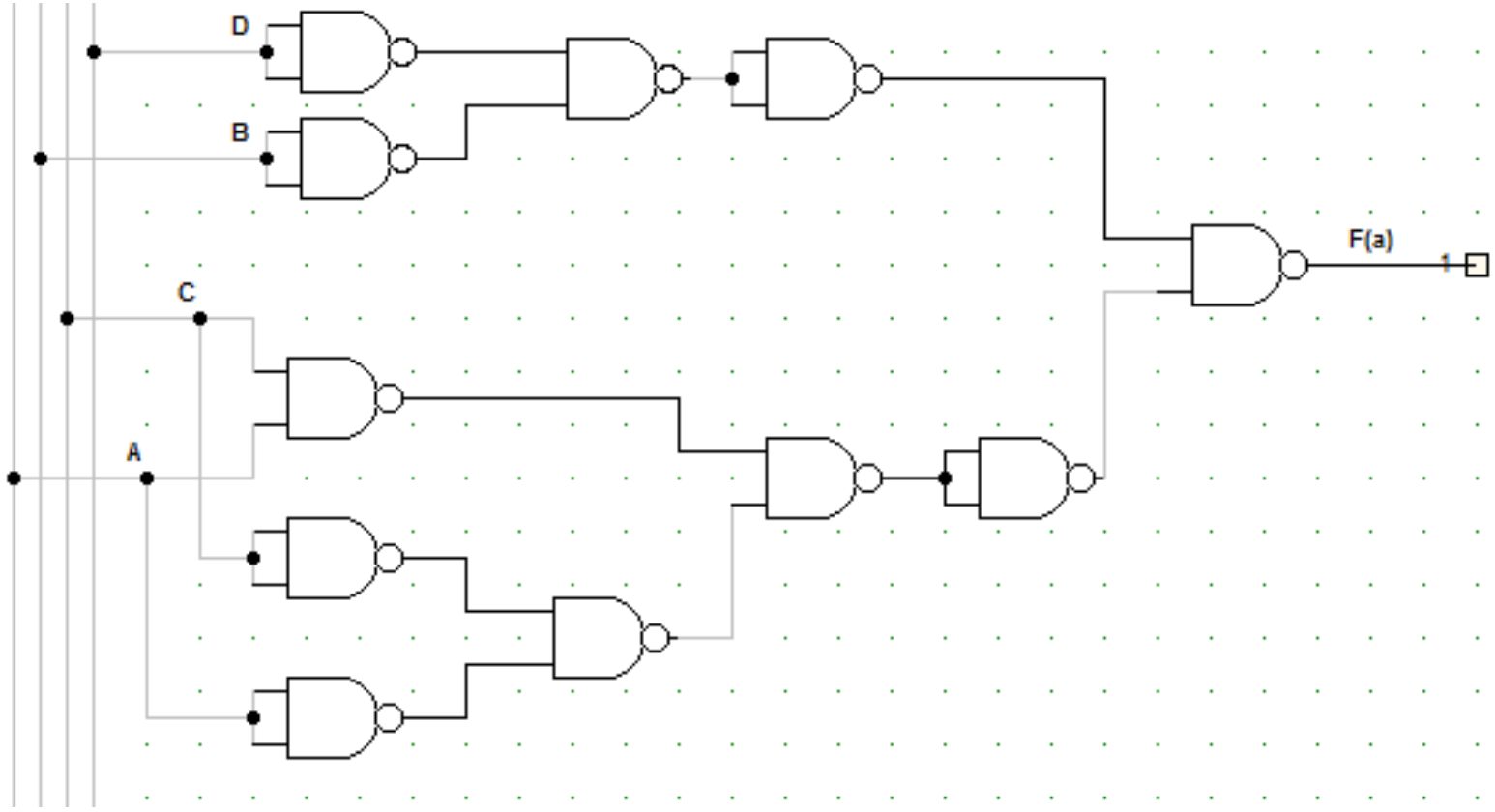
$$a = D + B + CA + \overline{C}\,\overline{A} = \overline{\overline{D + B + CA + \overline{C}\,\overline{A}}} = \overline{\overline{D} \bullet \overline{B} \bullet \overline{C} \bullet A \bullet \overline{C} \bullet \overline{A}}$$





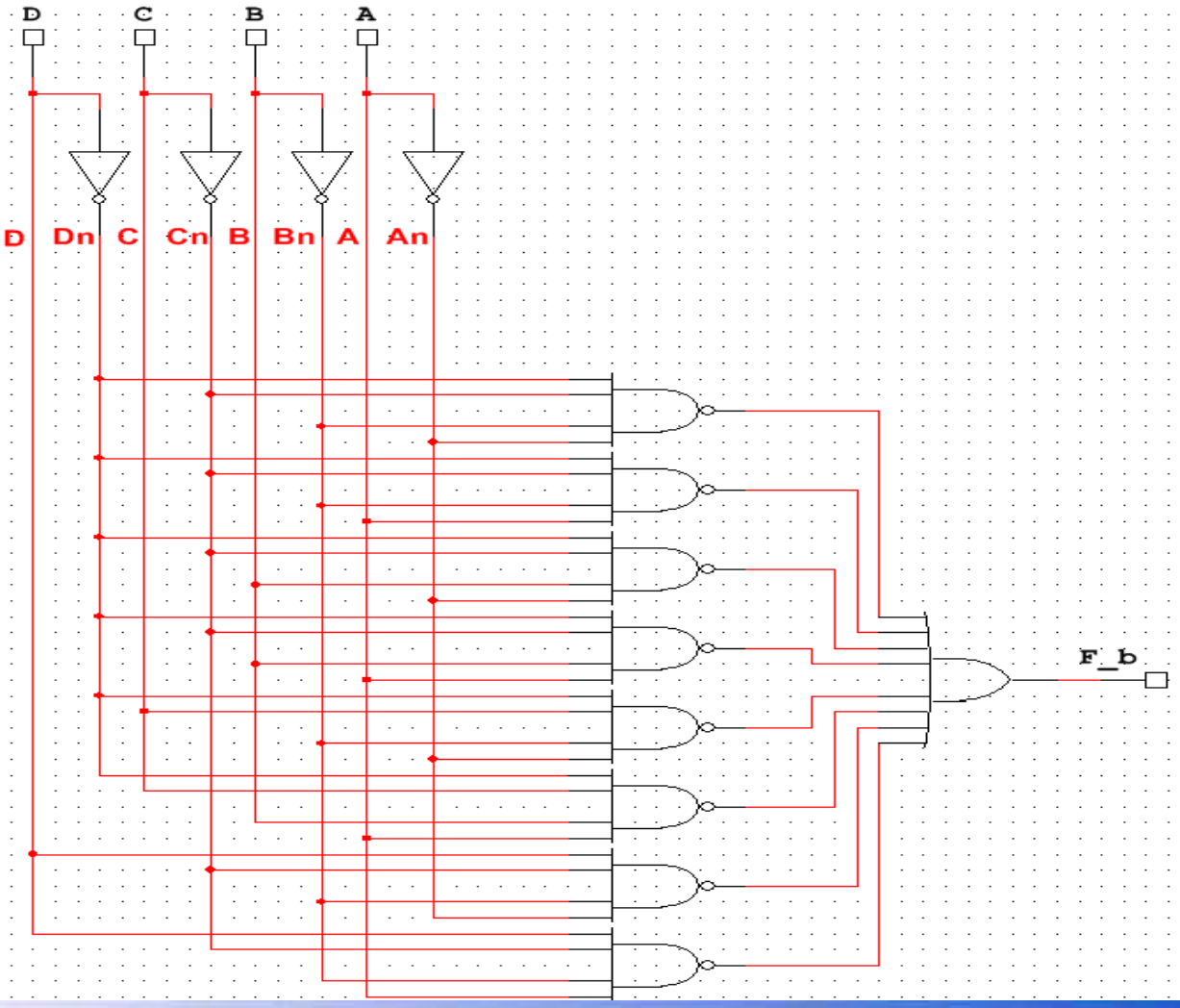
Decodificatorul BCD – 7 segmente IV

Realizarea cu porți NAND cu 2 intrări



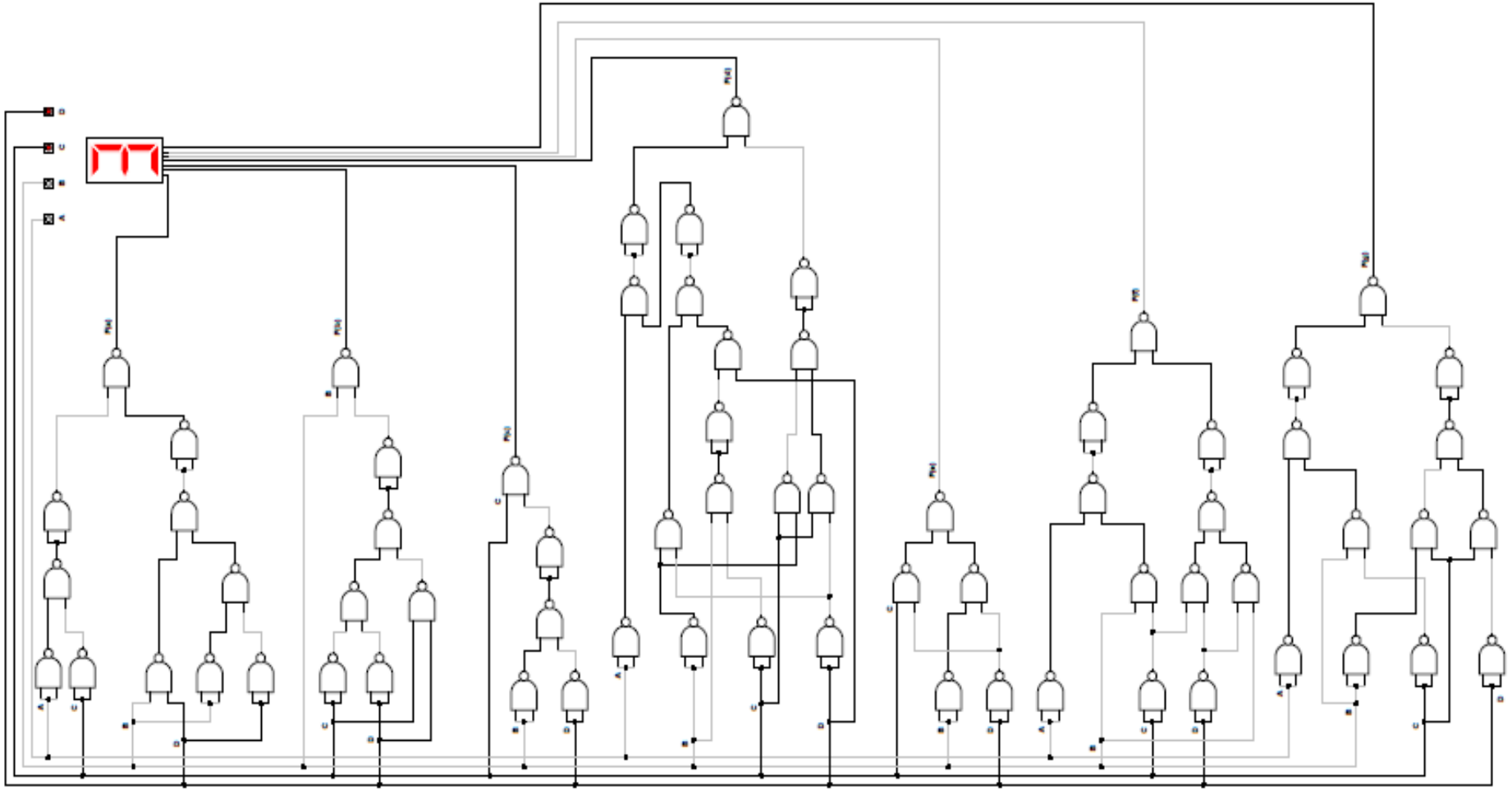
Decodificatorul BCD – 7 segmentul b

$$b = \overline{D}\overline{C}\overline{B}\overline{A} + \overline{D}\overline{C}\overline{B}A + \overline{D}\overline{C}B\overline{A} + \overline{D}\overline{C}BA + \overline{D}C\overline{B}\overline{A} + \overline{D}CBA + D\overline{C}\overline{B}\overline{A} + D\overline{C}\overline{B}A$$

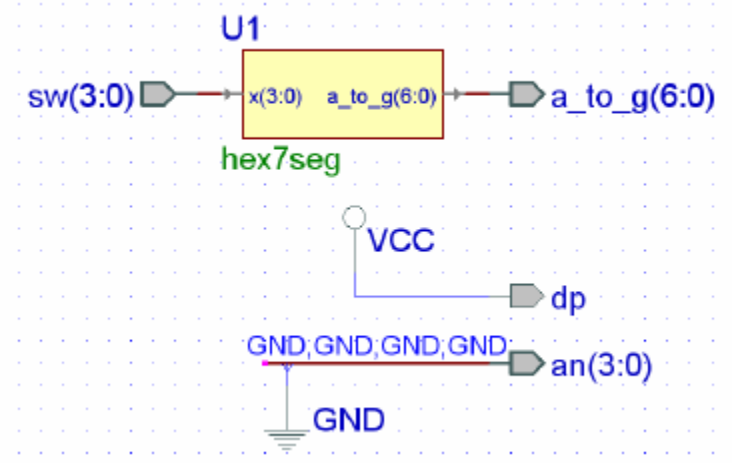


Decodificatorul BCD – 7 segmente V

Simulare folosind programul Digital Works



Decodificatorul BCD – 7 segmente VHDL

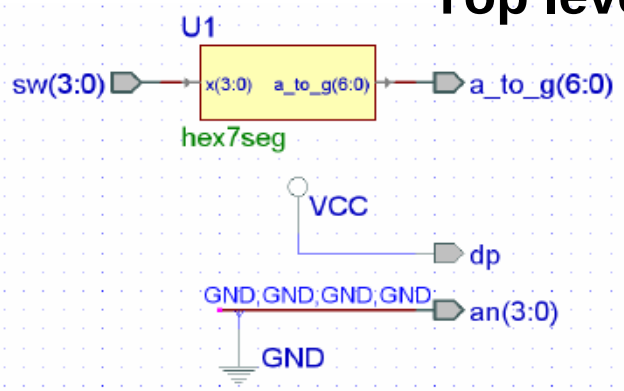



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity hex7seg is
       port(
            x : in STD_LOGIC_VECTOR(3 downto 0);
            a_to_g : out STD_LOGIC_VECTOR(6 downto 0)
           );
end hex7seg;

architecture hex7seg of hex7segbis
begin
    process(x)
    begin
        case x is
            when X"0" => a_to_g <= "0000001";    --0
            when X"1" => a_to_g <= "1001111";    --1
            when X"2" => a_to_g <= "0010010";    --2
            when X"3" => a_to_g <= "0000110";    --3
            when X"4" => a_to_g <= "1001100";    --4
            when X"5" => a_to_g <= "0100100";    --5
            when X"6" => a_to_g <= "0100000";    --6
            when X"7" => a_to_g <= "0001101";    --7
            when X"8" => a_to_g <= "0000000";    --8
            when X"9" => a_to_g <= "0000100";    --9
            when X"A" => a_to_g <= "0001000";    --A
            when X"B" => a_to_g <= "1100000";    --b
            when X"C" => a_to_g <= "0110001";    --C
            when X"D" => a_to_g <= "1000010";    --d
            when X"E" => a_to_g <= "0110000";    --E
            when others => a_to_g <= "0111000";  --F
        end case;
   end process;
end hex7seg;
```

Decodificatorul BCD – 7 segmente Verilog

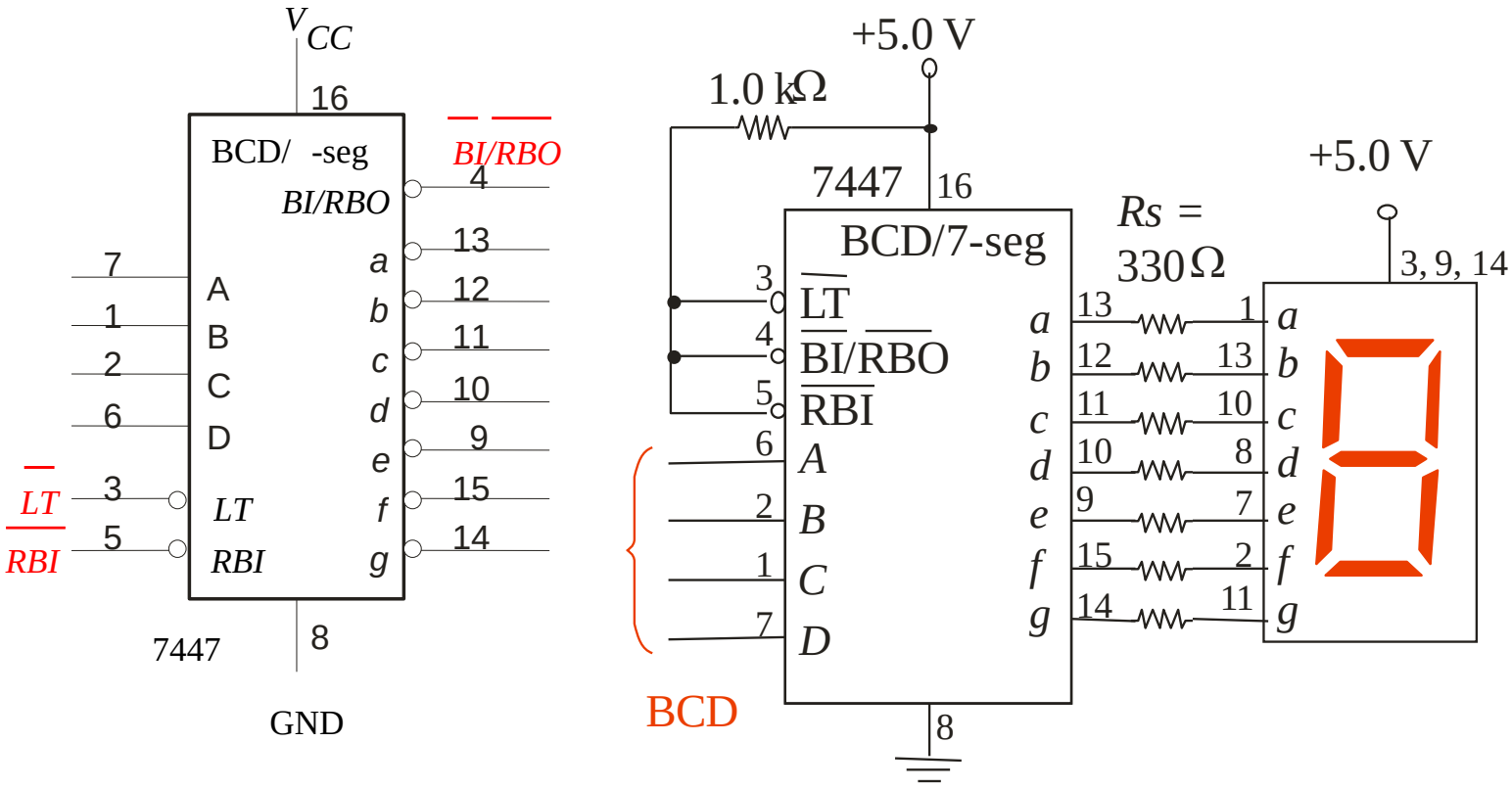

```
module hex7seg ( input [3:0] x, output reg [6:0]
a_to_g );
always @(*)
      case(x)
            0: a_to_g = 7'b0000001;
            1: a_to_g = 7'b1001111;
            2: a_to_g = 7'b0010010;
            3: a_to_g = 7'b0000110;
            4: a_to_g = 7'b1001100;
            5: a_to_g = 7'b0100100;
            6: a_to_g = 7'b0100000;
            7: a_to_g = 7'b0001111;
            8: a_to_g = 7'b0000000;
            9: a_to_g = 7'b0000100;
            'hA: a_to_g = 7'b0001000;
            'hb: a_to_g = 7'b1100000;
            'hC: a_to_g = 7'b0110001;
            'hd: a_to_g = 7'b1000010;
            'hE: a_to_g = 7'b0110000;
            'hF: a_to_g = 7'b0111000;
            default: a_to_g = 7'b0000001; // 0
      endcase
endmodule
```

Top level: hex7seg_top.v




```
module hex7seg_top (input [3:0] sw, output [6:0] a_to_g ,
output [3:0] an, output dp);

assign an = 4'b0000; // all digits on
assign dp = 1; // dp off

hex7seg D4 (.x(sw),.a_to_g(a_to_g));
Endmodule
```

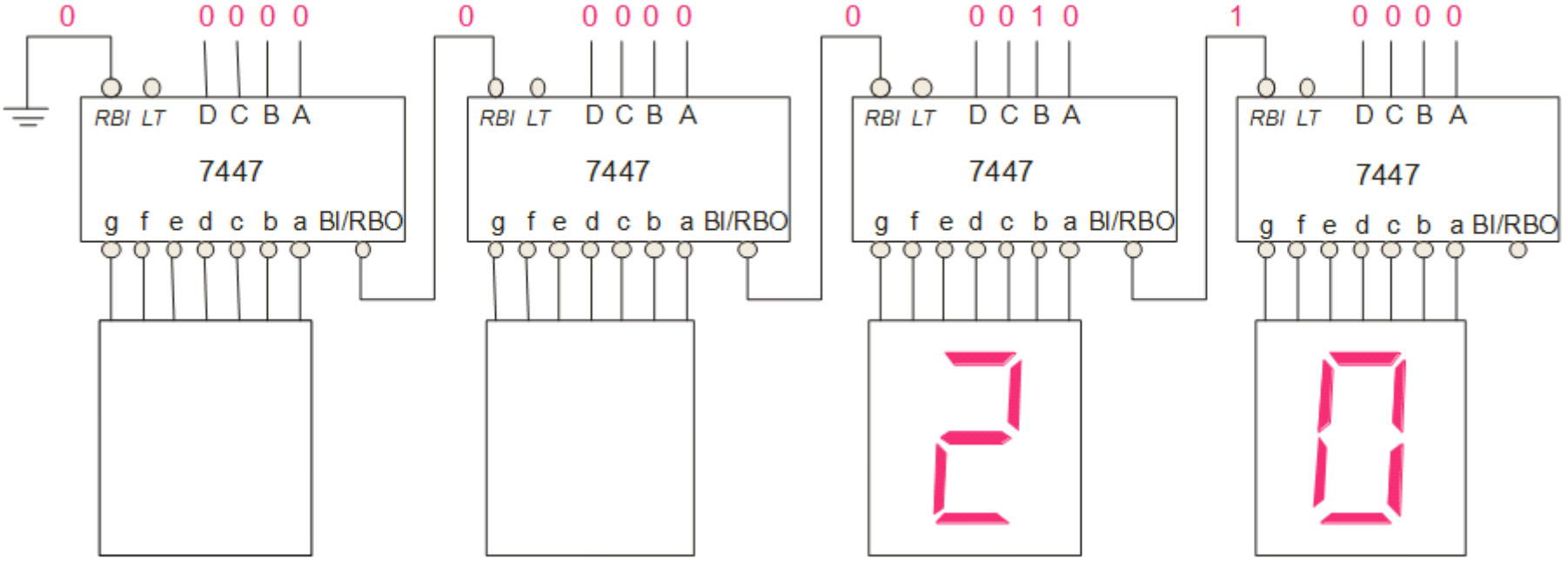
Decodificator BCD – 7 segmente integrat: 7447



Lamp Test ($\overline{LT}$) este utilizat pentru a verifica dacă toate segmentele afișajului sunt în stare de funcționare. Când se aplică un „0” pe intrarea $\overline{LT}$ și / este „1” toate cele 7 segmente vor fi aprinse.

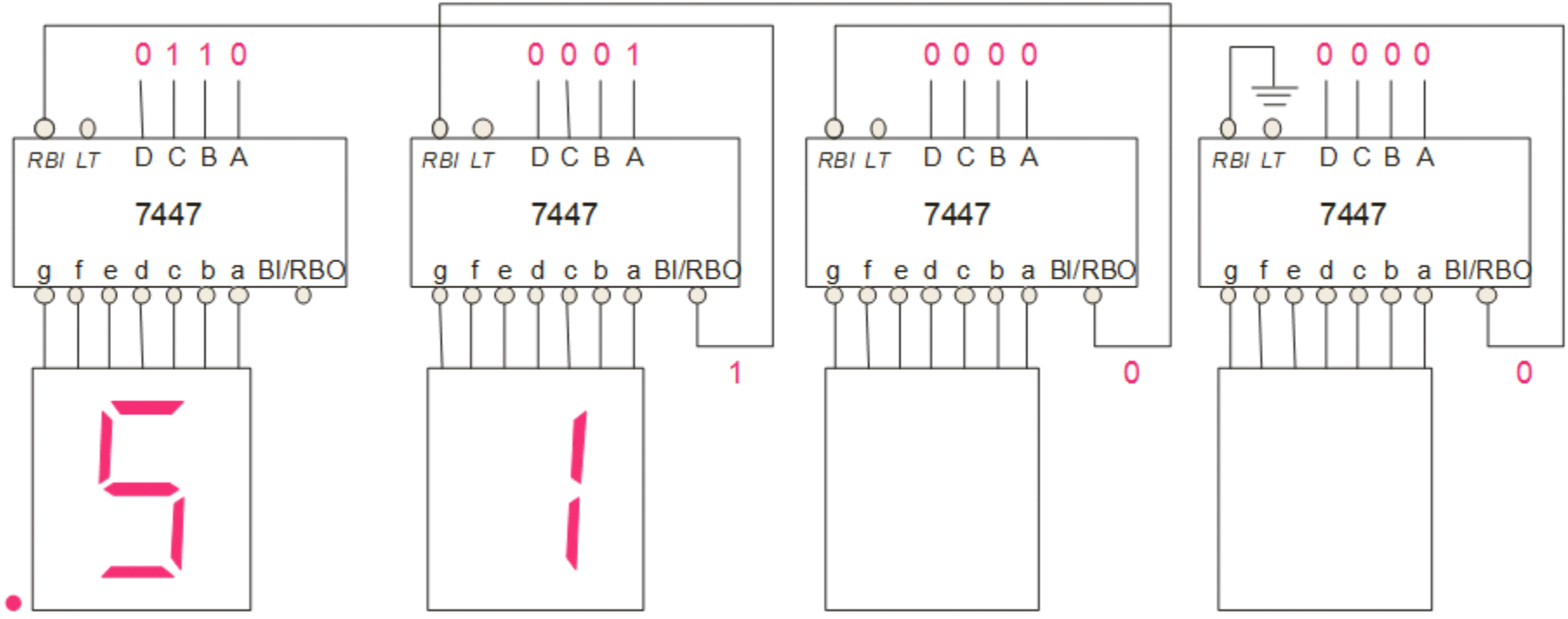
Suprimarea zerourilor I

- Suprimarea zerourilor este o funcție care permite afișajelor compuse din mai mulți digiți (mai multe celule de afișare cu 7 segmente) să nu afișeze zerourile care nu sunt necesare. De exemplu un afișaj cu 4 digiți ar putea afișa cifra 4.5 ca 04.50 daca nu se utilizează această funcție.
- Pentru implementarea acestei funcții se utilizează două terminale suplimentare, și anume (Ripple Blanking Input) și / (Ripple Blanking Output).
- Ex. Suprimarea zerourilor din față (leading):



Suprimarea zerourilor II

Ex. Suprimarea zerourilor din coadă (trailing):

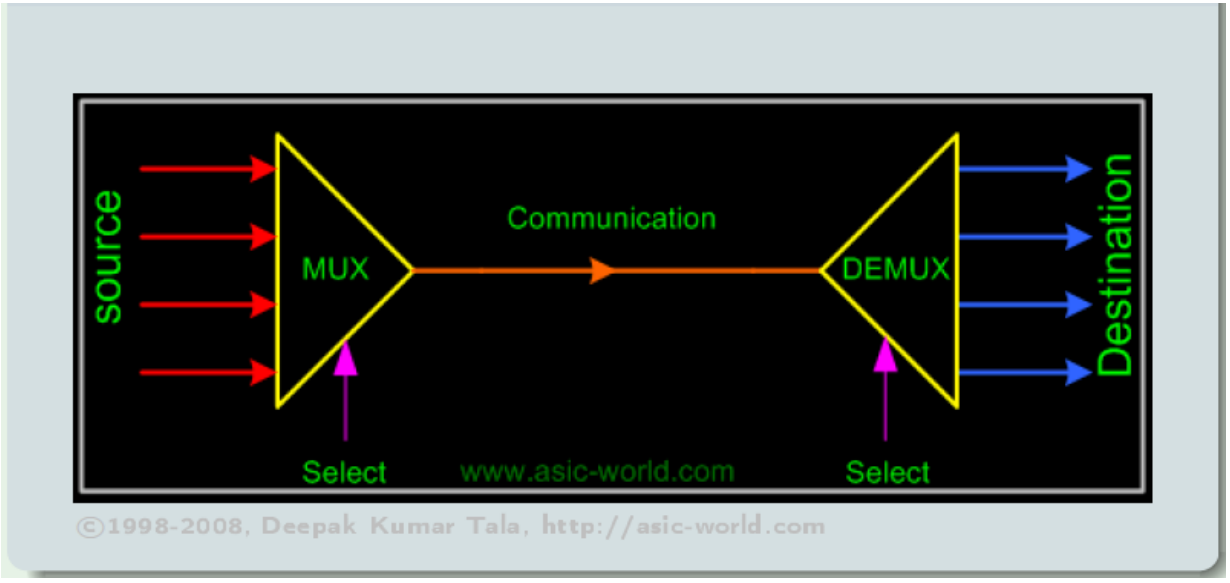


• Ultimul digit (de ordin cel mai mic) va afișa întotdeauna zero dacă la intrarea lui se aplică „0000” deoarece intrarea lui este conectată la masă

• Digiții de ordin mai mare vor afișa zero numai dacă intrările lor conectate la ieșirile ale digiților de ordin mai mic sunt zero.

MUX-DEMUX

- In cazul unui număr mai mic de canale de transmisie (fir, unde radio, etc.) utilizând tehnica de multiplexare-demultiplexare – se poate realiza transmisia unui număr mare de semnale.

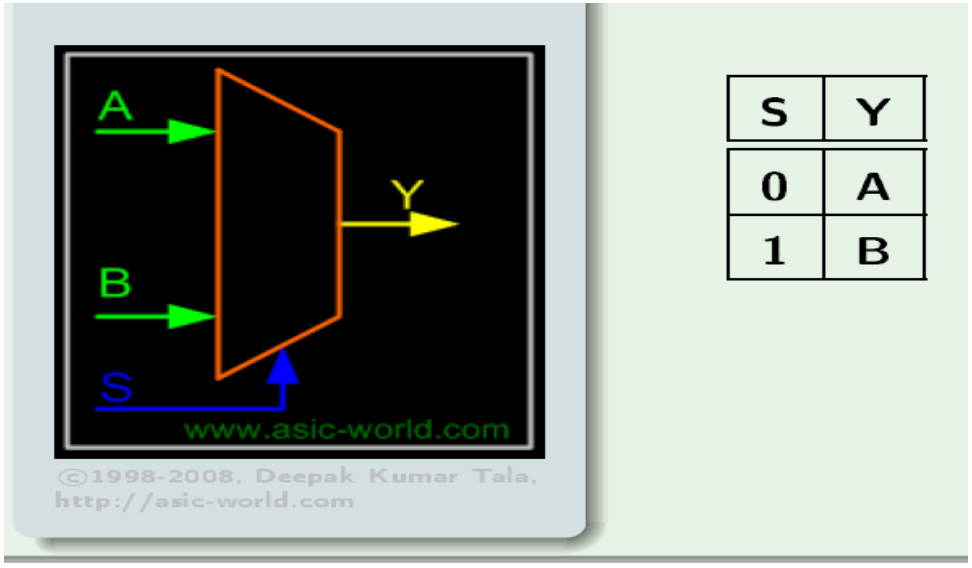


- Tipuri de multiplexare:
  - Multiplexare analogică, **multiplexare digitală**
  - Multiplexare datelor **organizate pe bit** sau byte
  - Multiplexare în timp sau funcție de **adresă**

Circuite de multiplexare (MUX)

- Circuitele de multiplexare sunt circuite logice combinaționale care permit trecerea datelor de la una din cele *m* ($m=2^n$) căi de intrare la o cale de ieșire unică.
- Pentru 2^n intrări sunt necesare 2^n adrese diferite
- 2^n adrese diferite pot fi generate folosind n biți.

- **Ex. Multiplexor cu 2 intrări**



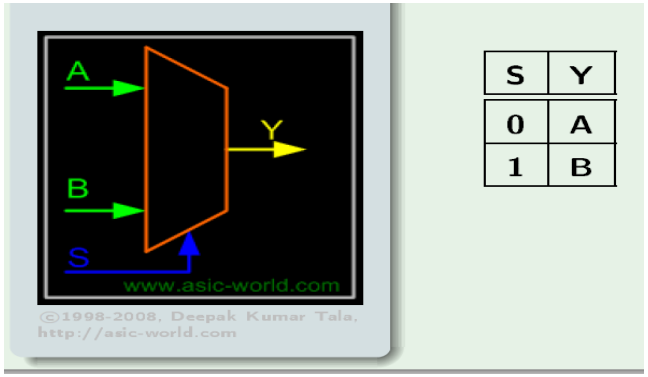
S	Y
0	A
1	B

Exemplu – MUX 2:1 în cod Verilog

```
module mux_21 (input A, B, S, output Y);
assign Y = (S==1'b1) ? B : A;
endmodule
```

```
module mux_21 (input A, B, S, output reg Y);
always @ (*)
if (S==1'b1) Y <= B;
else         Y <= A;
endmodule
```

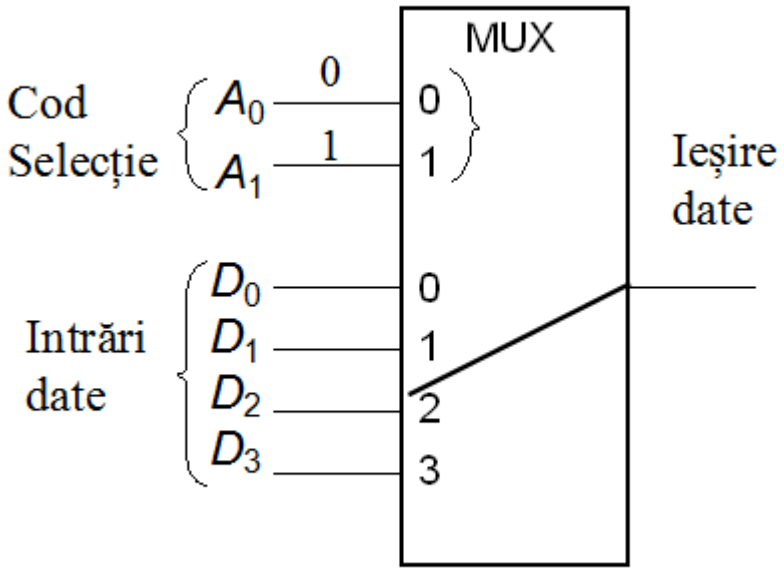
```
module mux_21 (input A, B, S, output reg Y);
always @ (*)
case(S)
    1'b0:    Y <= A;
    1'b1:    Y <= B;
endmodule
```



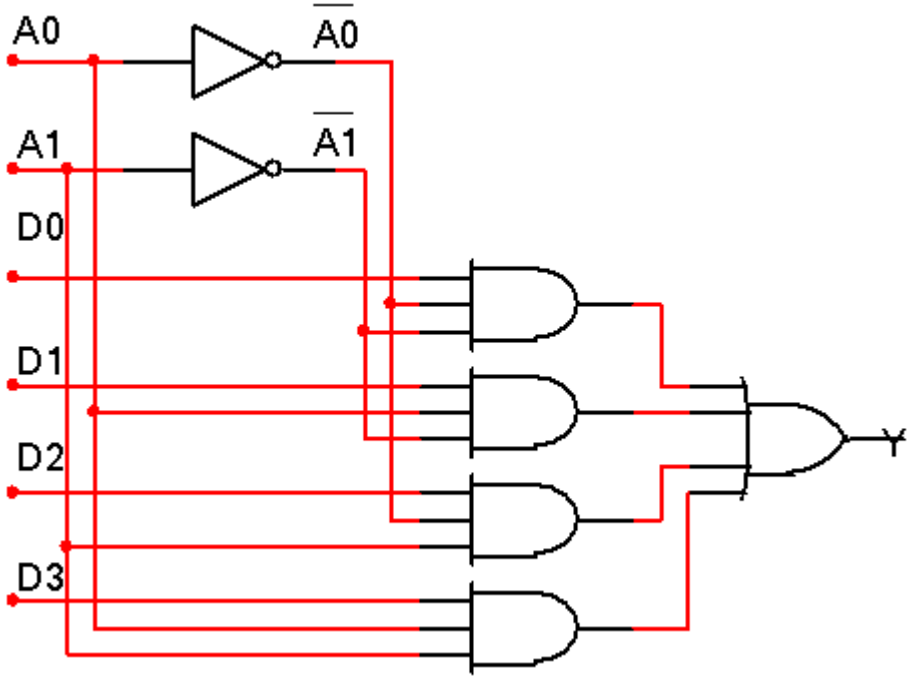
S	Y
0	A
1	B

Multiplexor cu 4 intrări

Schema bloc

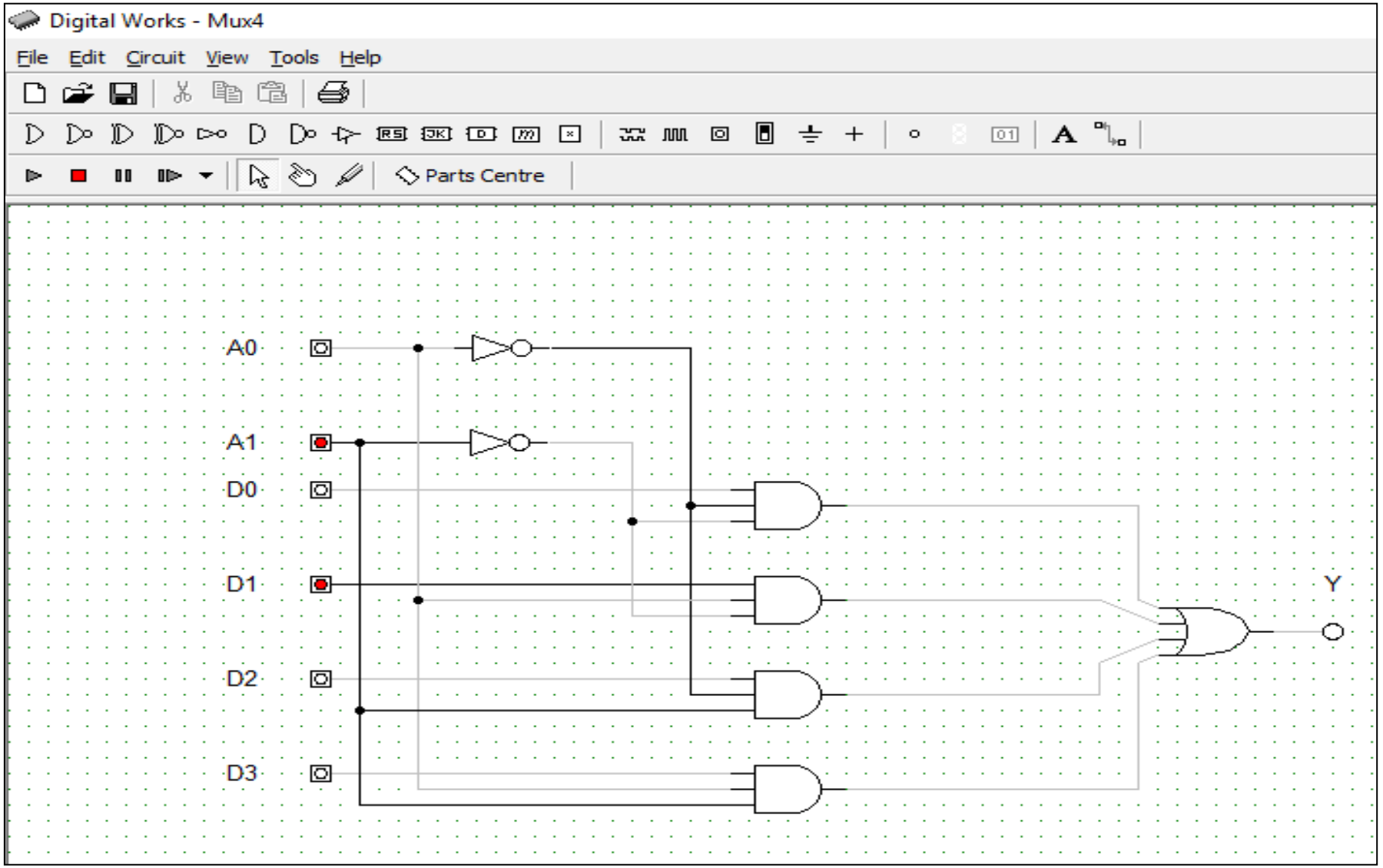


Schema detailată a MUX cu 4 intrări



Simularea unui circuit multiplexor

- http://www.play-hookey.com/digital/combinational/multiplexer_four_input.html
- Digital Works – program de simulare a circuitelor digitale



Multiplexor cu 4 intrări (VHDL)

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

entity multiplexers_1 is
    port (a, b, c, d : in std_logic;
          s : in std_logic_vector (1 downto 0);
          o : out std_logic);
end multiplexers_1;

architecture archi of multiplexers_1 is
begin
    process (a, b, c, d, s)
    begin
        if (s = "00") then o <= a;
        elsif (s = "01") then o <= b;
        elsif (s = "10") then o <= c;
        else o <= d;
        end if;
    end process;
end archi;
```

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

entity multiplexers_2 is
    port (a, b, c, d : in std_logic;
          s : in std_logic_vector (1 downto 0);
          o : out std_logic);
end multiplexers_2;

architecture archi of multiplexers_2 is
begin
    process (a, b, c, d, s)
    begin
        case s is
            when "00" => o <= a;
            when "01" => o <= b;
            when "10" => o <= c;
            when others => o <= d;
        end case;
    end process;
end archi;
```

Multiplexor cu 4 intrări (Verilog)

```
module v_multiplexers_1 (a, b, c, d, s, o);
    input a,b,c,d;
    input [1:0] s;
    output o;
    reg o;

    always @(a or b or c or d or s)
    begin
        if (s == 2'b00) o = a;
        else if (s == 2'b01) o = b;
        else if (s == 2'b10) o = c;
        else o = d;
    end
endmodule
```

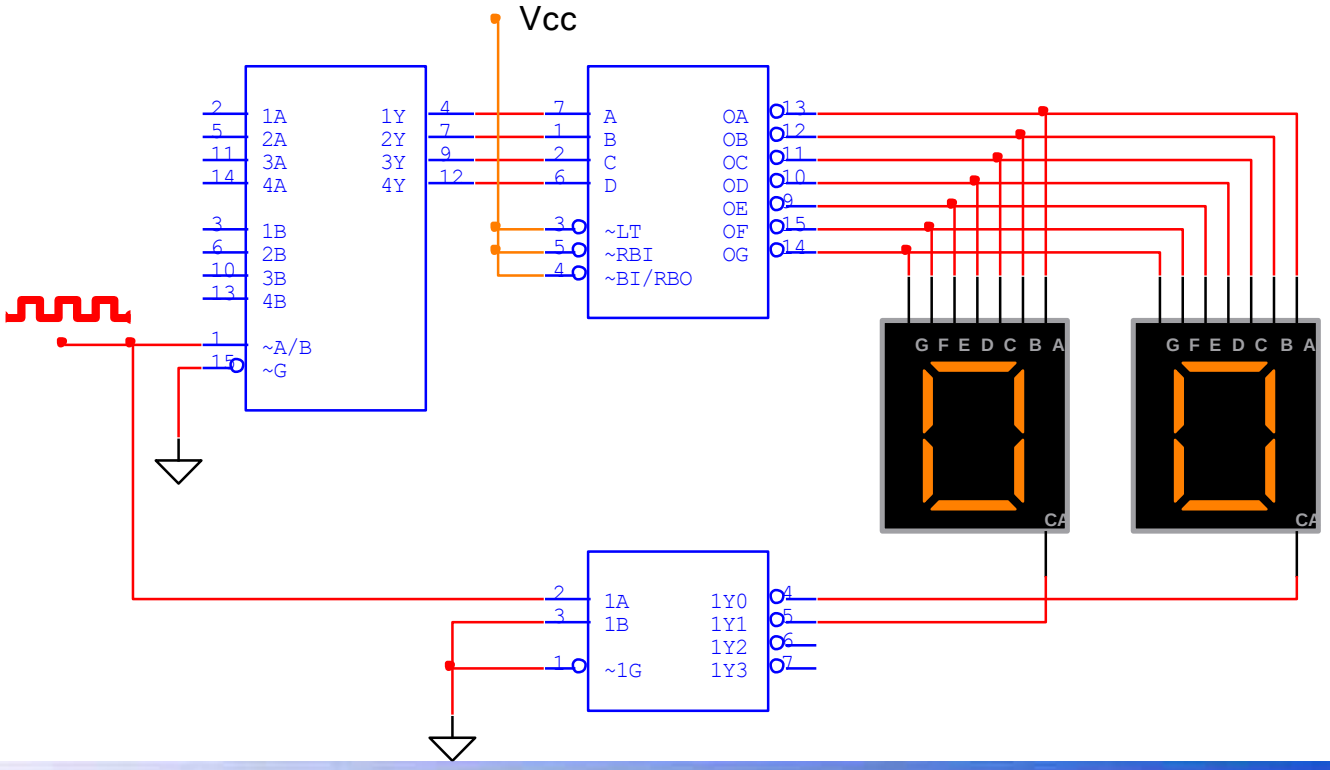
```
module v_multiplexers_2 (a, b, c, d, s, o);
    input a,b,c,d;
    input [1:0] s;
    output o;
    reg o;

    always @(a or b or c or d or s)
    begin
        case (s)
            2'b00 : o = a;
            2'b01 : o = b;
            2'b10 : o = c;
            default : o = d;
        endcase
    end
endmodule
```

Aplicații ale multiplexoarelor I

Utilizarea unui multiplexor pentru afișarea datelor

Afișaj cu mai mulți digiți utilizând un singur decodificator BCD/7 segmente

- multiplexor quadruplu cu câte doua intrări (A și B) - 74LS157
- Decodificator BCD/7 segmente - 74LS47
- Decodificator 2:4 74LS139

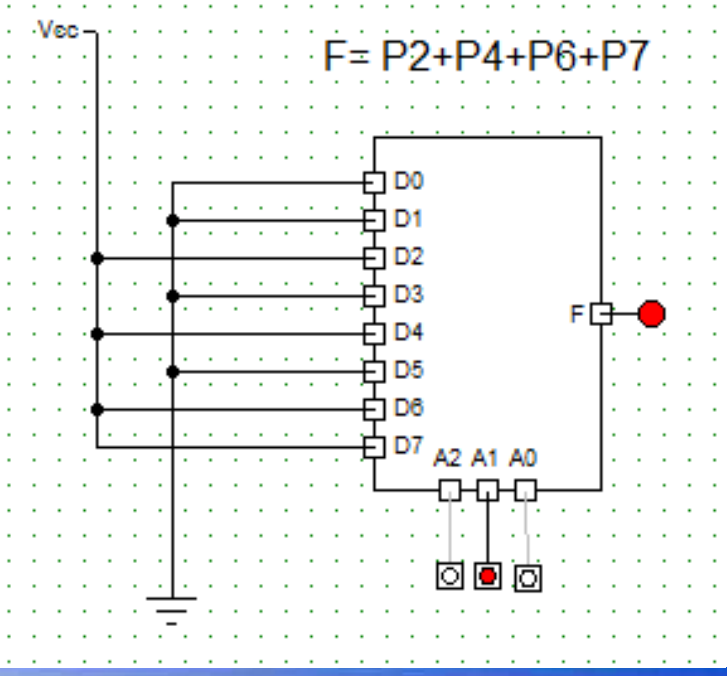


Aplicații ale multiplexoarelor II

Implementarea funcțiilor logice combinaționale

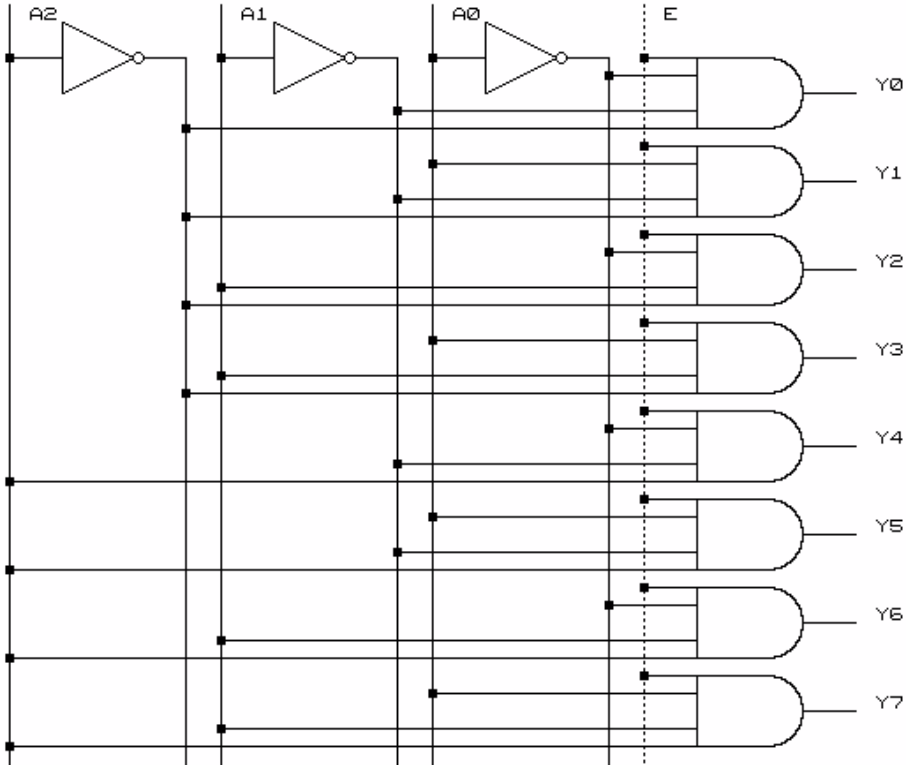
- Multiplexoarele sunt CLC realizate cu porți SI-NU deci pot servi pentru implementarea funcțiilor logice combinaționale in forma sumă de produse (SOP).
- la intrările de selecție ale multiplexorului se leagă variabilele funcției
- la intrările de date ale multiplexorului se leagă nivelele logice specificate în tabelul de adevăr pentru funcția respectivă

Intrări			Ieșire
C	B	A	Y
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	1



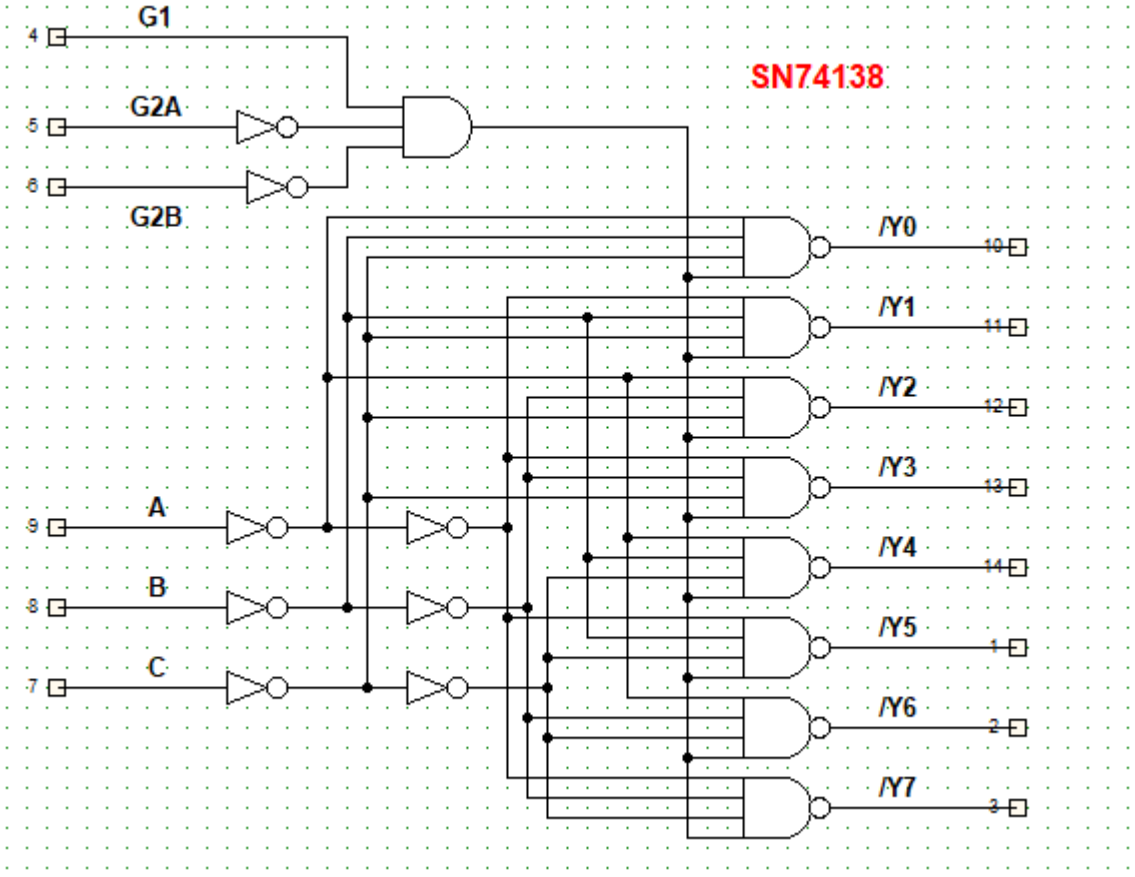
Demultiplexoare

- Demultiplexorul este un CLC care permite transmiterea datelor de pe o intrare de date comună pe una din cele 2^n ieșiri (cea selectată)
- Selectarea ieșirii se face printr-un cuvânt de cod numit adresă de n-biți.
- Intrarea de date a demultiplexorului este intrarea de autorizare a decodificatorului.



Simularea unui circuit demultiplexor

http://www.play-hookey.com/digital/combinational/decoder_demux_four.html

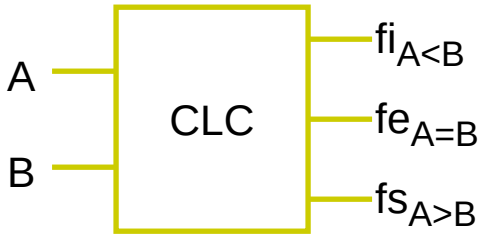


Circuite logice combinaționale II

Comparatoare digitale

- Indică relația dintre două numere (*egale, mai mic, mai mare)*.
- Dintre cele 3 doar una poate fi adevărată

Comparator de 1 bit

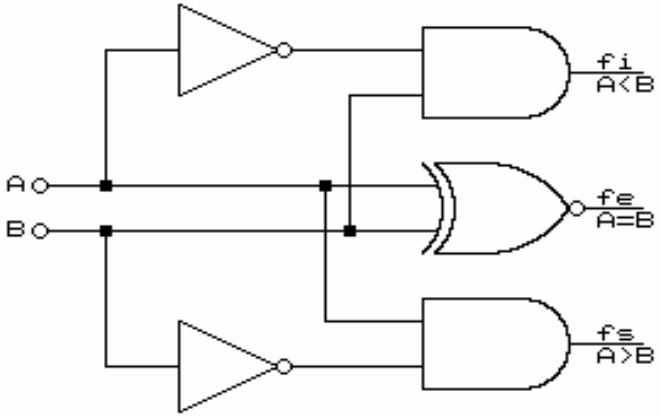


A	B	$fi_{A<B}$	$fe_{A=B}$	$fs_{A>B}$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

$$f_e = \overline{A \oplus B} = \overline{A \bullet \overline{B} + \overline{A} \bullet B} = A \bullet B + \overline{A} \bullet \overline{B}$$

$$f_i = \overline{A} \bullet B$$

$$f_s = A \bullet \overline{B}$$



Comparator pe 2 biți I

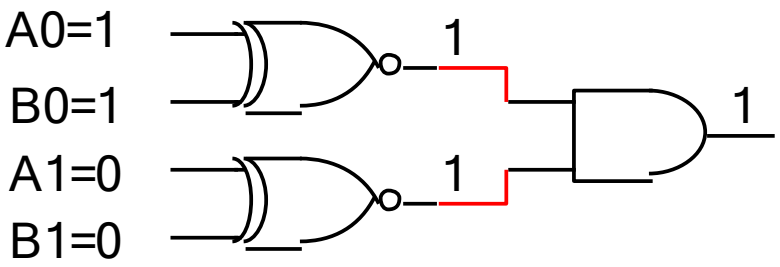
- Fie 2 numere:

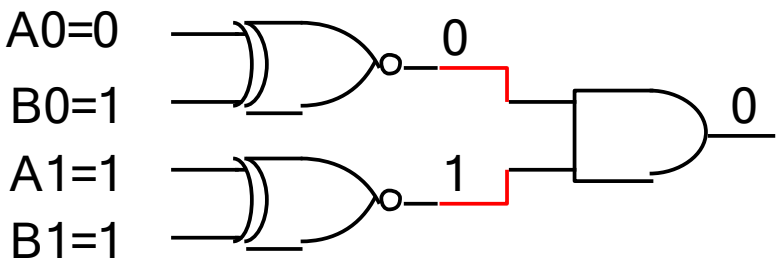
$$A = A_1 \cdot 2^1 + A_0 \cdot 2^0$$

$$B = B_1 \cdot 2^1 + B_0 \cdot 2^0$$

- Două numere de 2 biți sunt egale dacă biții corespunzători sunt egali

$$F_{A=B} = \overline{A_1 \oplus B_1} \bullet \overline{A_0 \oplus B_0} = (A_1B_1 + \overline{A_1}\,\overline{B_1})(A_0B_0 + \overline{A_0}\,\overline{B_0})$$





Comparator pe 2 biți II

- Relațiile care indică inegalitatea:

- A < B este adevărată, dacă A1 < B1,
  sau dacă A1 = B1 și A0 < B0

$$F_{A<B} = \overline{A_1} B_1 + (A_1 B_1 + \overline{A_1}\,\overline{B_1})\overline{A_0} B_0$$

- A > B este adevărată, dacăA1 > B1,
  sau dacă A1 = B1 și A0 > B0

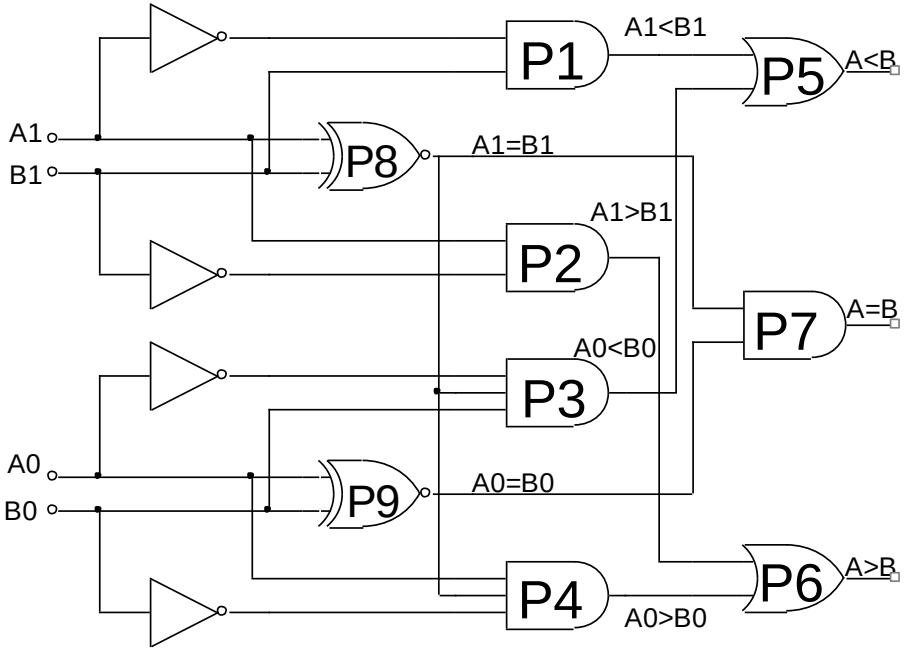
$$F_{A>B} = A_1 \overline{B_1} + (A_1 B_1 + \overline{A_1}\,\overline{B_1}) A_0 \overline{B_0}$$

Comparator pe 2 biți III

$$F_{A=B} = \overline{A_1 \oplus B_1} \bullet \overline{A_0 \oplus B_0}$$

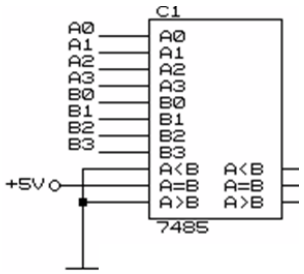
$$F_{A>B} = A_1 \overline{B_1} + (A_1 B_1 + \overline{A_1}\,\overline{B_1}) A_0 \overline{B_0}$$

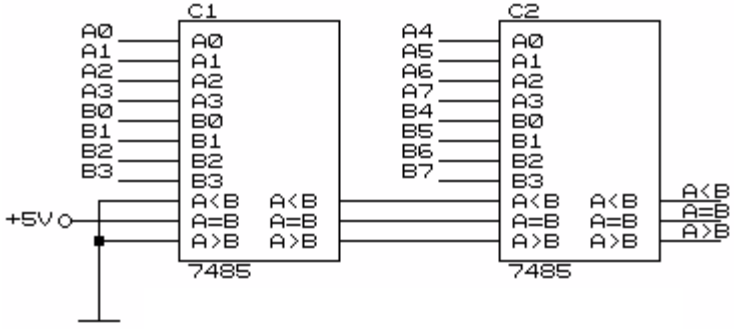
$$F_{A<B} = \overline{A_1} B_1 + (A_1 B_1 + \overline{A_1}\,\overline{B_1}) \overline{A_0} B_0$$



Comparatoare pe mai mulți biți

- Comparator pe 4 biți
  - CI de tipul SN 7485
  - Intrări
    - Biții celor două numere de comparat (A_0, A_1, A_2, A_3 si B_0, B_1, B_2, B_3)
    - **Intrări de expandare** $A_i<B_i$, $A_i=B_i$, $A_i>B_i$, la care se leagă rezultatul comparării celor patru biți mai puțini semnificativi.
  - Ieșiri, care arată care dintre relațiile (A<B, A=B, A>B) este adevărată.
  - Extinderea serială a comparatoarelor:





Comparatoare pe mai mulți biți VHDL

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

entity comparator_1 is
    port(A,B : in  std_logic_vector(7 downto 0);
         A_lt_B, A_eq_B, A_gt_B : out std_logic);
end comparator_1;

architecture archi of comparator_1 is
begin

    A_lt_B <= '1' when A < B else '0';
    A_eq_B <= '1' when A = B else '0';
    A_gt_B <= '1' when A > B else '0';

end archi;
```

Comparatoare pe mai mulți biți Verilog

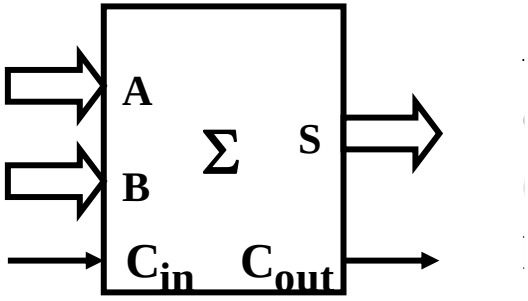
```
module v_comparator_2 (input  [7:0] A, B, output A_lt_B, A_eq_B, A_gt_B);

    assign A_lt_B = (A < B) ? 1'b1 : 1'b0;
    assign A_eq_B = (A = B) ? 1'b1 : 1'b0;
    assign A_gt_B = (A > B) ? 1'b1 : 1'b0;

endmodule
```

```
module compare_2 (output reg A_lt_B, A_gt_B, A_eq_B,
                  input [7:0] A,B);

always @ (A or B)
        begin
            A_lt_B = 0;
            A_gt_B = 0;
            A_eq_B = 0;

            if (A==B) A_eq_B = 1;
            else if (A>B) A_gt_B = 1;
            else A_lt_B = 1;

        end

endmodule
```

Sumatoare

- Porturile unui sumator (adder)
  - intrări
    - cele două numere de adunat A și B
    - transportul de la poziția mai puțin semnificativă (Cin-carry)
  - ieșiri
    - suma (S)
    - transportul (Cout)

- Semisumatoare (half adder)
- Sumatoare complete (full adder)



După modul de funcționare:

- Sumatoare seriale
- Sumatoare paralele

În funcție de codificarea operanzilor:

- Sumatoare binare
- Sumatoare BCD

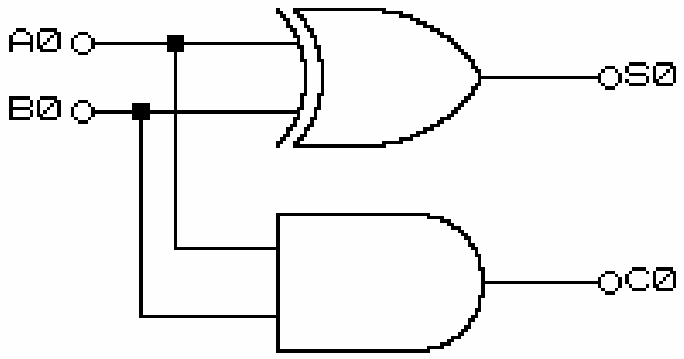
Semisumatoare

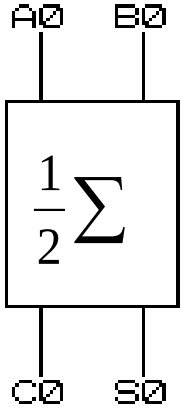
Nu țin cont de transportul de la bitul mai puțin semnificativ
Se pot folosi doar în poziția bitului cel mai puțin semnificativ

A_0	B_0	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S_0 = \overline{A}_0 \bullet B_0 + A_0 \bullet \overline{B}_0 = A \oplus B$$

$$C_0 = A_0 \bullet B_0$$





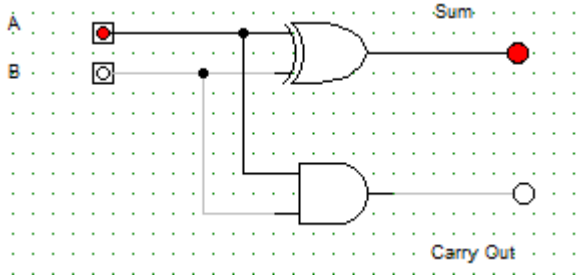
Semisumatoare

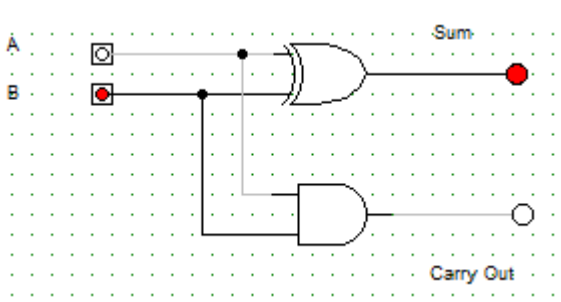
Nu țin cont de transportul de la bitul mai puțin semnificativ
Se pot folosi doar în poziția bitului cel mai puțin semnificativ

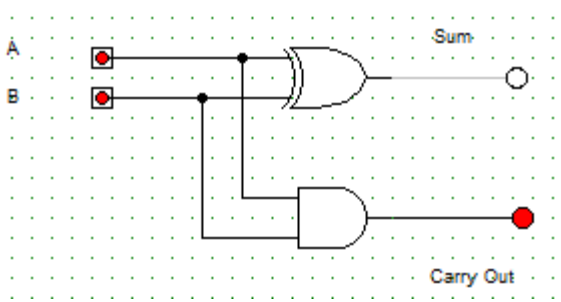
A_0	B_0	S_0	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S_0 = \overline{A}_0 \bullet B_0 + A_0 \bullet \overline{B}_0 = A \oplus B$$

$$C_0 = A_0 \bullet B_0$$







Sumatoare complete

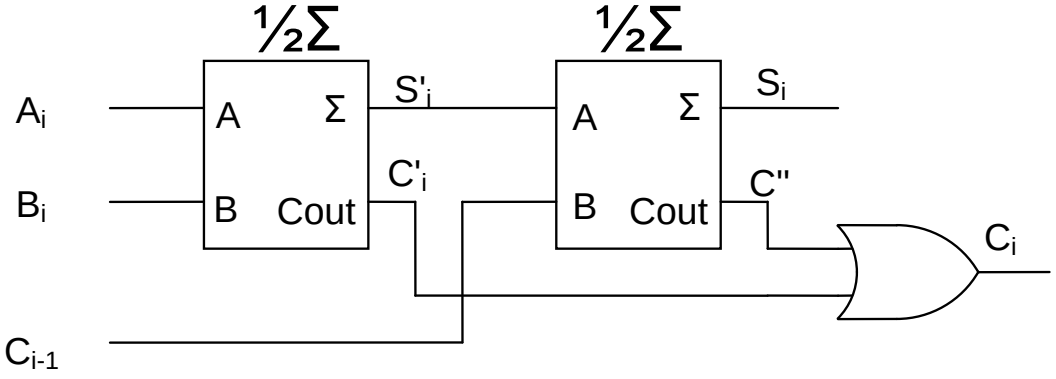
Țin cont de transportul de la bitul mai puțin semnificativ

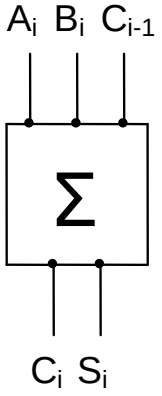
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

Un sumator complet se poate obține utilizând două semisumatoare:





Sumatoare complete

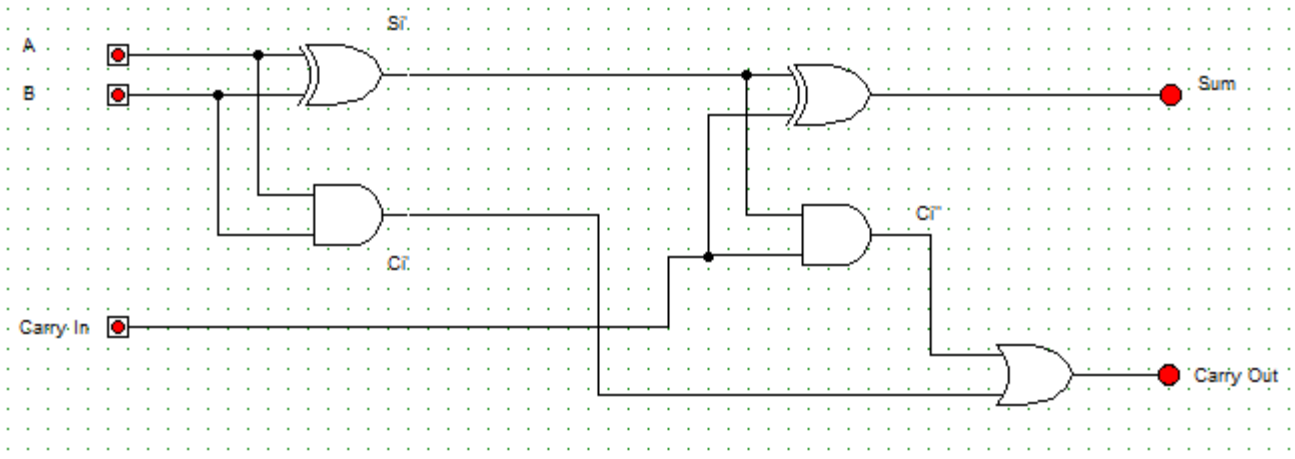
Țin cont de transportul de la bitul mai puțin semnificativ

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

Un sumator complet se poate obține utilizând două semisumatoare:



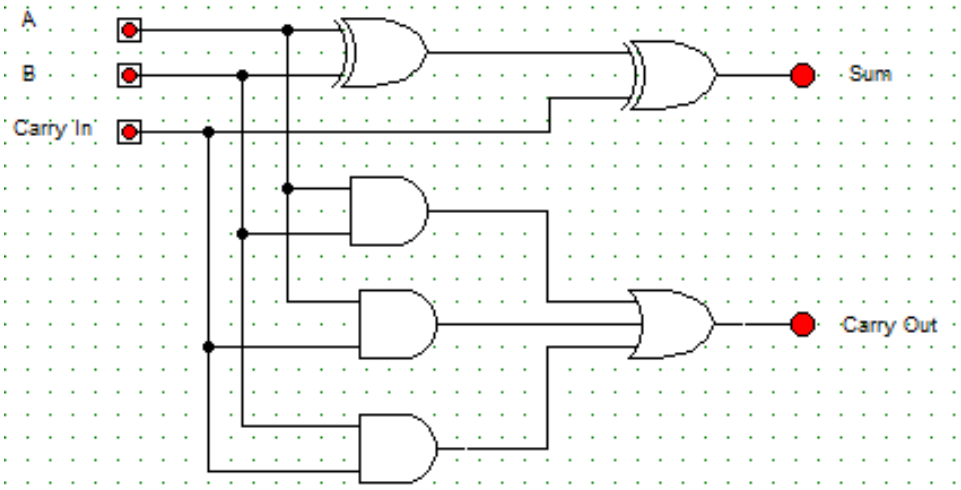
Sumatoare complete

Țin cont de transportul de la bitul mai puțin semnificativ

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

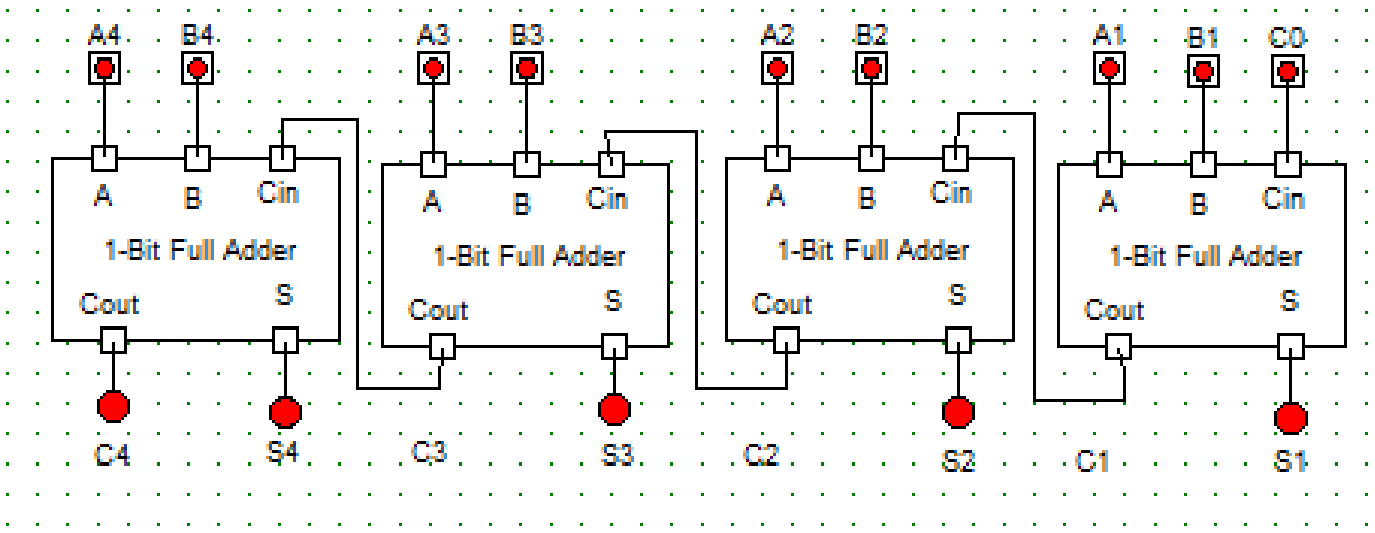
$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$



Sumatoare pe mai mulți biți I

Se pot realiza utilizând sumatoare complete

Sumator pe 4 biți cu transport succesiv (Ripple carry adder): 7483



Lent

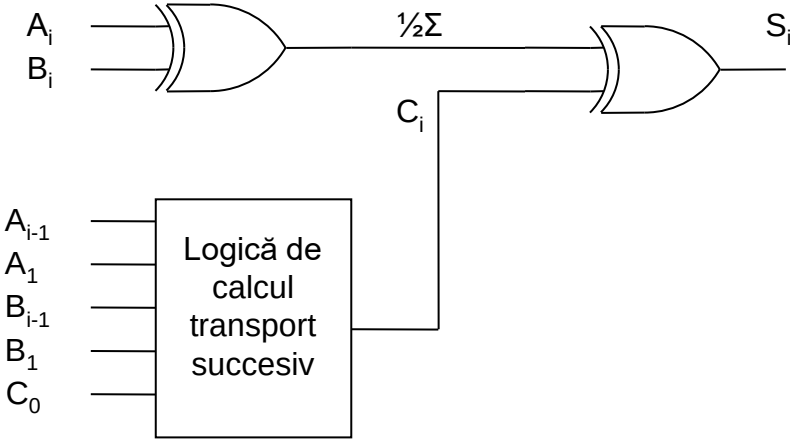
Rezultatele S_i și C_i se obțin doar după ce C_{i-1} de la etajul precedent este disponibil

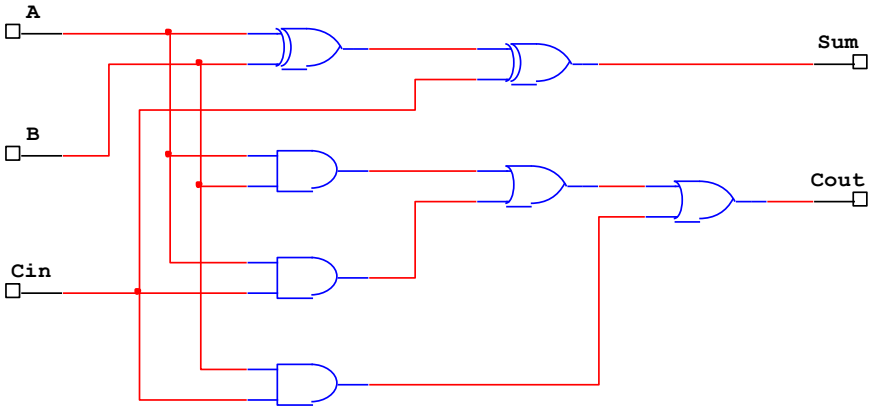
Sumatoare pe mai mulți biți II

Sumator pe 4 biți cu transport anticipat (Look-ahead carry adder)

$$C_i = A_iB_i + (A_i + B_i)C_{i-1} = (A_iB_i + A_iC_{i-1}) + B_iC_{i-1}$$

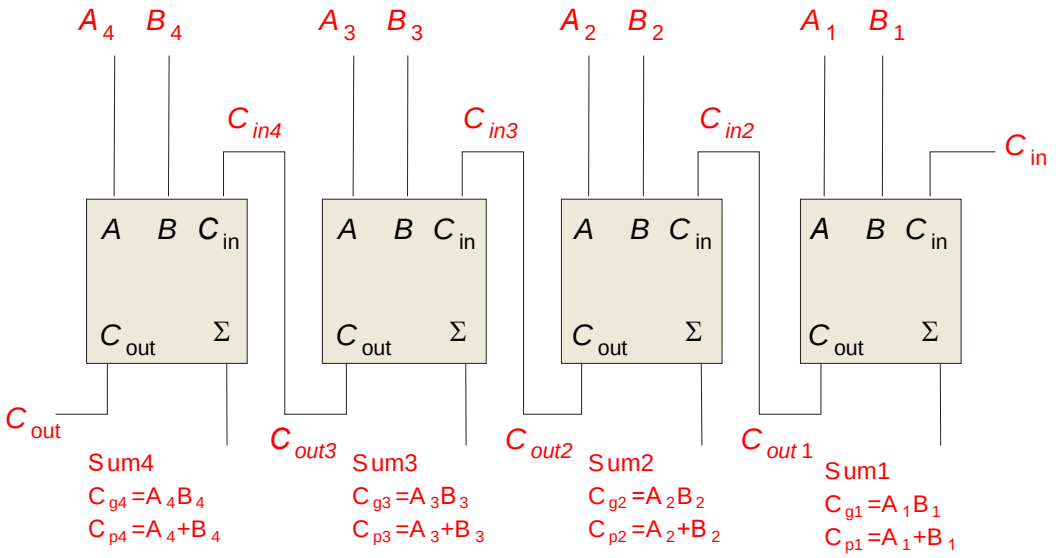
Transport generat — Cgi — Cpi — Transport propagat





Sumator paralel pe 4 biți

Look-ahead carry adder



- $C_{out1}=C_{g1}+C_{p1}C_{in1}$
- $C_{in2}=C_{out1}$
- $C_{out2}= C_{g2}+C_{p2}C_{in2}= C_{g2}+C_{p2}C_{out1}= C_{g2}+C_{p2}(C_{g1}+C_{p1}C_{in1})= C_{g2}+C_{p2}C_{g1}+ C_{p2}C_{p1}C_{in1}$
  $C_{out2}=A_2B_2+(A_2+B_2)\ A_1B_1+(A_2+B_2)\ (A_1+B_1)C_{in1}$
- $C_{in3}=C_{out2}$
- $C_{out3}= C_{g3}+C_{p3}C_{in3}= C_{g3}+C_{p3}C_{out2}= C_{g3}+C_{p3}(C_{g2}+C_{p2}C_{g1}+C_{p2}C_{p1}C_{in1})= C_{g3}+C_{p3}C_{g2}+ C_{p3}C_{p2}C_{g1}+ C_{p3}C_{p2}C_{p1}C_{in1}$
- $C_{in4}=C_{out3}$
- $C_{out4}=C_{g4}+C_{p4}C_{in4}=C_{g4}+C_{p4}C_{out3}= C_{g4}+C_{p4}(C_{g3}+C_{p3}C_{g2}+C_{p3}C_{p2}C_{g1}+C_{p3}C_{p2}C_{p1}C_{in1}) = C_{g4}+C_{p4}C_{g3}+ C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1}+ C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$

Sumator paralel pe 4 biți

$C_{out1}=C_{g1}+C_{p1}C_{in1}$

$C_{in2}=C_{out1}$

$C_{out2}= C_{g2}+C_{p2}C_{in2}= C_{g2}+C_{p2}C_{out1}= C_{g2}+C_{p2}(C_{g1}+C_{p1}C_{in1})= C_{g2}+C_{p2}C_{g1}+ C_{p2}C_{p1}C_{in1}$

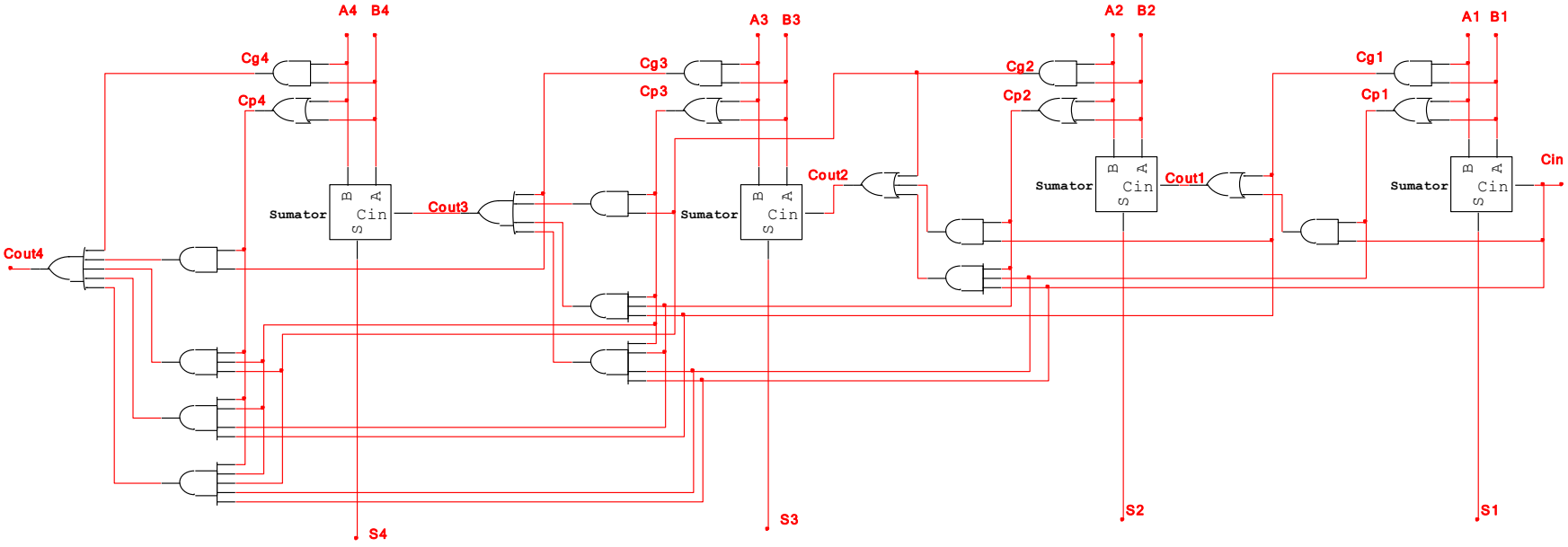
$C_{out2}=A_2B_2+(A_2+B_2)\ A_1B_1+(A_2+B_2)\ (A_1+B_1)C_{in1}$

$C_{in3}=C_{out2}$

$C_{out3}= C_{g3}+C_{p3}C_{in3}= C_{g3}+C_{p3}C_{out2}= C_{g3}+C_{p3}(C_{g2}+C_{p2}C_{g1}+C_{p2}C_{p1}C_{in1})= C_{g3}+C_{p3}C_{g2}+ C_{p3}C_{p2}C_{g1}+ C_{p3}C_{p2}C_{p1}C_{in1}$

$C_{in4}=C_{out3}$

$C_{out4}=C_{g4}+C_{p4}C_{in4}=C_{g4}+C_{p4}C_{out3}= C_{g4}+C_{p4}(C_{g3}+C_{p3}C_{g2}+C_{p3}C_{p2}C_{g1}+C_{p3}C_{p2}C_{p1}C_{in1}) = C_{g4}+C_{p4}C_{g3}+ C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1}+ C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$



Sumator pe 8 biți in limbaj VHDL

Fără transport

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

entity adders_1 is
    port(A,B : in std_logic_vector(7 downto 0);
         SUM : out std_logic_vector(7 downto 0));
end adders_1;

architecture archi of adders_1 is
begin

    SUM <= A + B;

end archi;
```

Cu intrare de transport

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

entity adders_2 is
    port(A,B : in std_logic_vector(7 downto 0);
         CI : in std_logic;
         SUM : out std_logic_vector(7 downto 0));
end adders_2;

architecture archi of adders_2 is
begin

    SUM <= A + B + CI;

end archi;
```

Sumator pe 8 biți in limbaj VHDL

Cu ieșire de transport

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

entity adders_3 is
    port(A,B : in std_logic_vector(7 downto 0);
         SUM : out std_logic_vector(7 downto 0);
         CO : out std_logic);
end adders_3;

architecture archi of adders_3 is
    signal tmp: std_logic_vector(8 downto 0);
begin

    tmp <= conv_std_logic_vector((conv_integer(A) +
conv_integer(B)),9);
    SUM <= tmp(7 downto 0);
    CO <= tmp(8);

end archi;
```

Cu intrare și ieșire de transport

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

entity adders_4 is
    port(A,B : in std_logic_vector(7 downto 0);
         CI : in std_logic;
         SUM : out std_logic_vector(7 downto 0);
         CO : out std_logic);
end adders_4;

architecture archi of adders_4 is
    signal tmp: std_logic_vector(8 downto 0);
begin

    tmp <=
conv_std_logic_vector((conv_integer(A) +
conv_integer(B) + conv_integer(CI)),9);
    SUM <= tmp(7 downto 0);
    CO <= tmp(8);

end archi;
```

Sumator pe 8 biți in limbaj Verilog

Fără transport

```
module v_adders_1(A, B, SUM);
    input [7:0] A;
    input [7:0] B;
    output [7:0] SUM;

    assign SUM = A + B;

endmodule
```

Cu intrare de transport

```
module v_adders_2(A, B, CI, SUM);
    input [7:0] A;
    input [7:0] B;
    input CI;
    output [7:0] SUM;

    assign SUM = A + B + CI;

endmodule
```

Sumator pe 8 biți in limbaj Verilog

Cu ieșire de transport

```
module v_adders_3(A, B, SUM, CO);
    input [7:0] A;
    input [7:0] B;
    output [7:0] SUM;
    output CO;
    wire [8:0] tmp;

    assign tmp = A + B;
    assign SUM = tmp [7:0];
    assign CO = tmp [8];

endmodule
```

Cu intrare și ieșire de transport

```
module v_adders_4(input [7:0] A, B, input CI,
                  output [7:0] SUM, output CO);

    wire [8:0] tmp;

    assign tmp = A + B + CI;
    assign SUM = tmp [7:0];
    assign CO = tmp [8];

endmodule
```

```
// 5_5b
module add4 (input [7:0] a, b, output [8:0] s);
assign s = a + b;
endmodule
```

Sumator pe 1 bit in limbaj Verilog

Descriere structurală explicită

```
// A version
module add1_full (input a, b, cin, output cout, s);
xor3_m xor(.i0(a), .i1(b), .i2(cin), .o(s));
wire a0, a1, a2;
and2_m and0(.i0(a), .i1(b), .o(a0));
and2_m and1(.i0(a), .i1(cin), .o(a1));
and2_m and2(.i0(b), .i1(cin), .o(a2));
or3_m or(.i0(a0), .i1(a1), .i2(a2) , .o(cout))
endmodule
```

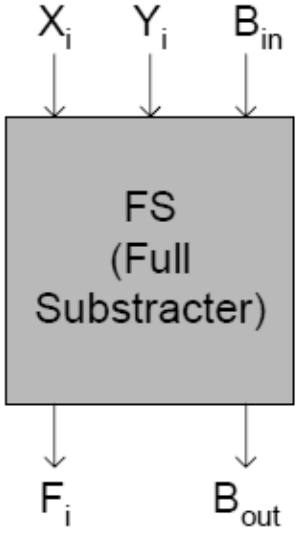
Descriere structurală implicită

```
// B version
module add1_full (input a, b, cin, output cout, s);
assign s = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

Descriere comportamentală

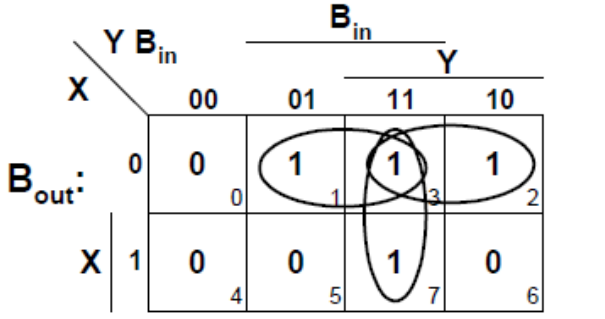
```
// C version
module add1_full (input a, b, cin, output cout, s);
assign {cout, s} = a + b + cin;
endmodule
```

Circuit pentru scăderea a două numere pe 1 bit

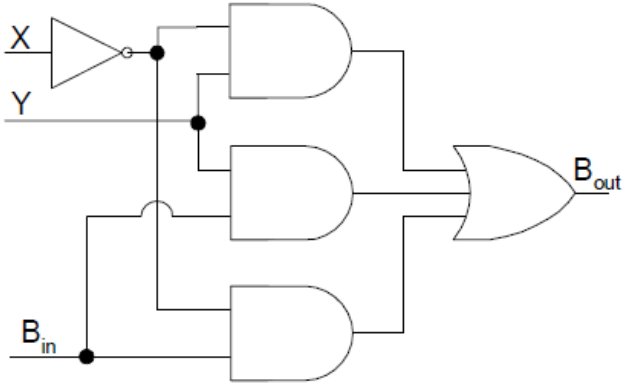


X_i	Y_i	Bin	F_i	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$F_i = X_i \oplus Y_i \oplus B_{in}$$



$$B_{out} = \overline{X_i} \cdot Y_i + \overline{X_i} \cdot B_{in} + Y_i \cdot B_{in}$$



Circuit de scădere complet pe 4 biți

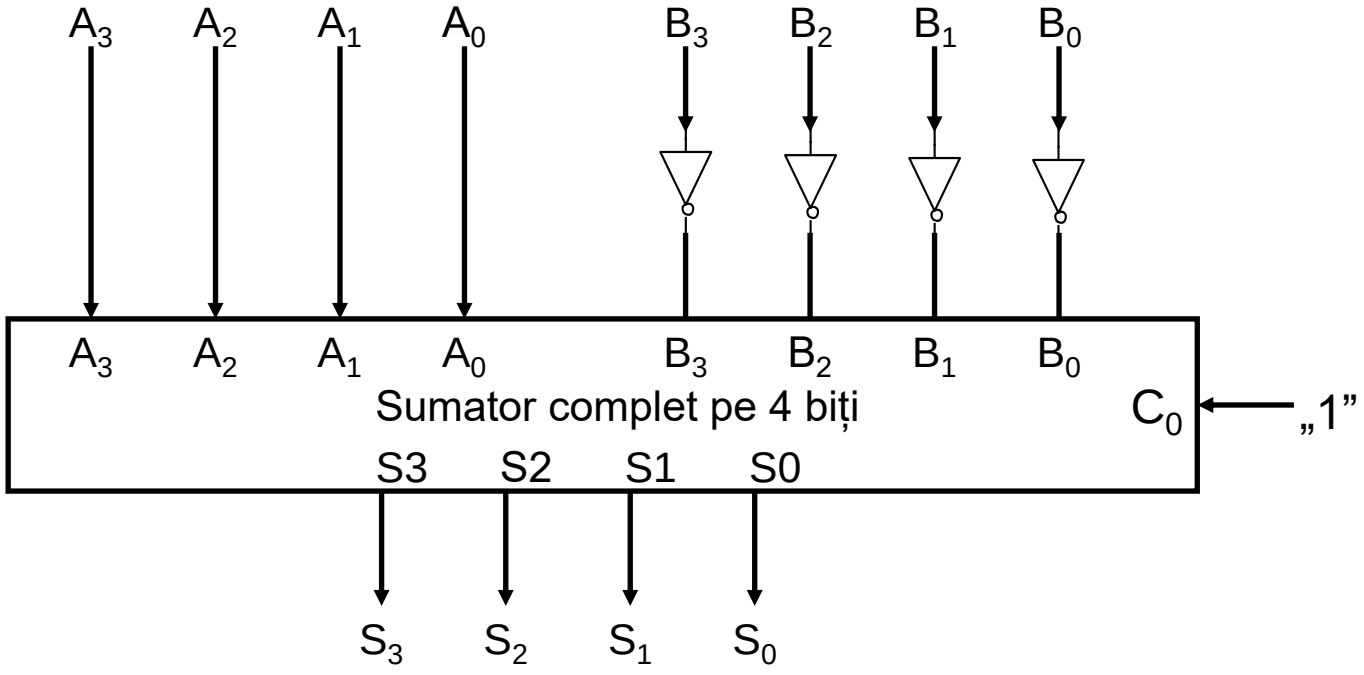
A-B=A+(-B)

-B= $B_{(C2)}$

A-B=A+ $B_{(C2)}$

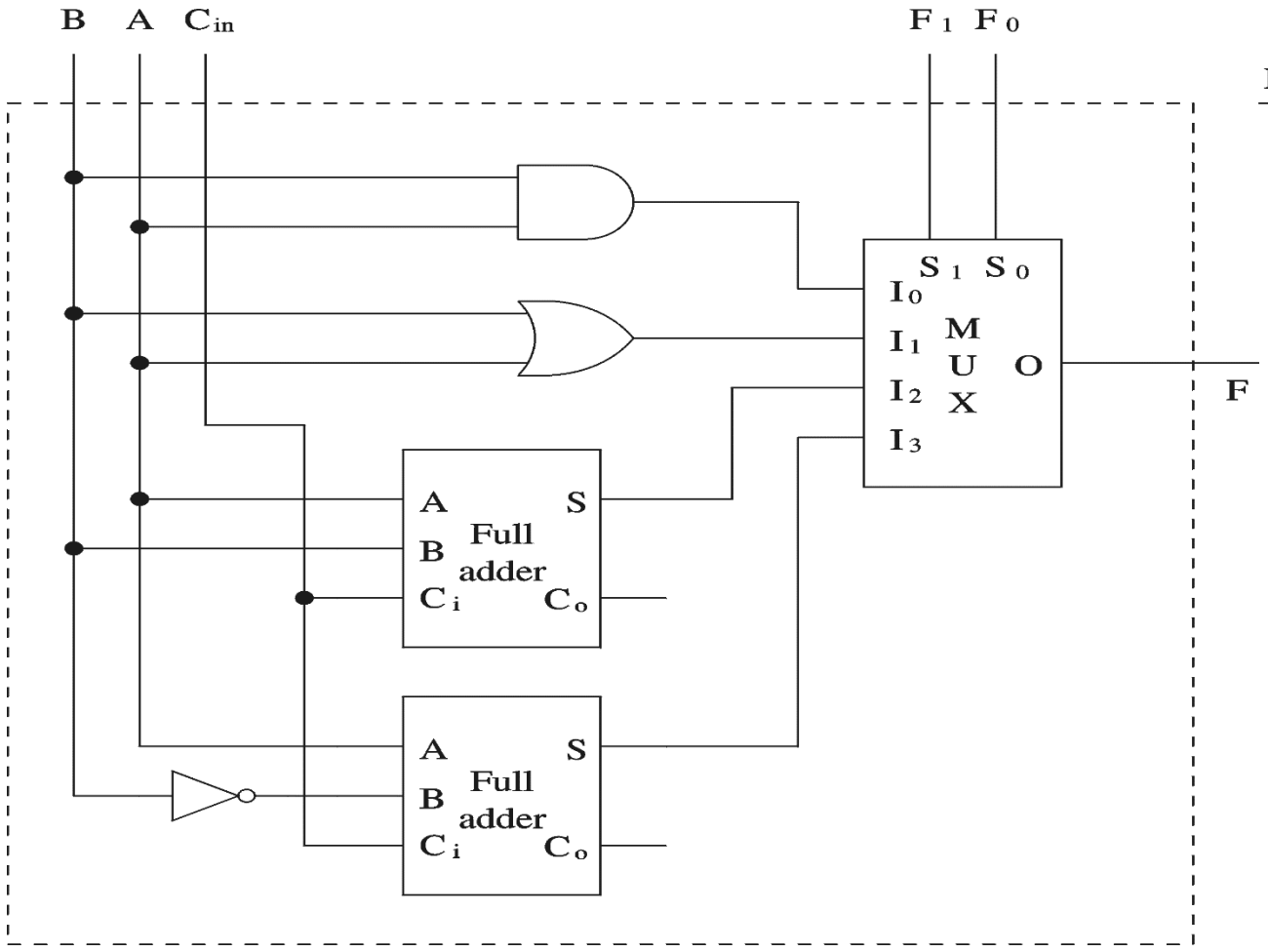
Scăderea se poate face prin adunarea complementului față de doi a scăzătorului la descăzut

$B_{(C2)}=\overline{B}+1$



ALU pe 1 bit

(ALU = arithmetic + logic unit)

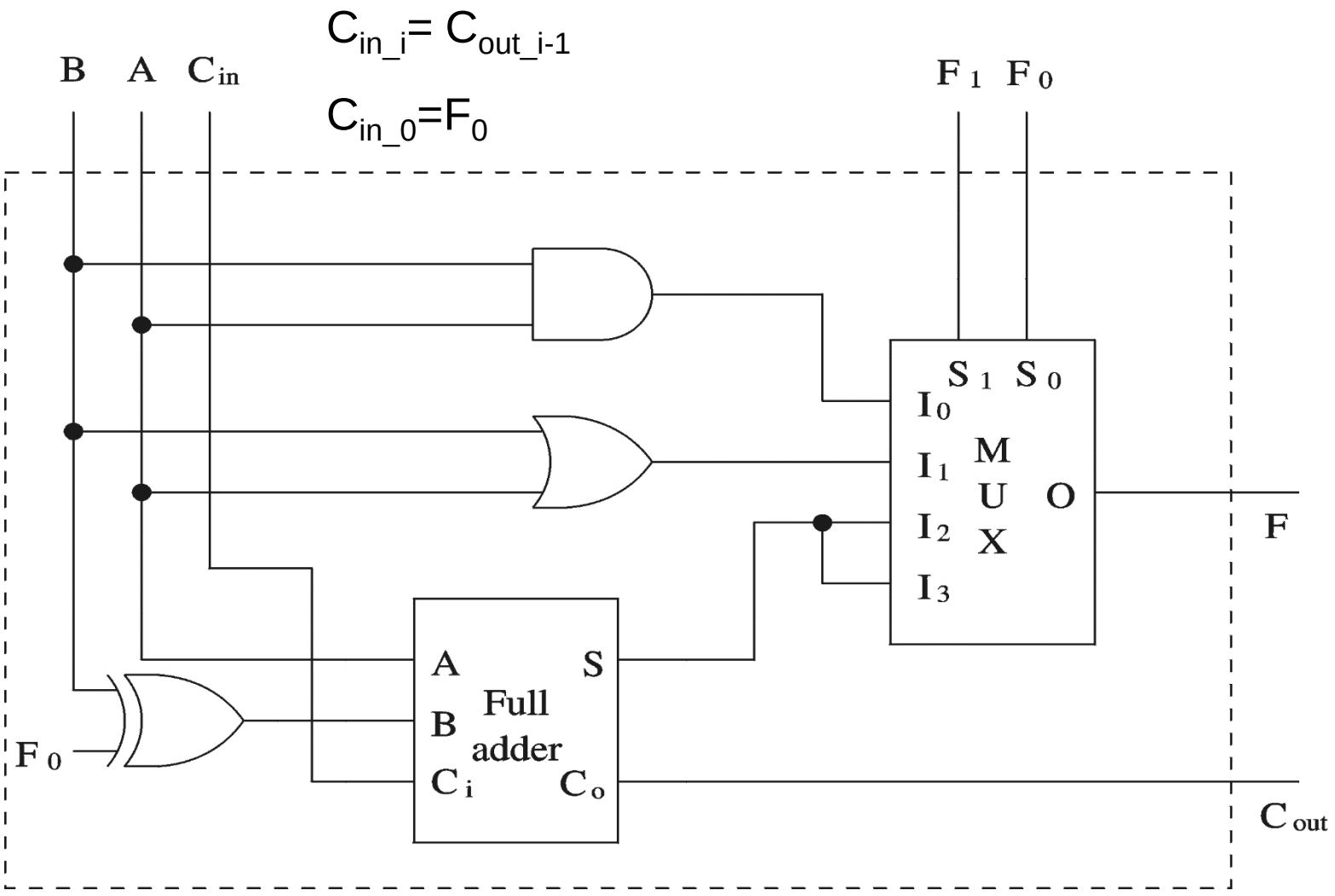


F_1	F_0	F
0	0	A and B
0	1	A or B
1	0	A + B
1	1	A − B

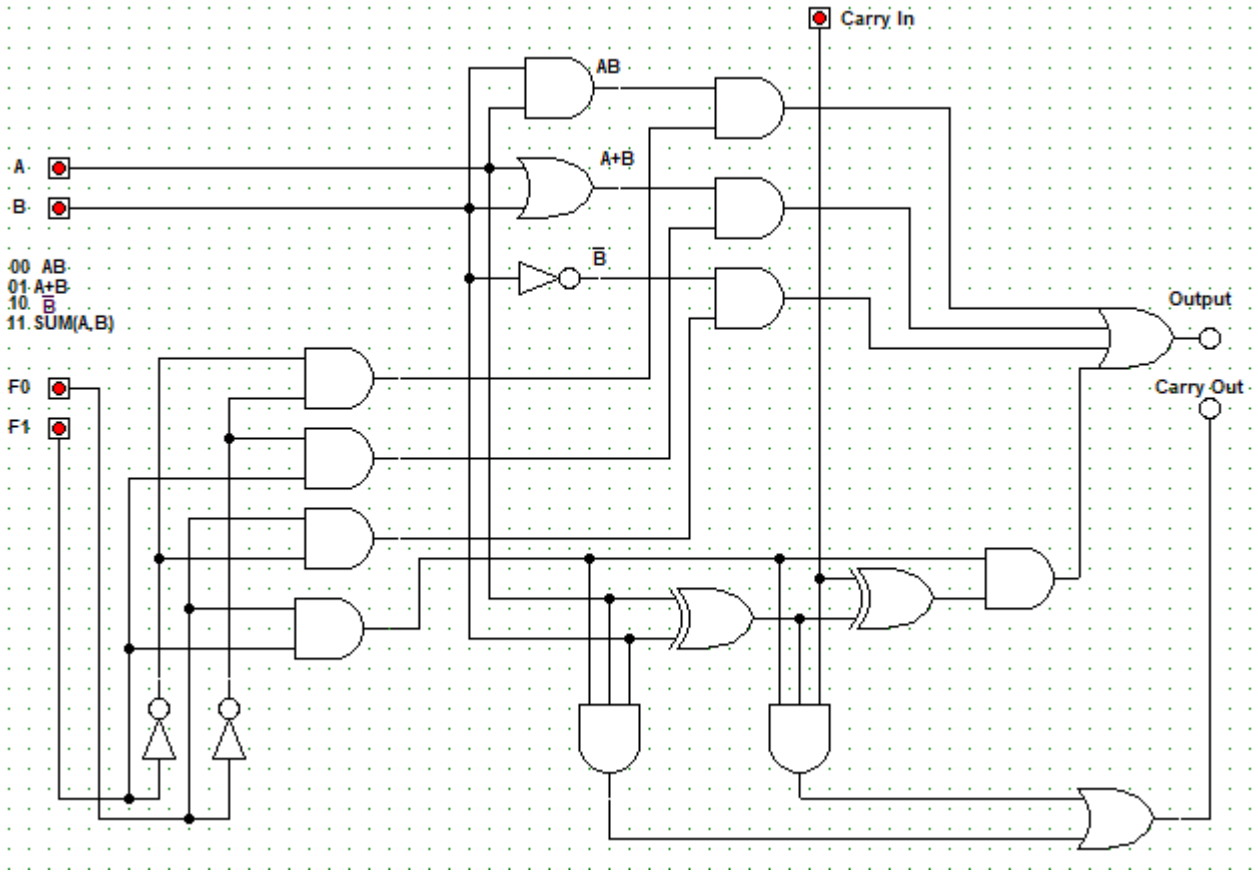
$C_{in\_i} = C_{out\_i-1}$

$C_{in\_0} = F_0$

ALU pe 1 bit optimizat



ALU 1 bit (simulare)

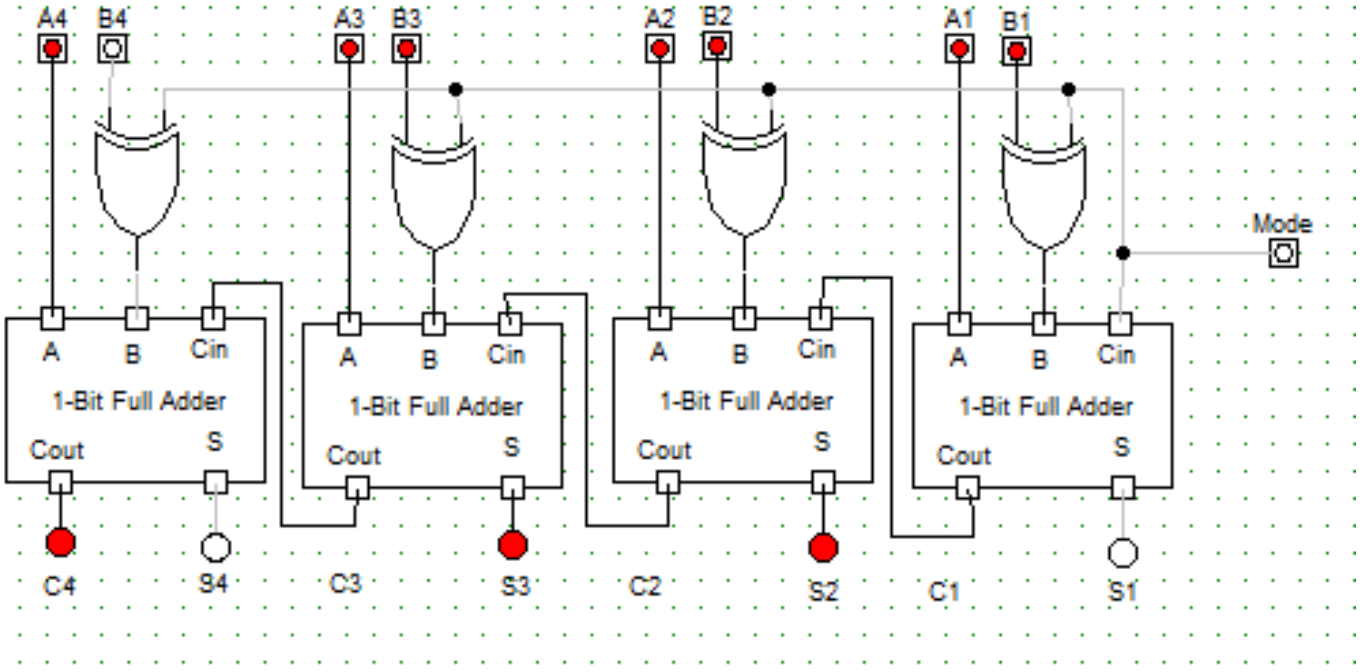


Simplified version of a circuit in Tanenbaum, Andrew S., Structured Computer Organization, Fourth Edition Prentice-Hall, 1999 [p.138]

Circuit de adunare/scădere pe 4 biți optimizat

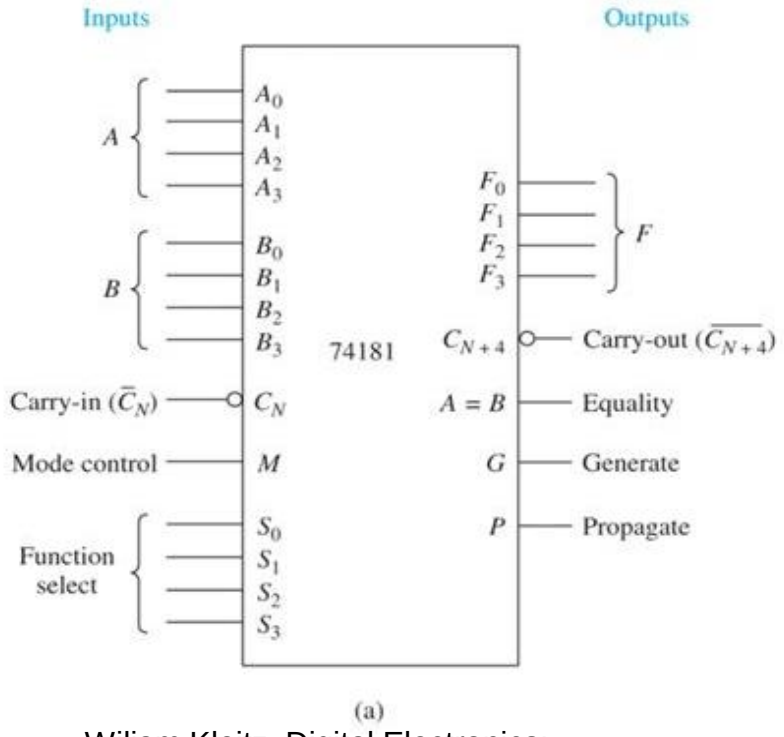
Adunare: Mode = 0 => $b_i' = b_i$

Scădere: Mode = 1 => $b_i' = \overline{b_i}$



ALU pe 4 biți - 74LS181

- Doi operanzi pe 4 biți (A, B)
- Rezultat pe 4 biți
- Carry In/ Out
- S0, S1, S2: intrări selecție operație



(a)

Wiliam Kleitz, Digital Electronics:
A practical Approach with VHDL, 9-th Edition

Mode select				Logic functions	Arithmetic operations
S_3	S_2	S_1	S_0	$(M = H)$	$(M = L)(\overline{C_n} = H)$
L	L	L	L	$F = \overline{A}$	$F = A$
L	L	L	H	$F = \overline{A + B}$	$F = A + B$
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$
L	L	H	H	$F = 0$	$F =$ minus 1 (2's comp.)
L	H	L	L	$F = \overline{AB}$	$F = A$ plus $A\overline{B}$
L	H	L	H	$F = \overline{B}$	$F = (A + B)$ plus $A\overline{B}$
L	H	H	L	$F = A \oplus B$	$F = A$ minus B minus 1
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ minus 1
H	L	L	L	$F = \overline{A} + B$	$F = A$ plus AB
H	L	L	H	$F = \overline{A \oplus B}$	$F = A$ plus B
H	L	H	L	$F = B$	$F = (A + \overline{B})$ plus AB
H	L	H	H	$F = AB$	$F = AB$ minus 1
H	H	L	L	$F = 1$	$F = A$ plus A*
H	H	L	H	$F = A + \overline{B}$	$F = (A + B)$ plus A
H	H	H	L	$F = A + B$	$F = (A + \overline{B})$ plus A
H	H	H	H	$F = A$	$F = A$ minus 1

*Each bit is shifted to the next-more-significant position.

Detectorul și generatorul de paritate

Coduri detectoare și corectoare de erori

Cea mai simpla metodă:
- bit de paritate
- La emisie se formează un nou cuvânt de cod prin adăugarea unui bit suplimentar la cei existenți, a.î. numărul de cifre de „1” din cuvântul nou format să fie par (sau impar).
- La recepție se verifică paritatea sau imparitatea numărului de cifre de „1” din cuvântul recepționat

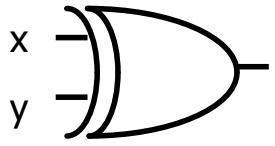
Paritate pară		Paritate impară	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	0100	0	0100
0	1001	1	1001

- paritate pară
- paritate impară

Dezavantaje:
– Nu pot corecta erorile, doar le detectează
– Poate detecta doar un număr impar de erori.

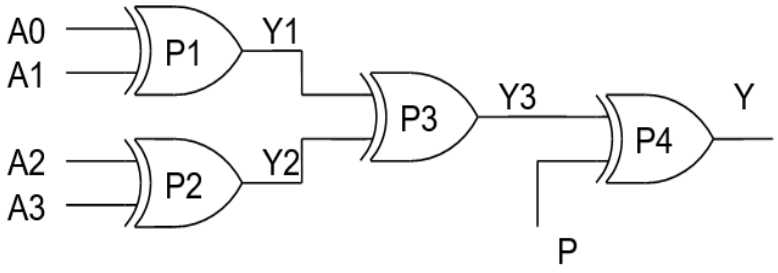
Ușor de realizat cu: poartă XOR

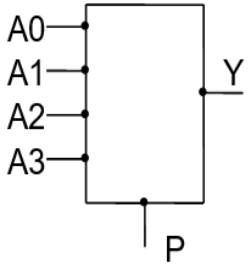
x y	x XOR y
0 0	0
0 1	1
1 0	1
1 1	0



Detectoare de paritate

Detector de paritate pt. cuvinte de 4 biți





A_0	A_1	A_2	A_3	Y_1	Y_2	Y_3
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	0	0

P – Semnal de setare a tipului de paritate:

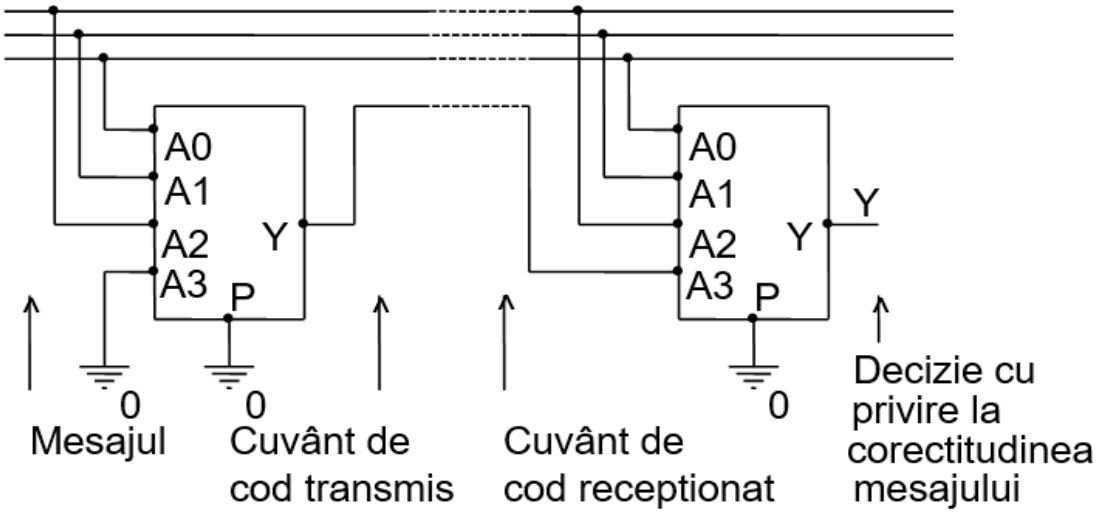
Y_3	P	Y
0	0	0
1	0	1
0	1	1
1	1	0

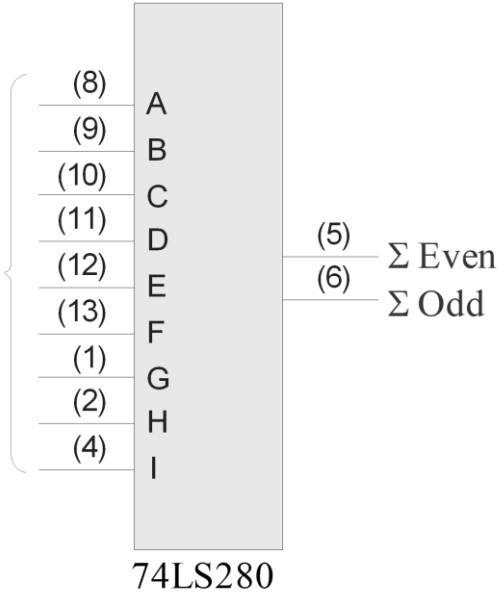
- P = 0 => Y = Y_3 => generator de paritate pară
- P = 1 => Y = $\overline{Y_3}$ => generator de paritate impară

Schemă de transmisie cu detectoare de paritate

Emițător – generator de paritate:
- Extinderea biților de date cu bitul de paritate => numărul de biți de 1 în cuvântul extins este par.

Receptor – detector de paritate:
- Verificarea parității cuvântului extins recepționat





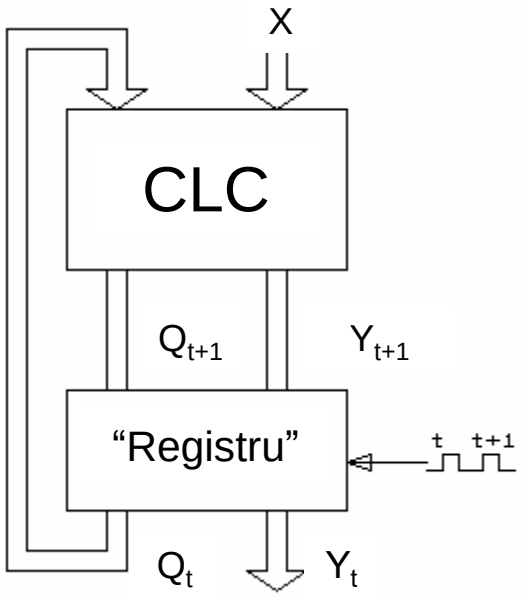
Circuite logice secvențiale

Circuite logice secvențiale

- **Definirea conceptului de Circuit logic secvențial**
- **Circuite basculante**
  - Circuite basculante bistabile - RS
  - Circuite basculante bistabile - JK
  - Circuite basculante bistabile de tip T și D
- **Numărătoare**
  - Numărătoare sincrone
  - Numărătoare asincrone
- **Registre**
  - http://www.play-hookey.com/digital/
  - http://www.asic-world.com/digital/seq.html

Conceptul de Circuit logic secvențial

- CLC- starea ieșirilor depinde doar de starea intrărilor din momentul de timp considerat.
- CLS - starea ieșirilor la un moment de timp dat depinde nu numai de vectorul de intrare, ci și de starea circuitului la un moment de timp imediat anterior

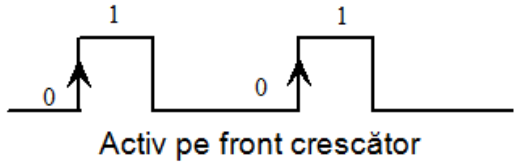


- Un CLC cu reacție simplă (independentă de timp):
  - ***Circuite secvențiale asincrone***
- Dacă reacția are loc la momente de timp bine stabilite - de exemplu la fiecare impuls de ceas
  - ***Circuite secvențiale sincrone.***

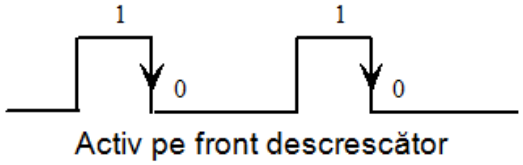
$$Y = f(X, Q)$$

Circuite basculante bistabile

- Moduri de funcționare:
  - Scriere (setare) SET înscrierea unui „1” logic
  - Ștergere (reset) RESET înscrierea unui „0” logic
  - Memorare STORE menținerea stării anterioare (0 sau 1)
- Tipuri:
  - Bistabil - RS
  - Bistabil - J-K
  - Bistabil - D
  - Bistabil - T
- Tipuri de comandă:
  - Bistabil asincron = latch
  - Bistabil activ pe nivel
  - Bistabil activ pe front
  - Bistabil cu comandă mixtă

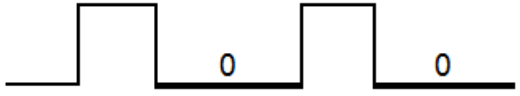


Activ pe front crescător



Activ pe front descrescător

Activ pe nivel 1

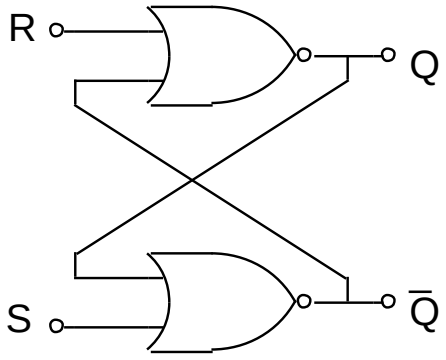


Activ pe nivel 0

Latch-ul de tip RS

Latch-ul RS este un circuit basculant asincron care are:

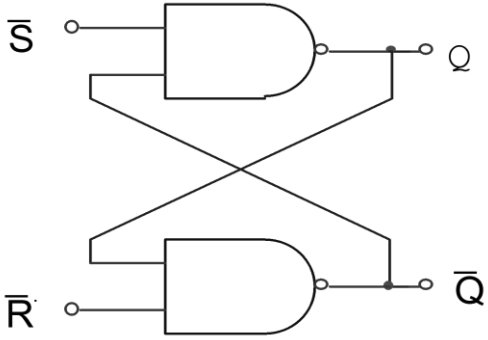
- două intrări de comandă (intrări de date)
  - S (de la SET—punere la 1)
  - R (de la RESET - punere la zero)
- două ieșiri Q şi /Q



R	S	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	Q_n	$\overline{Q}_n$
0	1	1	0
1	0	0	1
1	1	interzis	interzis

- Starea a n-a o notăm cu indicele n, starea următoare cu indicele n+1

- Lach-ul RS cu porți NOR
  - **Comanda S = R = 1 interzisă**
- Lach-ul RS cu porți NAND
  - **Comanda S = R = 0 interzisă**



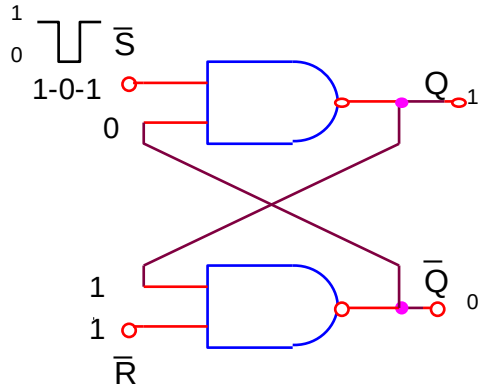
$\overline{S}$	$\overline{R}$	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	interzis	interzis
0	1	1	0
1	0	0	1
1	1	Q_n	$\overline{Q}_n$

RS NOR Latch http://www.play-hookey.com/digital/sequential/rs_nor_latch.html
RS NAND Latch http://www.play-hookey.com/digital/sequential/rs_nand_latch.html

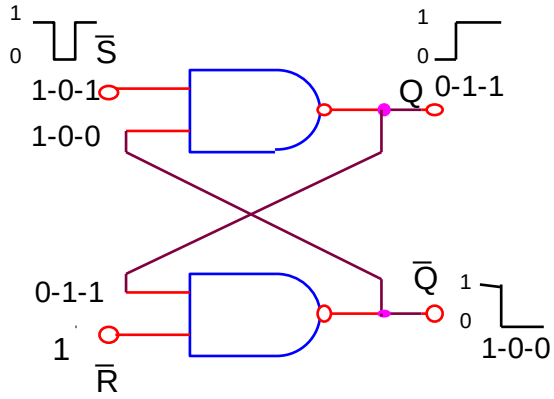
Latch-ul de tip RS *(cont.)*

SETARE

Stare anterioară presupusă Q=1

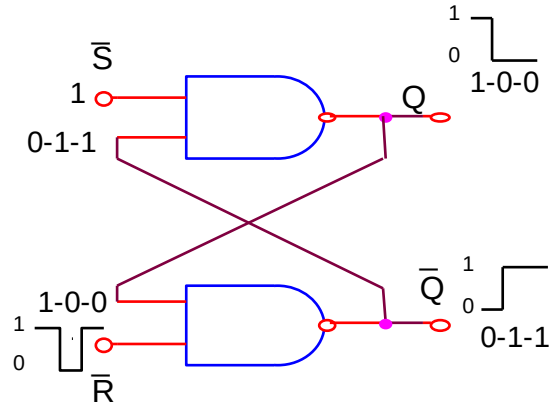


Stare anterioară presupusă Q=0

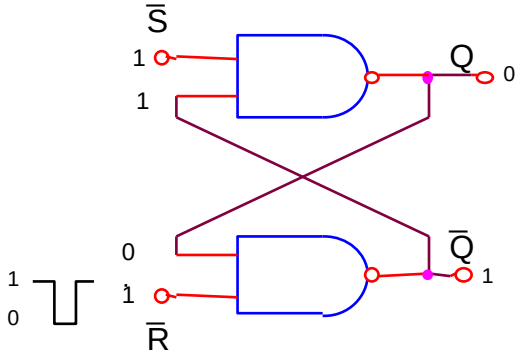


RESETARE

Stare anterioară presupusă Q=1

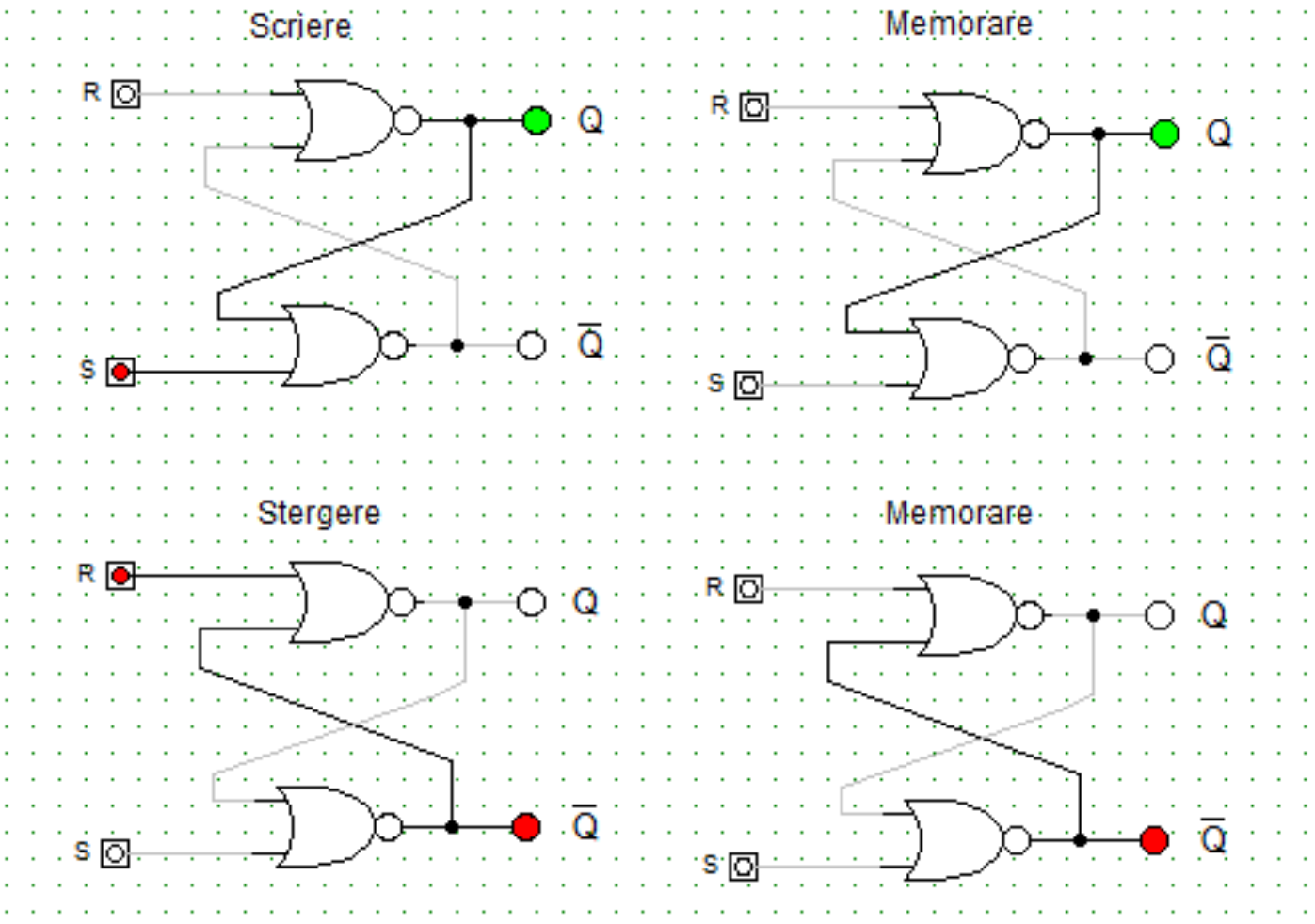


Stare anterioară presupusă Q=0



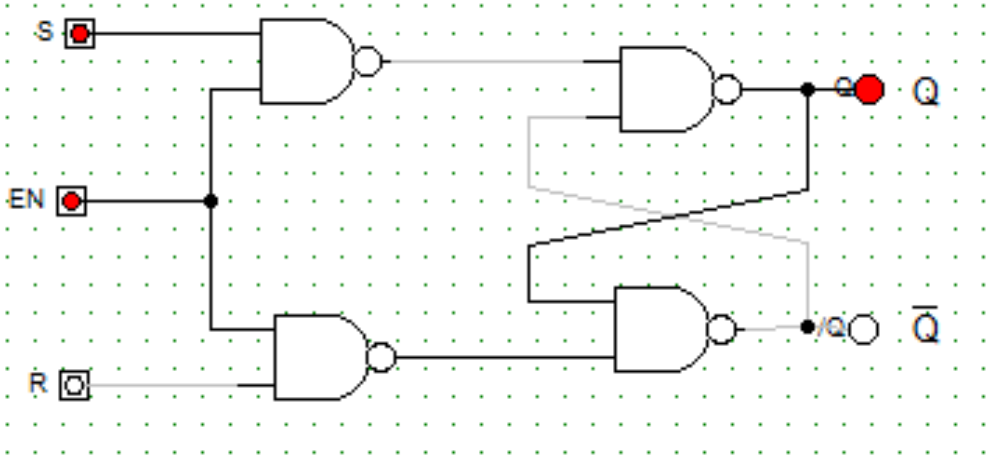
Latch-ul de tip RS (*cont.*)

Simulare cu programul DigitalWorks

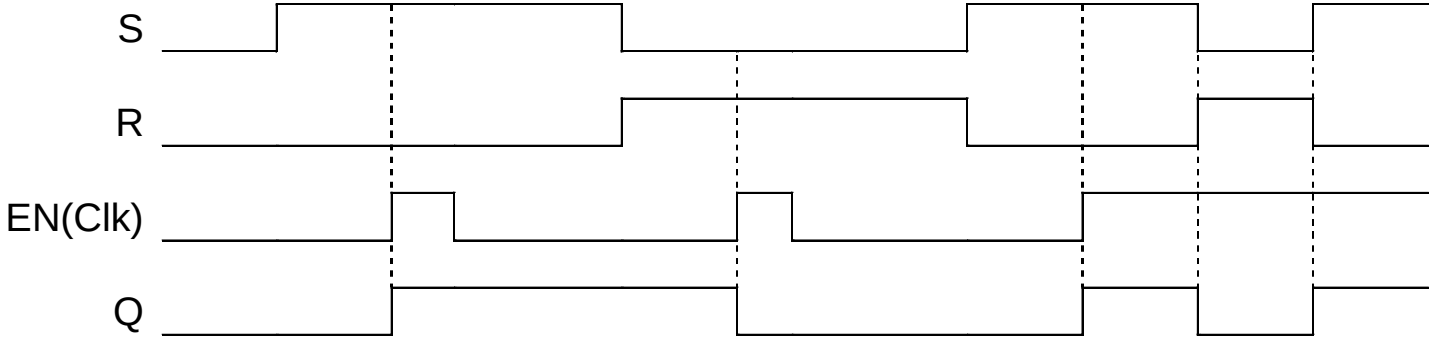


Latch-ul RS cu intrare de autorizare

- **“Gated RS latch”**
- **Intrare de autorizare EN**

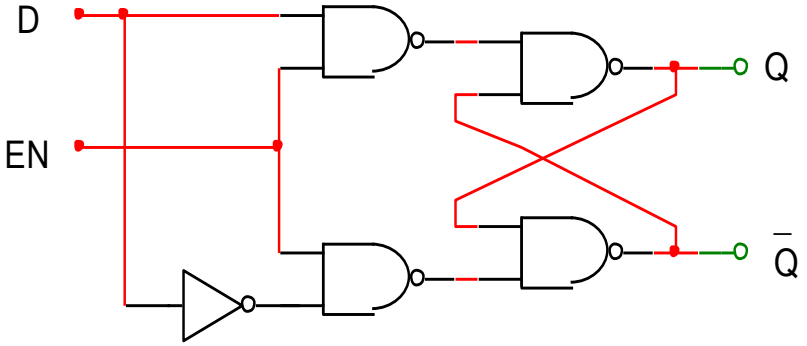


EN	S_n	R_n	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	1	0	1
1	0	1	0
1	1	1	Interzis

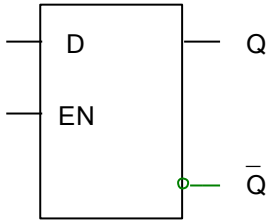


[Clocked RS Latch](http://www.play-hookey.com/digital/sequential/clocked_rs_latch.html) http://www.play-hookey.com/digital/sequential/clocked_rs_latch.html

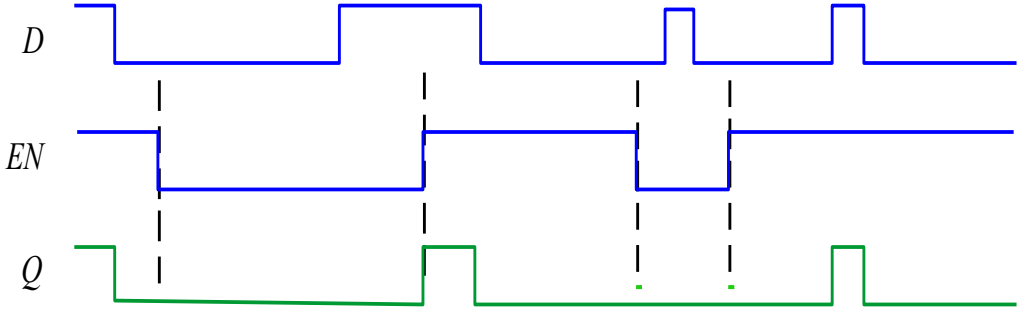
Latch-ul de tip D cu intrare de autorizare



EN	D	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	Q_n	Q_n
0	1	Q_n	Q_n
1	0	0	1
1	1	1	0



- O intrare = D
- Intrare de autorizare = EN
- Dacă EN = 0 => $Q_{n+1} = Q_n$
- Q =D dacă EN =1 (actív)
- „Transparent”



D Latch http://www.play-hookey.com/digital/sequential/d_nand_latch.html

Latch-ul de tip D cu intrare de autorizare

VHDL

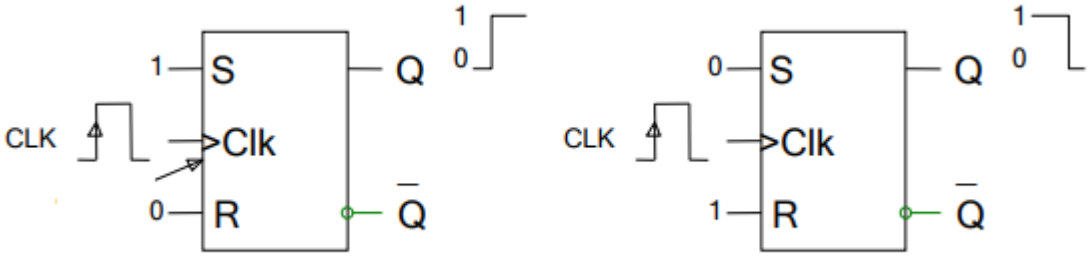
```
5   library ieee;
6   use ieee.std_logic_1164.all;
7
8   entity latches_1 is
9       port(G, D : in std_logic;
10           Q : out std_logic);
11  end latches_1;
12
13  architecture archi of latches_1 is
14  begin
15      process (G, D)
16      begin
17          if (G='1') then
18              Q <= D;
19          end if;
20      end process;
21  end archi;
```

Verilog

```
module v_latches_1 (input G, D, output reg Q);

    always @(G or D)
    begin
        if (G)
            Q = D;
    end
endmodule
```

Circuite bistabile RS cu comutare pe front

- Trei intrări:
  - S (Set), R (Reset)
  - Clk (Clock) (simbolizat printr-un triunghi)
- Ieșiri (Q), sau (Q_n,).
- La RESET, Q_{n+1} devine 0, la SET devine1.



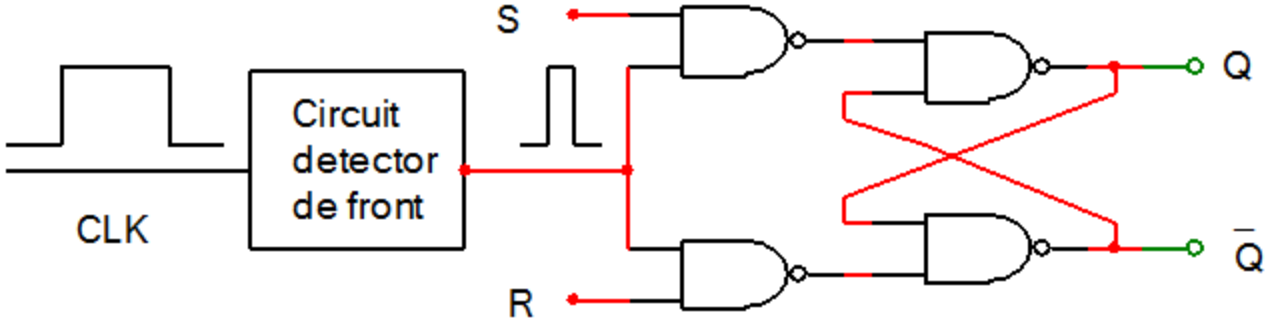
- Daca nu scriem sau nu ștergem, $Q_{n+1} = Q_n$.

- Funcționare sincronă

RS Flip-Flop http://www.play-hookey.com/digital/sequential/rs_nand_flip-flop.html

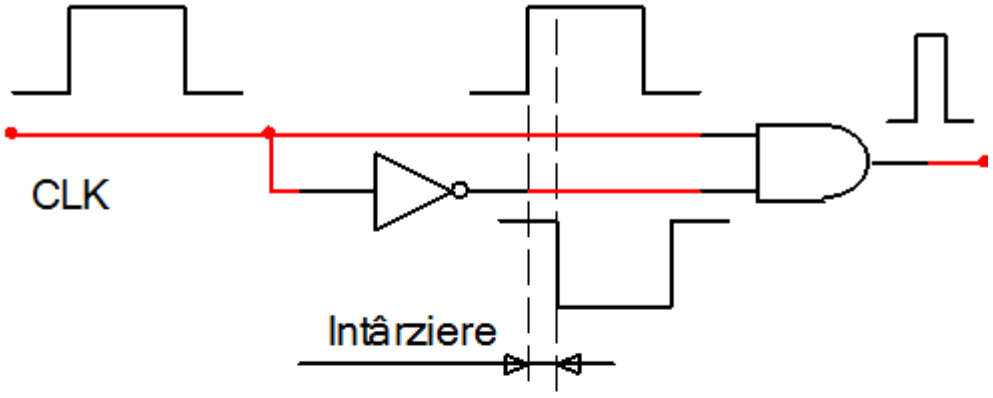
Circuite bistabile RS cu comutare pe front

Intrările S și R ale bistabilului RS se numesc intrări sincrone deoarece comenzile aplicate pe aceste intrări sunt transferate la ieșire sincron cu frontul activ al impulsului de tact

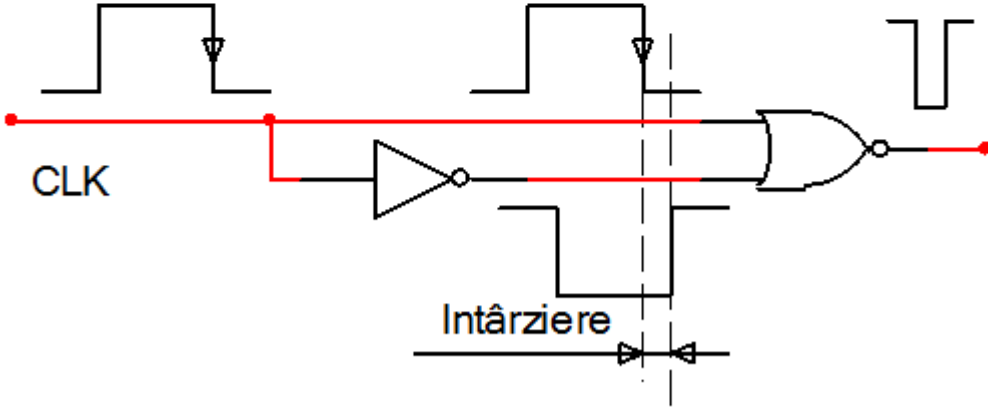


Schema simplificată a bistabilului RS activ pe front crescător

Circuite pentru detectarea frontului



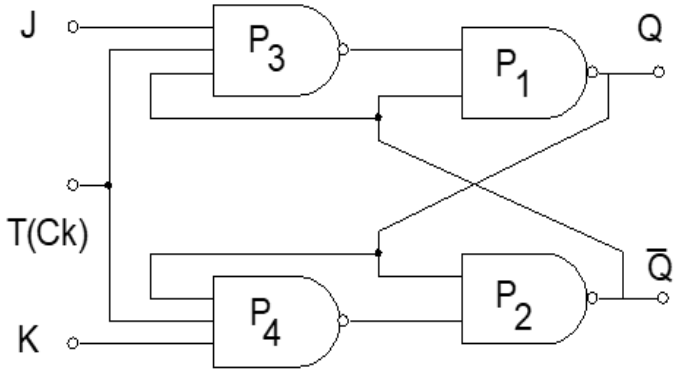
a) Circuitul pentru detectarea frontului crescător



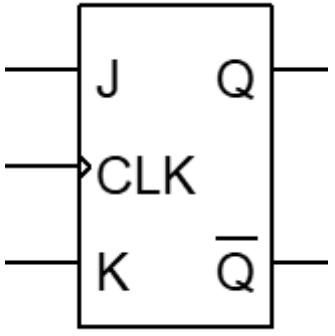
b) Circuitul pentru detectarea frontului descrescător

Bistabilul J-K

- Doar cu comutare pe front
- Trei intrări: J (setare), K (ștergere) și Clk (Clock) intrare de tact
- Înlătură nedeterminarea care apare în urma aplicării combinației logice $S_n=R_n=1$ la intrările bistabilului RS sincron prin utilizarea unor bucle suplimentare de reacție ce includ porțile P_3 și P_4
- Când $J_n=K_n=1$, starea ieșirii circuitului este complementată în urma aplicării impulsului de tact, $Q_{n+1}=\overline{Q}_n$



J_n	K_n	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	$\overline{Q}_n$

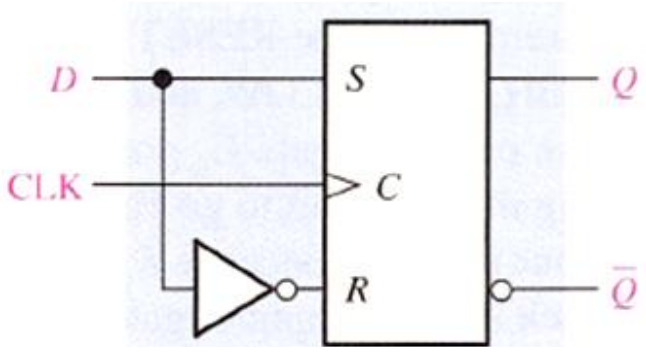


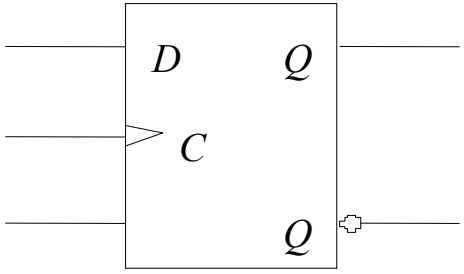
- Dacă durata impulsului de tact este mult mai mare decât timpul de propagare printr-o poartă, t_{pd}, în anumite situații circuitul oscilează.

JK Flip-Flop http://www.play-hookey.com/digital/sequential/jk_nand_flip-flop.html

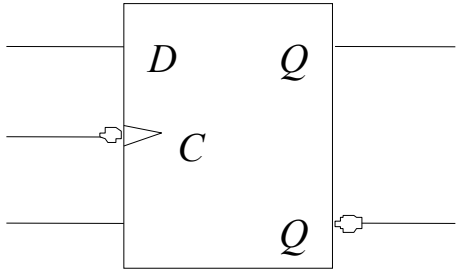
Bistabil de tip D cu comutare pe front

- Circuitul basculant sincron de tip D are o singură intrare de date D, o intrare de tact C_k
- Stocarea informației de pe o singură linie de semnal – elemente de memorie de 1 bit.





(a) Comanada pe front crescator



Comanada pe front descrescator

D Flip-Flop http://www.play-hookey.com/digital/sequential/d_nand_flip-flop.html

Bistabil de tip D cu comutare pe front

VHDL

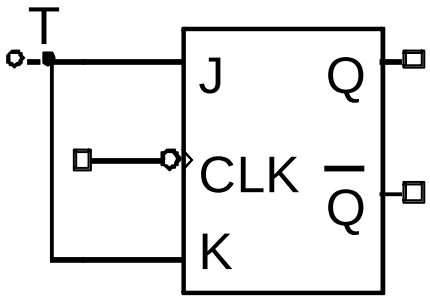
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity registers_1 is
    port(C, D : in std_logic;
         Q    : out std_logic);
end registers_1;

architecture archi of registers_1 is
begin

    process (C)
    begin
        if (C'event and C='1') then
            Q <= D;
        end if;
    end process;

end archi;
```

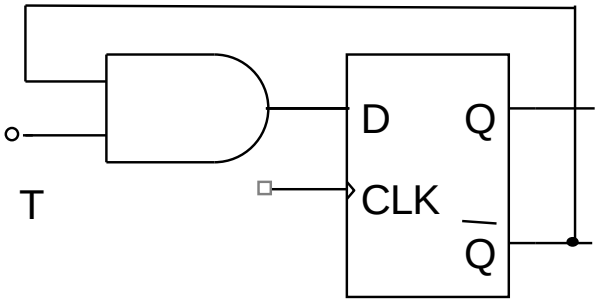
Verilog

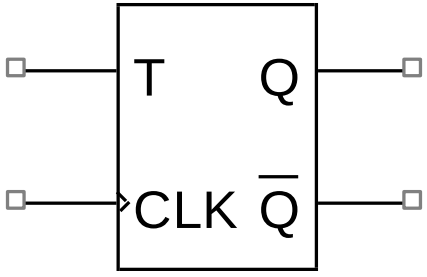
```verilog
module v_registers_1 (input C, D,
    output reg Q);

    always @(posedge C)
    begin
        Q <= D;
    end

endmodule
```

Bistabil de tip T cu comutare pe front

- Se obțin din bistabile de tip J-K având intrările J și K legate împreuna, numită intrare T
- La activarea intrării (T = 1), Q comută în starea complementară
- Dacă T=0, ieșirea își păstrează valoare
- Funcționare foarte simplă aplicații multiple (numărătoare).



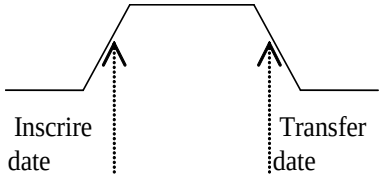


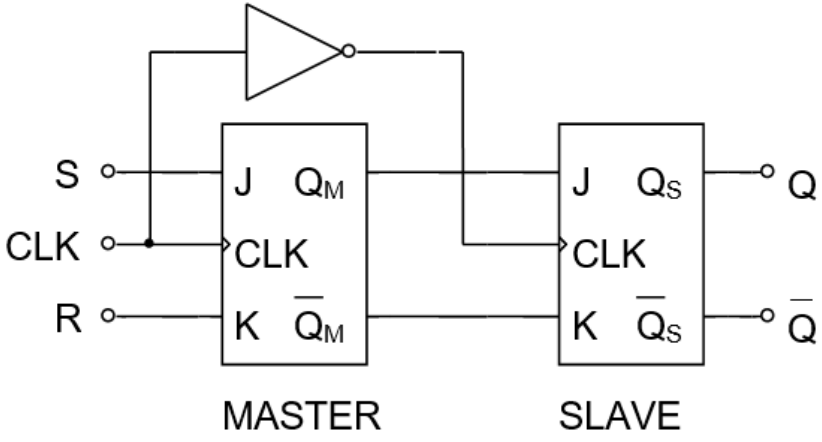


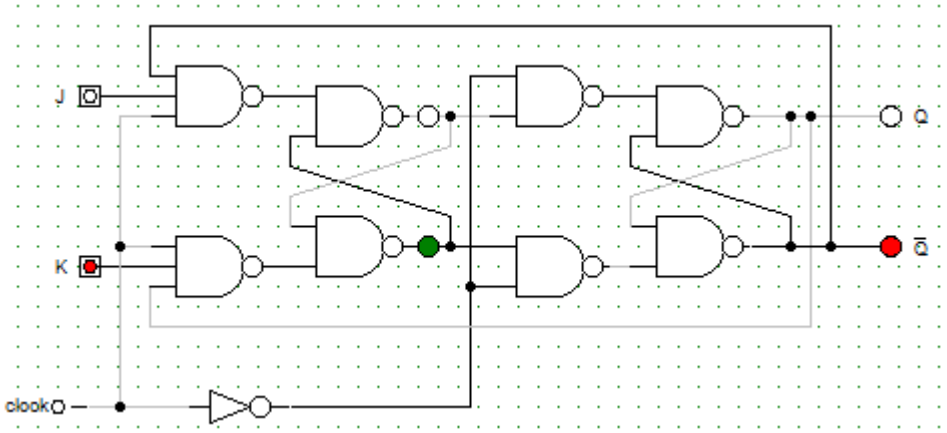
T	Q_{n+1}
0	Q_n
1	$\overline{Q}_n$

Bistabile Master-Slave

- Înlăturarea inconvenientelor bistabilului J-K (oscilația ieșirilor).
- Înlăturarea inconvenientelor rezultate, ca urmare a conectării directe a intrărilor unui circuit bistabil la ieșiri
- Impulsurile de tact sunt aplicate direct primului bistabil (master) și inversate celui de al doilea bistabil (slave)
- Pe frontul crescător înscrierea datelor în etajul master
- Pe frontul descrescător transferarea datelor din master în slave
- RS-MS / JK-MS

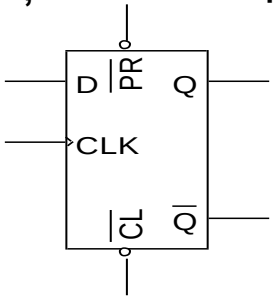


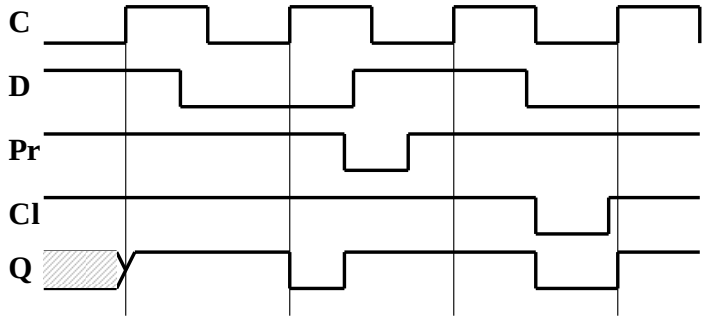


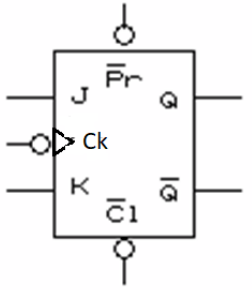


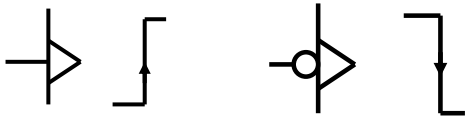
Bistabile cu comenzi multiple

- Intrări sincrone : S-R, J-K, D, T (comutare pe front)
- Intrări asincrone: **Cl (Clear) ștergere și Pr (Preset) înscriere.**
- **Pr** și **Cl** active pe **0.**

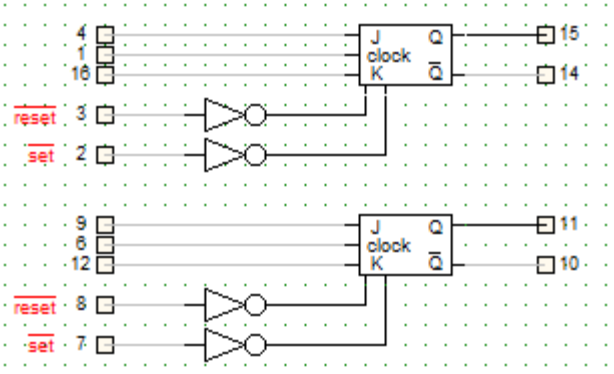






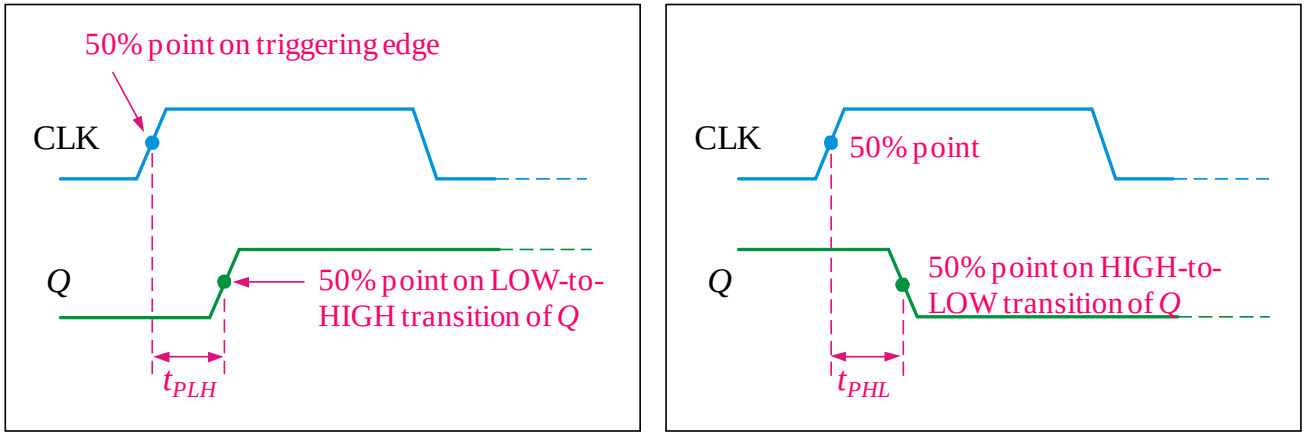


- **Exemple**
- 7472E - JK-MS FF - 3 intrări poartă ȘI
- 7473E – două JK-MS FF cu C_k și C_l separate
- 7476E - două JK-MS, cu intrări C_k , P_r, și C_l
- 7474 – două DFF intrări D, C_k, P_r , C_l , separate.
- 7475 – patru DFF, câte două cu intrări comune D și C_k
- 74HC76 - două JK-MS FF cu intrări set si reset

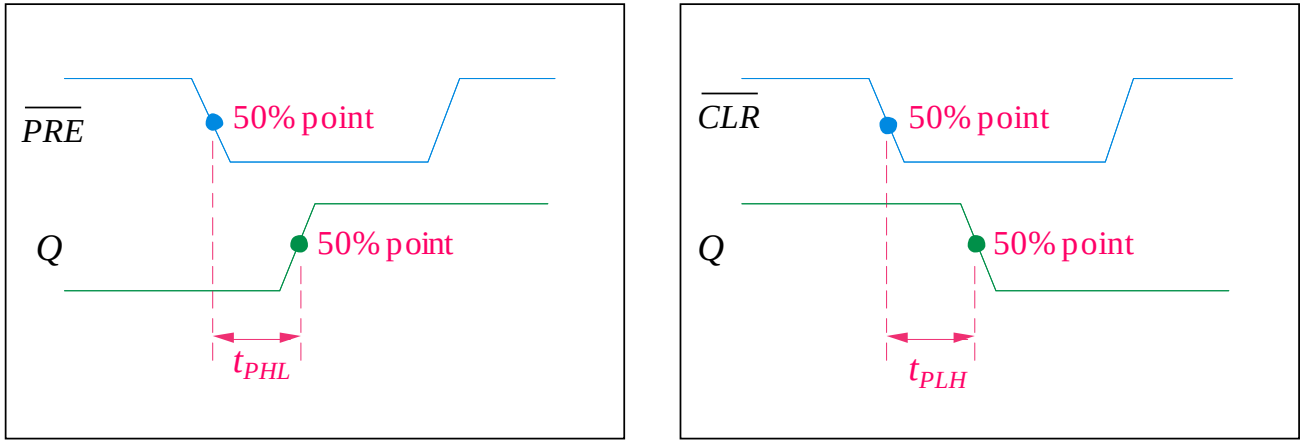


Caracteristicile bistabilelor (1)

- Timpul de propagare față de intrările sincrone (Ex. 4ns la familia 74AHC IC) [2]



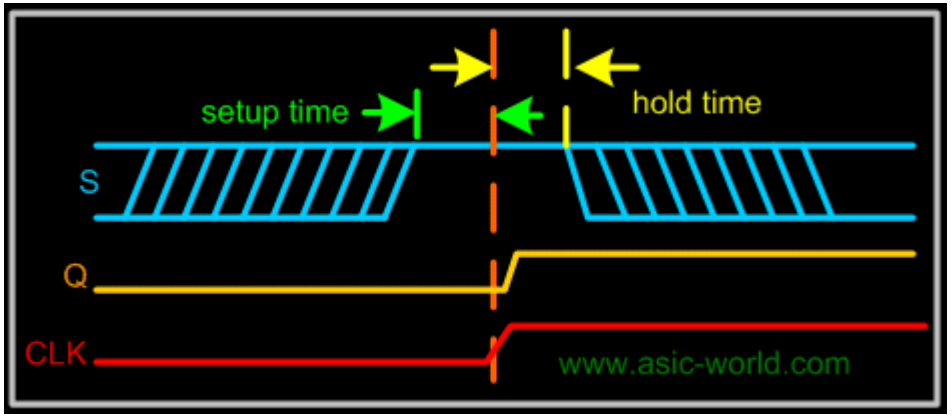
- Timpul de propagare față de intrările asincrone(Ex. 5ns la familia 74AHC IC) [2]

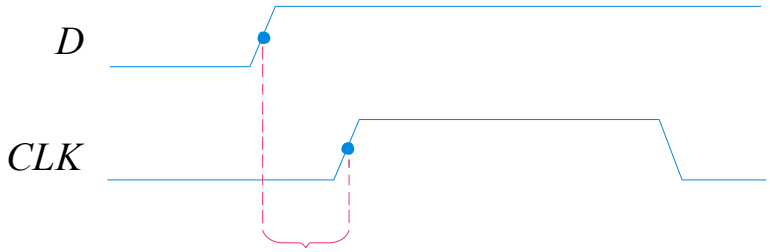


Caracteristicile bistabilelor(2)

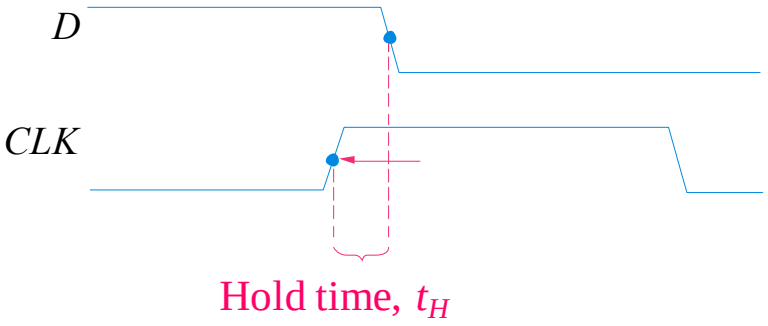
- Timp de setare –setup time – intervalul de timp minim necesar cu care datele trebuie să fie prezente pe intrări înainte de semnalul de clock.

- Timpul de menținere– hold time – intervalul de timp minim necesar pentru care datele trebuie să rămână stabile după semnalul de clock.





Timp de setare [2]



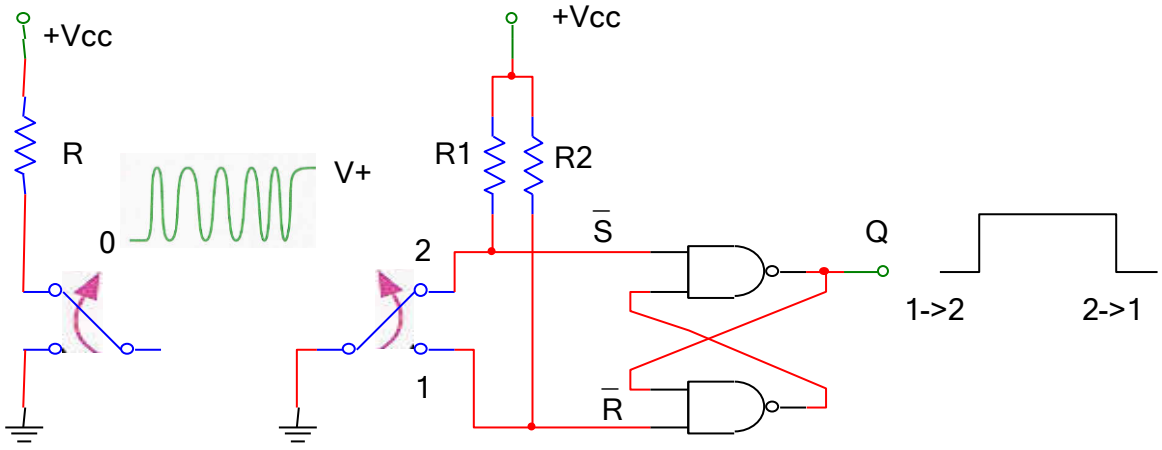
Timp de menținere [2]

Aplicații ale bistabilelor (1)

- Aplicații tipice:
  - Eliminarea comutărilor false la închiderea sau deschiderea unui contact
  - Memorii
  - Divizoare de frecvență
  - Numărătoare

Eliminarea comutărilor false la închiderea sau deschiderea unui contact

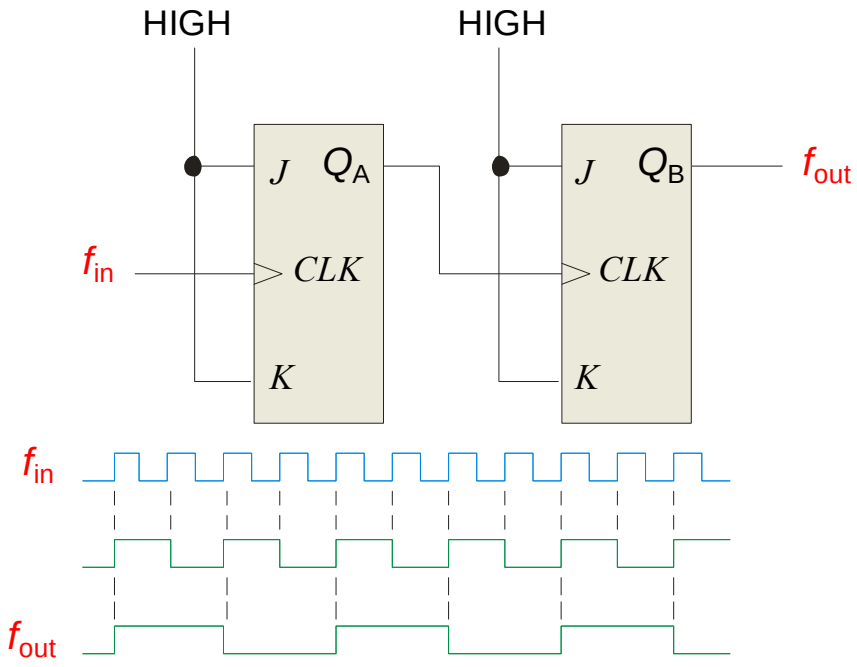
- Oscilații mecanice - „bounce”
- Debounce utilizând bistabil RS



Aplicații ale bistabilelor (2)

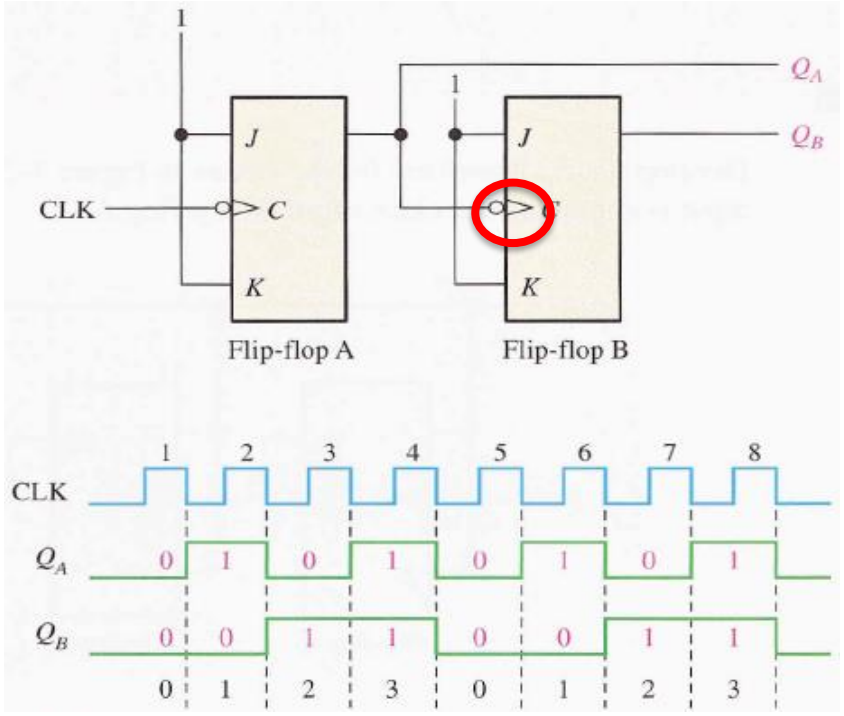
- La fiecare front de clock ieșirea bistabilelor T se complementează
- Un bistabil divizează cu doi semnalul de intrare, două bistabile cu patru, și așa mai departe

Divizare de frecvență



Bistabile utilizate ca divizoare de frecvență [2]

Numărător



Bistabile utilizate ca numărătoare [2]

Numărătoare

Numărătoare

Numărătoarele sunt circuite secvențiale care numără impulsurile sosite pe intrarea și ”afișează rezultatul ” pe ieșirea Q.

- După direcția de numărare:
  - numărătoare directe,
  - numărătoare inverse,
  - numărătoare reversibile.
- În funcție de codul în care numără:
  - binar,
  - BCD (zecimal codificat binar)
- După modul de funcționare :
  - numărătoare asincrone
  - numărătoare sincrone
- Alte facilități:
  - ștergere sincronă/asincronă
  - presetare sincronă/asincronă

Numărătoare asincrone

- Fiecare FF-T divizează cu 2 frecvența impulsului aplicat la intrarea sa.
- **Daca interconectăm ”n” celule T** => n x TFF, divizare cu 2^n .
- **Capacitatea** unui numărător reprezintă numărul maxim cu care se termină ciclul de numărare.
- Numărătorul are **m = k + 1** stări distincte.
- Numărul **m** este numit **modulul** numărătorului (**numărător module m)**.
- De ex. 4 TFF – numără între 0-15 => m=16

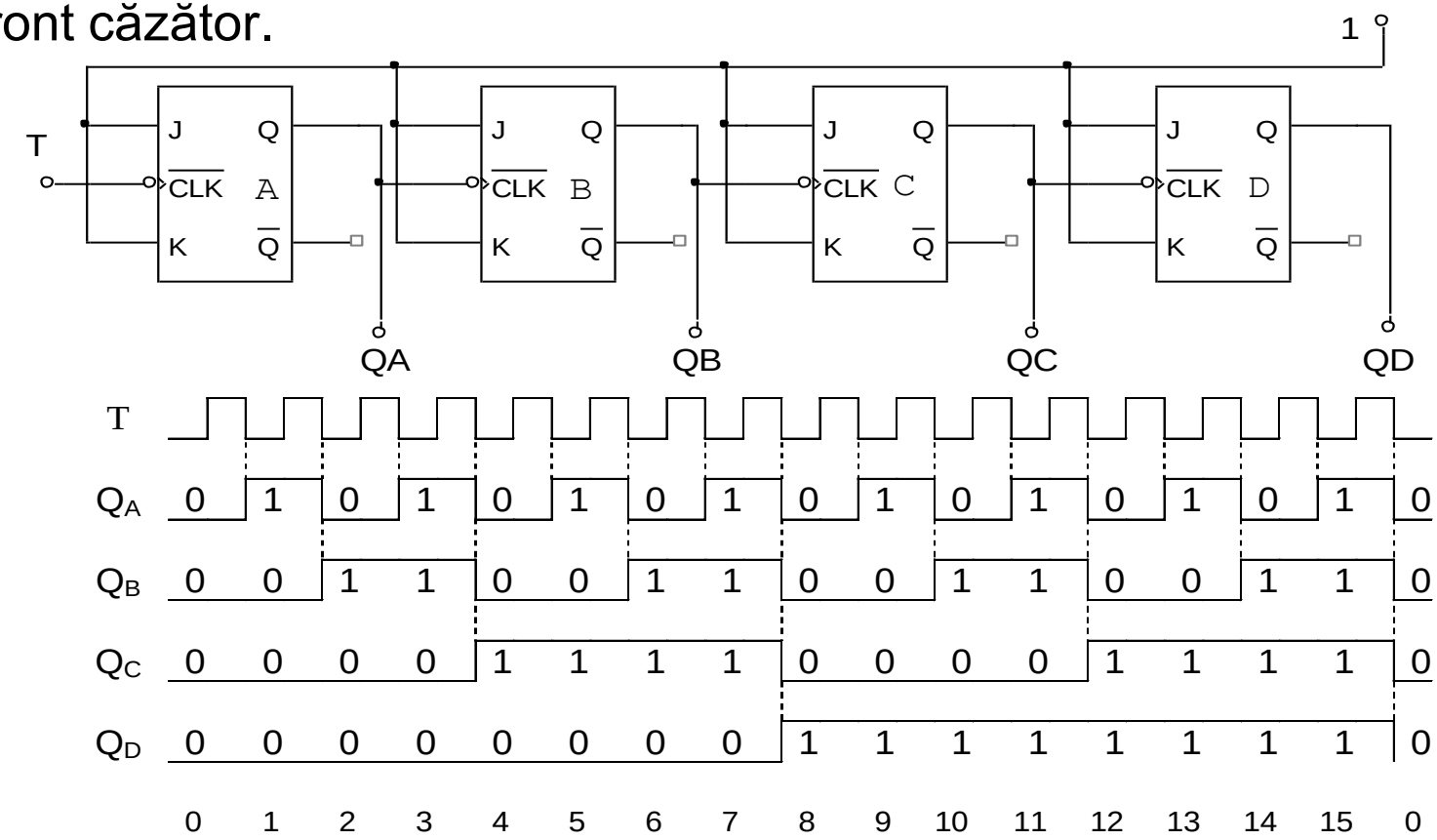
Nr. Imp.tact	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16 (0)	0	0	0	0
17 (1)	0	0	0	1

Funcționarea ***asincronă*** = *bistabilele nu comută sincron (simultan). Semnalul aplicat la intrare declanșează modificarea stării bistabililor. In continuare un FF îl provoacă basculareа următorului****.***

=> Pentru un timp scurt există o combinație greșită pe ieșiri

Numărătoare asincrone directe

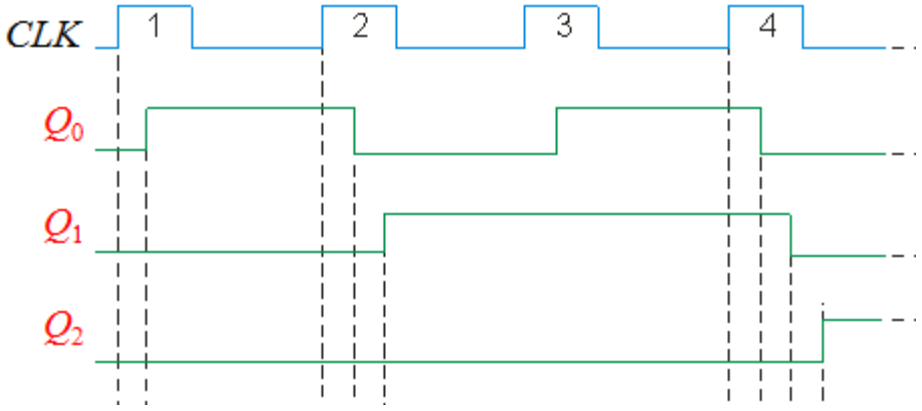
- Lanţ de bistabile de tip T ce au ieşirile Q conectate la intrările de tact ale bistabilului următor, iar intrările T conectate la nivel logic 1.
- Numărarea se face în sens direct dacă se folosesc bistabile active pe front căzător.



$$N_x = Q_D*2^3 + Q_C*2^2 + Q_B 2^1 + Q_A*2^0$$

Caracteristicile numărătoarelor asincrone

- În perioada de tranziție a stărilor apar si combinații nedorite pe ieșiri



- Datorită simplității circuitului are dezavantajul funcționării la frecvențe limitate.
- Daca timpul de propagare tipic pentru un FF este de T_p = 10 ns atunci timpul de propagare total pentru un numărător asincron pe 4 biți este:

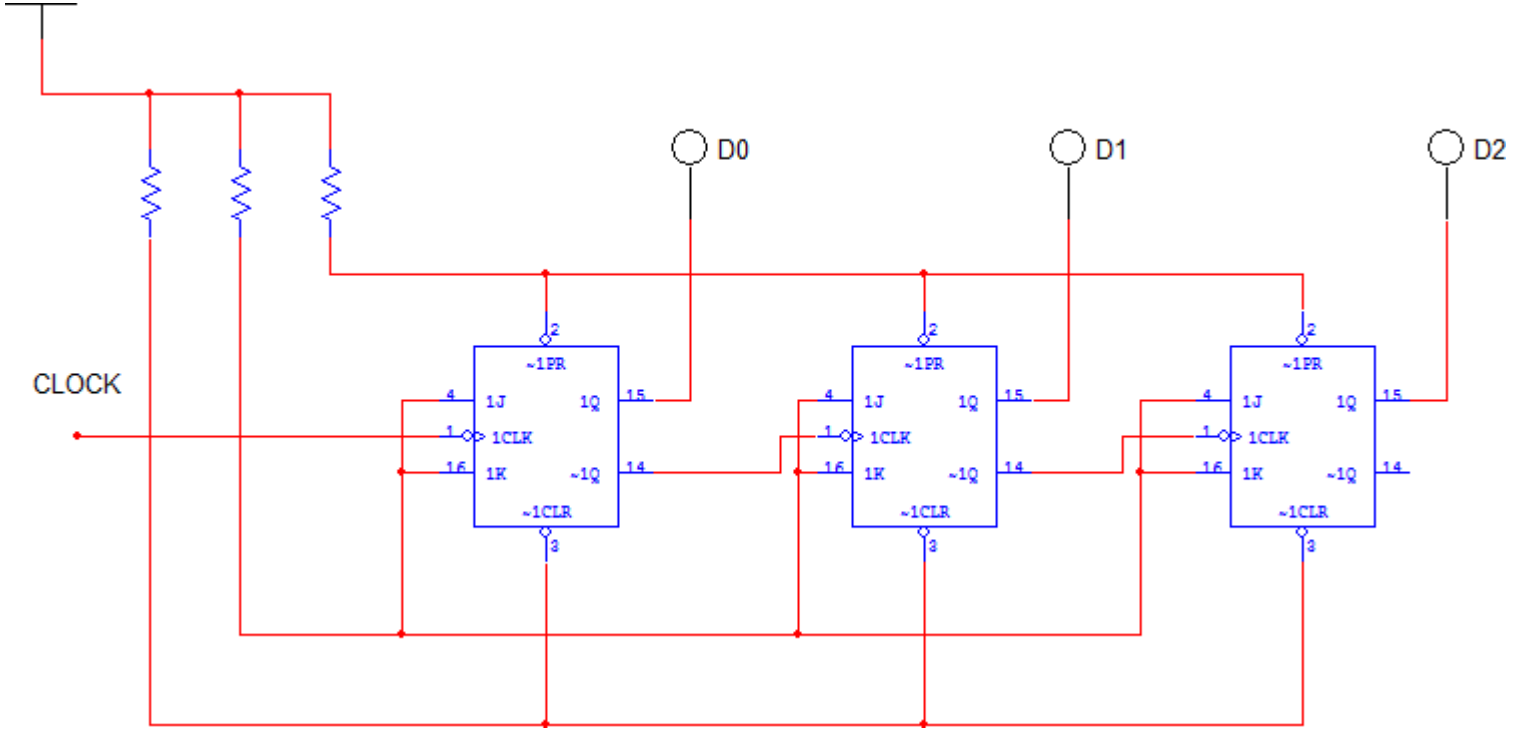
  - $T_{p(tot)} = 4 \times T_p = 4 \times 10 \text{ ns} = 40 \text{ ns}$

- Frecvența maximă de funcționare:

  - $f_{max} = 1/ T_{p(tot)} = 1/40 \text{ ns} = 25 \text{ MHz}$

Numărătoare asincrone inverse

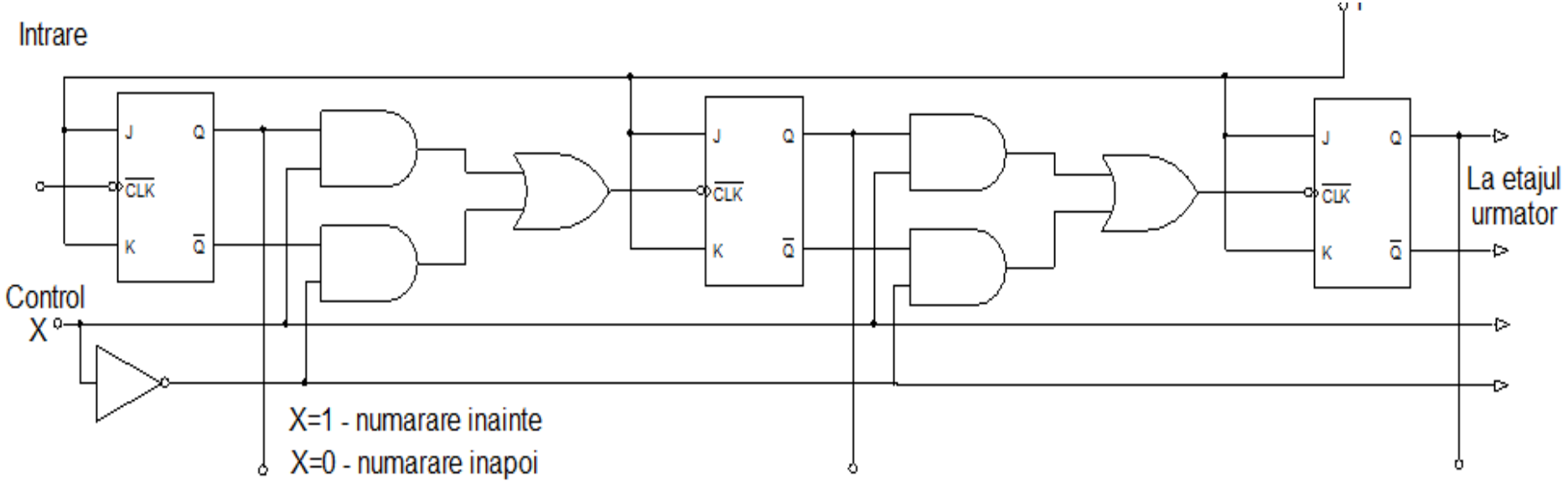
- Se obțin prin legarea ieșirii $\overline{Q}$ a bistabilului T la intrarea bistabilului următor.
- Folosim bistabile active pe front căzător.



Numărător asincron invers. Simulare cu programul Multisim.

Numărătoare reversibile

- Daca conectăm ieșirea Q a FF-T la intrarea FF-T următor – numărare directă
- Daca conectăm ieșirea $\overline{Q}$ a FF-T la intrarea FF-T următor – numărare inversă
- Un **numărător reversibil** se obține dacă cu ajutorul unui semnal de control (X) și a unor porți logice realizăm legarea ieșirii Q la intrarea următoare când X =1 și a ieșirii $\overline{Q}$ când X=0.



Scurtarea ciclului de numărare

- Proiectarea unor numărătoare Modulo-N, N ≠ putere a lui 2
- La cel de al N-lea impuls, numărătorul revine în starea inițială
- Procedura de proiectare a numărătorului divizor cu N este următoarea :
- Se caută numărul N de bistabile necesar (n),

  $2^{n-1} \leq N \leq 2^{n}$;
- Se leagă toate bistabilele într-o schemă de numărător asincron cu transport succesiv
- Se leagă toate ieşirile bistabilelor care au Q=1 după al N-lea impuls de tact la intrările unei porţi ŞI-NU. Se leagă ieşirea porţii ŞI-NU la terminalele $\overline{CL}$ ale bistabililor.

De ex: ciclurile unui nr. pe 4 biți

BINAR:

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$

ZECIMAL:

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$

Numărător zecimal asincron

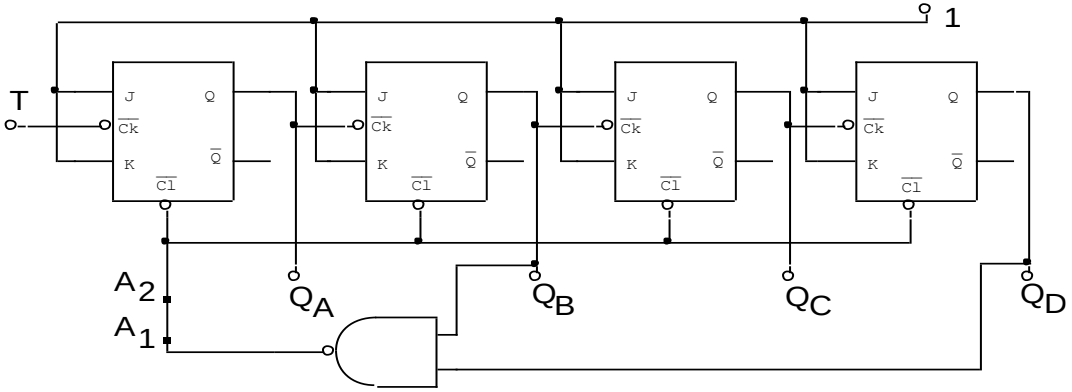
$2^{n-1} \leq N \leq 2^n$

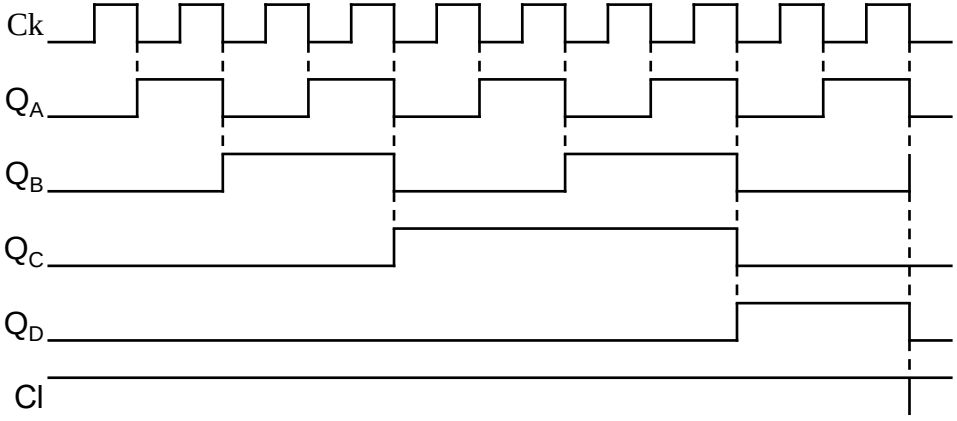
$N=10 \Rightarrow n=4$

$2^{4-1} \leq 10 \leq 2^4$;

- circuitul este format din 4 bistabile ($2^3<10<2^4$) cu intrare de ștergere Cl
- completăm circuitul cu o poartă NAND care la apariția stării **1010** (10 în zecimal) șterge toate bistabilele și aduce numărătorul în starea inițială **0000**

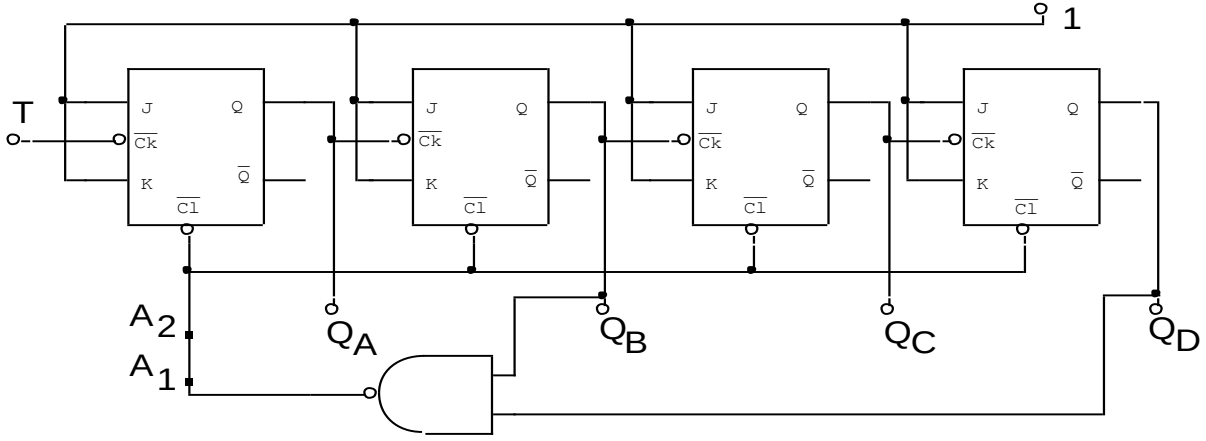
	Q_D	Q_C	Q_B	Q_A
(Stare) 0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (0)	1(0)	0	1(0)	0

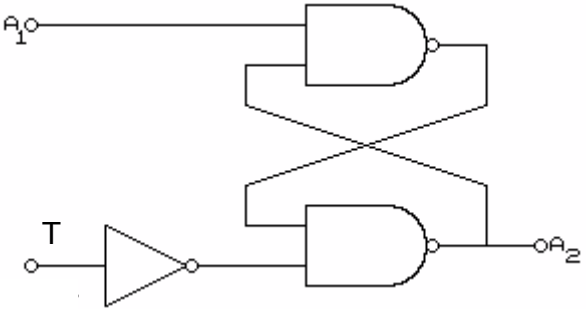




Numărător zecimal asincron (2)

- Dacă timpul de propagare de la apariția semnalului de inițializare la ștergerea bistabilului variază puternic de la un etaj la altul, impulsul de ștergere poate să se termine înainte ca toate bistabilele să fie aduse la zero





Între punctele A_1 și A_2 inserăm un circuit ce memorează ieșirea porții ŞI-NU după cel de-al M-lea impuls

Numărătoare asincrone integrate

- **Numărător BCD - SN7490**. Numărător divizor cu 10. Ieșirile circuitului sunt în cod BCD. Poate fi utilizat de asemenea ca divizor cu 2 sau 5. Circuitul este prevăzut cu intrări speciale pentru aducerea la 0 și pentru inițializarea combinației 1001.
  Un 1 logic aplicat la intrarea R_{0i} toate bistabilele sunt șterse.
  Un 1 logic aplicat la intrarea R_{9i} ieșirile circuitului sunt aduse în starea 1 0 0 1 (9 zecimal).
- **Numărător binar pe 4 biți - SN7493.**
  Numărător divizor cu 16 - dacă impulsurile de numărat se aplică la intrarea In.A și ieșirea Q_A se conectează la intrarea In.B .
- Daca impulsurile de intrare se aplică la intrarea In.B și legăm între ele Q_D și In.A, obținem un divizor, la care la ieșirea Q_A se obțin impulsuri dreptunghiulare simetrice.

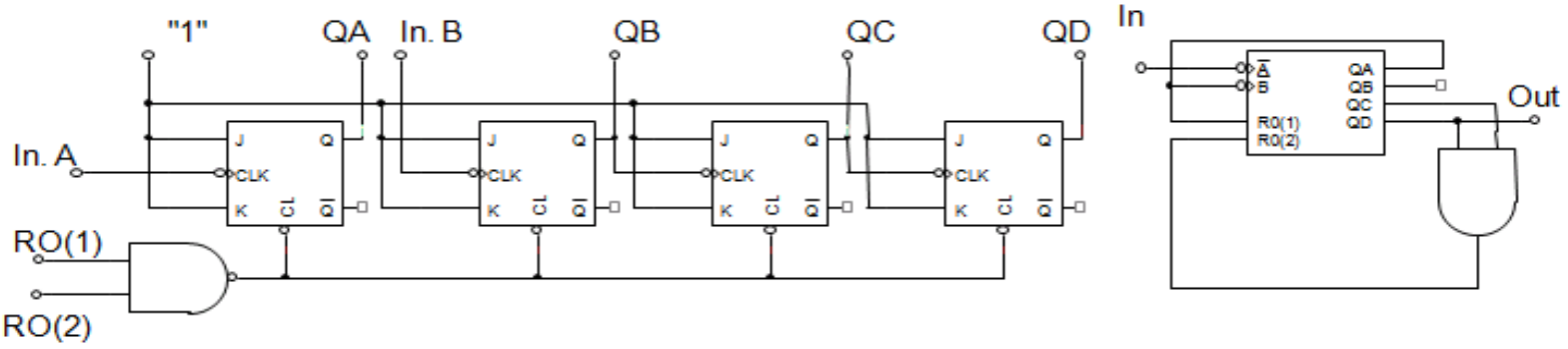


Figura din dreapta reprezintă un numărător modulo - ?

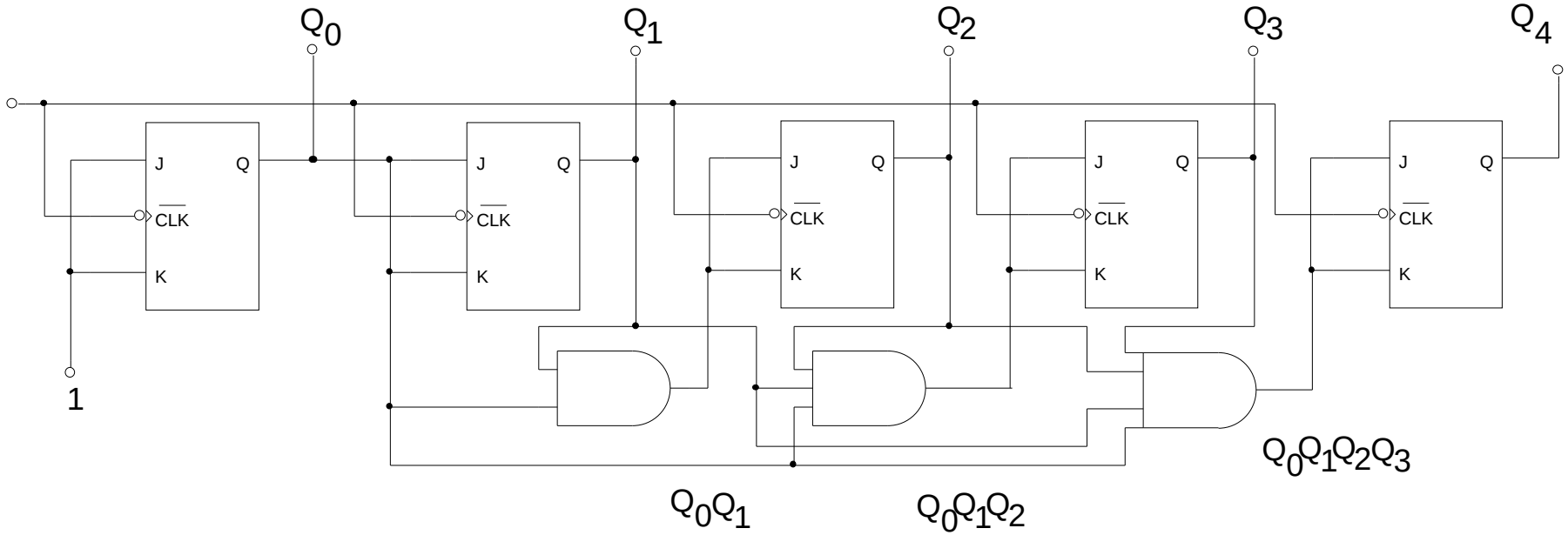
$Cl=R_{01}\&R_{02}=Q_A\&(Q_C\&Q_D)$ M = 13

Numărătoare binare sincrone

- Elimină neajunsurile numărătoarelor sincrone.
- Toate bistabilele comută sincron cu semnalul de clock extern
- Q_0 își modifică starea la fiecare impuls de clock. Acest lucru se obține daca:
  - $T_0=J_0=K_0=1$
- FF1 comută numai dacă $Q_0=1$, în consecință:
  - $T_1=J_1=K_1= Q_0$ (intrările lui FF1 se leagă la Q_0)
- FF2 comută dacă $Q_0=Q_1=1$, în consecință:
  - $T_2=Q_0Q_1$
  - $T_3=Q_0Q_1Q_2$
  - $T_4=Q_0Q_1Q_2Q_3$

Nr.	Q_0	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	1	0	0	0
4	0	0	1	0	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	1	1	0	0
8	0	0	0	1	0
9	1	0	0	1	0
10	0	1	0	1	0
11	1	1	0	1	0
12	0	0	1	1	0
13	1	0	1	1	0
14	0	1	1	1	0
15	1	1	1	1	0
16	0	0	0	0	1
17	1	0	0	0	1

Numărător sincron cu transport paralel



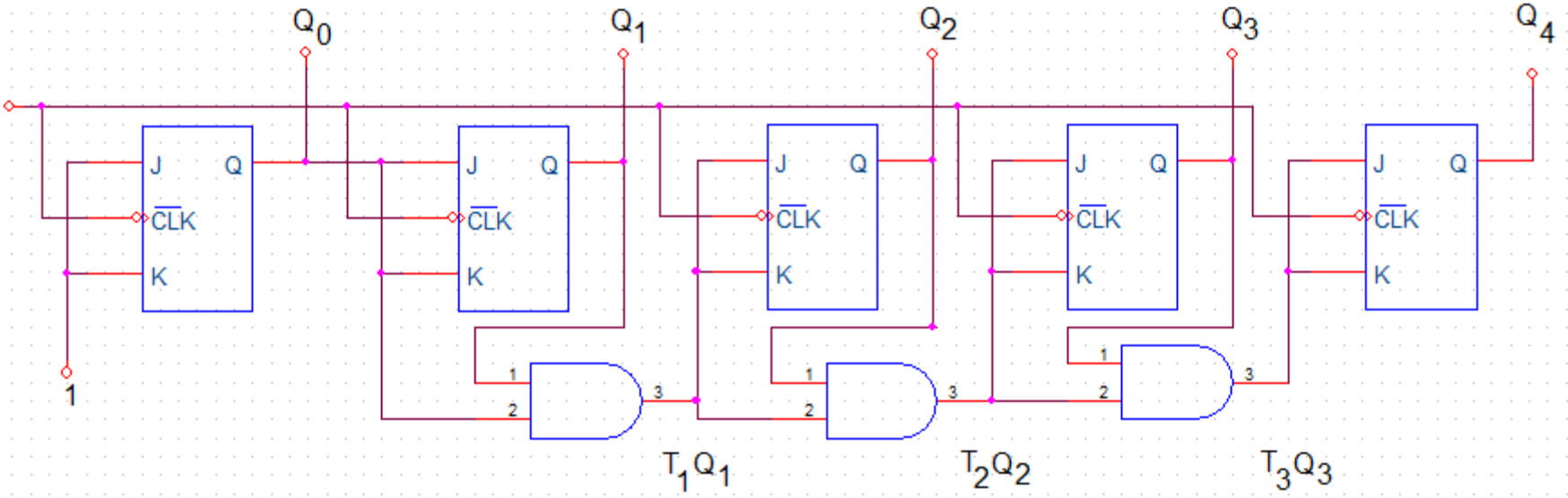
$T_0=1$
$T_1=Q_0$
$T_2=Q_0Q_1$
$T_3=Q_0Q_1Q_2$
$T_4=Q_0Q_1Q_2Q_3$

$T_{min}=t_{pb}+t_{pp}$.

Pentru etajul *n este nevoie de o poartă ȘI cu n-1 intrări*

Numărător sincron cu transport succesiv

$T_1=Q_0$
$T_2=Q_0Q_1=T_1Q_1$
$T_3=Q_0Q_1Q_2=T_2Q_2;$
$T_4=Q_0Q_1Q_2Q_3=T_3Q_3$

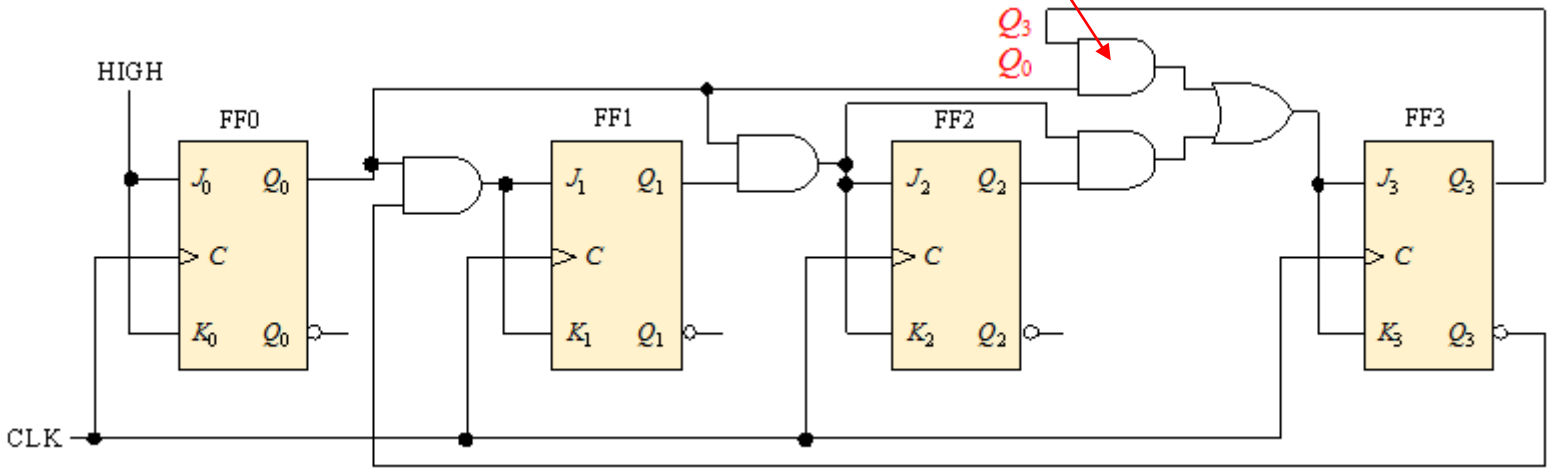


- Dezavantaj: timpii de propagare prin porțile ȘI se adună

$T_{min}=t_{pb}+(N-2)\ t_{pp}$

Numărător BCD sincron

- Prin adăugarea de logică suplimentară care detectează atingerea stării 1001, numărătorul este readus la starea inițială 0000.
- Această poartă detectează starea 1001 și provoacă basculareа lui FF3 pe următorul impuls de clock.
- FF0 basculează la fiecare impuls de clock. Astfel numărarea reîncepe de la 0000.

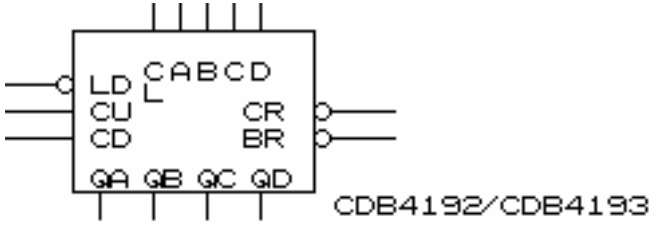


Numărător BCD sincron [2]

Numărătoare sincrone integrate

SN74192 – numărător reversibil în cod BCD

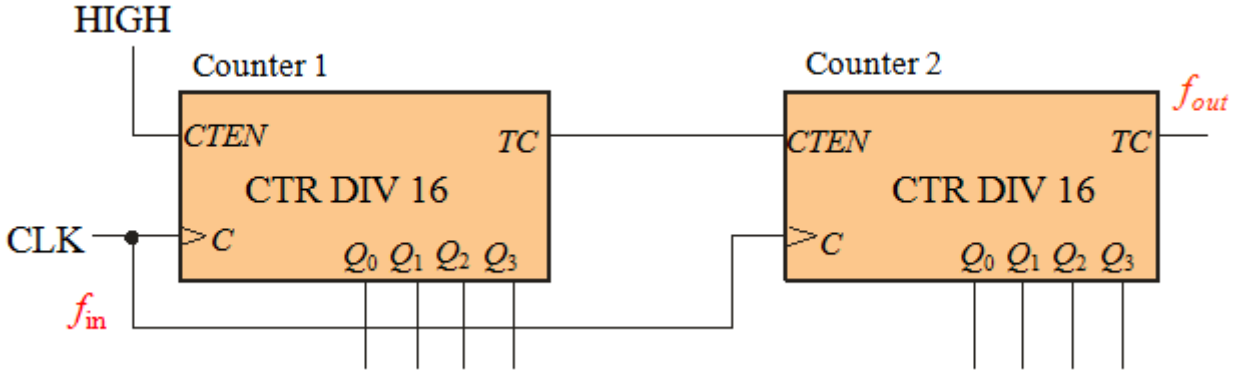
SN74193 - numărător reversibil sincron binar de 4 biţi



- QA, QB,QC,QD – ieșiri.
- CR (carry - transport) indică prin nivel logic 0 atingerea combinației 1001, respectiv 1111 la ieșirile QDQCQBQA ale circuitelor CDB4192E, respectiv CDB4193E atunci când numărarea se face înainte.
- BR (borrow - împrumut) comută pe nivelul logic 0 atunci când ieșirile numărătoarelor ating starea 0000, iar numărarea se face înapoi.
- Cele 8 intrări ale circuitului sunt:
  - LD (load - încărcare) comandă încărcarea asincronă a combinației prezente la intrările DCBA.
  - CL (clear) șterge conținutul tuturor numărătoarelor atunci când este trecut în starea logică 1
  - CU (count up - numără înainte) intrare de tact pentru sensul de numărare direct.
  - CD (count down - numără înapoi) intrare de tact pentru sensul de numărare invers.
  - D,C,B,A - intrări de date.

Legarea în cascadă a numărătoare

- Este o metodă de obținere a numărătoarelor cu modul mai mare.
- Următorul numărător este autorizat doar când numărătorul anterior a ajuns la valoarea sa maximă.



Întrebări:

a) Care este modulul numărătorului din obținut prin cascadarea a două de numărătoare modulo 16?

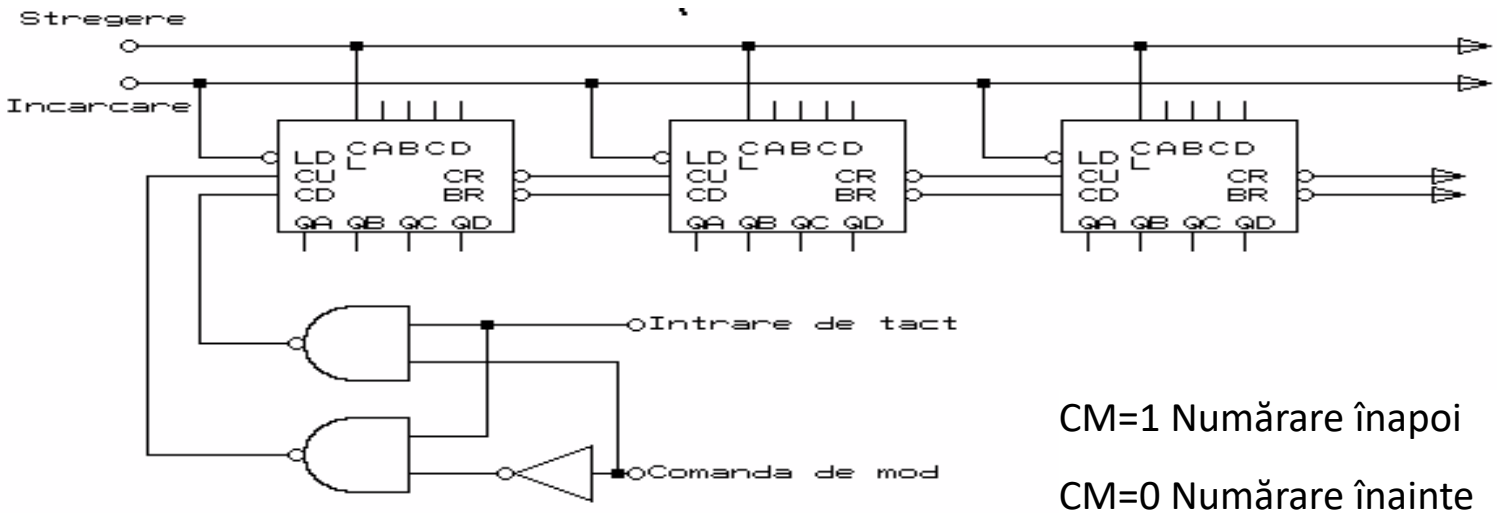
b) Dacă f_{in} =100 kHz, cât este f_{out}?

Răspunsuri:

a) Fiecare numărător divide frecvența cu 16. Astfel modulul este: 16^2 = 256.

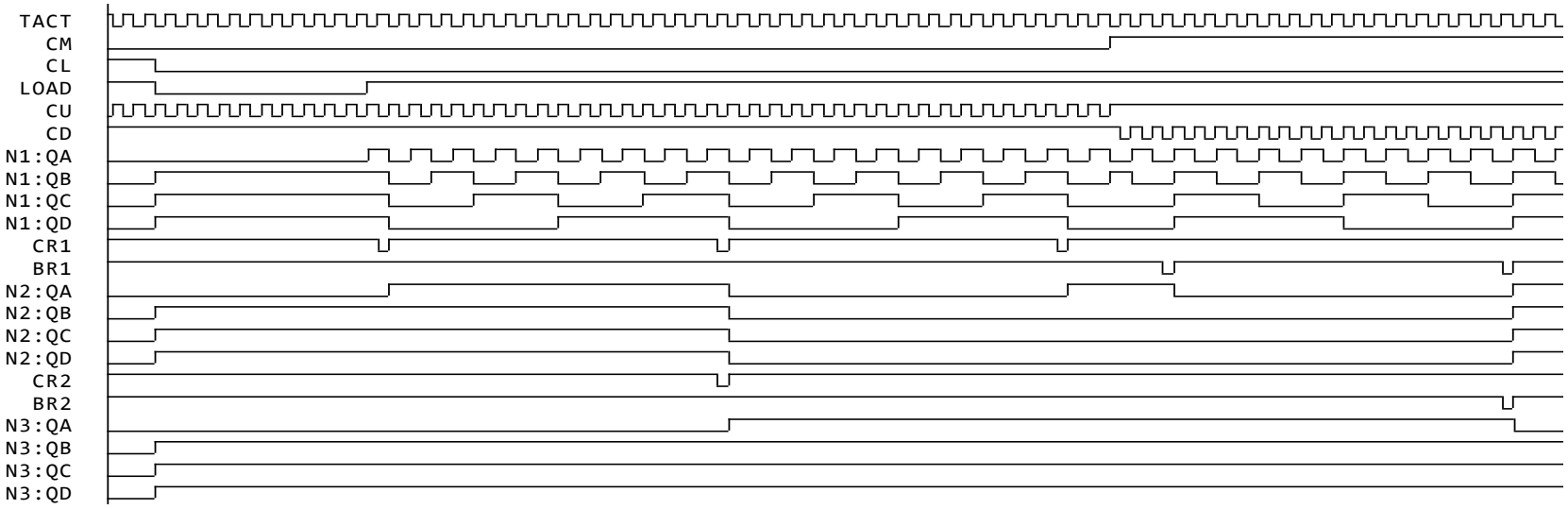
b) Frecvența de ieșire este: 100 kHz/256 = 391 Hz

Numărător sincron reversibil de 12 biți cu transport succesiv



CM=1 Numărare înapoi

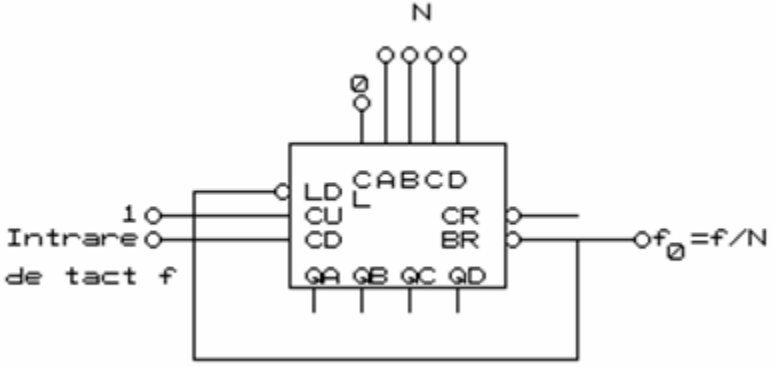
CM=0 Numărare înainte



Exemple de utilizare a numărătoarelor sincrone

Divizoare de frecvență:

- La intrările de date ale modulului se aplică raportul de divizare dorit N.
- Numărătorul numără în jos și în momentul în care ieșirile sale devin egale cu 0 se generează semnalul BR, prin intermediul căruia se reinițializează circuitul.
- Semnalul de ieșire BR are durata tipică de 30 ns și frecvența f/N.



Generarea complementului față de 2:

- Numărul care se dorește a fi convertit se aplică sub formă de impulsuri la intrarea CD a numărătorului sincron.
- Dacă circuitul a fost inițial adus la 0, se obține la ieșirile numărătorului, complementul față de 2 a numărului introdus.

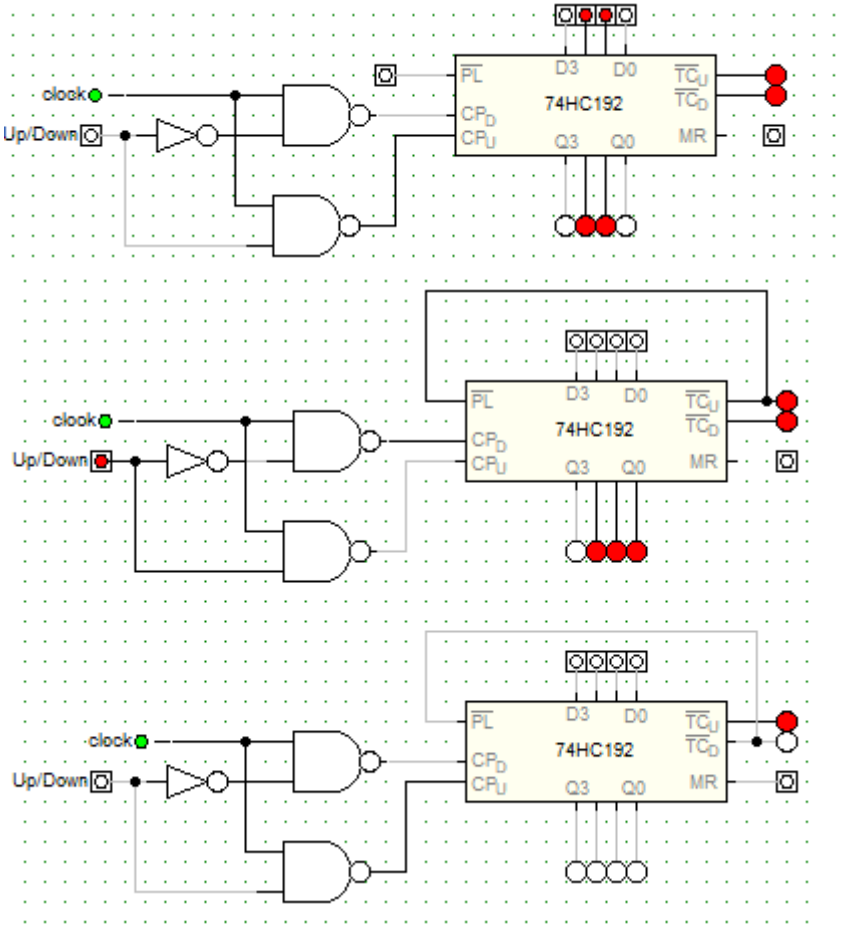
Simularea numărătoarelor

1. /PL = 1 => Încărcare număr prezent pe intrările D

2. /PL = 0

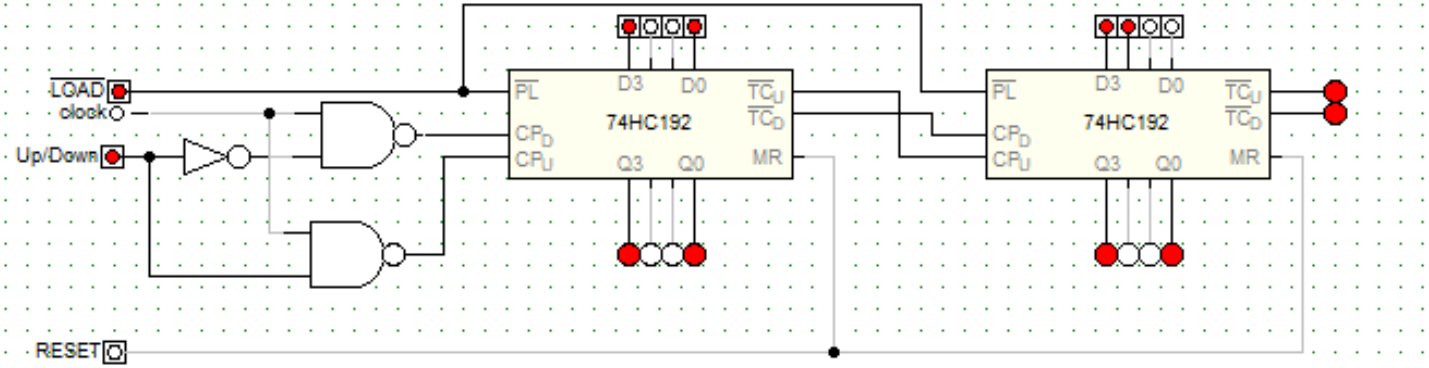
   a. Up/Down = 1 Numărare înainte

   b. Up/Down = 0 Numărare înapoi

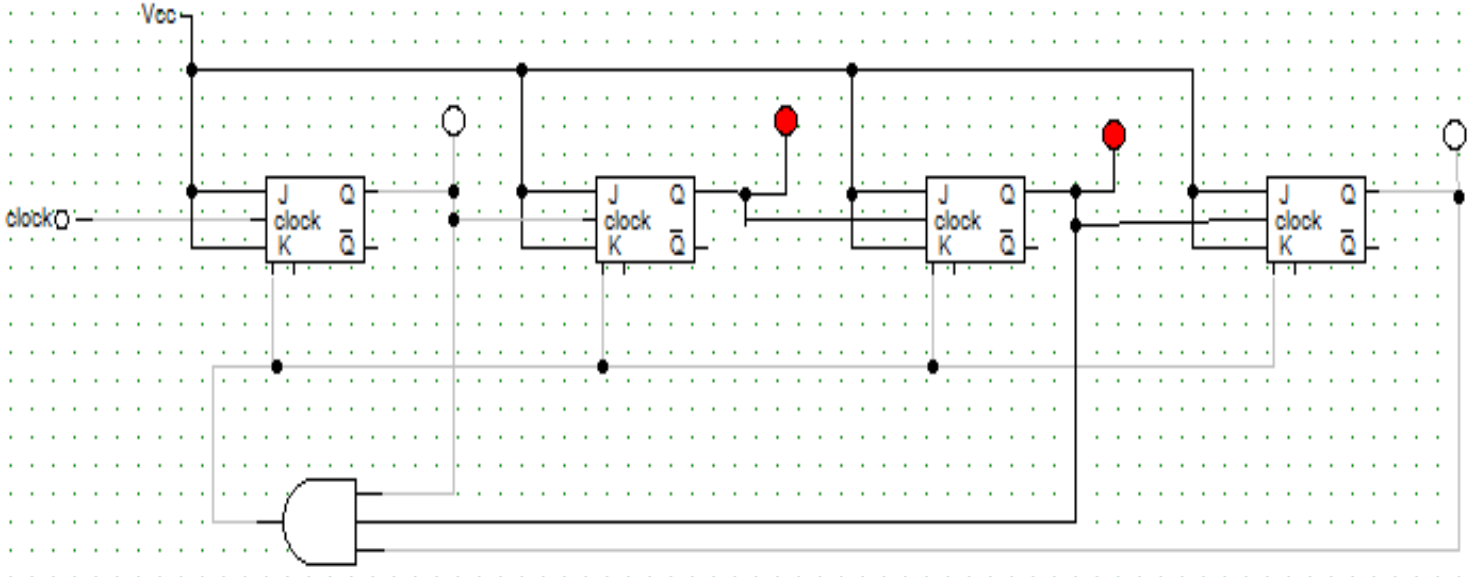


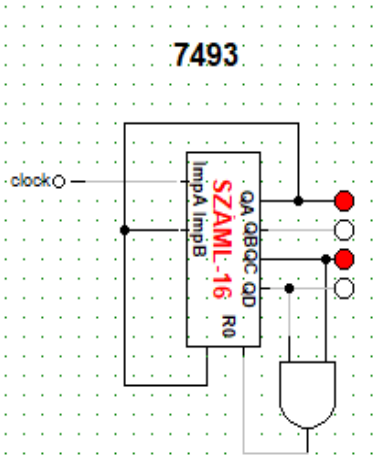
Simularea numărătoarelor (2)

Legarea în cascadă



Numărător modulo 13

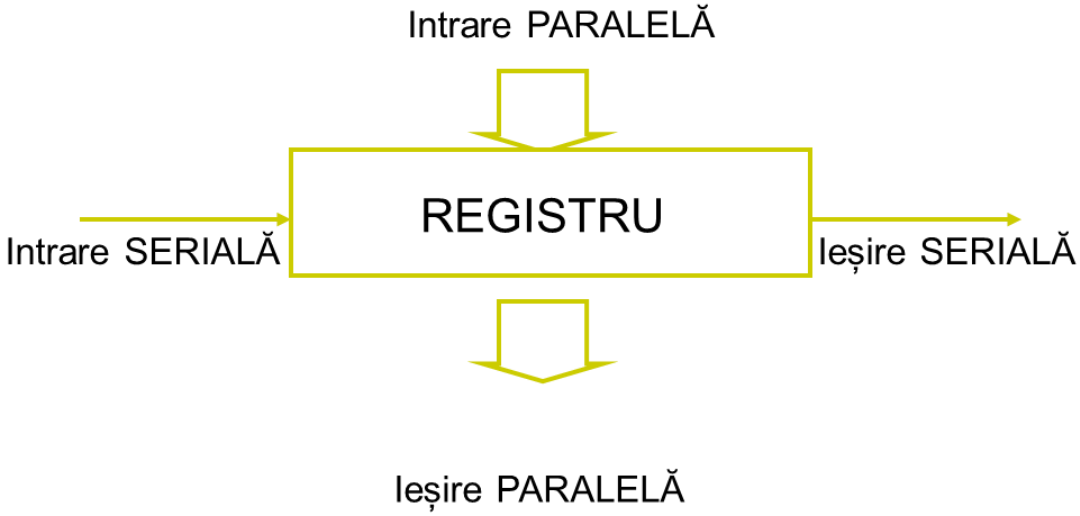




Registre

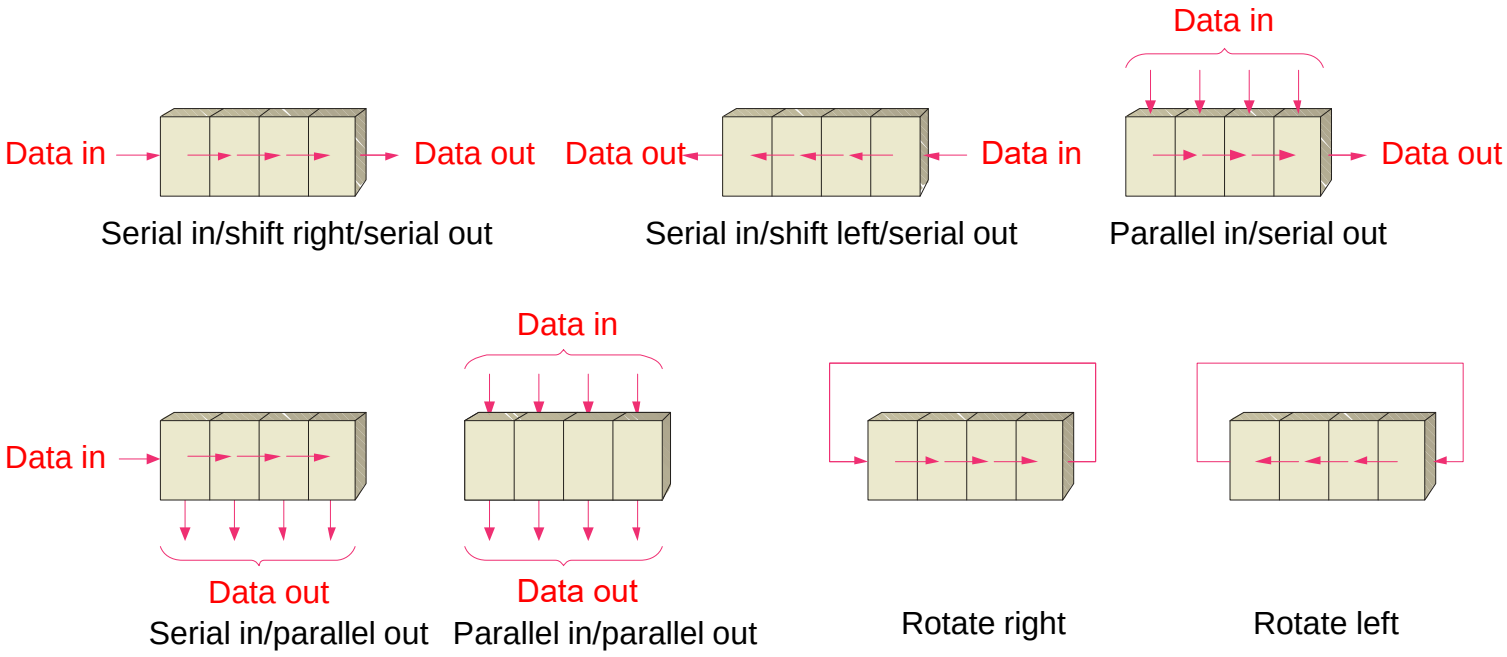
Registre

- Un registrul este un ansamblu de celule elementare de memorie în care se conservă provizoriu un grup de informații binare.
- Se realizează din bistabile RS, JK, sau D.
- Utilizări: memorarea și manipularea datelor
- Funcții:
  - introducerea datelor
    - serie
    - paralel
  - stocarea datelor
  - extragerea datelor
    - serie
    - paralel



Tipuri de registre

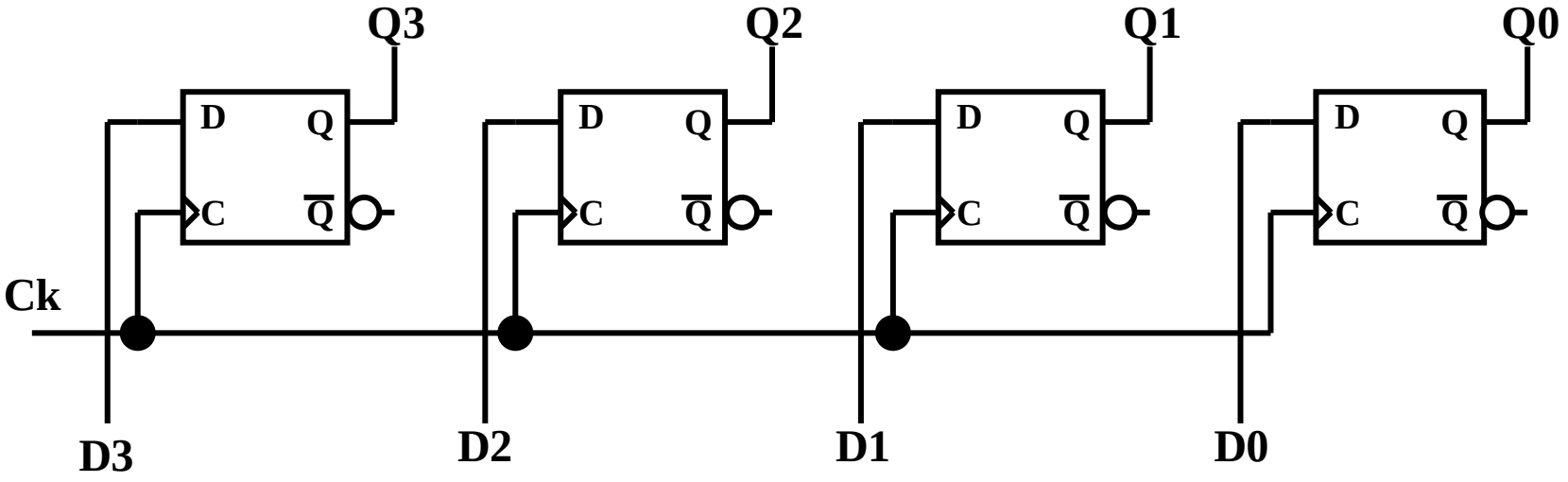
- Principalele tipuri de registre:



Tipuri de registre [2]

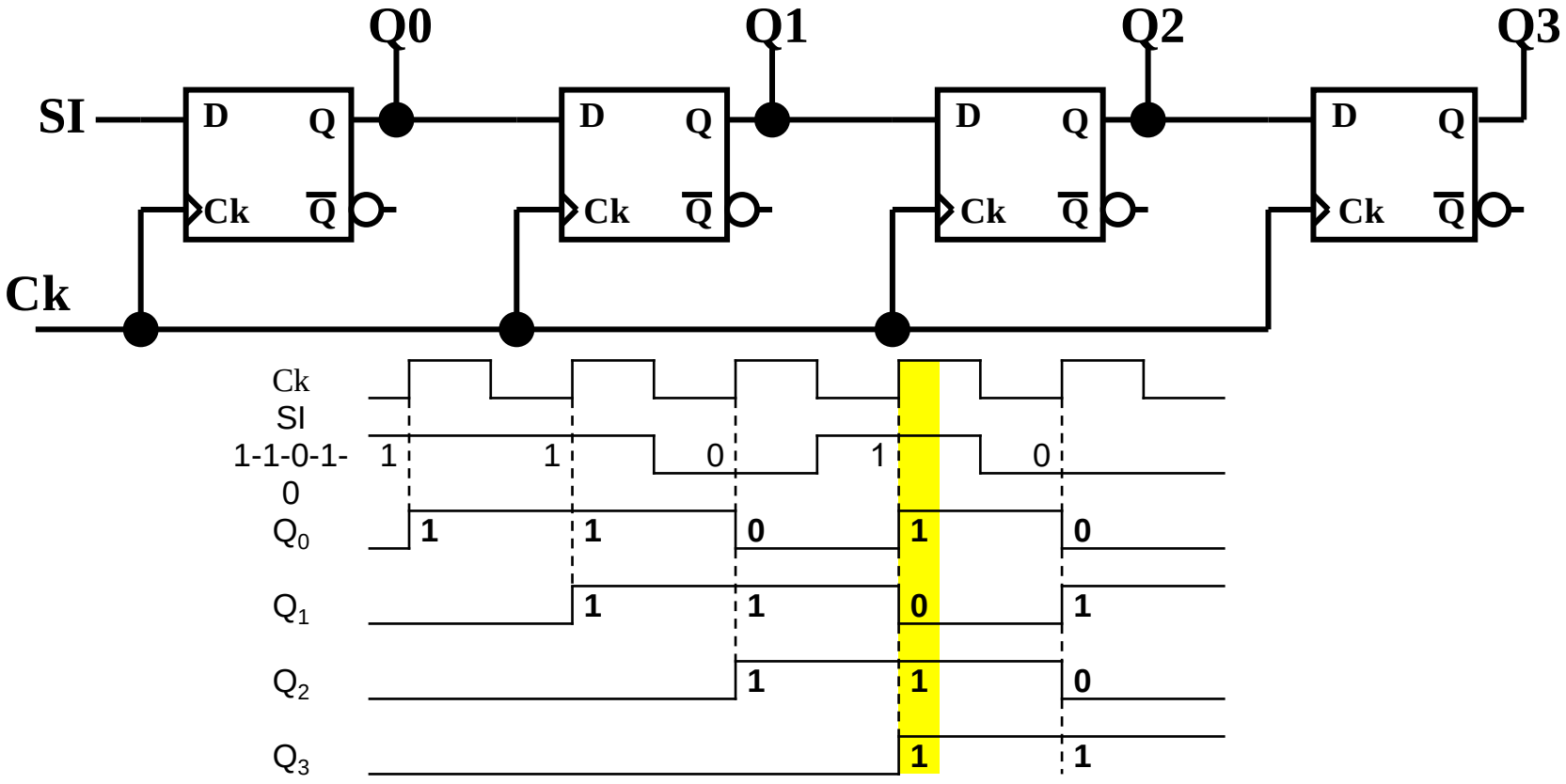
Registre tampon (buffer)

- Stocare temporară
- Înscriere paralel
- Extragere paralel
- Se utilizează bistabile cu intrare de autorizare sau active pe front
- Exemplu de registru tampon utilizând latch-ul de tip D cu intrare de autorizare

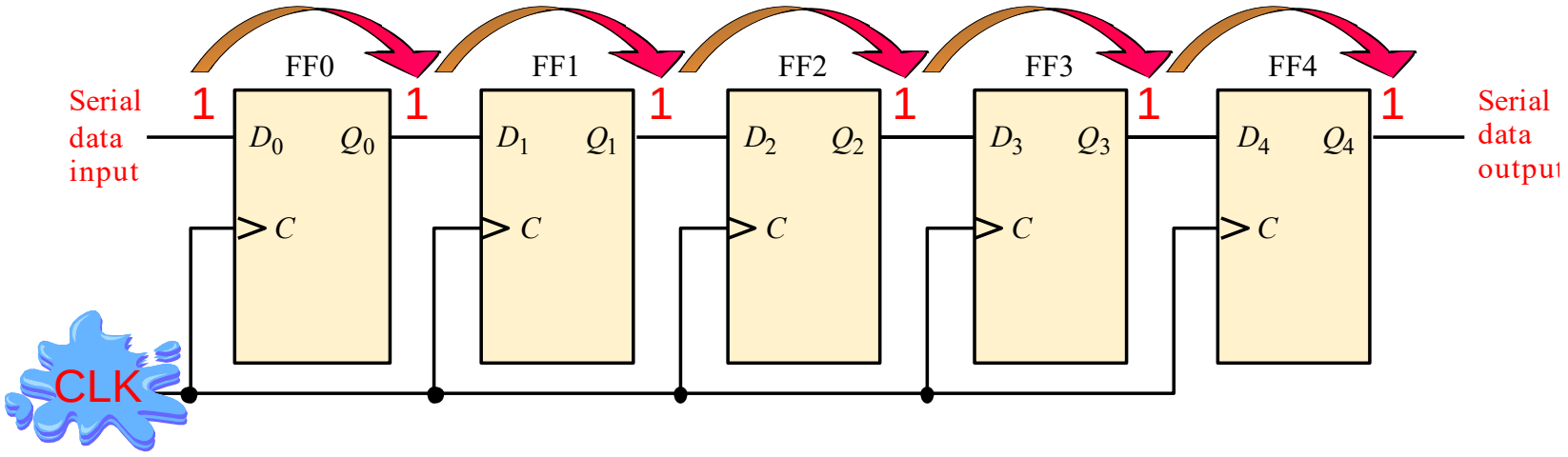


Registre de deplasare

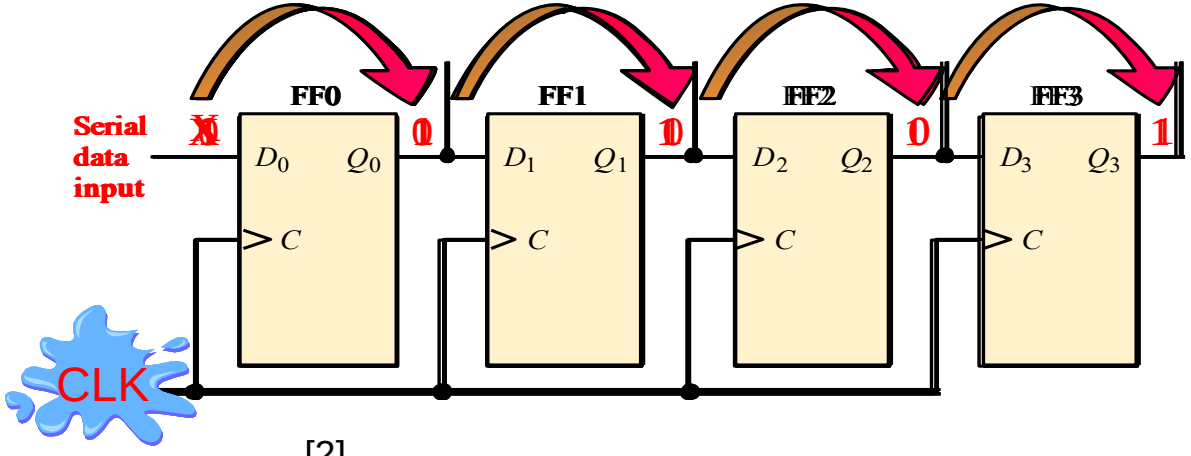
- Registre de deplasare sau șiftare
- Intrare și/sau ieșire serie
- La fiecare impuls de clock bitul de la intrare este transferat următorului flip-flop



Registre de deplasare



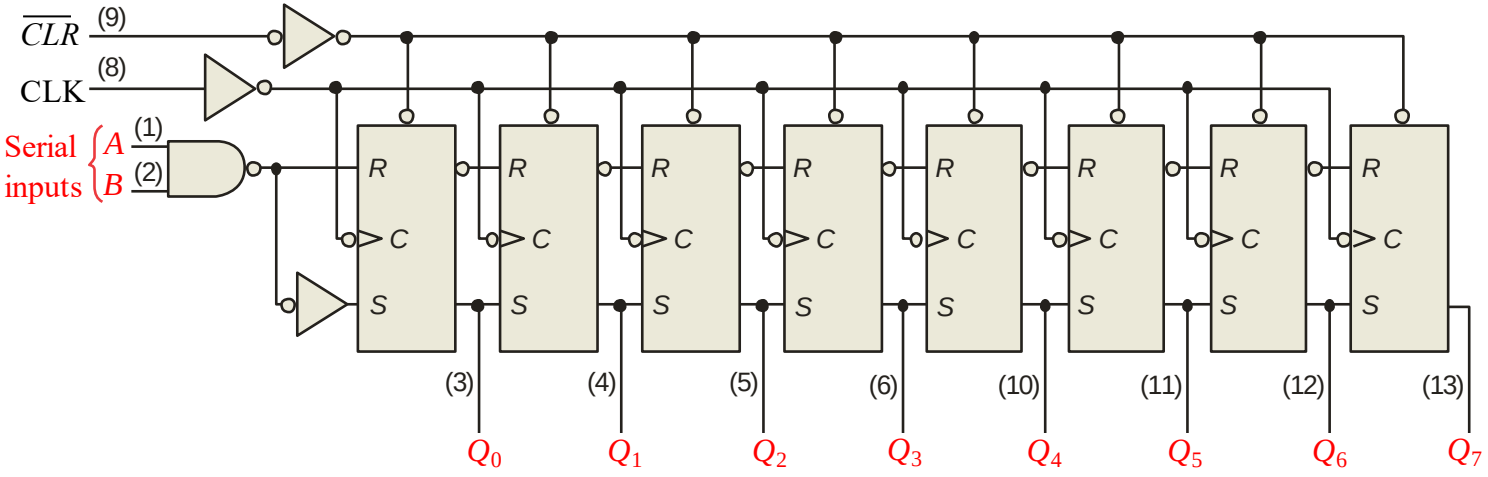
1011



[2]

Registru de deplasare S-P pe 8 biți

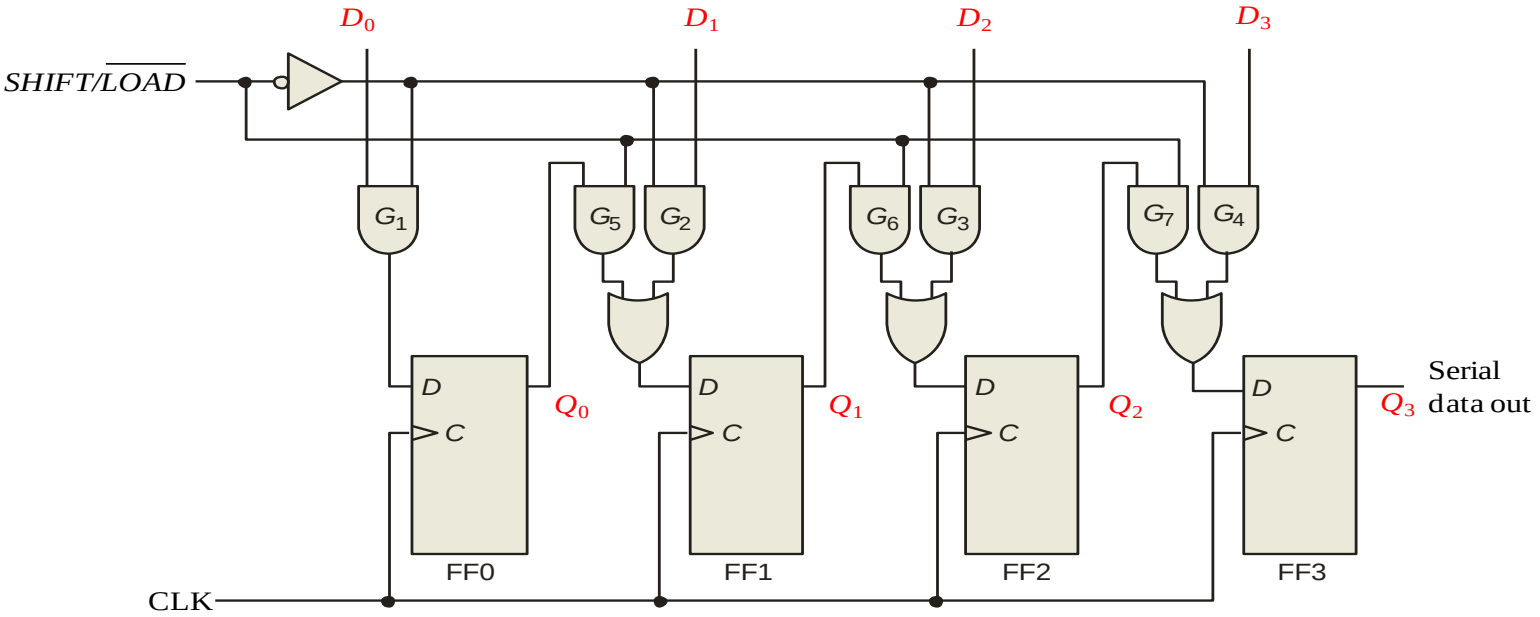
- Transformare serial/paralel
- 74HC164A – registru de deplasare S-P pe 8 biți în tehnologie CMOS
- intrări: A și B prin poartă ȘI-NU,
  - o intrare de date
  - a doua intrare poate fi:
    - intrare de autorizare
    - V_{cc}
- ștergere asincronă activă pe 0.



74HC164A – registru de deplasare S-P [2]

Registre de deplasare P-S

- Pot fi folosite la transformarea **paralel/serie a datelor**

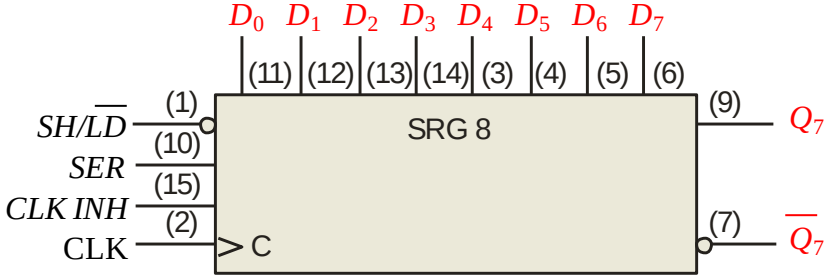


Registru de șiftare P-S pe 4 biți [2]

74HC165A – registru de șiftare P-S pe 8 biți

SH/LD = LOW Încărcare paralelă asincronă

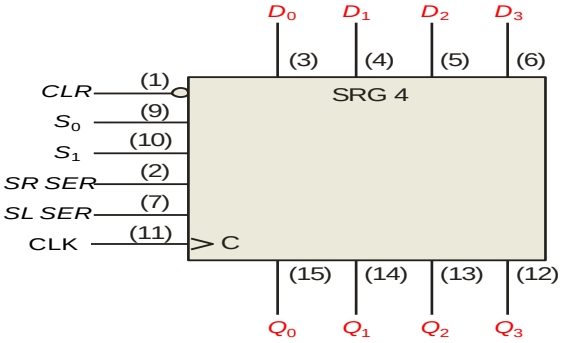
SH/LD = HIGH Deplasare sincronă



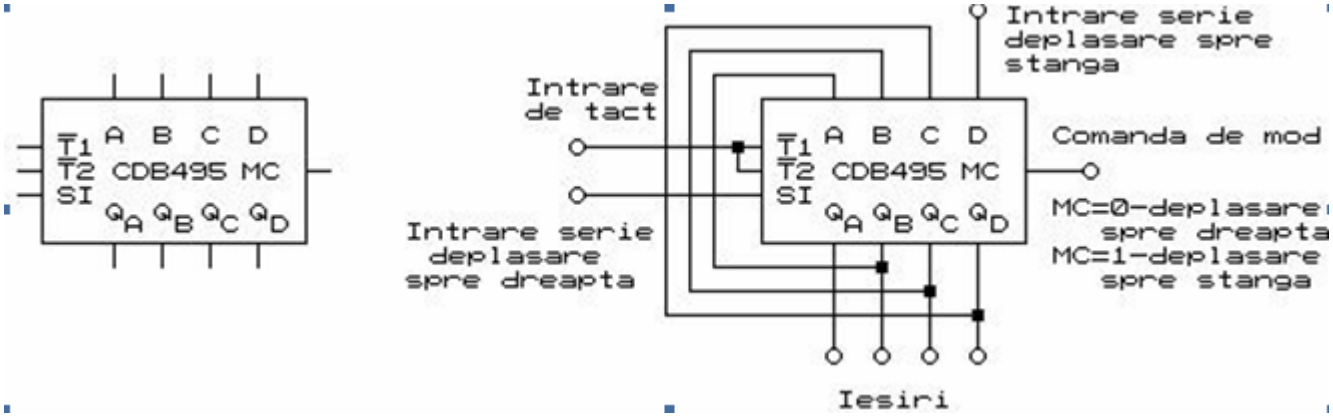
74HC165A – registru de șiftare P-S pe 8 biți [2]

Registre de deplasare universale

- Pot efectua încărcarea serie și paralel a datelor, extragerea paralel, deplasarea in ambele direcții și ștergerea datelor
- 74HC194A – registru de deplasare universal (CMOS)



S1	S2	Mod de lucru
0	0	ștergere sincronă
0	1	deplasare stânga
1	0	deplasare dreapta
1	1	înscriere paralel



- SN7495 – Registru de deplasare universal TTL

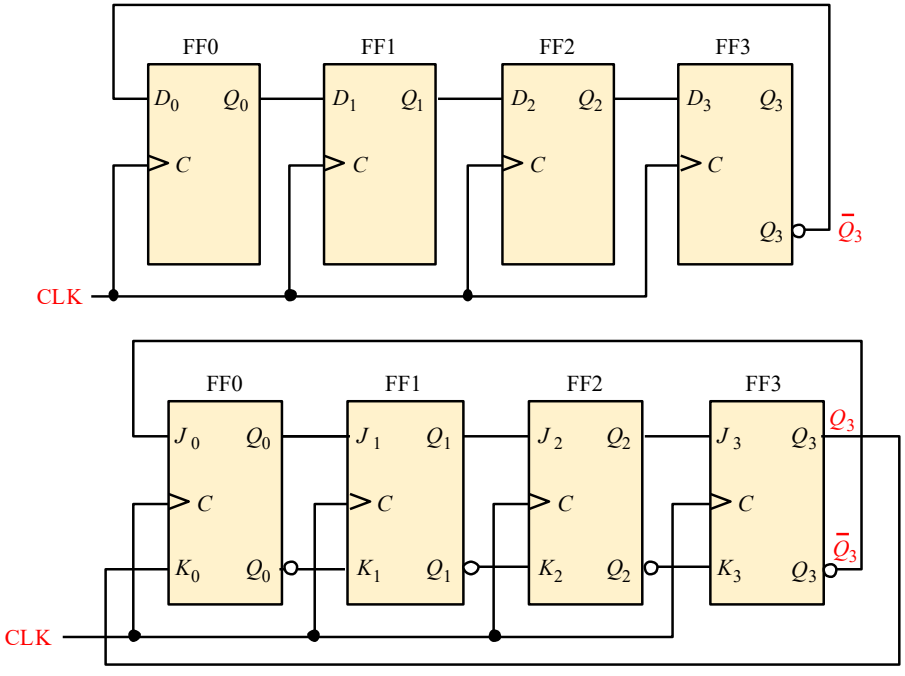
Numărătoare realizate cu registre de deplasare

•Registre de deplasare prevăzute cu reacție

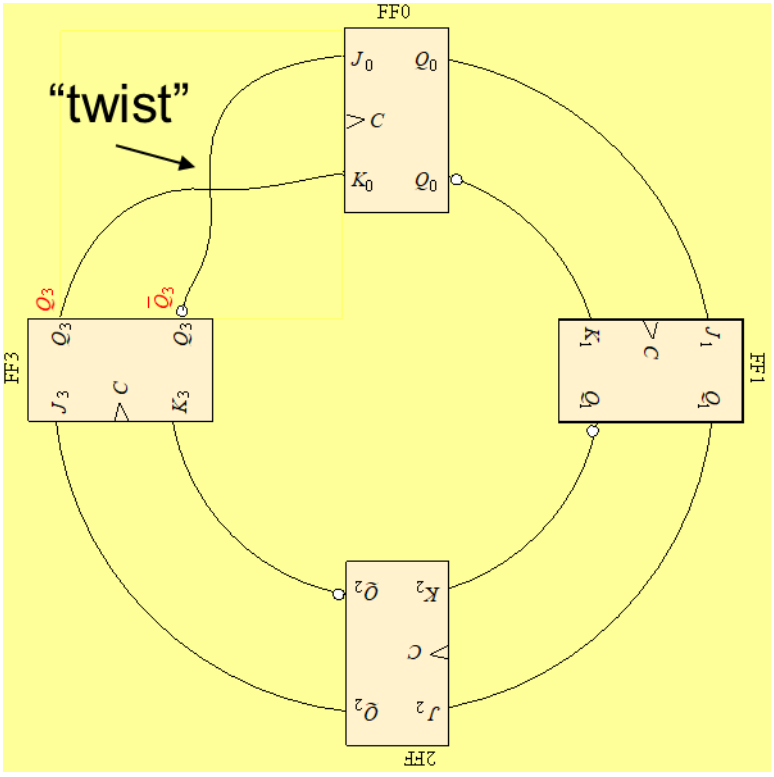
Se pot realiza numărătoare prin recircularea unui tipar de 0 sau 1.

Două tipuri importante: numărător *Johnson și numărător în inel*.

Numărătorul Johnson se poate realiza cu bistabile de tip D sau JK

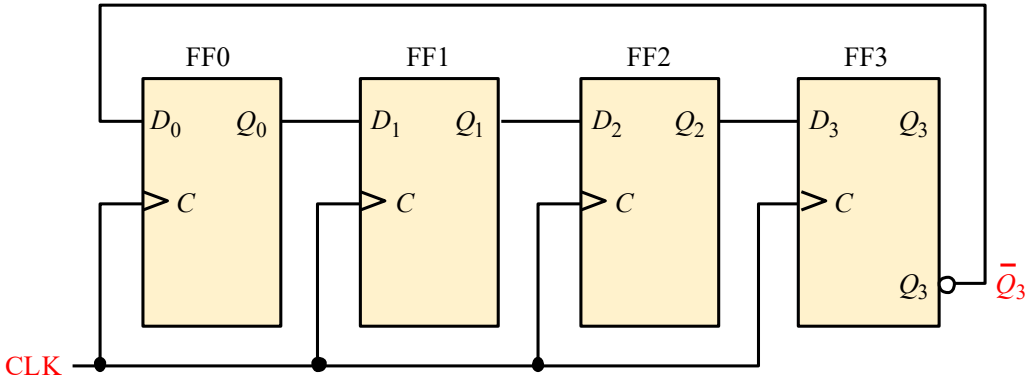


Numărător Johnson pe 4 biți [2]

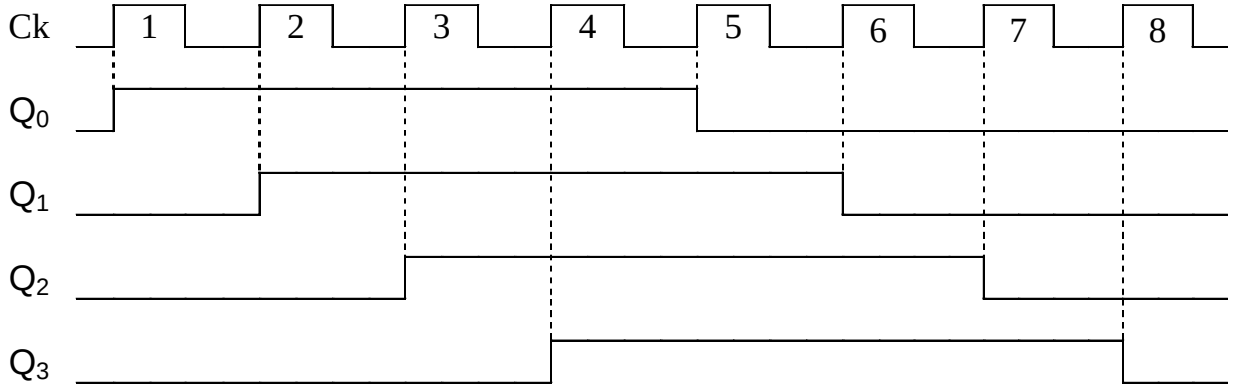


Numărătoare Johnson

- reacție de la ultima ieșire negată pe prima intrare
- capacitatea de numărare cu n bistabile este 2n (≠2^n)
- ieșirile trebuie decodate



CLK	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1



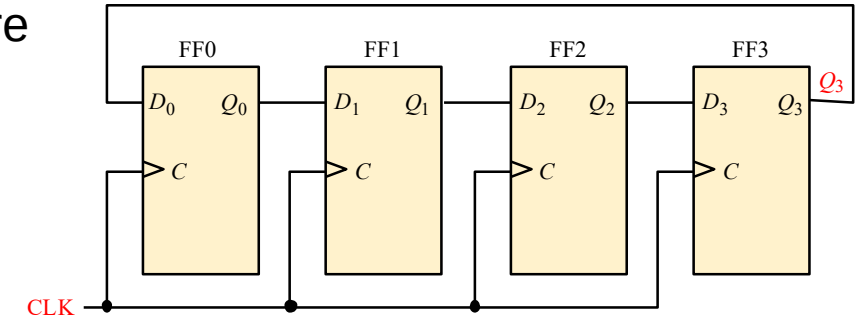
Forme de undă numărător Johnson pe 4 biți [2]

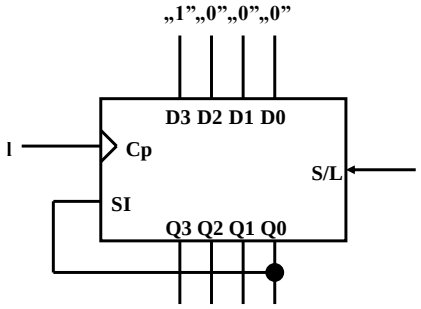
Numărătoare în inel

Numărătoare 1 din N

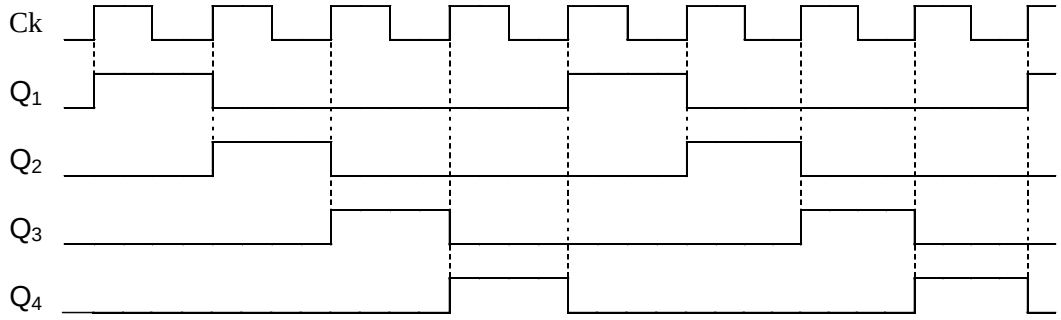
Ultima ieșire se conectează la prima intrare

Q_3	Q_2	Q_1	Q_0	Cicluri de clock
1	0	0	0	Stare inițială
0	1	0	0	1. clock
0	0	1	0	2. clock
0	0	0	1	3. clock
1	0	0	0	4. clock
⋮	⋮	⋮	⋮	



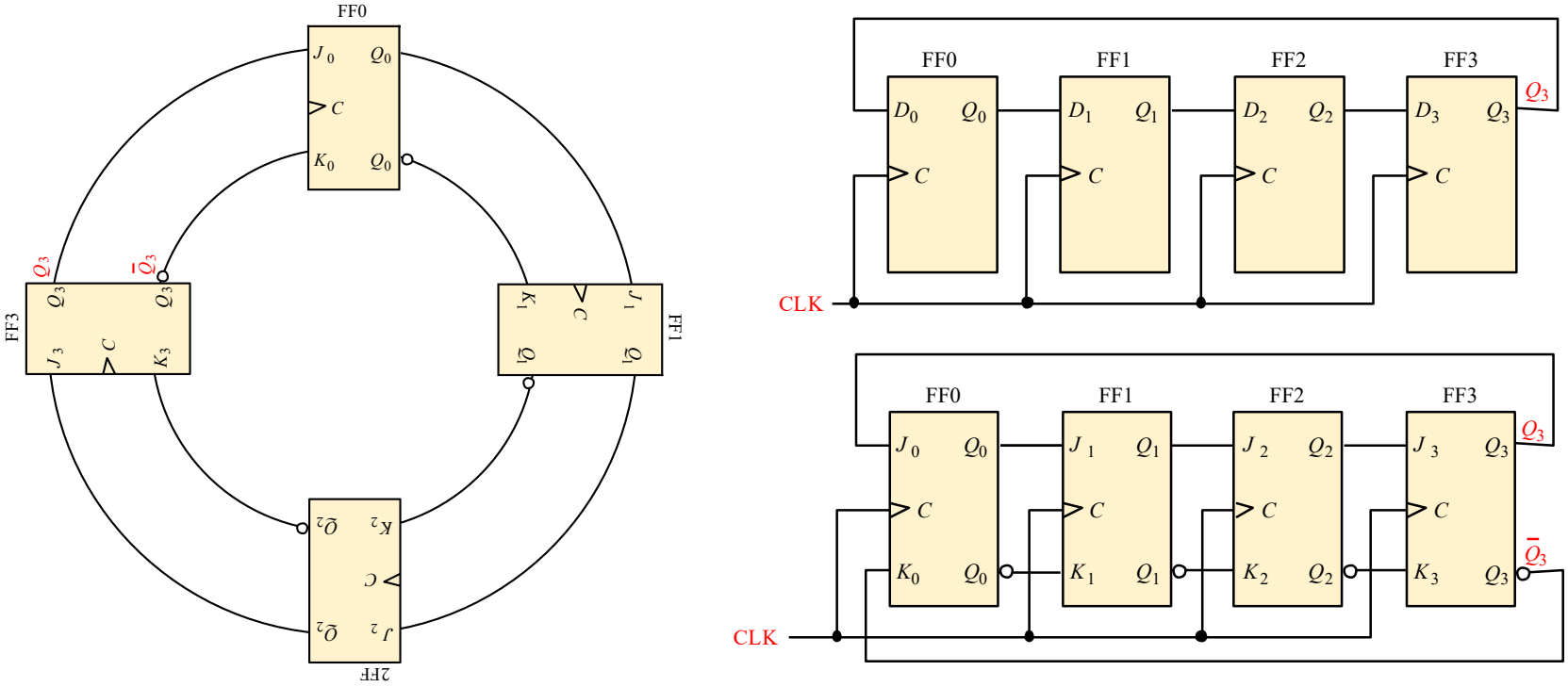


- n bistabile numără până la n (≠2^n)
- ieșire nu trebuie decodată
- numărarea este sincronă
- viteza de numărare mai mare



Numărătoare în inel

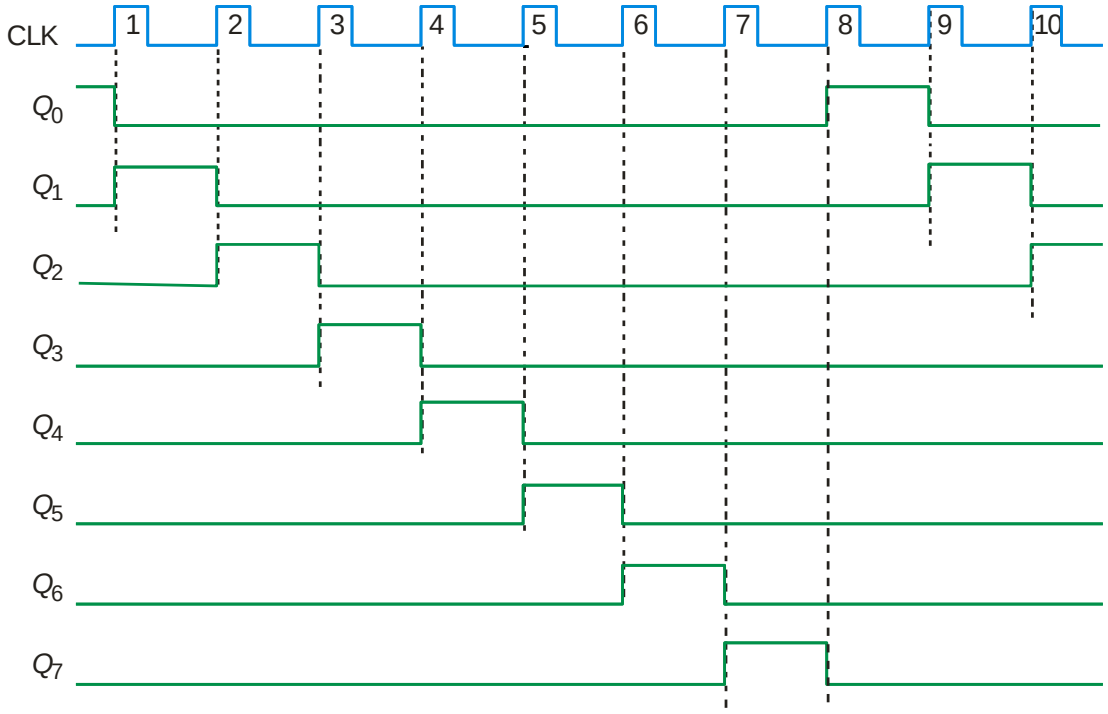
Numărătorul în inel se poate realiza cu bistabile de tip D sau JK



Numărător în inel [2]

Numărătoare în inel

Un tipar obișnuit pentru un numărător în inel este un singur 1 sau un singur 0. Formele de undă prezentate se referă la un numărător în inel de 8 biți încărcat cu un 1.



Forme de undă numărător în inel pe 8 biți [2]

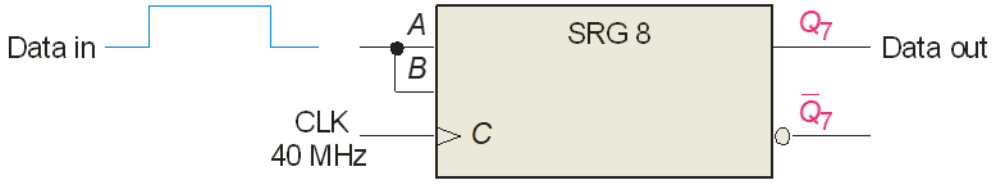
Aplicații ale registrelor de deplasare

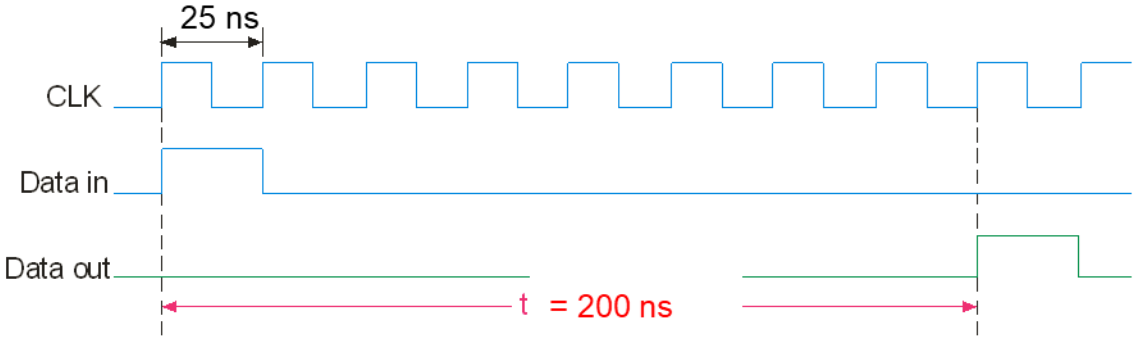
Întârzierea unui semnal digital cu o valoare stabilită.

Exemplu: Un registru de deplasare serial in/serial out de 8 biți are un semnal de clock de 40 MHz. Care este întârzierea totală prin registru?

Soluție: Întârzierea pentru fiecare clock este de 1/40 MHz = 25 ns

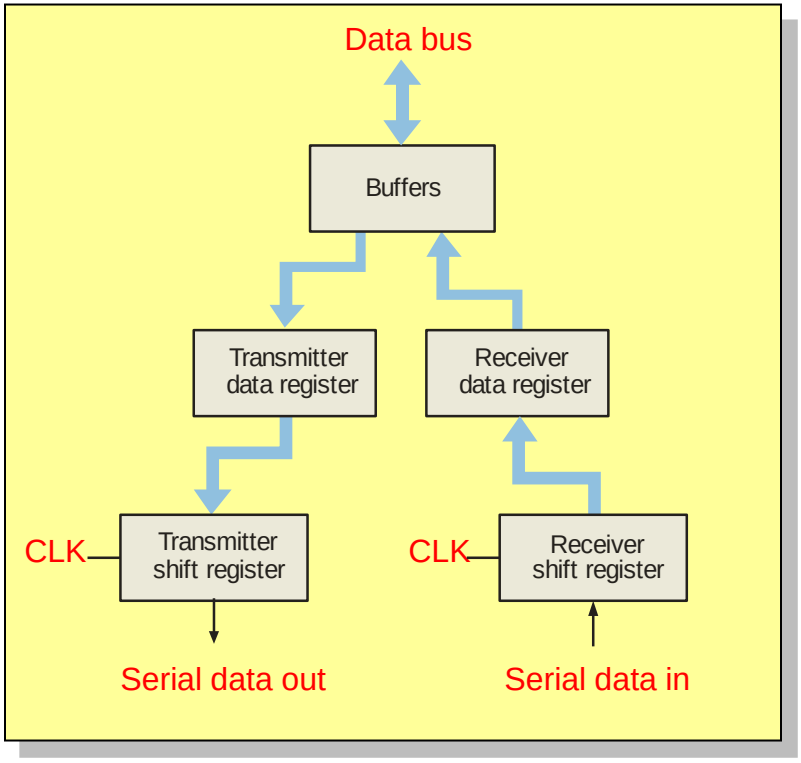
Întârzierea totală este de 8 x 25 ns = 200 ns



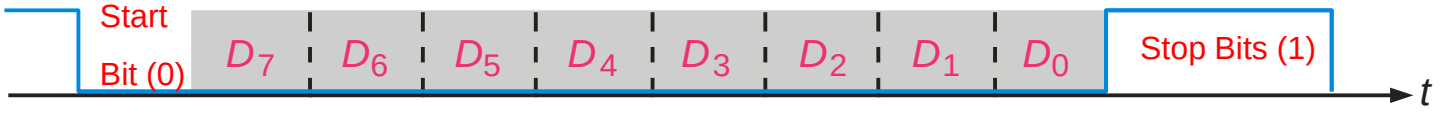


Aplicații ale registrelor de deplasare

- A UART (Universal Asynchronous Receiver Transmitter) este un convertor serial-paralel și un convertor paralel-serial.
- UART-urile sunt folosite în sistemele care trebuie să comunice între ele. Datele paralele sunt convertite in date seriale asincrone și transmise.

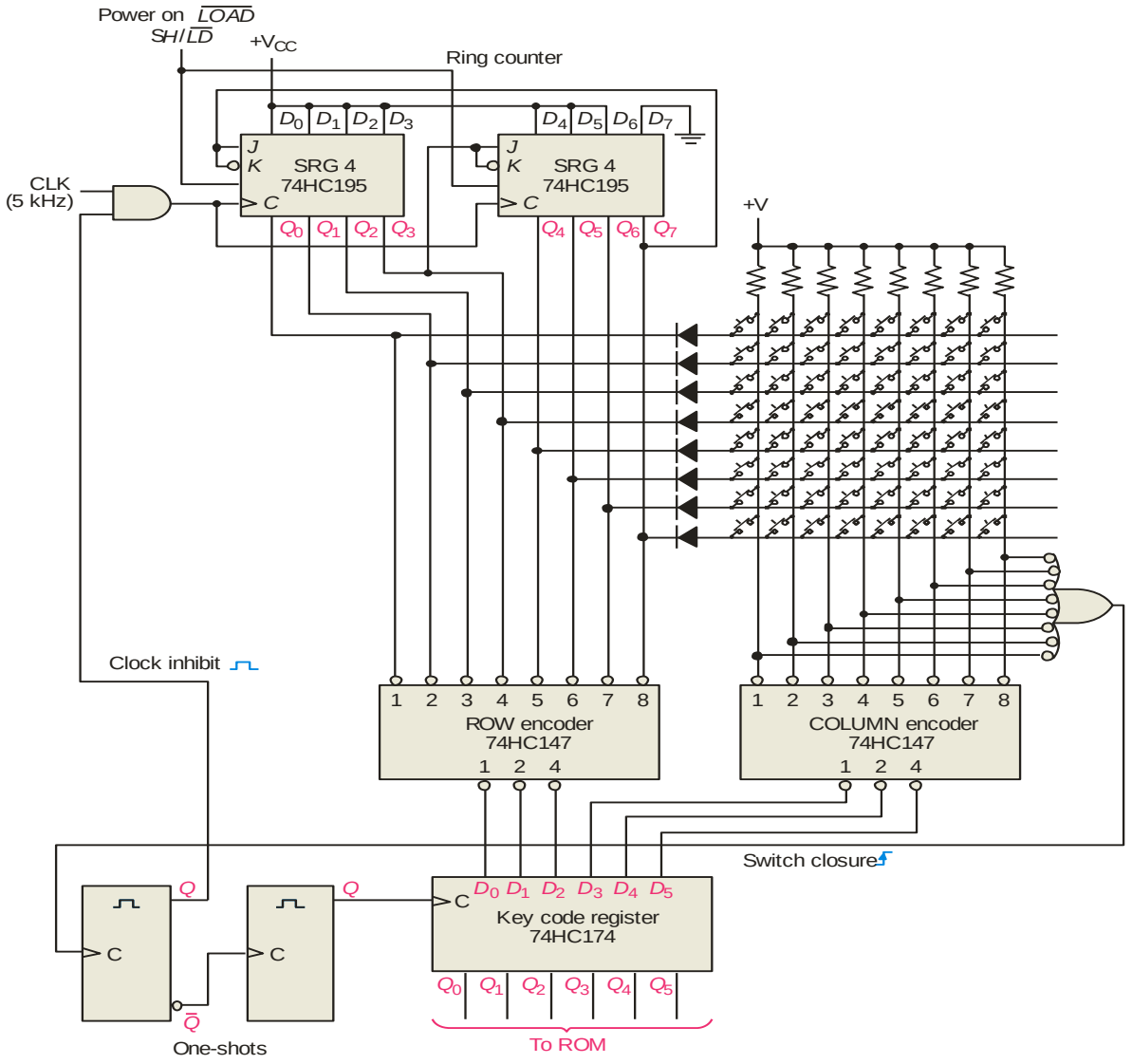


Formatul serial al datelor este:



Aplicații ale registrelor de deplasare

Codificator tastatură [2]



Introducere în limbajul Verilog

Limbaje HDL

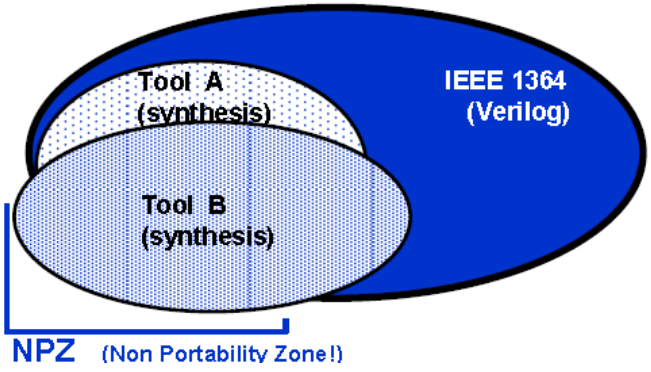
- Standarde HDL (hardware description language)
  - Verilog
    - 1984: Gateway Design Automation Inc.
    - 1990: Cadence -> Open Verilog International
    - 1995: IEEE standardization
    - 2001: Verilog 2001
  - VHDL
    - 1983-85: IBM, Texas Instruments
    - 1987: IEEE standardization
    - 1994: VHDL-1993

Alte limbaje HDL

- Dezvoltarea în limbaj HDL necesită timp mai îndelungat în comparație cu alte limbaje
- Există mulți programatori C/C++ knowledge, mult mai puțini proiectanți HDL
- Limbaje de descriere hardware de nivel înalt
  - Celoxica Handel-C: bazat pe ANSI-C cu caracteristici speciale
  - SystemC: standardizat, bazate pe limbajul C++
  - Mentor Catapult-C: poate genera hardware din cod standard C
- Simulare mai rapidă
- HW/SW co-design

Scopul limbajelor HDL

- Modelarea comportamentului circuitelor hardware
  - O mare parte a limbajului poate fi folosită doar pentru simulare și nu pentru generare de hardware (sinteză)
  - Partea sintetizabilă depinde de programul de sintetizare
- Înlocuiește proiectarea grafică bazată pe schematic (care este foarte consumatoare de timp)
- Descriere RTL (Register Transfer Level)
  - Permite sinteza automată a circuitului hardware
  - Crește productivitatea

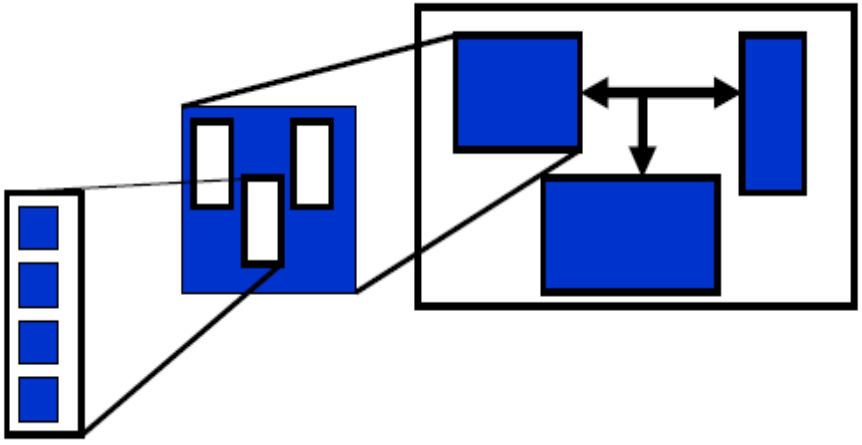


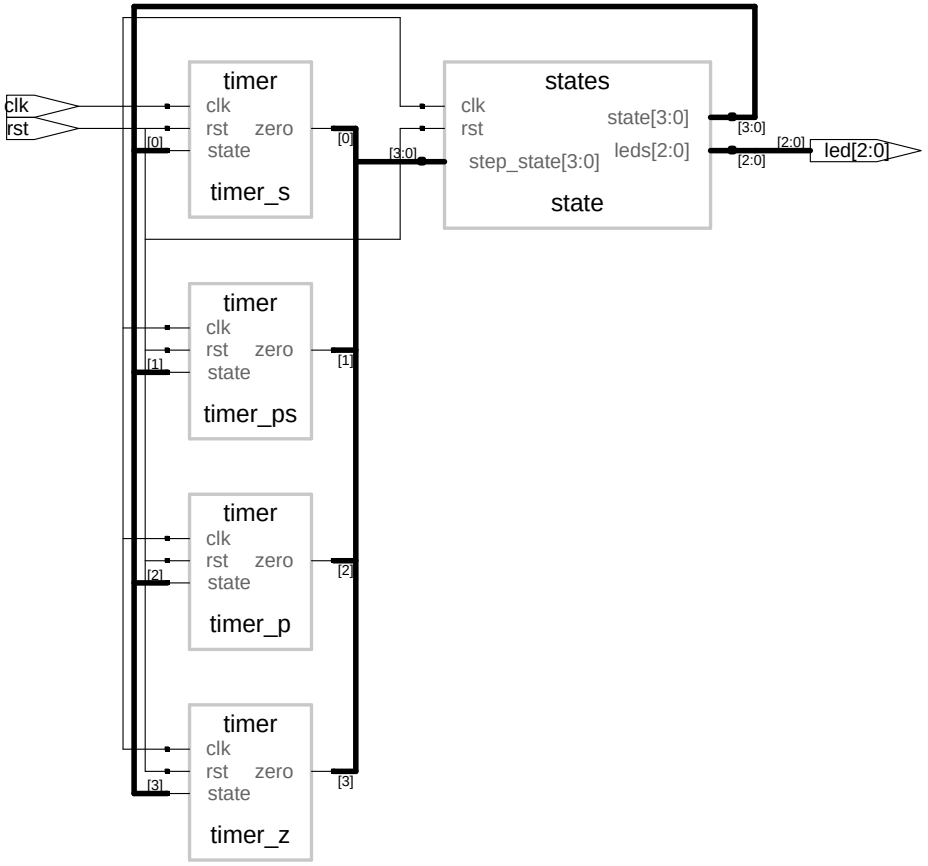
Limbaje HDL

- Sunt limbaje modulare
- HDL modul
  - Definirea porturilor de intrare – ieșire
  - Funcțiile logice între intrări și ieșiri
- Spre diferență de limbajele de programare software, limbajele HDL NU reprezintă coduri executate secvențial
- Ele descriu o **funcționare paralelă**

Module

- Reprezintă blocuri din care se realizează sistemele complexe ierarhice
- Descriere ierarhică, partiționarea funcțiilor
- Proiectare
  - Top-down
  - Down-to-top

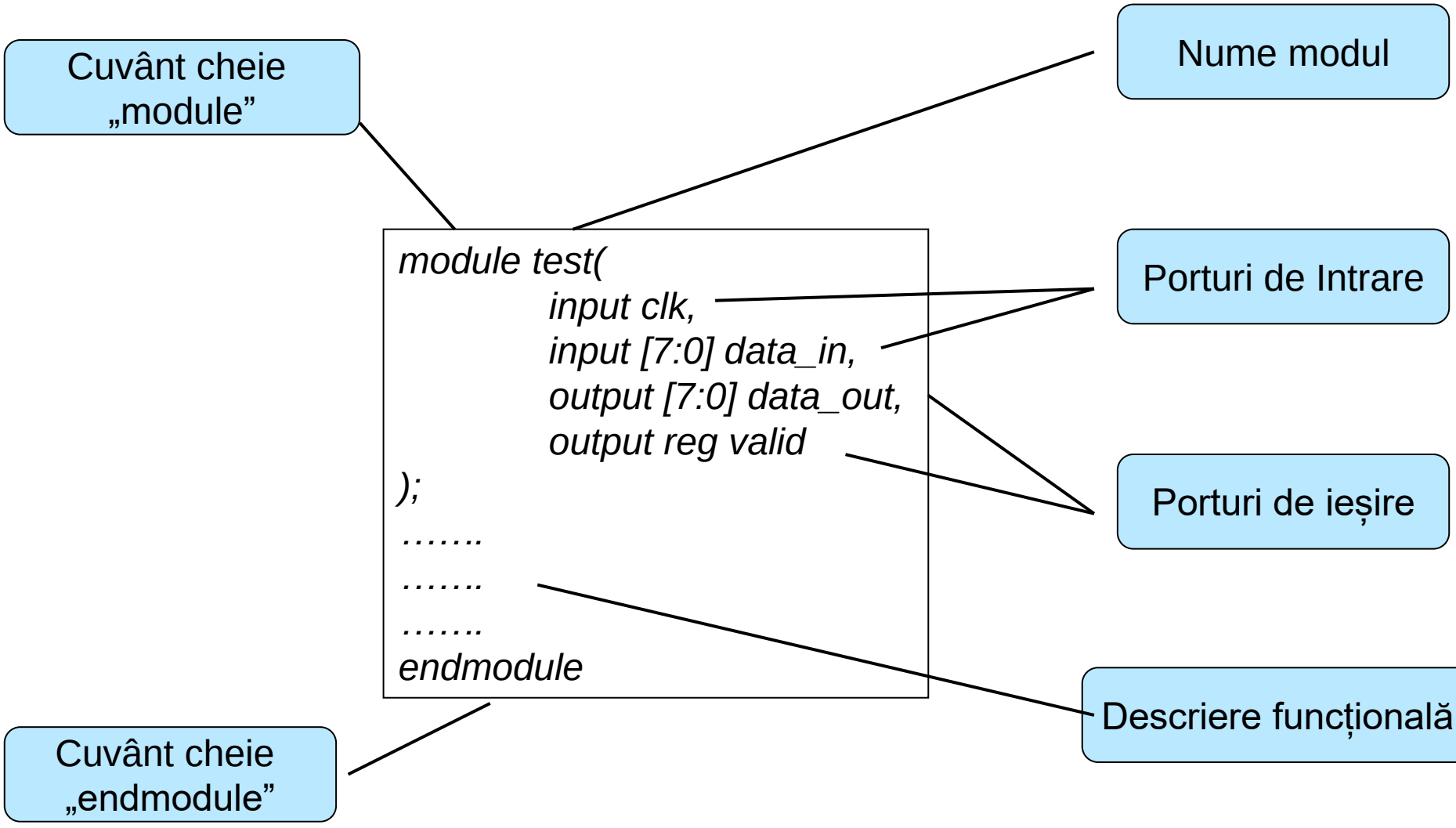




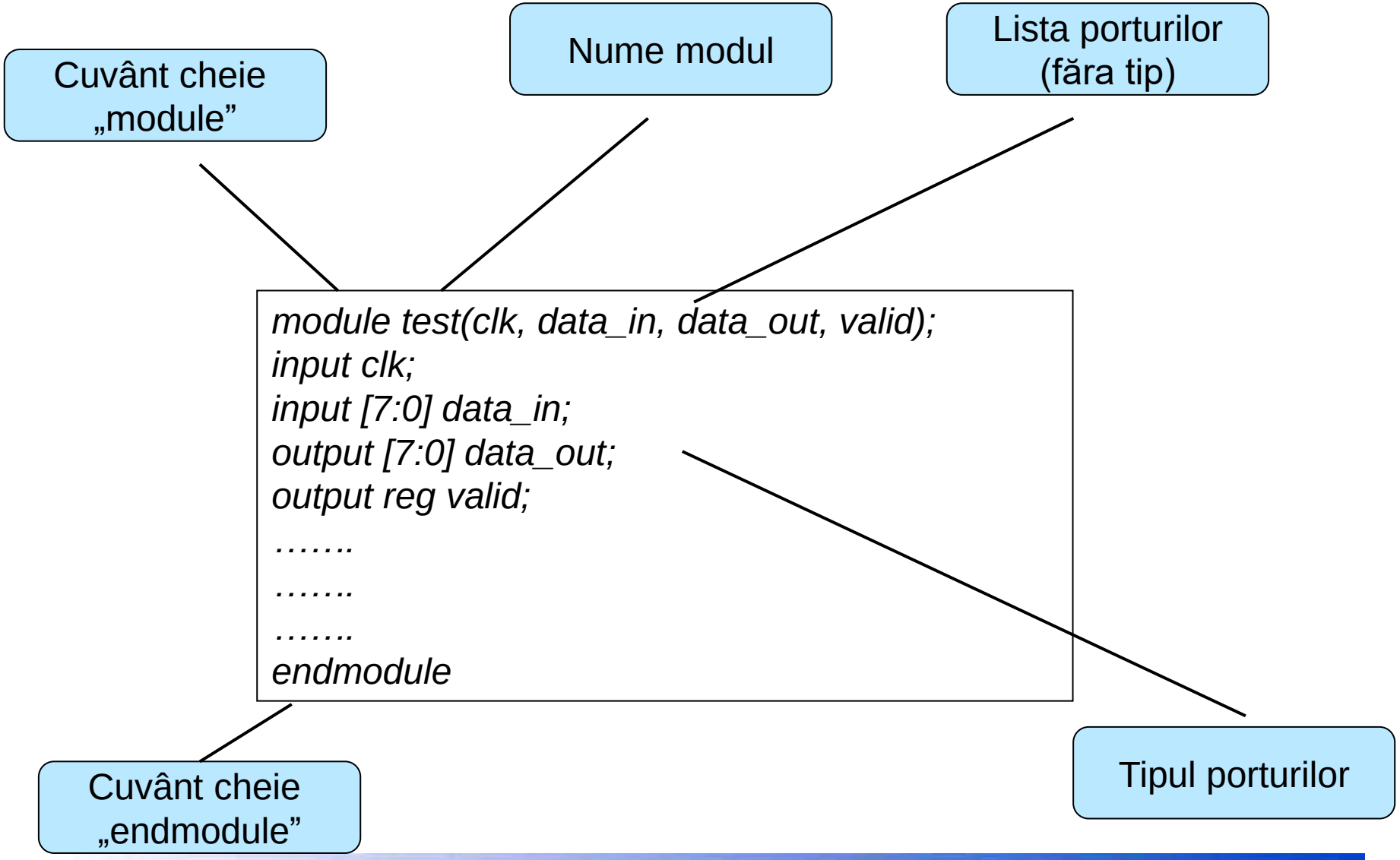
Sintaxa Verilog

- Comentarii (ca în C)
  - // o linie
  - /* */ mai multe linii
- Constante
  - <nr. biți><‘baza><valoarea>
    - 5’b00100: 00100 valoarea zecimală 4, exprimat pe 5 biți
    - 8’h4e: 01001110 valoarea zecimală78, exprimat pe 8 biți
    - 4’bZ: ZZZZ Z= înaltă impedanță (deconectat)

Verilog: descriere modul (2001)



Verilog: descriere modul (1995)



Operații la nivel de bit (Bit operations)

- ~, &, |, ^, ~^ (negate, and, or, xor, xnor)
- Operații Bitwise pe vectori:
  - 4’b1101 & 4’b0110 = 4’b0100
- Dacă numărul de biți ai operanzilor nu sunt egali, cel mai mic este extins cu zerouri
  - 2’b11 & 4’b1101 = 4’b0001
- Ooperatori logici: !, &&, ||

Operatori tip Bit reduction

- Operează pe toții biții vectorului, rezultatul este pe un singur bit

- &, ~&, |, ~|, ^, ~^ (and, nand, or, nor, xor, xnor)

  - &4’b1101 = 1’b0
  - |4’b1101 = 1’b1

  - Se utilizează de exemplu în operațiile de verificarea parității.

Comparații

- La fel ca în limbajul C
- Egal, diferit
  - ==, !=
  - ===: egalitate când se consideră si valorile „Z”, „X”
  - !==: diferit când se consideră si valorile „Z”, „X”
- Comparații
  - <, >, <=, >=

Operații aritmetice

- La fel ca în limbajul C
- Operatori: +, -, *, /, %
  - Nu toate sunt sintetizabile
    - De exemplu împărțirea este sintetizabilă numai când împărțitorul este un număr putere a lui 2
  - Numerele negative sunt reprezentate în complement față de doi

Alte operații

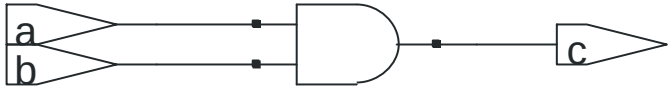
- Concatenare: {}
  Ex.:
  - {4’b0101, 4’b1110} = 8’b01011110
- Shift:
  - <<, >>
- Selectare biți
  - Partea selectată trebuie să fie constantă
  - data[5:3]

Tipuri de date

- wire
  - Se comportă ca o fir conductor adevărat (logică combinațională)
  - Declararea unui wire pe 8 biți: wire [7:0] data;
- reg
  - După sintetizare se poate materializa într-un
    - Wire
    - Latch
    - Flip-flop
  - Ex.: reg [7:0] data;

Assign

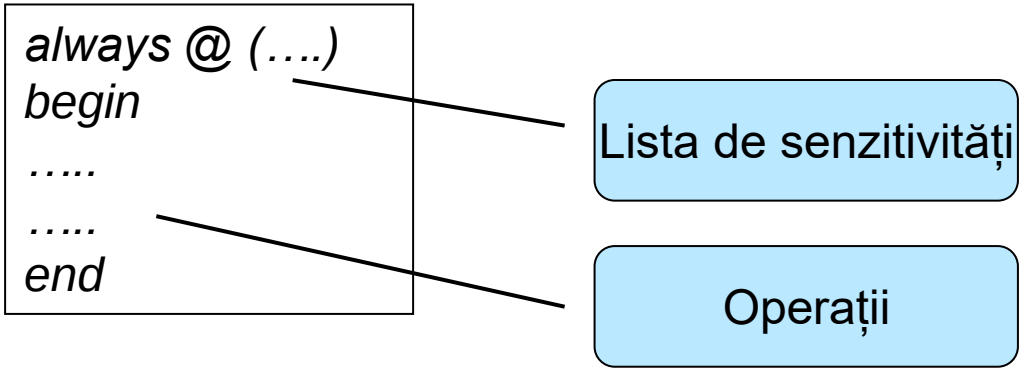
- Assign se poate utiliza doar pentru atribuirea de valori pentru date de tip wire
- Este o atribuire continuă
  - Operandul din partea stângă primește în mod continuu valori noi
- Ex.
  - assign c = a & b;



- Unei variabile i se poate atribui valoare doar folosind o singură operație assign
- Mai multe operații assign operează în paralel
- Pot fi folosite pentru a descrie logică combinațională

Always

- Sintaxa:



- O variabilă trebuie scrisă doar într-un singur bloc de tip always
- Lista de senzitivități nu poate să conțină variabile de ieșire ale blocului always
- Assign nu poate fi utilizat într-un bloc always
- Mai multe blocuri always sunt executate în paralel

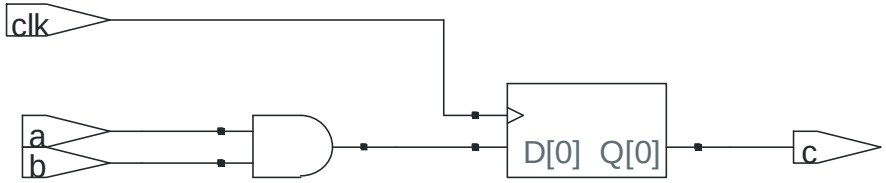
Atribuirile din cadrul blocului Always

- Blocking: =
  - Blochează execuția operației următoare pe timpul execuției operației curente -> execuție secvențială (evitați utilizarea ei dacă nu este neapărat necesar)
- Nonblocking: <=
  - Atribuirile Nonblocking sunt executate în paralel -> operații specifice funcționării hardware

Always – Flip Flop

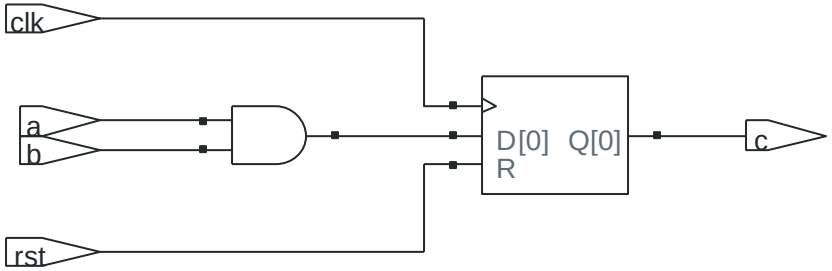
- Flip Flop: elemente de stocare cu acționare pe front

```
always @ (posedge clk)
        c <= a & b;
```



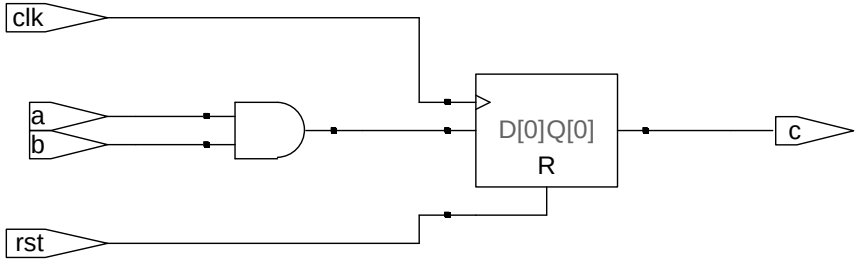
- Reset sincron

```
always @ (posedge clk)
if (rst)
        c <= 1'b0;
else
        c <= a & b;
```



- Reset asincron

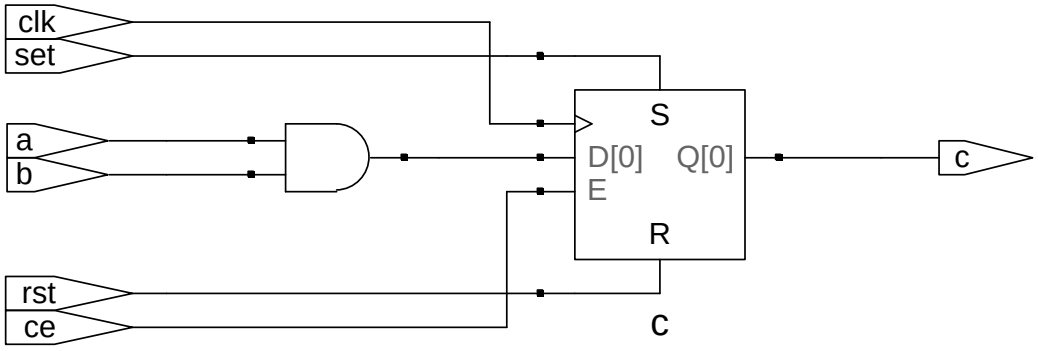
```
always @ (posedge clk, posedge rst)
if (rst)
        c <= 1'b0;
else
        c <= a & b;
```



Always – Flip Flop

- In FPGA-urile Xilinx
  - Reset și set pot fi sincrone sau asincrone
  - În cazul semnalelor sincrone prioritatea este:
    - reset, set, ce
- Exemplu asincron:

```
always @ (posedge clk, posedge rst, posedge set)
if (rst)
        c <= 1'b0;
else if (set)
        c <= 1'b1;
else if (ce)
        c <= a & b;
```

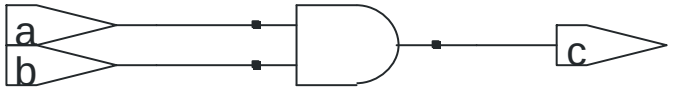


Always – logică combinațională

- Rezultatul este calculat continuu – dacă oricare dintre intrări se modifică, ieșirea se modifică imediat

```
always @ (a, b)
   c <= a & b;
```

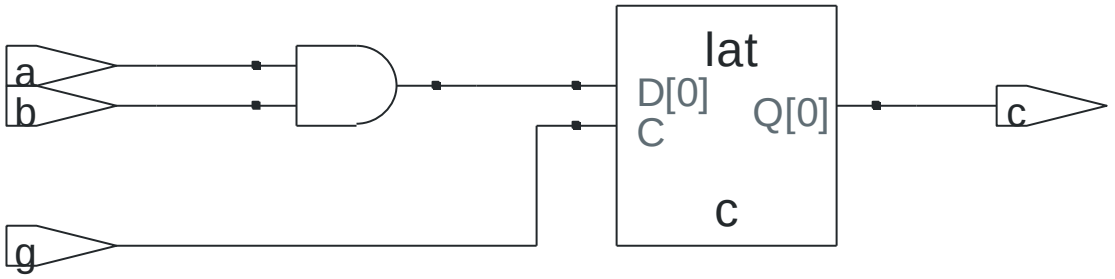
```
always @ (*)
   c <= a & b;
```



Always – latch

- Latch-ul: este un element de stocare senzitiv pe nivel
  - Cât timp semnalul „gate” este ‘1’, intarea (D) the input este înscrisă în latch
  - Dacă „gate” este ‘0’, valoarea anterioară înscrisă este stocată

```
always @ (*)
If (g)
   c <= a & b;
```

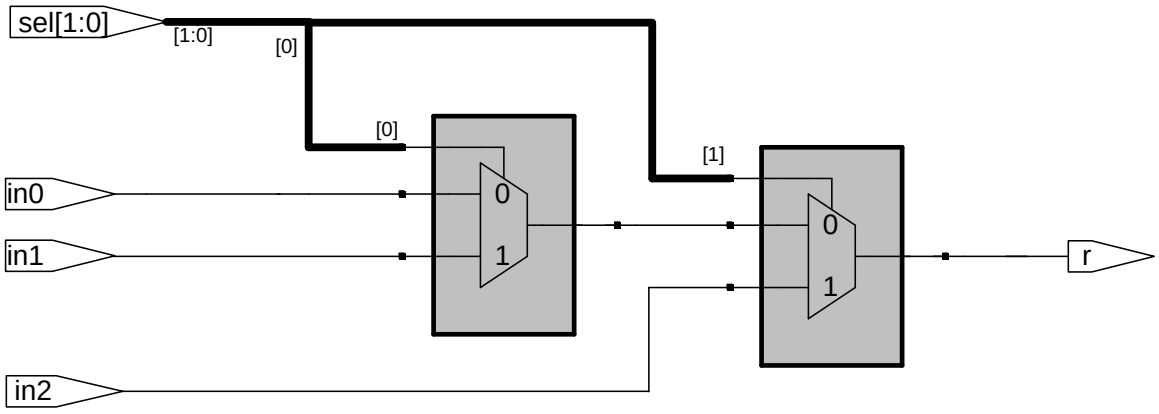


Always – construcții corecte if/case

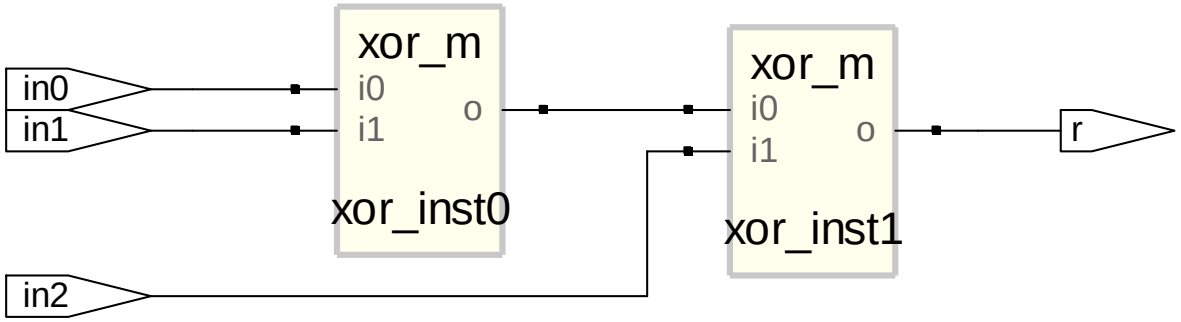
- Sintaxa corectă pentru instrucțiunile if și case: trebuie să se încheie cu default în cazul case și else pentru instrucțiunea if.

```
always @ (*)
case (sel)
   2'b00: r <= in0;
   2'b01: r <= in1;
   2'b10: r <= in2;
   default: r <= 'bx;
endcase
```

```
always @ (*)
if (sel==0)
   r <= in0;
else if (sel==1)
   r <= in1;
else
   r <= in2;
```



Descriere structurală



- Conectarea modulelor pe baza numelor porturilor

```
module top_level (input in0, in1, in2, output r);
wire xor0;
xor_m xor_inst0(.i0(in0), .i1(in1), .o(xor0));
xor_m xor_inst1(.i0(xor0), .i1(in2), .o(r));
endmodule
```

- Conectarea modulelor pe baza poziției porturilor

```
module top_level (input in0, in1, in2, output r);
wire xor0;
xor_m xor_inst0(in0, in1, xor0);
xor_m xor_inst1(xor0, in2, r);
endmodule
```

Exemple – MUX (1.)

- 2:1 multiplexer

```
module mux_21 (input in0, in1, sel, output r);
assign r = (sel==1'b1) ? in1 : in0;
endmodule
```

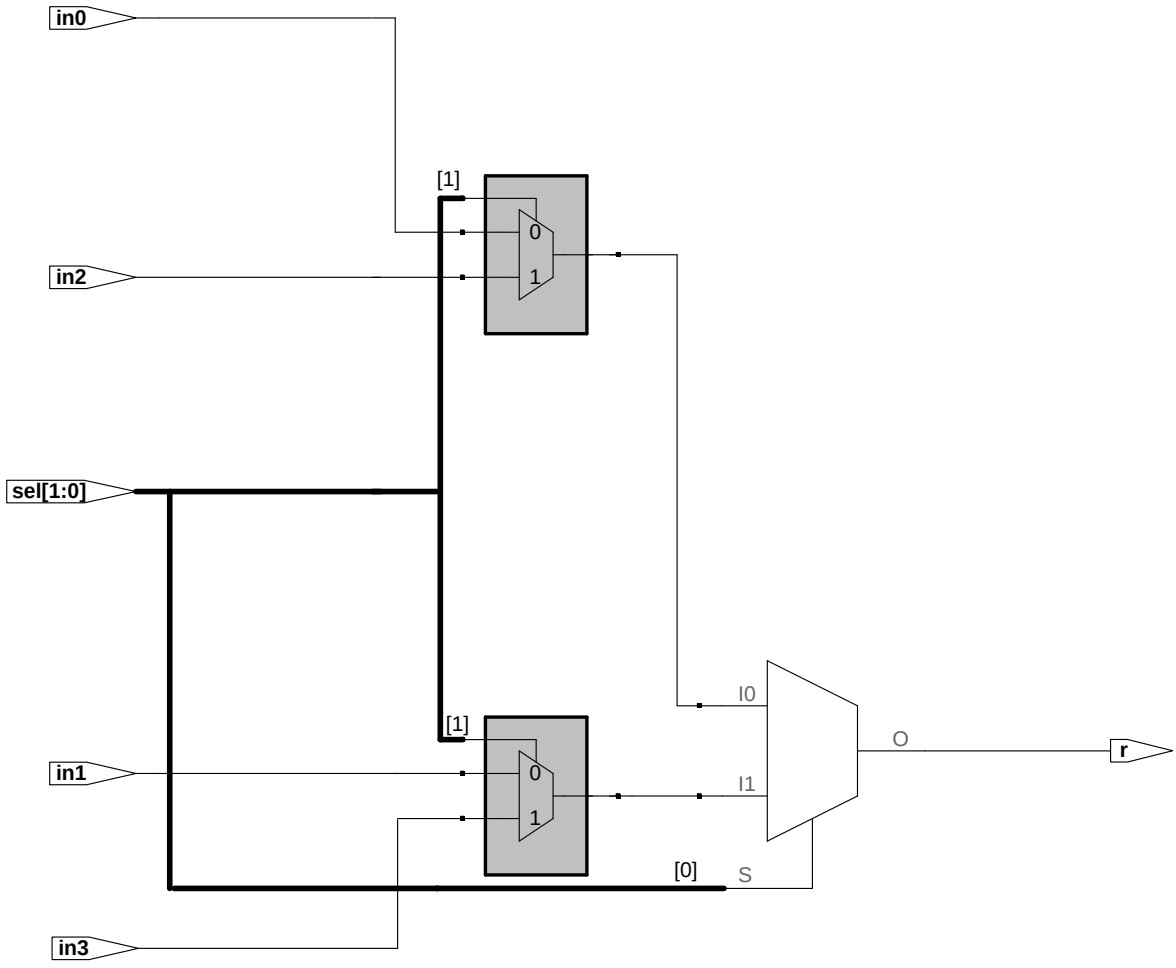
```
module mux_21 (input in0, in1, sel, output reg r);
always @ (*)
if (sel==1'b1) r <= in1;
else           r <= in0;
endmodule
```

```
module mux_21 (input in0, in1, sel, output reg r);
always @ (*)
case(sel)
    1'b0:      r <= in0;
    1'b1:      r <= in1;
endmodule
```

Exemplu – MUX (2.)

- 4:1 multiplexer

```
module mux_41 (input in0, in1, in2, in3, input [1:0] sel, output reg r);
always @ (*)
case(sel)
    2'b00: r <= in0;
    2'b01: r <= in1;
    2'b10: r <= in2;
    2'b11: r <= in3;
endcase
endmodule
```



Exemple – sumator complet pe 1 bit

```
module add1_full (input a, b, cin, output cout, s);
xor3_m xor(.i0(a), .i1(b), .i2(cin), .o(s));
wire a0, a1, a2;
and2_m and0(.i0(a), .i1(b), .o(a0));
and2_m and1(.i0(a), .i1(cin), .o(a1));
and2_m and2(.i0(b), .i1(cin), .o(a2));
or3_m or(.i0(a0), .i1(a1), .i2(a2) , .o(cout))
endmodule
```

```
module add1_full (input a, b, cin, output cout, s);
    assign s = a ^ b ^ cin;
    assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

```
module add1_full (input a, b, cin, output cout, s);
    assign {cout, s} = a + b + cin;
endmodule
```

Exemple – sumator pe 4 biți, structural

```
module add4 (input [3:0] a, b, output [4:0] s);
wire [3:0] cout;
add1_full add0(.a(a[0]), .b(b[0]), .cin(1'b0),    .cout(cout[0]), .s(s[0]));
add1_full add1(.a(a[1]), .b(b[1]), .cin(cout[0]), .cout(cout[1]), .s(s[1]));
add1_full add2(.a(a[2]), .b(b[2]), .cin(cout[1]), .cout(cout[2]), .s(s[2]));
add1_full add3(.a(a[3]), .b(b[3]), .cin(cout[2]), .cout(s[4]),    .s(s[3]));
endmodule
```

```
module add4 (input [3:0] a, b, input cin, output cout, output [3:0] sum);
assign {cout, sum} = a + b + cin;
endmodule
```

Exemplu –registru de deplasare

- Registru de deplasare pe 16 biți (pentru introducerea unei întârzieri)

```
module shr (input clk, sh, din, output dout);

reg [15:0] shr;
always @ (posedge clk)
if (sh)
     shr <= {shr[14:0], din};

assign dout = shr[15];

endmodule
```

Exemplu – Numărător

- Numărător binar cu reset sincron, clock enable, load și intrare de direcție

```
module m_cntr (input         clk, rst, ce, load, dir,
                input  [7:0] din,
               output [7:0] dout);

reg [7:0] cntr_reg;
always @ (posedge clk)
if (rst)
      cntr_reg <= 0;
else if (ce)
      if (load)
            cntr_reg <= din;
      else if (dir)
            cntr_reg <= cntr_reg – 1;
      else
            cntr_reg <= cntr_reg + 1;

assign dout = cntr_reg;

endmodule
```

Exemplu – Numărător secunde

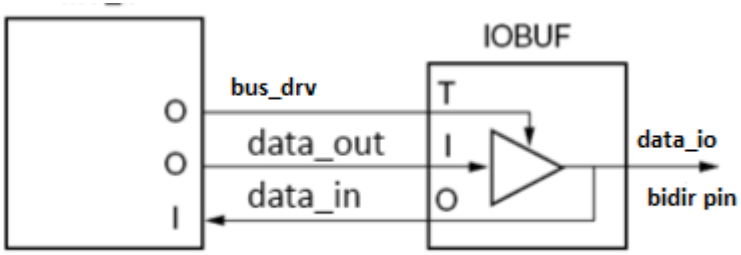
- Frecvență de clock 50 MHz, 1 sec = 50 000 000 impulsuri de clock

```
module sec (input clk, rst, output [6:0] dout);

reg [25:0] clk_div;
wire tc;
always @ (posedge clk)
If (rst)
        clk_div <= 0;
else
        if (tc)
                clk_div <= 0;
        else
                clk_div <= clk_div + 1;

assign tc = (clk_div == 49999999);

reg [6:0] sec_cntr;
always @ (posedge clk)
If (rst)
        sec_cntr <= 0;
else if (tc)
        if (sec_cntr==59)
                sec_cntr <= 0;
        else
                sec_cntr <= sec_cntr + 1;

assign dout = sec_cntr;

endmodule
```

Linii Tri-state

- Se folosesc în cazul liniilor bidirecționale
  - De exemplu magistrala de date externă a memoriilor

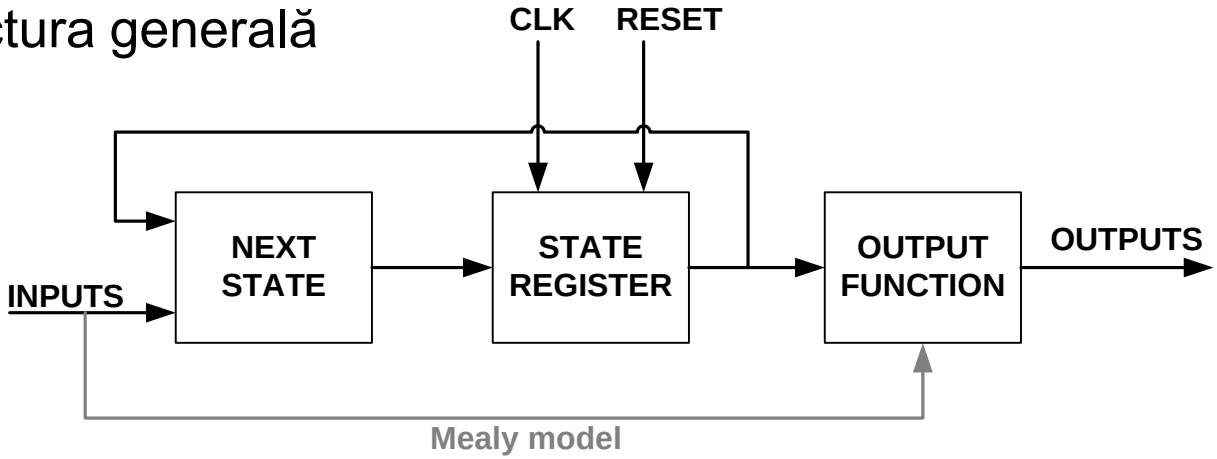
```
module tri_state (input clk, inout [7:0] data_io);

wire [7:0] data_in, data_out;
wire bus_drv;

assign data_in = data_io;
assign data_io = (bus_drv) ? data_out : 8'bz;

endmodule
```



- Semnalul de autorizare a ieșirii (bus_drv) condiționează punerea datelor pe magistrala de ieșire. Atenție la generarea lui!

Automate cu stări finite (FSM)

- FSM – Finite State Machine sunt folosite pentru a crea structuri de control complexe
- Structura generală

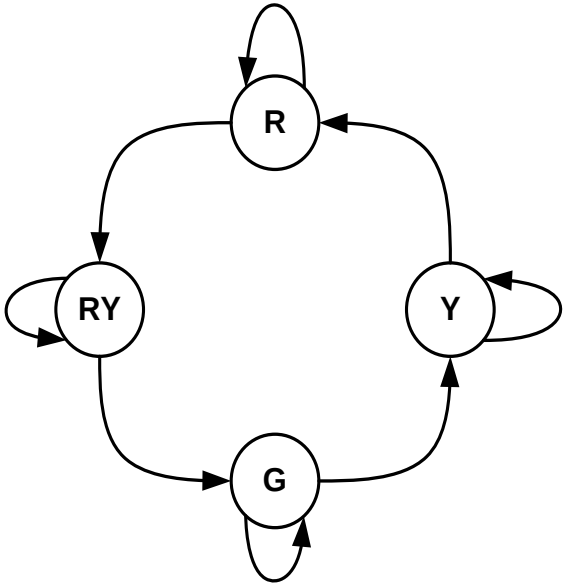


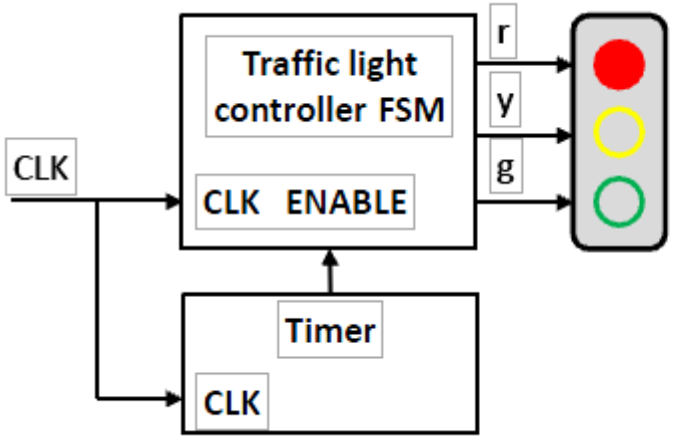
- Registru de stare (State register): pentru memorarea stării curente
- Starea viitoare (Next state): CLC pentru determinarea stării viitoare
- Funcția de ieșire (Output function): CLC pentru determinarea valorilor ieșirilor
  - Moore: pe baza registrului de stare
  - Mealy: pe baza registrului de stare și a intrărilor curente

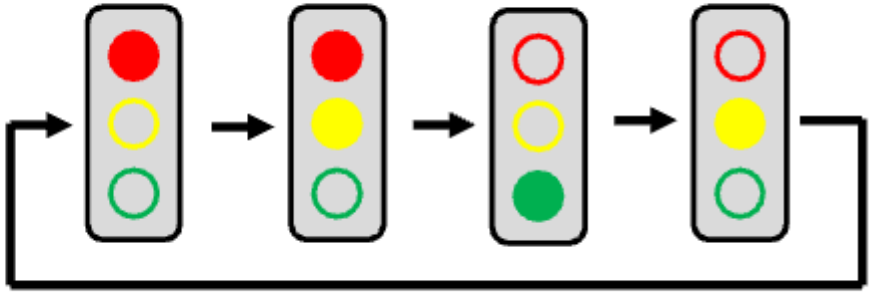
Exemplu FSM

Semafor (simplu)

- Stări: roșu, roșu-galben, verde, galben
- Intrări: temporizatoare pentru diferitele stări
- Ieșiri: starea







Exemplu FSM – Verilog (1)

```
module light(
        input clk, rst,
        output reg [2:0] led);

parameter RED        = 2'b00;
parameter RY         = 2'b01;
parameter GREEN    = 2'b10;
parameter YELLOW  = 2'b11;

reg [15:0] timer;
reg [1:0] state_reg;
reg [1:0] next_state;

always @ (posedge clk)
if (rst)
        state_reg <= RED;
else
        state_reg <= next_state;
```

```
always @ (*)
case(state_reg)
        RED: begin
          if (timer == 0)
            next_state <= RY;
          else
            next_state <= R;
          end
        RY: begin
          if (timer == 0)
            next_state <= GREEN;
          else
            next_state <= RY;
          end
        YELLOW: begin
          if (timer == 0)
            next_state <= RED;
          else
            next_state <= YELLOW;
          end
        GREEN: begin
          if (timer == 0)
            next_state <= YELLOW;
          else
            next_state <= GREEN;
          end
        default:
          next_state <= 3'bxxx;
endcase
```

Exemplu FSM – Verilog (2)

```
always @ (posedge clk)
case(state_reg)
        RED: begin
          if (timer == 0)
            timer <= 500;       //next_state <= RY;
          else
            timer <= timer - 1;
          end
        RY: begin
          if (timer == 0)
            timer <= 4000;      //next_state <= GREEN;
          else
            timer <= timer - 1;
          end
        YELLOW: begin
          if (timer == 0)
            timer <= 4500;      //next_state <= RED;
          else
            timer <= timer - 1;
          end
        GREEN: begin
          if (timer == 0)
            timer <= 500;       //next_state <= YELLOW;
          else
            timer <= timer - 1;
          end
endcase
```

- Încarcă noua valoare când se schimbă starea
- Numărare în jos
- ==0: schimbare stare

```
always @ (*)
case (state_reg)
        RY   :          led <= 3'b110;
        RED:            led <= 3'b100;
        YELLOW:         led <= 3'b010;
        GREEN:          led <= 3'b001;
        default:        led <= 3’b100;
endcase

endmodule
```

Module parametrizabile

- Sumator parametrizabil

```
module add(a, b, s);
    parameter width = 8;
    input [width-1:0] a, b;
    output [width:0] s;

assign s =  a + b;

endmodule
```

- Instanțierea modulului parametrizabil

```
wire [15:0] op0, op1;
wire [16:0] res;

add #(
    .width(16)
)
add_16(
    .a(op0),
    .b(op1),
    .s(res)
);
```

Simulare

- Creare testbench: se face cu în fișier Verilog Test Fixture
    - Semnalele de intrare se generează în limbaj Verilog
- Simulatoare
    - ISE Simulator
    - Modelsim (MXE)

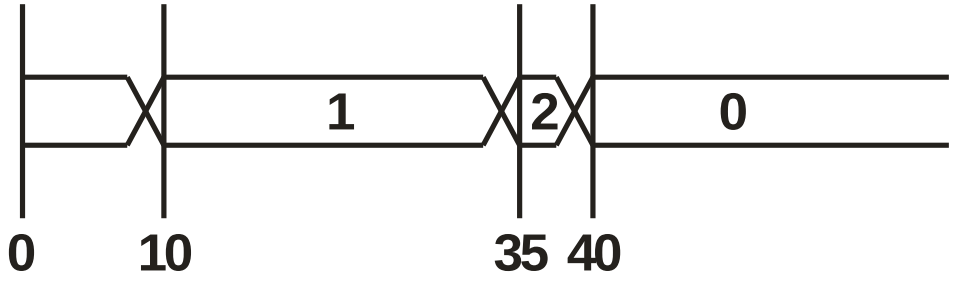
Verilog Test Fixture

- Test Fixture
  - Test Fixture este un modul Verilog
  - Modulul de testat UUT (Unit under test) este un submodul al modulului test fixture
  - Se pot folosi toate construcțiile sintactice din limbajul Verilog
  - Există construcții nesintetizabile
- Unitatea de timp
  - ‘timescale 1ns/1ps
    - Unitatea de timp este 1 ns
    - Rezoluția simulării este de 1 ps

Test Fixture - initial

- Blocul „initial”
  - Execuția începe la momentul de timp „0”
  - Se execută o singură dată și are rol de inițializare a semnalelor
  - Dacă există mai multe blocurile „initial” ele sunt executate în paralel, și în paralel cu blocurile always și atribuirile assign
- În cadrul unui bloc „initial” întârzierile se cumulează

```
initial
begin
    a <= 0;
    #10 a <= 1;
    #25 a <= 2;
    #5   a <= 0;
end
```



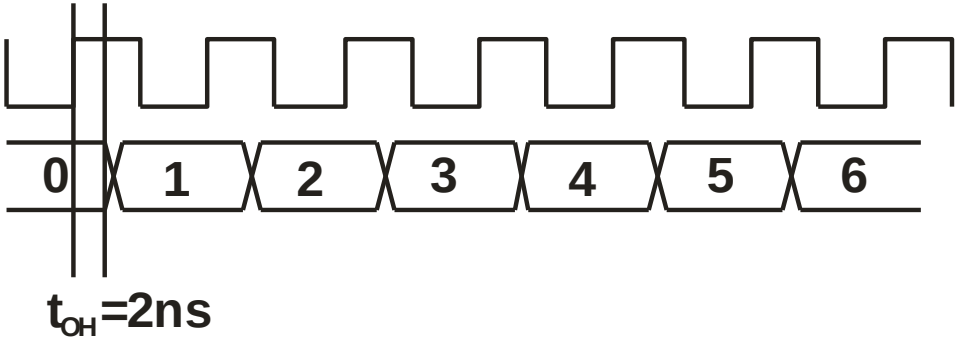
Test Fixture - always

- Generarea semnalului de clock

```
initial
    clk <= 1;

always #5
    clk <= ~clk;
```

- Descrierea timpului de propagare

```
initial cntr <= 0;
always @ (posedge clk)
    #2 cntr <= cntr + 1;
```



Bibliografie selectivă

[1] Didier Teifreto, Cours d'architecture des ordinateurs, Université Franche Compté

[2] Thomas L. Floyd, Digital fundamentals - editia 10, Prentice Hall, 2009

[3] Ștefan Oniga, Circuite digitale, Editura Risoprint Cluj Napoca, 2002

[4] Dan Nicula - Electronica digitala. Carte de invățătură - Editura Universității Transilvania din Brașov, 2012

[5] John F. Wakerly - Circuite digitale. Principiile și practicile folosite în proiectare, Editura Teora, București, 2002. (Original English language title: Digital Design: Principles and Practices, Third Edition, Prentice Hall, 2000)

[6] Gheorghe Ștefan - Circuite și sisteme digitale, Editura Tehnica, București, 2000

[7] Gheorghe Toacșe, Dan Nicula - Electronică digitală. Dispozitive, circuite, proiectare, vol.1, Editura Tehnică, 2005

[8] Gheorghe Toacșe, Dan Nicula - Electronică digitală. Verilog HDL, vol. 2, Editura Tehnică, 2005

[9] Digital design exercise collection www.play-hookey.com/digital/

[10] Digital Works 3.04 program de simulare: http://www.mecanique.co.uk/shop/index.php?route=product/category&path=89

[11] Digital technics exercise collection http://www.inf.u-szeged.hu/projectdirs/digipeldatar/