

Octavian Creț

Lucia Văcariu

**Probleme de
Proiectare logică
- a sistemelor numerice -**

***Logic Design
Problems***

- for digital systems -

Octavian Creț

Lucia Văcariu

Probleme de Proiectare logică

- a sistemelor numerice -

*Logic Design
Problems*

- for digital systems -



U. T. PRESS
Cluj-Napoca, 2008

**Editura U.T.PRESS**

Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel./Fax: 0264 - 430408
e-mail: utpress@biblio.utcluj.ro

Director:

Prof.dr.ing. Traian Onet

Consilier științific:

Prof.dr.ing. Virgil Maier

Consilier editorial:

Ing. Călin D. Câmpean

Descrierea CIP a Bibliotecii Naționale a României**CRETĂ, OCTAVIAN**

Probleme de proiectare logică a sistemelor numerice = Logic design problems for digital systems / Octavian Creț, Lucia Văcariu. -
Cluj-Napoca : Editura U.T. Press, 2008

ISBN 978-973-662-412-4

I. Văcariu, Lucia

004.383.3

Copyright © 2008 Editura U.T.PRESS

Toate drepturile asupra versiunii în limba română aparțin Editurii U.T.PRESS.
Reproducerea integrală sau parțială a textului sau ilustrațiilor din această carte este posibilă numai cu acordul prealabil scris al editurii U.T.PRESS.
Tiparul executat la Atelierul de multiplicare al UTCN.

ISBN 978-973-662-412-4

Bun de tipar: 19.11.2008

Tiraj: 700 exemplare

Prefață

Abstract

Culegerea de probleme constituie un material de bază pentru pregătirea în domeniul proiectării și utilizării sistemelor numerice. Ea este destinată în primul rând studenților din secțiile de inginerie (studii de licență) care au în programa de studiu materii care asigură calificări pentru proiectarea dispozitivelor numerice. Lucrarea poate fi folosită și de către cadrele didactice, ca un ajutor la seminarii sau proiecte.

Materialul culegerii de probleme este structurat în 5 capitole, ultimul capitol având un caracter sintetic.

Capitolul 1 este focalizat pe principalele sisteme de numerație și coduri binare folosite în sistemele numerice. Capitolul 2 prezintă noțiunile fundamentale referitoare la algebra Booleană și funcții Booleene. Capitolele 3 și 4 tratează problemele specifice proiectării circuitelor logice combinaționale, respectiv secvențiale. Capitolul 5 conține doar probleme propuse, pentru rezolvarea cărora sunt necesare cunoștințe din toate celelalte capitole.

Fiecare capitol conține o parte teoretică introductivă, în care sunt prezentate conceptele de bază necesare, urmată de o secțiune de

The present Problems textbook constitutes a fundamental support for study in the field of the design and utilization of digital systems. It is primarily intended to help students in Engineering curricula (undergraduate), for disciplines that provide competences in the field of digital systems design. The textbook can also be used by teachers as an auxiliary for seminars or projects.

The book is structured on four chapters, plus a final chapter with a synthetic character.

Chapter 1 focuses on the main binary number systems and codes used in digital systems. Chapter 2 presents the basic concepts about Boolean algebra and Boolean functions. Chapters 3 and 4 treat problems that are specific to the design of combinational and sequential logic circuits, respectively. Chapter 5 contains only proposed problems; for solving these problems, it is necessary to apply knowledge from all the other chapters.

Each chapter contains a theoretical introductory part, in which the basic concepts are exposed, followed by a section of solved problems and by a

probleme rezolvate și de o ultimă parte în care sunt propuse probleme pentru muncă individuală.

Scopul prezentării acestor probleme îl constituie însușirea de către studenți a cunoștințelor referitoare la conceptele de bază din proiectarea logică a sistemelor numerice.

Sperăm ca această lucrare să vină în sprijinul pregătirii studenților, permîțându-le să aprofundeze noțiunile fundamentale studiate pe baza exemplelor prezentate aici *in extenso*. Lecturarea sa poate fi considerată, într-un anumit sens, similară participării la seminarii de sisteme numerice.

Autorii

final section in which a series of problems are proposed for individual homework.

The goal of these problems' presentation is to help students assimilate basic knowledge from the logic design of digital (numerical) systems.

We hope this book helps the preparation of students, allowing them to go thoroughly into the main fundamental concepts based on the examples that are presented here *in extenso*. Its reading can be considered, in a certain sense, similar to participating in a Logic Design seminar.

The Authors

CUPRINS

TABLE OF CONTENTS

Capitolul 1. Sisteme de numerație și coduri	7
Chapter 1. Number systems and codes	7
1.1. Sisteme de numerație binare	7
1.1. Binary numbering systems.....	7
1.2. Coduri binare.....	10
1.2. Binary codes.....	10
1.2.1. Coduri ponderate.....	11
1.2.1. Weighted codes.....	11
1.2.2. Coduri neponderate	11
1.2.2. Unweighted codes	11
1.2.3. Coduri detectoare de erori.....	12
1.2.3. Error detecting codes	12
1.2.4. Coduri corectoare de erori.....	13
1.2.4. Error correcting codes.....	13
1.3. Probleme rezolvate.....	14
1.3. Solved problems.....	14
1.4. Probleme propuse.....	24
1.4. Proposed problems	24
Capitolul 2. Algebra Booleană. Funcții Booleene	27
Chapter 2. Boolean Algebra. Boolean Functions.....	27
2.1. Algebra Booleană.....	27
2.1. Boolean algebra.....	27
2.2. Proprietățile algebrei Booleene	29
2.2. Properties of Boolean algebra	29
2.3. Funcții Booleene	30
2.3. Boolean functions	30
2.3.1. Funcții Booleene elementare.....	31
2.3.1. Elementary Boolean functions	31
2.3.2. Reprezentarea funcțiilor Booleene	33
2.3.2. Boolean functions representation.....	33
2.3.3. Minimizarea funcțiilor Booleene	39
2.3.3. Minimizing Boolean functions.....	39
2.4. Probleme rezolvate.....	56

2.4. Solved problems.....	56
2.5. Probleme propuse.....	75
2.5. Proposed problems.....	75
Capitolul 3. Circuite logice combinaționale	80
Chapter 3. Combinational logic circuits	80
3.1. Definiții	80
3.1. Definitions.....	80
3.2. Analiza CLC	81
3.2. CLC analysis	81
3.3. Sinteza CLC	83
3.3. CLC synthesis	83
3.4. Probleme rezolvate.....	95
3.4. Solved problems.....	95
3.5. Probleme propuse.....	131
3.5. Proposed problems.....	131
Capitolul 4. Circuite logice secvențiale	136
Chapter 4. Sequential logic circuits	136
4.1. Definiții	136
4.1. Definitions.....	136
4.2. Circuite basculante bistabile	138
4.2. Latches and Flip-Flops.....	138
4.3. Aplicații ale circuitelor basculante bistabile	150
4.3. Applications of Latches and Flip-Flops	150
4.4. Sinteza circuitelor secvențiale sincrone	158
4.4. Sequential logic circuits synthesis	158
4.5. Probleme rezolvate.....	161
4.5. Solved problems.....	161
4.6. Probleme propuse.....	237
4.6. Proposed problems	237
Capitolul 5. Probleme propuse finale.....	248
Chapter 5. Final proposed problems	248
5.1. Probleme propuse.....	248
5.1. Proposed problems	248
Bibliografie	257
References	257

Capitolul 1.

Sisteme de numerație și coduri

1.1. Sisteme de numerație binare

Sistemele de numerație binare sunt utilizate în aproape toate sistemele numerice, inclusiv în aplicațiile de procesare a semnalelor digitale (DSP), în rețele și în sistemele de calcul. Înainte de a alege un sistem de numerație, este important să-i înțelegem avantajele și dezavantajele și, de asemenea, să știm cum să convertim numerele dintr-un sistem în altul.

Principalele sisteme de numerație binare folosite la ora actuală sunt:

- Întregi fără semn
- Complementul față de 2
- Fracționare fără semn
- Fracționare cu semn în Complementul față de 2
- Cod Gray
- Mărime și semn
- Complementul față de 2 deplasat
- Complementul față de 1
- Virgulă mobilă

Tabelul următor redă sintetic principalele lor proprietăți:

Chapter 1.

Number systems and codes

1.1. Binary numbering systems

Binary numbering systems are used in virtually all digital systems, including digital signal processing (DSP), networking, and computers. Before choosing a numbering system, it is important to understand the advantages and disadvantages of each system and to know how to convert between different systems.

The main binary numbering systems most widely used at this moment are the following:

- Unsigned integer
- Two's complement
- Unsigned fractional
- Two's complement signed fractional
- Gray code
- Signed-magnitude
- Offset Two's complement
- One's complement
- Floating point

The next table presents synthetically their main properties:

Sistem (System)	Interval de reprezentare (Number range)	Avantaje (Advantages)	Dezavantaje (Disadvantages)
Întregi fără semn (<i>Unsigned integer</i>)	$[0 - 2^N - 1]$	Sistem de numerație universal. Operațiile aritmetice (adunare, scădere) sunt ușor de realizat. (<i>Universal numbering system. Easy to perform arithmetic operations like additions and subtractions</i>)	Nu poate stoca numere negative. (<i>Cannot store negative numbers</i>).
Complementul față de 2 (<i>Two's complement</i>)	$[-2^{(N-1)} - 2^{(N-1)} - 1]$	Reprezintă atât numere pozitive cât și numere negative. Operațiile aritmetice sunt ușor de realizat cu sumatoare clasice. (<i>Stores both positive and negative numbers. Easy to perform arithmetic with regular adders.</i>)	Necesită un bit de stocare suplimentar chiar și în cazul în care se reprezintă numai numere pozitive. (<i>Requires one extra bit of storage space when only positive numbers are necessary</i>).
Fracționare fără semn (<i>Unsigned fractional</i>)	$[0 - 2^{-N} - 2^{-M}]$	Operațiile sunt identice cu cele pe Întregi fără semn. (<i>Operations are identical to unsigned integer operations</i>).	Nu poate stoca numere negative. (<i>Cannot store negative numbers</i>).
Fracționare cu semn în Complementul față de 2 (<i>Two's complement signed fractional</i>)	$[-2^{(N-1)} - 2^{(N-1)} - 2^{-M}]$	Stochează atât numere pozitive cât și numere negative. Operațiile sunt identice cu cele pe numere în Complementul față de 2. (<i>Stores positive and negative numbers both greater than and less than 1. Operations are identical to Two's complement operations.</i>)	-
Cod Gray (<i>Gray code</i>)	$[0 - 2^{(N-1)}]$	Cuvintele de cod consecutive sunt adiacente (diferă printr-un singur bit), ceea ce facilitează interfațarea	Operațiile aritmetice sunt greu de realizat; este necesară conversia între unul dintre sistemele prezentate aici. (<i>Difficult</i>)

		cu sistemele fizice. <i>(Only one bit changes between adjacent numbers, which facilitates interfaces with physical systems.)</i>	<i>to perform arithmetic operations without first converting to one of the systems listed above.)</i>
Mărime și semn <i>(Signed-magnitude)</i>	$[-2^{(N-1)} - 1 - 2^{(N-1)} - 1]$	Util în cazul aplicațiilor unde trebuie ca valoarea absolută a numărului să fie stocată independent de semnul acestuia. <i>(Useful for applications that require the magnitude to be distinct from the sign.)</i>	Operațiile aritmetice sunt greu de realizat, dar totuși mai ușor decât în cazul codului Gray. <i>(Difficult to perform arithmetic operations, but easier than with Gray code.)</i>
Complementul față de 2 deplasat <i>(Offset Two's complement)</i>	$[-2^{(N-1)} - 2^{(N-1)} - 1]$	Sistemul este utilizat în cadrul convertoarelor analogic-numerice (CAN) și numeric-analogice (CNA). Operațiile aritmetice sunt ușor de implementat. <i>(Used by many analog-to-digital (A/D) and digital-to-analog (D/A) converters. Easy to perform arithmetic operations.)</i>	—
Complementul lui 1 <i>(One's complement)</i>	$[-2^{(N-1)} - 1 - 2^{(N-1)} - 1]$	Negătiile sunt ușor de realizat. <i>(Easy to perform negations.)</i>	Operațiile aritmetice sunt greu de realizat, cu excepția negătiilor. <i>(Difficult to perform arithmetic operations, other than negations.)</i>
Virgulă mobilă <i>(Floating point)</i>		Domeniul de reprezentare a numerelor este foarte mare. <i>(Very large dynamic range.)</i>	Implementarea operațiilor aritmetice consumă mai multe resurse hardware. <i>(Requires more hardware to perform arithmetic.)</i>

Tabelul de mai jos prezintă, sub formă de exemplu, codificarea primelor 8 numere întregi în binar și în cele mai folosite sisteme de numerație binare prezentate mai sus.

The table below shows, as an example, the encoding of the first 8 integers in binary and in the other most used binary numbering systems (from those presented above).

Număr binar (Binary number)	Înregi fără semn (Unsigned integer)	Complementul față de 2 (Two's complement)	Cod Gray (Gray code)	Mărime și semn (Signed-magnitude)	Complementul față de 2 deplasat (Offset Two's complement)	Complementul față de 1 (One's complement)
000	0	0	0	0	-4	0
001	1	1	1	1	-3	1
010	2	2	3	2	-2	2
011	3	3	2	3	-1	3
100	4	-4	7	-0	0	-3
101	5	-3	6	-1	1	-2
110	6	-2	4	-2	2	-1
111	7	-1	5	-3	3	-0

1.2. Coduri binare

Deși sistemul de numerație binar are multe avantaje practice și este foarte utilizat în calculatoarele numerice, în multe cazuri este convenabil să lucrăm cu sistemul zecimal, în special acolo unde comunicația între om și mașină este intensă.

Pentru a simplifica problema comunicării au fost definite un număr de coduri astfel încât cifrele zecimale să fie reprezentate prin succesiuni de cifre binare.

Pentru a reprezenta cele 10 cifre zecimale este suficient să folosim 4 cifre binare. Codurile binare se pot împărti în două clase:

- Coduri ponderate;
- Coduri neponderate.

1.2. Binary codes

Even though the binary numbering system has many practical advantages and is very widely used in digital computing systems, it is often preferable to work with the decimal system, especially where the human-computer communication is intense.

In order to simplify the communication problem, binary codes have been defined for representing the decimal digits by sequences of binary digits.

To represent the ten decimal digits it is sufficient to use 4 bits. We can divide binary codes into two major classes:

- Weighted codes;
- Unweighted codes.

1.2.1. Coduri ponderate

Caracteristica principală a codurilor ponderate este aceea că fiecarei cifre binare îi este asociată o „pondere“. Pentru fiecare grup de 4 biți suma ponderilor acelor cifre binare a căror valoare este 1 este egală cu cifra zecimală pe care o reprezintă. O cifră zecimală într-un cod ponderat se scrie astfel:

$$N = \sum_{i=0}^3 a_i b_i$$

unde $a_i = 0$ sau 1 .

În tabelul de la pagina 12 se dau trei exemple de coduri binare ponderate. Primul cod se numește BCD (*Binary Coded Decimal*), sau 8421, deoarece pentru obținerea codului fiecare cifră zecimală este convertită în binar.

1.2.2. Coduri neponderate

Cele mai utilizate coduri neponderate sunt *Exces 3* și codul *Gray*. Codul Exces 3 este format prin adăugarea lui 0011 la fiecare cuvânt de cod din BCD. Din acest cod s-a eliminat combinația „0000”, care ar putea fi confundată cu lipsa de informație.

În multe aplicații practice, cum ar fi conversia analog-digitală, este de dorit a se folosi codurile în care toate cuvintele de cod succesive

1.2.1. Weighted codes

The main feature of the weighted codes is that each binary digit has an associated “weight”.

For each group of 4 bits, the sum of the weights of those binary digits whose value is ‘1’ equals the decimal digit whose value it represents. A decimal digit in a weighted code is written as follows:

$$N = \sum_{i=0}^3 a_i b_i$$

where $a_i = 0$ or 1 .

In the from page 12 three examples of weighted binary codes are given. The first code is BCD (Binary Coded Decimal), or 8421, because in order to obtain the code each decimal digit is converted in binary.

1.2.2. Unweighted codes

The most used unweighted codes are *Excess 3* and the *Gray* code. The Excess 3 code is formed by adding 0011 to each BCD word. The combination “0000” was eliminated from this code, since it could be mistaken with the lack of information.

In many practical applications, like the analog-to-digital conversion, it is preferable to use codes in which all the successive words differ by

diferă doar printr-o cifră. Codurile care au o astfel de proprietate se numesc *coduri ciclice*. Un astfel de cod este codul Gray. Codul Gray de n biți face parte din clasa *codurilor reflectate*. Termenul „reflectate“ este folosit pentru a desemna coduri care au următoarea proprietate: cuvântul de cod de n biți este generat prin reflectarea codului de $n-1$ biți.

În tabelul de mai jos sunt prezentate, în afara celor trei exemple de coduri ponderate, cele două coduri neponderate amintite.

one single digit. Codes having this property are called *cyclic codes*. Such a code is the *Gray code*. The n -bit Gray code belongs to the *reflected codes* category. The term “reflected” is used here to designate those codes that have the following property: their n -bit words are generated by reflecting the words of the $n-1$ -bit code.

In the table below are presented, besides the three examples of weighted codes, the two unweighted codes mentioned above.

Cifra zecimală (decimal digit)	b ₃	b ₂	b ₁	b ₀	b ₃	b ₂	b ₁	b ₀	Pondere negativă (negative weight)				Exces 3 (Excess 3)				Gray				
	8	4	2	1	2	4	2	1	6	4	2	-3	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1	0	0	1
3	0	0	1	1	0	0	1	1	1	0	0	1	0	1	1	0	0	0	1	0	0
4	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	1	0	1	1	0
5	0	1	0	1	1	0	1	1	1	0	1	1	1	0	0	0	0	1	1	1	1
6	0	1	1	0	1	1	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0
7	0	1	1	1	1	1	0	1	1	1	0	1	1	0	1	0	0	1	0	1	0
8	1	0	0	0	1	1	1	0	1	0	1	0	1	0	1	1	1	1	1	0	0
9	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1

1.2.3. Coduri detectoare de erori

Codurile detectoare de erori au următoarea proprietate: *apariția unei singure erori transformă un cuvânt valid într-un cuvânt invalid*.

O metodă pentru detecția erorilor este folosirea *bitului de paritate*. Ideea de bază în controlul de paritate este de a adăuga o cifră binară în plus la fiecare

1.2.3. Error detecting codes

The error detecting codes have the following property: *the apparition of a single error transforms a valid word into an invalid word*.

A method for detecting errors is to use the *parity bit*. The main idea in parity control is to add an extra binary digit at each word of a given

cod cuvânt al unui cod dat, pentru a face ca numărul de biți de „1” din fiecare cuvânt să devină impar sau par.

Un cod detector de erori este „*2 din 5*“ care are ponderile 0, 1, 2, 4, 7. Cu excepția cuvântului de cod 0 zecimal, codul este ponderat și arată ca în tabelul de mai jos.

code, in order to make the number of bits of ‘1’ from each word to become odd or even.

An error detecting code is the code called “*2 out of 5*” whose weights are 0, 1, 2, 4, 7. Except for the word 0 (in decimal), this code is weighted and can be seen in the table below.

Cifra zecimală (<i>Decimal digit</i>)	0	1	2	4	7
0	0	0	0	1	1
1	1	1	0	0	0
2	1	0	1	0	0
3	0	1	1	0	0
4	1	0	0	1	0
5	0	1	0	1	0
6	0	0	1	1	0
7	1	0	0	0	1
8	0	1	0	0	1
9	0	0	1	0	1

1.2.4. Coduri corectoare de erori

Spunem că un cod este corector de eroare dacă cuvântul de cod corect poate fi întotdeauna dedus din cuvântul eronat. O clasă de coduri de corecție este cunoscută sub numele de *coduri Hamming*.

Numărul minim de biți de control necesari pentru corectarea erorilor singulare se determină conform relației lui Hamming:

$$2^k \geq m + k + 1$$

unde m este numărul de biți de informație, iar k – numărul de biți de control.

1.2.4. Error correcting codes

We say that a code is an error correcting one if the correct word can always be deduced from the altered word. Such error correcting codes are the *Hamming codes*.

The minimal number of control bits that are necessary to correct singular errors is determined according to Hamming's relation:

$$2^k \geq m + k + 1$$

where m is the number of information bits, and k – the number of control bits.

Vom explica acum construcția unui cod Hamming cu $m = 4$. Din relația lui Hamming rezultă $k = 3$, deci 3 biți de control trebuie adăugați celor 4 biți de informație. Pozițiile bițiilor sunt numerotate de la 1 la 7, după cum se poate vedea mai jos:

1	2	3	4	5	6	7
c_1	c_2	b_1	c_3	b_2	b_3	b_4

Biții de pe pozițiile puteri ale lui 2 sunt folosiți ca biți de control: c_1, c_2, c_3 . Celelalte poziții corespund bițiilor de informație: de la b_1 la b_4 . Biți de control se calculează astfel:

$$\begin{aligned}c_1 &= b_1 \oplus b_2 \oplus b_4 \\c_2 &= b_1 \oplus b_3 \oplus b_4 \\c_3 &= b_2 \oplus b_3 \oplus b_4\end{aligned}$$

Exemplu:

$$\begin{aligned}b_1 b_2 b_3 b_4 &= 0100 \\c_1 &= 1, c_2 = 0, c_3 = 1 \\ \Rightarrow \text{Secvența de cod va fi: } &1001100.\end{aligned}$$

1.3. Probleme rezolvate

1. Care este cel mai mare și cel mai mic număr întreg cu semn / fără semn care poate fi exprimat pe 12 biți?

We will explain now the method for creating a Hamming code with $m = 4$. From Hamming's relation we have $k = 3$, so we must add 3 control bits to the 4 information bits. The bits' positions are numbered from 1 to 7, as shown below:

The bits on the positions powers of 2 are used as control bits: c_1, c_2, c_3 . The other positions correspond to the information bits: from b_1 to b_4 . The control bits are computed as follows:

$$\begin{aligned}c_1 &= b_1 \oplus b_2 \oplus b_4 \\c_2 &= b_1 \oplus b_3 \oplus b_4 \\c_3 &= b_2 \oplus b_3 \oplus b_4\end{aligned}$$

Example:

$$\begin{aligned}b_1 b_2 b_3 b_4 &= 0100 \\c_1 &= 1, c_2 = 0, c_3 = 1 \\ \Rightarrow \text{The code sequence will be: } &1001100.\end{aligned}$$

1.3. Solved problems

1. What are the largest and smallest signed / unsigned numbers that can be expressed with 12 bits?

Soluție

Conform Tabelului 1.1:

- a) în reprezentarea numerelor întregi cu semn, cel mai mare număr întreg care poate fi reprezentat pe 12 biți este $2^{(N-1)}-1$, adică $2^{11}-1 = 2047$, iar cel mai mic număr întreg care poate fi reprezentat pe 12 biți este $-2^{(N-1)}-1$, adică -2047 .
- b) în reprezentarea numerelor întregi fără semn, cel mai mare număr întreg care poate fi reprezentat pe 12 biți este 2^N-1 , adică $2^{12}-1 = 4095$, iar cel mai mic număr întreg care poate fi reprezentat pe 12 biți este 0.

2. Convertiți numărul hexazecimal 68BE în binar și apoi din binar convertiți-l în octal.

Solutie

a) Pentru a realiza conversii din hexazecimal (baza 16) în binar (baza 2), observăm că între cele două baze există relația: $16 = 2^4$. Prin urmare, o cifră hexazecimală corespunde unei grupări de 4 cifre binare (4 biți).

Astfel, numărul $NR_{16} = 68BE$ se scrie în binar astfel:
 $0110 | 1000 | 1011 | 1110$,
deci $NR_2 = 0110100010111110$.

Solution

According to Table 1.1:

- a) in the Signed-magnitude numbering system, the largest integer that can be represented on 12 bits is $2^{(N-1)}-1$, i.e. $2^{11}-1 = 2047$, while the smallest integer that can be represented on 12 bits is $-2^{(N-1)}-1$, i.e. -2047 .
- b) in the Unsigned integer numbering system, the largest integer that can be represented on 12 bits is 2^N-1 , i.e. $2^{12}-1 = 4095$, while the smallest integer that can be represented on 12 bits is 0.

2. Convert the hexadecimal number 68BE to binary and then from binary convert it to octal.

Solution

a) In order to make conversions from hexadecimal (base 16) to binary (base 2), we observe that the following relationship stands between the two bases: $16 = 2^4$. Therefore, a hexadecimal digit corresponds to a group of 4 binary digits (4 bits).

The number $NR_{16} = 68BE$ is then written in binary as follows:
 $0110 | 1000 | 1011 | 1110$,
so $NR_2 = 0110100010111110$.

b) În aceeași idee, observăm că între cele două baze există relația: $8 = 2^3$. Așadar, fiecărui grup de 3 cifre binare îi corespunde o cifră octală:

$$NR_2 = 000 \mid 110 \mid 100 \mid 010 \mid 111 \mid \\ 110, \text{ deci } NR_8 = 064276 = 64276.$$

Observație: Dacă numărul de biți nu este multiplu de 3, se completează cu zerouri pe poziția cea mai semnificativă (în exemplul de mai sus, am adăugat două zerouri – evidențiate cu *italice*).

3. Convertiți numărul zecimal 34,4375 în binar și hexazecimal.

Soluție

Întâi observăm că numărul este fracționar. Conversia părții întregi și a părții fracționale se face după algoritmi diferenți.

a) Partea întreagă este $N_{10} = 34$. Pentru conversia în baza 2, vom împărti succesiv partea întreagă a numărului la baza 2:

$$34 : 2 = 17, \text{ rest } 0$$

$$17 : 2 = 8, \text{ rest } 1$$

$$8 : 2 = 4, \text{ rest } 0$$

$$4 : 2 = 2, \text{ rest } 0$$

$$2 : 2 = 1, \text{ rest } 0$$

$$1 : 2 = 0, \text{ rest } 1$$

N_2 se obține citind resturile în ordinea inversă obținerii lor:

b) In the same idea, we notice that the following relationship stands between the two bases: $8 = 2^3$. As a consequence, an octal digit corresponds to each group of 3 bits:

$$NR_2 = 000 \mid 110 \mid 100 \mid 010 \mid 111 \mid \\ 110, \text{ so } NR_8 = 064276 = 64276.$$

Remark: If the number of bits is not a multiple of 3, we must pad the number with zeroes on the most significant position (in the example above, we padded it with two zeroes – highlighted in *italics*).

3. Convert the decimal number 34.4375 to binary and hexadecimale.

Solution

First, we notice that this number is fractional. The conversion of the integer part and of the fractional part is done after different algorithms.

a) The integer part is $N_{10} = 34$. To convert into base 2, we will successively divide the integer part of the number to base 2:

$$34 : 2 = 17, \text{ rest } 0$$

$$17 : 2 = 8, \text{ rest } 1$$

$$8 : 2 = 4, \text{ rest } 0$$

$$4 : 2 = 2, \text{ rest } 0$$

$$2 : 2 = 1, \text{ rest } 0$$

$$1 : 2 = 0, \text{ rest } 1$$

N_2 is obtained by reading the rests

$$N_2 = 100010.$$

Partea fracționară este $M_{10} = 0,4375$. Pentru conversia în baza 2, vom înmulți succesiv partea fracționară a numărului cu baza 2. Vom obține un nou număr care va avea o parte întreagă și o parte fracționară. Partea întreagă o vom memora, iar partea fracționară o vom înmulții în mod repetat cu baza 2:

$$0,4375 \times 2 = 0,875$$

$$0,875 \times 2 = 1,75$$

$$0,75 \times 2 = 1,5$$

$$0,5 \times 2 = 1,0$$

$$0,0 \times 2 = 0$$

M_2 se obține citind părțile întregi în ordinea normală a obținerii lor:

$M_2 = 0111$ (ultimul 0 nu mai contează).

Așadar, numărul în baza 2 este:

$$NR_2 = 100010,0111.$$

b) Observăm că numărul este un număr fracționar fără semn de forma (N,M) , unde N și M reprezintă numărul de biți ai părții întregi și respectiv ai părții fracționale. În acest caz, avem un număr (6,4).

Pentru conversia în baza 16, vom pleca de la reprezentarea numărului în baza 2 obținută anterior. La fel ca la problema 2, vom realiza grupări de câte 4 biți, fiecărei grupări corespunzându-i o cifră hexazecimală. *Atenție:* Grupările se realizează plecând de la virgula binară, spre stânga și respectiv spre

in reverse order:

$$N_2 = 100010.$$

The fractional part is $M_{10} = 0.4375$. In order to convert it into base 2, we will successively multiply the fractional part of the number with base 2. We will obtain a new number that will have an integer part and a fractional part. The integer part will be stored, while the fractional integer part will be repeatedly multiplied by base 2:

$$0.4375 \times 2 = 0.875$$

$$0.875 \times 2 = 1.75$$

$$0.75 \times 2 = 1.5$$

$$0.5 \times 2 = 1.0$$

$$0.0 \times 2 = 0$$

M_2 is obtained by reading the integer parts in normal order:

$M_2 = 0111$ (the last 0 is discarded).

So, the number in base 2 is:

$$NR_2 = 100010.0111.$$

b) We notice that the number is an $(N.M)$ unsigned fractional, where N and M represent the number of bits of the integer part and the fractional part, respectively. In this particular case, we have a (6,4) number.

To convert it into base 16, we start from the number's representation in base 2, which was obtained previously. As for problem 2, we will realize groups of 4 bits; a hexadecimal digit corresponds to each group. *Remark:* The groups are constituted starting from the binary point, towards the left and towards

dreapta!

Așadar, $NR_2 = 0010 | 0010 |,0111$.

Atunci, $NR_{16} = 22,7$.

4. Exprimăți următoarele numere în zecimal: $N_1 = (10110,0101)_2$, $N_2 = (16,5)_{16}$ și $N_3 = (26,24)_8$

Solutie

Pentru a converti numerele în baza 10, ele se exprimă ca un polinom în puterile bazei actuale (2, 16 și respectiv 8) și apoi se evaluează folosind aritmetică în baza 10.

Așadar:

$$\begin{aligned}N_1 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 \\&+ 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} \\&+ 1 \times 2^{-4} = (22,3125)_{10}.\end{aligned}$$

$$\begin{aligned}N_2 &= 1 \times 16^1 + 6 \times 16^0 + 5 \times 16^{-1} = \\&(22,3125)_{10}.\end{aligned}$$

$$\begin{aligned}N_3 &= 2 \times 8^1 + 6 \times 8^0 + 2 \times 8^{-1} + 4 \times \\&8^{-2} = (22,3125)_{10}.\end{aligned}$$

Observație: Se putea observa încă de la început că este vorba despre același număr, realizând în N_1 grupări de câte 4 cifre binare (astfel, reprezentând fiecare grup de 4 cifre în hexazecimal, se obține numărul N_2) sau grupări de câte 3 cifre binare (astfel, reprezentând fiecare grup de 3 cifre în octal, se obține numărul N_3).

the right respectively!

So, $NR_2 = 0010 | 0010 |,0111$.

$NR_{16} = 22.7$.

4. Express the following numbers in decimal: $N_1 = (10110.0101)_2$, $N_2 = (16.5)_{16}$, and $N_3 = (26.24)_8$

Solution

To convert the numbers in base 10, they have to be expressed as a polynomial in the powers of the current base (2, 16, and 8, respectively) and then evaluated using the arithmetic in base 10:

$$\begin{aligned}N_1 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 \\&+ 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} \\&+ 1 \times 2^{-4} = (22,3125)_{10}.\end{aligned}$$

$$\begin{aligned}N_2 &= 1 \times 16^1 + 6 \times 16^0 + 5 \times 16^{-1} = \\&(22.3125)_{10}.\end{aligned}$$

$$\begin{aligned}N_3 &= 2 \times 8^1 + 6 \times 8^0 + 2 \times 8^{-1} + 4 \times \\&8^{-2} = (22.3125)_{10}.\end{aligned}$$

Remark: We could have noticed from the beginning that it is the same number, by making in N_1 groupings of 4 binary digits (this way, by representing each group of 4 digits in hexadecimal, we obtain the number N_2) or groupings of 3 binary digits (this way, by representing each group of 3 digits in octal, we obtain the number N_3).

5. Adunați și înmulțiiți numerele binare 1011 și 101 fără a le converti în zecimal.

Soluție

Adunarea și înmulțirea în baza 2 se fac similar cu operațiile din baza 10:

$$\begin{array}{r} 1011 \\ + \quad 101 \\ \hline 10000 \end{array}$$

5. Add and multiply the binary numbers 1011 and 101 without converting them into decimal.

Solution

Addition and multiplication in binary are done similarly with the operations in decimal:

$$\begin{array}{r} 1011 \times \\ 101 \\ \hline 1011 \\ 0000 \\ \hline 1011 \\ \hline 110111 \end{array}$$

6. Obțineți complementul față de 1 și față de 2 ale următoarelor numere binare întregi fără semn:

- a) $N_1 = 11101010$
- b) $N_2 = 01111110$
- c) $N_3 = 00000000$

Soluție

Complementul față de 1 se obține negând toți biții numărului în cauză. Complementul față de 2 se obține negând toți biții numărului în cauză și adunând apoi '1' (cu alte cuvinte, complementul față de 2 se obține adunând '1' la expresia numărului în Complementul față de 1).

Așadar:

6. Obtain 1's and 2's complement of the following unsigned integer binary numbers:

- a) $N_1 = 11101010$
- b) $N_2 = 01111110$
- c) $N_3 = 00000000$

Solution

One's complement is obtained by negating all the number's bits.
 Two's complement is obtained by negating all the number's bits and then adding '1' (in other words, two's complement is obtained by adding '1' to the number's expression in One's complement).

As a matter of consequence:

- a) Complementul față de 1 al lui $N_1 = 00010101$; Complementul față de 2 al lui $N_1 = 00010110$.
- b) Complementul față de 1 al lui $N_2 = 10000001$; Complementul față de 2 al lui $N_2 = 10000010$.
- c) Complementul față de 1 al lui $N_3 = 11111111$; Complementul față de 2 al lui $N_3 = 00000000$. Observăm aici că pentru reprezentarea lui N_3 în Complementul față de 2, se trece pe 9 biți, dar transportul la nivelul bitului cel mai semnificativ este ignorat, deci reprezentarea lui „0” în Complementul față de 2 și în sistemul Întregi fără semn este identică, pe când în Complementul față de 1 există două reprezentări posibile ale lui „0”: „00000000” (,+0’) și „11111111” (-0’).

7. Convertiți numărul zecimal +61 și numărul zecimal +27 în binar folosind reprezentarea în Complementul față de 2 (care este un sistem de numerație cu semn) și suficient de mulți biți pentru a alinia numerele. Apoi realizați operațiile binare echivalente următoarelor operații:

- $(27) + (-61)$
- $(-27) + (+61)$
- $(-27) + (-61)$.

Convertiți apoi rezultatele înapoi în zecimal și verificați corectitudinea lor.

- a) One's complement of $N_1 = 00010101$; Two's complement of $N_1 = 00010110$.
- b) One's complement of $N_2 = 10000001$; Two's complement of $N_2 = 10000010$.
- c) One's complement of $N_3 = 11111111$; Two's complement of $N_3 = 00000000$. We can notice here that for the representation of N_3 in Two's complement, we pass on 9 bits, but carry at the level of the most significant bit is ignored, so the representation of '0' in Two's complement and in the Unsigned integers numbering system is identical, while in the One's complement there are two possible representations of '0': "00000000" (+0') and "11111111" (-0').

7. Convert the decimal number +61 and the decimal number +27 to binary using the 2's complement representation (which is a signed numbering system) and enough digits to accommodate the numbers. Then perform the binary operations that are equivalent to the following operations:

- $(27) + (-61)$,
- $(-27) + (+61)$,
- $(-27) + (-61)$.

Convert then the results back to decimal and verify that they are correct.

Soluție

Să notăm $N_1 = +61$ și $N_2 = +27$.

În Întregi fără semn, $N_1 = 111101$, iar $N_2 = 11011$. Pentru a-l alinia pe N_2 la N_1 , $N_2 = 011011$.

În Complementul față de 2, vom avea nevoie de 7 biți pentru a reprezenta numerele:

$N_1 = 0111101$, iar $N_2 = 0011011$.

- $N_1 = -61 = 1000011$.

- $N_2 = -27 = 1100101$.

$$a) (27) + (-61) = N_2 + (-N_1) = -34.$$

$$\begin{array}{r} 0011011 \\ + 1000011 \\ \hline 1011110 \end{array}$$

$$b) (-27) + (+61) = -N_2 + N_1 = +34.$$

$$\begin{array}{r} 1100101 \\ + 0111101 \\ \hline 0100010 \end{array}$$

Ignorat!

După cum se știe, *depășirea* apare când transportul din dreapta și transportul spre stânga bitului cel mai semnificativ sunt diferite. Întrucât nu este cazul aici (ambii biți de transport sunt '1'), nu avem depășire și rezultatul corect este pe 7 biți.

$$c) (-27) + (-61) = -N_2 + (-N_1) = -88.$$

Solution

Let's denote $N_1 = +61$ and $N_2 = +27$.

In the unsigned integers numbering system, $N_1 = 111101$, and $N_2 = 11011$. To align N_2 to N_1 , $N_2 = 011011$.

In Two's Complement, we will need 7 bits to represent the numbers:

$N_1 = 0111101$, and $N_2 = 0011011$.

- $N_1 = -61 = 1000011$.

- $N_2 = -27 = 1100101$.

$$a) (27) + (-61) = N_2 + (-N_1) = -34.$$

$$\begin{array}{r} 0011011 \\ + 1000011 \\ \hline 1011110 \end{array}$$

$$b) (-27) + (+61) = -N_2 + N_1 = +34.$$

$$\begin{array}{r} 1100101 \\ + 0111101 \\ \hline 0100010 \end{array}$$

Ignored!

As it is known, *overflow* occurs when the carry-in and carry-out of the most significant bit are different. Since this is not the case here (both carry bits are '1'), we don't have an overflow situation and the correct result is on 7 bits.

$$c) (-27) + (-61) = -N_2 + (-N_1) = -88.$$

Necesari
pentru
extensie de
semn!

Nu este ignorat!

$$\begin{array}{r} 1000011 \\ + 1100101 \\ \hline 0101000 \end{array}$$

Needed for
sign
extension!

Not ignored!

$$\begin{array}{r} 1000011 \\ + 1100101 \\ \hline 0101000 \end{array}$$

În acest caz, transportul din dreapta bitului cel mai semnificativ este '0', iar transportul spre stânga este '1'. Dacă rezultatul se menține pe 7 biți, el este incorect (+40). Pentru a avea rezultatul corect, trebuie să-l considerăm pe 8 biți.

8. a) Convertiți numerele zecimale 126 și 348 în cod BCD, și efectuați adunarea lor folosind codul BCD.
 b) Cum se pot converti numerele din binar în BCD? Exemplificați pentru numărul 11111111 (255 în baza 10).

Soluție

a) Să notăm $N_1 = 126$ și $N_2 = 348$. În BCD, $N_1 = 000100100110$, iar $N_2 = 001101001000$.

Adunarea în BCD se efectuează pe grupuri de 4 cifre:

$$6 + 8 = 4, \text{ transport } 1.$$

$$2 + 4 + 1 = 7, \text{ transport } 0.$$

$$1 + 3 = 4, \text{ transport } 0.$$

Așadar, $N_1 + N_2 = 474$.

b) O tehnică eficientă de conversie din binar în BCD este algoritmul ADUNĂ-3. Vom converti numărul binar originar într-un număr BCD

In this case, the carry-in bit of the most significant bit is '0', while the carry-out bit is '1'. If the result is kept on 7 bits, it is incorrect (+40). To get the correct result, we must consider it on 8 bits.

8. a) Convert the decimal numbers 126 and 348 to BCD code, and perform their addition using the BCD code.
 b) How can be converted numbers from binary to BCD? Exemplify for 11111111 (255 in base 10).

Solution

a) Let's denote $N_1 = 126$ and $N_2 = 348$. In BCD, $N_1 = 000100100110$, and $N_2 = 001101001000$.

Addition in BCD is done on groups of 4 digits:

$$6 + 8 = 4, \text{ carry } 1.$$

$$2 + 4 + 1 = 7, \text{ carry } 0.$$

$$1 + 3 = 4, \text{ carry } 0.$$

So, $N_1 + N_2 = 474$.

b) An efficient technique for binary-to-BCD conversion is that of the so-called ADD-3 algorithm, and will convert the original binary number

de trei cifre (SUTE, ZECI, UNITĂȚI). Algoritmul poate fi exprimat astfel:

1. Se inițializează SUTE = 0, ZECI = 0, UNITĂȚI = 0, COUNT=8.
2. Dacă vreo cifră BCD este 5 sau mai mare, se adună trei la acea cifră.
3. Se decrementează COUNT și se efectuează o deplasare la stânga. Bitul deplasat din numărul binar pe 8 biți este transportat în UNITĂȚI, bitul deplasat din UNITĂȚI este transportat în ZECI etc.
4. Dacă COUNT nu este egal cu 0, salt la pasul 2, altfel STOP.

Exemplu:

SUTE (HUNDREDS)	ZECI (TENS)	UNITĂȚI (UNITS)	BINAR (BINARY)	Pași (Steps)
0000	0000	0000	11111111	Start
0000	0000	0001	11111110	Deplasare (<i>shift</i>) 1
0000	0000	0011	11111100	Deplasare (<i>shift</i>) 2
0000	0000	0111	11111000	Deplasare (<i>shift</i>) 3
0000	0000	1010	11111000	ADUNĂ 3 la UNITĂȚI (<i>ADD-3 to UNITS</i>)
0000	0001	0101	11110000	Deplasare (<i>shift</i>) 4
0000	0001	1000	11110000	ADUNĂ 3 la UNITĂȚI (<i>ADD-3 to UNITS</i>)
0000	0011	0001	11100000	Deplasare (<i>shift</i>) 5
0000	0110	0011	11000000	Deplasare (<i>shift</i>) 6
0000	1001	0011	11000000	ADUNĂ 3 la ZECI (<i>ADD-3 to TENS</i>)
0001	0010	0111	10000000	Deplasare (<i>shift</i>) 7
0001	0010	1010	10000000	ADUNĂ 3 la UNITĂȚI (<i>ADD-3 to UNITS</i>)
0010	0101	0101	00000000	Deplasare (<i>shift</i>) 8

Rezultat (*result*): $(11111111)_2 = (FF)_{16} = (255)_{10}$.

into three BCD digits (HUNDREDS, TENS, UNITS). The algorithm can be expressed as follows:

1. Set HUNDREDS=0, TENS=0, UNITS=0, COUNT=8.
2. If any BCD digit is 5 or greater, add three to that digit.
3. Decrement COUNT and shift left. The bit shifted from the 8-bit binary is carried into UNITS, the bit shifted from UNITS is carried to TENS, etc.
4. If count not equal to 0, go to step 2, else STOP.

Example:

1.4. Probleme propuse

1. Codificați cifrele zecimale 0, 1, 2, 3 ... 9 cu ajutorul următorului cod ponderat: 6 3 1 -1.
2. Realizați conversia:
 $(11010100)_4 \rightarrow$ în baza 8.
3. Realizați conversia:
 $(1476503)_8 \rightarrow$ în baza 2.
4. Codificați cifrele zecimale 0, 1, 2, 3 ... 9 cu ajutorul următorului cod ponderat: 7 3 1 -2.
5. Codificați cifrele zecimale 0, 1, 2, 3 ... 9 cu ajutorul următorului cod ponderat: 5 4 -2 -1.
6. Determinați bazele posibile ale numerelor implicate în următoarea operație:
 $\sqrt{41} = 5$
7. Realizați conversia:
 $(10011100010101)_4 \rightarrow$ în baza 16.
8. Știind că:
a) $(16)_{10} = (100)_b$,
b) $(292)_{10} = (1204)_b$,
determinați valoarea lui b în fiecare caz independent.
9. Determinați bazele posibile ale numerelor implicate în următoarea operație: $41 / 3 = 13$.

1.4. Proposed problems

1. Encode the decimal digits 0, 1, 2, 3 ... 9 with the following weighted code: 6 3 1 -1.
2. Make the conversion:
 $(11010100)_4 \rightarrow$ in base 8.
3. Make the conversion:
 $(1476503)_8 \rightarrow$ in base 2.
4. Encode the decimal digits 0, 1, 2, 3 ... 9 with the following weighted code: 7 3 1 -2.
5. Encode the decimal digits 0, 1, 2, 3 ... 9 with the following weighted code: 5 4 -2 -1.
6. Determine the possible bases of the numbers involved in the following operation:
 $\sqrt{41} = 5$
7. Make the conversion:
 $(10011100010101)_4 \rightarrow$ in base 16.
8. Knowing that:
a) $(16)_{10} = (100)_b$,
b) $(292)_{10} = (1204)_b$,
determine the value of b in each independent case.
9. Determine the possible bases of the numbers from the following operation: $41 / 3 = 13$.

10. O sursă de date transmite către un echipament receptor. Fiecare cifră zecimală este codificată independent.

Arătați cum este codificat numărul zecimal 7593 în cazul utilizării, pentru transmisie, a următoarelor coduri:

- a) BCD
- b) 2 din 5
- c) Gray
- d) Hamming

e) În cazul d), presupunând că se transmite numai cifra zecimală 8 în BCD, dacă destinatarul transmisiei recepționează cuvântul de cod „1110010”, care era cuvântul de cod inițial transmis de către sursă? Dar dacă se recepționează cuvântul de cod „1110000”?

Observație: Pe canalul de comunicare poate să apară cel mult o eroare (un bit alterat).

11. Următorul mesaj a fost codificat în cod Hamming și transmis pe un canal cu zgomote. Decodificați mesajul presupunând că în fiecare cod cuvânt a survenit cel mult o singură eroare:

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

10. A data source transmits towards a receiving device. Each decimal digit is independently encoded.

Show how the decimal number 7593 is encoded in case for the transmission the source uses the following codes:

- a) BCD
- b) 2 out of 5
- c) Gray
- d) Hamming
- e) In case d), assuming that only the decimal digit 8 is transmitted, in BCD, if the transmission's destination receives the binary word „1110010”, which was the initial binary word transmitted by the source? And what if the „1110000” binary word is received?

Remark: On the communication channel, at most one error (one altered bit) can occur.

11. The following message has been encoded in Hamming code and transmitted on a noisy channel. Decode the message assuming that in each word at most one error has occurred:

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

12. În ce condiții adunarea a două numere BCD este corectă (rezultatul este tot un număr BCD)? Exemplificați pentru $7 + 2$, apoi pentru $7 + 8$.

Indicație: Ca regulă generală, în cazul apariției transportului, este necesar să se adune 6 la rezultat.

12. In which conditions the addition of two BCD numbers is correct (the result is also a BCD number)? Exemplify for $7 + 2$, then for $7 + 8$.

Hint: As a general rule, in case a carry occurs, it is necessary to add 6 to the final result.

Capitolul 2. Algebra Booleană. Funcții Booleene

2.1. Algebra Booleană

Bazele algebrei logice au fost puse de matematicianul englez George Boole (1815-1864) și ca urmare se mai numește și algebră Booleană. Ea a fost concepută ca o metodă simbolică pentru tratarea funcțiilor logicii formale, dar a fost apoi dezvoltată și aplicată și în alte domenii ale matematicii. În 1938 Claude Shannon a folosit-o pentru prima dată în analiza circuitelor de comutație.

Algebra Booleană este o algebră formată din:

- elementele {0,1};
- două operații binare numite SAU și ȘI, notate simbolic + (sau \vee) și \cdot (sau \wedge);
- o operație unară numită NU (negație), notată simbolic $\bar{}$ (sau \neg).

Operațiile se definesc astfel:

ȘI	SAU	NU
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\bar{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\bar{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Chapter 2. Boolean Algebra. Boolean Functions

2.1. Boolean algebra

Fundamentals of logic algebra have been created by the British mathematician George Boole (1815-1864); this is why it is also called Boolean algebra. It has been conceived as a symbolic method for processing formal logic's functions, but then it was developed and applied in other mathematical fields. In 1938 Claude Shannon used it for the first time in the analysis of the switching circuits.

Boolean algebra is composed of:

- the elements {0,1};
- two binary operations called OR and AND, symbolically denoted by + (or \vee) and \cdot (or \wedge);
- one unary operation called NOT (negation), symbolically denoted by $\bar{}$ (or \neg).

These operations are defined as follows:

AND	OR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\bar{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\bar{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Axiomele algebrei Booleene sunt următoarele:

Fie o mulțime M compusă din elementele x_1, x_2, \dots, x_n , împreună cu operațiile \cdot și $+$. Această mulțime formează o algebră dacă:

- 1) Mulțimea M conține cel puțin 2 elemente distincte $x_1 \neq x_2$ ($x_1, x_2 \in M$);
- 2) Pentru $\forall x_1 \in M, x_2 \in M$ avem:

$$x_1 + x_2 \in M \text{ și } x_1 \cdot x_2 \in M$$

- 3) Operațiile \cdot și $+$ au următoarele proprietăți:

- a. sunt comutative

$$x_1 \cdot x_2 = x_2 \cdot x_1$$

$$x_1 + x_2 = x_2 + x_1$$

- b. sunt asociative

$$x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$$

$$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$$

- c. sunt distributive una față de cealaltă

$$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$$

$$x_1 + (x_2 \cdot x_3) = (x_1 + x_2) \cdot (x_1 + x_3)$$

- 4) Ambele operații admit câte un element neutru cu următoarele proprietăți:

$$x_1 + 0 = 0 + x_1 = x_1$$

$$x_1 \cdot 1 = 1 \cdot x_1 = x_1$$

unde 0 este elementul nul al mulțimii, iar 1 este elementul unitate al mulțimii.

- 5) Dacă mulțimea M nu conține decât două elemente, acestea trebuie să fie obligatoriu elementul nul 0 și elementul unitate 1; atunci pentru $\forall x \in M$ există un element unic notat

The axioms of Boolean algebra are:

Be given a set M composed of the elements x_1, x_2, \dots, x_n , along with the operations ‘ \cdot ’ and ‘ $+$ ’. This set forms an algebra if:

- 1) The set M contains at least 2 distinct elements $x_1 \neq x_2$ ($x_1, x_2 \in M$);

- 2) For $\forall x_1 \in M, x_2 \in M$ we have:

$$x_1 + x_2 \in M \text{ and } x_1 \cdot x_2 \in M$$

- 3) The operations ‘ \cdot ’ and ‘ $+$ ’ have the following properties:

- a. Commutative laws

$$x_1 \cdot x_2 = x_2 \cdot x_1$$

$$x_1 + x_2 = x_2 + x_1$$

- b. Associative laws

$$x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$$

$$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$$

- c. Distributive laws:

$$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$$

$$x_1 + (x_2 \cdot x_3) = (x_1 + x_2) \cdot (x_1 + x_3)$$

- 4) Both operations admit a neutral element with the following properties:

$$x_1 + 0 = 0 + x_1 = x_1$$

$$x_1 \cdot 1 = 1 \cdot x_1 = x_1$$

where 0 is the null element of the set, while 1 is its unit element.

- 5) If the set M only contains two elements, they have to be the null element (0) and the unit element (1); then, for $\forall x \in M$ there is a unique element denoted by \bar{x} with the following properties:

cu \bar{x} , cu proprietățile:

$x \bullet \bar{x} = 0$ principiul contradicției
 $x + \bar{x} = 1$ principiul terțului exclus
 \bar{x} este inversul elementului x .

$x \bullet \bar{x} = 0$ contradiction principle

$x + \bar{x} = 1$ third exclusion principle

\bar{x} is the inverse of the element x .

2.2. Proprietățile algebrei Booleene

Plecând de la axiome se deduc o serie de proprietăți, care vor forma reguli de calcul în cadrul algebrei Booleene. Aceste proprietăți sunt:

1) Prinzipiul dublei negații

$\underline{\underline{x}} = x$ dubla negație duce la o afirmație

2) Idempotență

$$x \bullet x = x$$

$$x + x = x$$

3) Absorbția

$$x_1 \bullet (x_1 + x_2) = x_1$$

$$x_1 + (x_1 \bullet x_2) = x_1$$

4) Proprietățile elementelor neutre

$$x \bullet 0 = 0 \qquad \qquad x \bullet 1 = x$$

$$x + 0 = x \qquad \qquad x + 1 = 1$$

5) Formulele lui De Morgan

$$\underline{\underline{x_1 \bullet x_2}} = \underline{\underline{x_1}} + \underline{\underline{x_2}}$$

$$\underline{\underline{x_1 + x_2}} = \underline{\underline{x_1}} \bullet \underline{\underline{x_2}}$$

These formulas are very useful because of the possibility they offer to transform the logic product into a logic sum and vice versa.

6) Prinzipiul dualității – dacă în axiomele și proprietățile algebrei Booleene se interschimbă „0” cu „1” și „+” cu „•”, sistemul de axiome

2.2. Properties of Boolean algebra

Starting from the axioms one can deduce a series of properties that will form calculus rules inside the Boolean algebra. These properties are the following ones:

1) Double negation principle

$\underline{\underline{x}} = x$ double negation is equivalent to an affirmation

2) Idempotency

$$x \bullet x = x$$

$$x + x = x$$

3) Absorption

$$x_1 \bullet (x_1 + x_2) = x_1$$

$$x_1 + (x_1 \bullet x_2) = x_1$$

4) Neutral elements properties

$$x \bullet 0 = 0 \qquad \qquad x \bullet 1 = x$$

$$x + 0 = x \qquad \qquad x + 1 = 1$$

5) De Morgan's laws

$$\underline{\underline{x_1 \bullet x_2}} = \underline{\underline{x_1}} + \underline{\underline{x_2}}$$

$$\underline{\underline{x_1 + x_2}} = \underline{\underline{x_1}} \bullet \underline{\underline{x_2}}$$

These formulas are very useful because of the possibility they offer to transform the logic product into a logic sum and vice versa.

6) Duality principle – if in the axioms and properties of the Boolean algebra we interchange ‘0’ with ‘1’ and ‘+’ with ‘•’, the system

rămâne același, în afara unor permutări.

Verificarea proprietăților se poate face cu ajutorul tabelelor de adevăr și cu observația că două funcții sunt egale dacă iau aceleași valori în toate punctele domeniului de definiție. Prin tabelul de adevăr se stabilește o corespondență între valorile de adevăr ale variabilelor și valoarea de adevăr a funcției.

Observație

- 1) Comutativitatea și asociativitatea pot fi extinse la un număr arbitrar, dar finit, de termeni, indiferent de ordinea lor.
- 2) Formulele lui De Morgan pot fi generalizate la un număr arbitrar de termeni:

$$\begin{aligned} \overline{x_1 \bullet x_2 \bullet \dots \bullet x_n} &= \overline{x_1} + \overline{x_2} + \dots + \overline{x_n} \\ \overline{x_1 + x_2 + \dots + x_n} &= \overline{x_1} \bullet \overline{x_2} \bullet \dots \bullet \overline{x_n} \end{aligned}$$

2.3. Funcții Booleene

O funcție $f: B^n \rightarrow B$, unde $B = \{0,1\}$, se numește funcție Booleană. Această funcție Booleană $y = f(x_1, x_2, \dots, x_n)$ are drept caracteristică faptul că atât variabilele de intrare cât și funcția nu pot lua decât două valori distincte, ,0' sau ,1'. Funcția va pune în corespondență fiecărui element al produsului cartezian n -dimensional, valorile ,0' sau ,1'. Astfel de funcții sunt utilizate pentru

of axioms remains the same, except for some permutations.

The verification of these properties can be done using truth tables, noticing that two functions are equal if they take the same values in all the points of their definition domain. By means of the truth table a correspondence is established between the variables' truth values and the function's truth value.

Remark

- 1) Commutativity and associativity can be extended to an arbitrary, but finite number of terms, regardless to their order.
- 2) De Morgan's laws can be generalized to an arbitrary number of terms:

2.3. Boolean functions

A function $f: B^n \rightarrow B$, where $B = \{0,1\}$, is called a Boolean function. This Boolean function $y = f(x_1, x_2, \dots, x_n)$ has as main feature the fact that both the input variables and the function can only take two distinct values, '0' or '1'. The function will put in correspondence each element of the n -dimensional Cartesian product with the values '0' or '1'. Such functions are used

caracterizarea funcționării unor dispozitive (circuite) construite cu elemente de circuit având două stări (ex.: un întrerupător închis sau deschis, un tranzistor blocat sau în conductie; funcționarea unui astfel de circuit va fi descrisă de o variabilă booleană x_i).

2.3.1. Funcții Booleene elementare

Să examinăm din nou forma generală a unei funcții Booleene de n variabile: $y = f(x_1, x_2, \dots, x_n)$.

Domeniul de definiție este format din $m = 2^n$ puncte. Deoarece în fiecare din aceste puncte funcția poate lua doar valorile '0' și '1', rezultă că numărul total al funcțiilor Booleene de n variabile este $N = 2^m$.

Toate funcțiile Booleene se pot realiza cu ajutorul unor funcții de bază. Acestea le vor corespunde și niște circuite logice elementare, cu ajutorul cărora se poate construi practic orice circuit. Înțînd cont de faptul că circuitele logice de comutație au 2 stări stabile *LOW* (L) și *HIGH* (H), atribuind lui L $\leftarrow 0$ și lui H $\leftarrow 1$ se poate întocmi un tabel al funcțiilor elementare.

for characterizing the functioning of some devices (circuits) built from two-state circuit elements (i.e. a switch that is open or closed, a transistor that is locked or conducting; the functioning of such a circuit will be described by a Boolean variable x_i).

2.3.1. Elementary Boolean functions

Let's consider again the general form of an n -variable Boolean function: $y = f(x_1, x_2, \dots, x_n)$.

The definition domain is composed of $m = 2^n$ points. Since in each one of these points the function can only take the values '0' and '1', the total number of n -variable Boolean functions is $N = 2^m$.

All the Boolean functions can be realized with some basic functions. These functions also have associated some elementary logic circuits, with the help of whom one can build practically any circuit.

Since the switching (logic) circuits have two stable states *LOW* (L) and *HIGH* (H), by assigning L $\leftarrow 0$ and H $\leftarrow 1$ one can create a table of the elementary functions.

Denumire (Name)	Funcție (Function)	Simbol (Symbol)	Tabel de adevăr (Truth table)	Tabel de definiție (Definition table)																														
Inversor - NU (Inverter - NOT)	$f = \bar{a}$		<table border="1"> <tr><td>a</td><td>f</td></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>	a	f	0	1	1	0	<table border="1"> <tr><td>a</td><td>f</td></tr> <tr><td>L</td><td>H</td></tr> <tr><td>H</td><td>L</td></tr> </table>	a	f	L	H	H	L																		
a	f																																	
0	1																																	
1	0																																	
a	f																																	
L	H																																	
H	L																																	
Poartă řI (AND gate)	$f(a,b) = a \bullet b$		<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	f	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>L</td><td>L</td><td>L</td></tr> <tr><td>L</td><td>H</td><td>L</td></tr> <tr><td>H</td><td>L</td><td>L</td></tr> <tr><td>H</td><td>H</td><td>H</td></tr> </table>	a	b	f	L	L	L	L	H	L	H	L	L	H	H	H
a	b	f																																
0	0	0																																
0	1	0																																
1	0	0																																
1	1	1																																
a	b	f																																
L	L	L																																
L	H	L																																
H	L	L																																
H	H	H																																
Poartă SAU (OR gate)	$f(a,b) = a + b$		<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	f	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>L</td><td>L</td><td>L</td></tr> <tr><td>L</td><td>H</td><td>H</td></tr> <tr><td>H</td><td>L</td><td>H</td></tr> <tr><td>H</td><td>H</td><td>H</td></tr> </table>	a	b	f	L	L	L	L	H	H	H	L	H	H	H	H
a	b	f																																
0	0	0																																
0	1	1																																
1	0	1																																
1	1	1																																
a	b	f																																
L	L	L																																
L	H	H																																
H	L	H																																
H	H	H																																
Poartă řI-NU (NAND gate)	$f(a,b) = \overline{a \bullet b}$		<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	a	b	f	0	0	1	0	1	1	1	0	1	1	1	0	<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>L</td><td>L</td><td>H</td></tr> <tr><td>L</td><td>H</td><td>H</td></tr> <tr><td>H</td><td>L</td><td>H</td></tr> <tr><td>H</td><td>H</td><td>L</td></tr> </table>	a	b	f	L	L	H	L	H	H	H	L	H	H	H	L
a	b	f																																
0	0	1																																
0	1	1																																
1	0	1																																
1	1	0																																
a	b	f																																
L	L	H																																
L	H	H																																
H	L	H																																
H	H	L																																
Poartă SAU-NU (NOR gate)	$f(a,b) = \overline{a + b}$		<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	a	b	f	0	0	1	0	1	0	1	0	0	1	1	0	<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>L</td><td>L</td><td>H</td></tr> <tr><td>L</td><td>H</td><td>L</td></tr> <tr><td>H</td><td>L</td><td>L</td></tr> <tr><td>H</td><td>H</td><td>L</td></tr> </table>	a	b	f	L	L	H	L	H	L	H	L	L	H	H	L
a	b	f																																
0	0	1																																
0	1	0																																
1	0	0																																
1	1	0																																
a	b	f																																
L	L	H																																
L	H	L																																
H	L	L																																
H	H	L																																
Poartă SAU EXCLUSIV (XOR gate)	$f(a,b) = a \oplus b$		<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	a	b	f	0	0	0	0	1	1	1	0	1	1	1	0	<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>L</td><td>L</td><td>L</td></tr> <tr><td>L</td><td>H</td><td>H</td></tr> <tr><td>H</td><td>L</td><td>H</td></tr> <tr><td>H</td><td>H</td><td>L</td></tr> </table>	a	b	f	L	L	L	L	H	H	H	L	H	H	H	L
a	b	f																																
0	0	0																																
0	1	1																																
1	0	1																																
1	1	0																																
a	b	f																																
L	L	L																																
L	H	H																																
H	L	H																																
H	H	L																																
Poartă COINCIDENTĂ (XNOR gate)	$f(a,b) = a \otimes b$		<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	a	b	f	0	0	1	0	1	0	1	0	0	1	1	1	<table border="1"> <tr><td>a</td><td>b</td><td>f</td></tr> <tr><td>L</td><td>L</td><td>H</td></tr> <tr><td>L</td><td>H</td><td>L</td></tr> <tr><td>H</td><td>L</td><td>L</td></tr> <tr><td>H</td><td>H</td><td>H</td></tr> </table>	a	b	f	L	L	H	L	H	L	H	L	L	H	H	H
a	b	f																																
0	0	1																																
0	1	0																																
1	0	0																																
1	1	1																																
a	b	f																																
L	L	H																																
L	H	L																																
H	L	L																																
H	H	H																																

2.3.2. Reprezentarea funcțiilor Booleene

Există două modalități de reprezentare a funcțiilor Booleene:

1. *Modalități grafice* – se caracterizează printr-o reprezentare intuitivă, ușor de reținut, dar sunt inadecvate pentru funcții Booleene cu un număr de variabile mai mare decât 4;
2. *Modalități analitice* – sunt mai greoaie, dar permit metode automate, deci algoritmi de simplificare a funcției; se folosesc în general pentru funcții Booleene cu mai mult de 5 variabile.

Modalități de reprezentare grafică: tabel de adevăr, diagramă Karnaugh, schemă logică, diagramă de timp.

1. Tabel de adevăr – se marchează într-un tabel corespondența dintre valorile de adevăr ale variabilelor de intrare și valoarea de adevăr a funcției, în fiecare punct al domeniului de definiție. Pentru o funcție cu n variabile de intrare vom avea 2^n combinații.

Există situații în care, pentru anumite combinații ale variabilelor de intrare, valoarea funcției nu este specificată. Aceste funcții se numesc *incomplet definite*. În tabel, în locul în care funcția nu este specificată, se notează cu “X” (indiferent – *don't care*).

2.3.2. Boolean functions representation

There are two ways of representing Boolean functions:

1. *Graphical ways* – characterized by an intuitive, easy to understand representation, but inadequate for Boolean functions of more than four variables.
2. *Analytical ways* – more difficult to understand, but allowing automatic methods, i.e. function simplification algorithms; generally used for Boolean functions of more than five variables.

Graphical representation ways: truth table, Karnaugh map, state diagram, waveform.

1. Truth table -- we will mark in a table the correspondence between the truth values of the input variables, in each point of the definition domain. For an n -variable function we will have 2^n combinations.

There are situations in which, for some input variables' combinations, the function's value is not specified. These functions are called *incompletely defined*. In the table, the locations where the function is not specified are marked with an “X” (*don't care*). If a Boolean

Dacă o funcție Booleană este incomplet definită pentru m combinații ale variabilelor de intrare, atunci se pot defini 2^m funcții distincte prin alegerea arbitrară a valorilor incomplet definite.

2. Diagramă Karnaugh – O diagramă Karnaugh a unei funcții Booleene de n variabile se desenează sub forma unui pătrat sau dreptunghi împărțit în 2^n celule sau compartimente. Fiecare compartiment este rezervat unui termen canonic al funcției. Diagrama Karnaugh este organizată astfel încât două compartimente vecine pe o linie sau pe o coloană corespund la doi termeni canonici care diferă printr-o singură variabilă, care apare în unul adevărată, iar în celălalt negată (la două n -uple adiacente). Se consideră vecine și compartimentele aflate la capetele unei linii, respectiv coloane.

Diagrama Karnaugh se adnotează indicând pe linie și coloană n -uple de zerouri și unități, corespunzătoare unui compartiment din diagramă, precum și ordinea variabilelor. De asemenea, se indică domeniul fiecărei variabile. Numerotarea liniilor și coloanelor se face în cod Gray.

Pentru a putea reprezenta ușor funcții exprimate în mod convențional prin indicii termenilor canonici se poate nota fiecare

function is incompletely defined for m combinations of the input variables, then we can define 2^m distinct functions by arbitrarily choosing the incompletely defined values.

2. Karnaugh map – A Karnaugh map of an n -variable Boolean function is drawn as a square or as a rectangle divided in 2^n cells. Each cell is reserved for a canonical term of the function. The Karnaugh map is organized such as two neighbor cells on a row or on a column correspond to two canonical terms that differ by only one variable, that appears in its TRUE form in one cell and in its NEGATED form in the other one (at two adjacent n -uples). We will also consider as adjacent the neighbor cells located at the ends of a row and of a column, respectively.

The Karnaugh map is labeled by indicating on each row and column n -uples of zeroes and units that correspond to a cell in the diagram, and also the variables' order. It is also necessary to indicate the domain of each variable. The numbering of the rows and columns is done in Gray code.

In order to be able to easily represent functions expressed conventionally by the indexes of the canonical terms, we can denote each

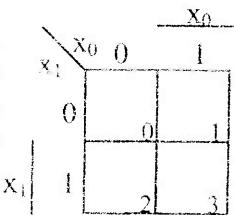
compartiment cu indicele termenului corespondent, ținând cont de o ordine dată a variabilelor.

Exemplu:

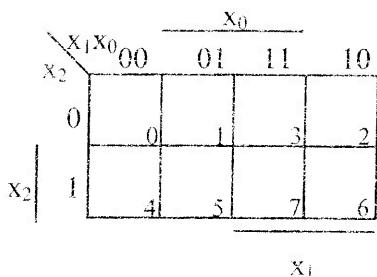
1) Tabelul de adevăr și diagrama Karnaugh pentru funcția de 2 variabile:

$$f(x_1, x_0) = \overline{x_1} \cdot x_0 + x_1 \cdot \overline{x_0}$$

x_1	x_0	f
0	0	0
0	1	1
1	0	1
1	1	0



2) Diagrama Karnaugh pentru funcția de 3 variabile: $y = f(x_2, x_1, x_0)$



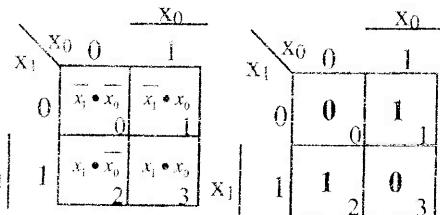
3) Diagrama Karnaugh pentru funcția de 4 variabile $y = f(x_3, x_2, x_1, x_0)$ – prin săgeți am marcat vecinătățile punctelor de coordonate 0010 și 0101.

cell by the index of its corresponding term, taking into account the given variables order.

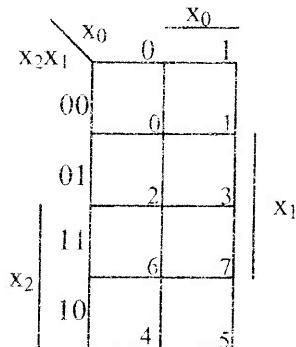
Examples:

1) Truth table and Karnaugh map for the 2-variable function:

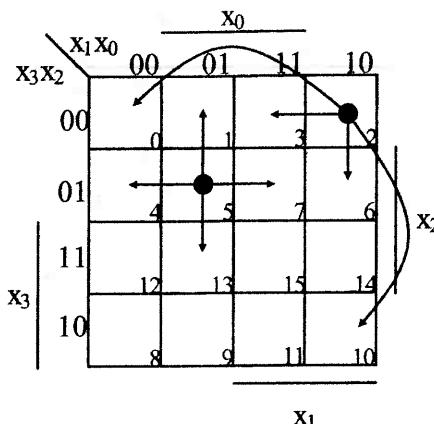
$$f(x_1, x_0) = \overline{x_1} \cdot x_0 + x_1 \cdot \overline{x_0}$$



2) The Karnaugh map for the 3-variable function: $y = f(x_2, x_1, x_0)$



3) The Karnaugh map for the 4-variable function: $y = f(x_3, x_2, x_1, x_0)$ – we marked by arrows the neighbors of the cells of coordinates 0010 and 0101.



Diagramele Karnaugh pentru funcții de mai mult de 4 variabile se construiesc din diagrame de 4 variabile considerate ca diagrame elementare.

3. Schemă logică – aceasta este o reprezentare realizată cu ajutorul simbolurilor circuitelor logice.

4. Diagramă de timp (cronogramă) – aceasta este o reprezentare utilă pentru studiul unor forme tranzitorii de hazard în circuitele logice. Se reprezintă funcții logice în a căror evoluție intervine factorul timp.

Modalități de reprezentare analitică: principalele modalități sunt formele canonice și formele minime.

Fie o funcție Booleană $f(X)$, unde $X = (x_1, x_2, \dots, x_n)$. Se definește

The Karnaugh maps for functions of more than 4 variables are built from 4-variable Karnaugh maps considered as elementary maps.

3. State diagram – this is a representation made with the symbols of the logic circuits.

4. Waveform diagram (chronogram) – this representation is useful for studying some transient hazard forms in logic circuits. We represent logic functions in whose evolution time is an important factor.

Analytical representation ways: the main ways are the canonical forms and the minimized forms.

Be given a Boolean function $f(X)$, where $X = (x_1, x_2, \dots, x_n)$. We define

numărul $i = x_1 \cdot 2^0 + x_2 \cdot 2^1 + \dots + x_n \cdot 2^{n-1}$ ca număr de combinație.

Fie funcția $\mathcal{P}_i: B^n \rightarrow B$ (unde $B = \{0,1\}$), $\mathcal{P}_i(x_1, x_2, \dots, x_n) = 1$ dacă numărul de combinație este i , 0 în caz contrar.

Această funcție se numește *constituuent al unității*. Se poate arăta că orice funcție Booleană dată prin tabelul de adevăr se poate scrie ca o sumă de constituenți ai unității:

$$f(x_1, x_2, \dots, x_n) = \mathcal{P}_{i1} + \mathcal{P}_{i2} + \dots + \mathcal{P}_{ip} = \sum_{i, j \in M_1} \mathcal{P}_{ij}.$$

unde M_1 este mulțimea tuturor combinațiilor argumentelor pentru care funcția ia valoarea 1. Această formă de scriere se numește *forma canonica disjunctivă FCD*, iar termenii constituenți se numesc *mintermi* (se mai numește și forma *sumă de produse*).

Fie acum funcția $S_i: B^n \rightarrow B$, $S_i(x_1, x_2, \dots, x_n) = 0$ dacă numărul de combinație este i , 1 în caz contrar.

Se poate demonstra că orice funcție Booleană poate fi adusă la forma:

$$f(x_1, x_2, \dots, x_n) = S_{i1} \bullet S_{i2} \bullet \dots \bullet S_{iq} = \prod_{i, j \in M_0} S_{ij}$$

unde M_0 este mulțimea tuturor combinațiilor argumentelor pentru care funcția ia valoarea 0. Această formă de scriere se numește *forma canonica conjunctivă FCC*, iar factorii

the number $i = x_1 \cdot 2^0 + x_2 \cdot 2^1 + \dots + x_n \cdot 2^{n-1}$ as the combination number.

Be given the function $\mathcal{P}_i: B^n \rightarrow B$ (where $B = \{0,1\}$), $\mathcal{P}_i(x_1, x_2, \dots, x_n) = 1$ if the combination number is i , otherwise 0.

This function is called *one's constituent*. It can be proven that any Boolean function given by its truth table can be written as a sum of *one's constituents*:

where M_1 is the set of all the arguments' combinations for which the function takes the value 1. This way of writing the function is called the *disjunctive canonical form DCF*, and the constituent terms are called *minterms* (it is also called the *sum of products* form).

Let's consider now the function $S_i: B^n \rightarrow B$, $S_i(x_1, x_2, \dots, x_n) = 0$ if the combination number is i , otherwise 1.

It can be proven that any Boolean function can be brought to the form:

where M_0 is the set the set of all the arguments' combinations for which the function takes the value 0. This way of writing the function is called the *conjunctive canonical form*

constituenți se numesc termeni canonici conjunctivi sau *maxtermi* (se mai numește și forma produs de sume). Se poate demonstra că $S_i = \overline{P}_i$.

Algoritmi de obținere a formelor canonice pe baza tabelului de adevăr sau a diagramei Karnaugh:

Forma canonică disjunctivă

- se determină toate combinațiile variabilelor pentru care valoarea funcției este 1;
- se scriu mintermii corespunzători (o variabilă apare nenegată dacă are valoarea 1 și negată dacă are valoarea 0);
- se însumează mintermii obținuți anterior.

Forma canonică conjunctivă

- se determină toate combinațiile variabilelor pentru care valoarea funcției este 0;
- se scriu maxtermii corespunzători prin însumarea variabilelor (o variabilă apare nenegată dacă are valoarea 0 și negată dacă are valoarea 1);
- se înmulțesc maxtermii obținuți anterior.

CCF, and the constituent terms are called *maxterms* (it is also called the *product of sums* form).

It can be proven that $S_i = \overline{P}_i$.

Algorithms for obtaining the canonical forms based on the truth table or on the Karnaugh map:

Disjunctive canonical form

- determine all the variables' combinations for which the function's value is 1;
- write the corresponding minterms (a variable appears not negated if it has the value 1 and negated if it has the value 0);
- sum the previously obtained minterms.

Conjunctive canonical form

- determine all the variables' combinations for which the function's value is 0;
- write the corresponding maxterms (a variable appears not negated if it has the value 0 and negated if it has the value 1);
- multiply the previously obtained maxterms.

Exemplu:

x_2	x_1	x_0	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Example:

$$\text{minterm: } \overline{x_2} \bullet \overline{x_1} \bullet \overline{x_0}$$

$$\text{maxterm: } x_2 + x_1 + \overline{x_0}$$

$$\text{maxterm: } x_2 + \overline{x_1} + x_0$$

$$\text{minterm: } \overline{x_2} \bullet x_1 \bullet x_0$$

$$\text{maxterm: } \overline{x_2} + x_1 + x_0$$

$$\text{minterm: } x_2 \bullet \overline{x_1} \bullet x_0$$

$$\text{maxterm: } \overline{x_2} + \overline{x_1} + x_0$$

$$\text{minterm: } x_2 \bullet x_1 \bullet x_0$$

$$f_{FCD} = \overline{x_2} \bullet \overline{x_1} \bullet \overline{x_0} + \overline{x_2} \bullet x_1 \bullet x_0 + x_2 \bullet \overline{x_1} \bullet x_0 + x_2 \bullet x_1 \bullet x_0.$$

$$f_{FCD} = \Sigma(0, 3, 5, 7)$$

$$f_{FCD} = P_0 + P_3 + P_5 + P_7.$$

$$f_{FCC} = (x_2 + x_1 + \overline{x_0}) \bullet (x_2 + \overline{x_1} + x_0) \bullet (\overline{x_2} + x_1 + x_0) \bullet (\overline{x_2} + \overline{x_1} + x_0).$$

$$f_{FCD} = \prod(1, 2, 4, 6)$$

$$f_{FCD} = S_1 \bullet S_2 \bullet S_4 \bullet S_6.$$

2.3.3. Minimizarea funcțiilor Booleene

Algebra Booleană se folosește la analiza și sinteza dispozitivelor numerice (circuite de comutație). Între gradul de complexitate al circuitului și cel al funcției care îl descrie există o legătură directă. Aceasta este motivația pentru care, în *etapa de sinteză* a circuitelor de comutație, după definirea acestora,

2.3.3. Minimizing Boolean functions

Boolean algebra is used for the analysis and the synthesis of digital devices (switching circuits). There is a direct link between the level of complexity of the circuit and the level of complexity of the function that describes it. This is why, in the switching circuits *synthesis step*, after their definition, it is mandatory

urmează în mod obligatoriu etapa de minimizare a funcției, având drept scop obținerea unor forme echivalente mai simple (forma minimă).

Prin aplicarea metodelor de minimizare (de simplificare) se ajunge la expresii minimale sub forma unor sume de produse ori a unor produse de sume.

Criteriile utilizate în vederea minimizării sunt:

- reducerea numărului de variabile;
- reducerea numărului de termeni;
- reducerea pe ansamblu a variabilelor și termenilor, astfel ca suma lor să devină minimă.

Minimizarea constă în principal în transformarea formelor canonice și a formelor elementare parțial simplificate în forme elementare minimale.

Metodele de minimizare pot fi grupate în metode algebrice și metode grafice.

Minimizarea prin metode algebrice

Aceste metode se bazează pe axiomele și pe proprietățile algebrei Booleene. Ele sunt adecvate pentru funcții cu număr mare de variabile (mai mult de 4). Deși aplicarea lor este mai greoaiă pentru un proiectant uman, ele se pretează foarte bine implementării pe calculator.

to continue with a function's minimization step, whose aim is to obtain simpler equivalent forms (the minimal form).

By applying the minimization (simplification) methods we obtain minimal expressions in the form of sums of products or product of sums.

The criteria used for minimization are the following:

- minimization of the number of variables;
- minimization of the number of terms;
- global minimization of the variables and the terms, so that their sum becomes minimal.

Minimization mainly consists of transforming the canonical forms and the partially simplified elementary forms in elementary minimal forms.

The minimization methods can be grouped in algebraic methods and graphical methods.

Algebraic methods for minimization

These methods are based on Boolean algebra's axioms and properties. They are appropriate for functions of many variables (more than 4). Even though their application is somehow more difficult for a human designer, they are very adequate for a computer implementation.

Minimizarea pe baza metodei Quine-McCluskey

Etapele de minimizare prin această metodă sunt următoarele:

1. Se construiește un tabel cu 3 coloane: „termeni canonici”, „repräsentare binară” și „index”. Se grupează termenii canonici astfel încât termenii din fiecare grupă să conțină același număr de biți de 1, respectiv 0. Numărul de biți de 1 din grupă dă indexul grupei respective.
2. Se compară fiecare termen din grupa de index i cu toți cei din grupa de index $i+1$. De fiecare dată când va fi posibil, se va realiza combinarea între 2 termeni pe baza relației: $x_1 \bullet x_2 + x_1 \bullet \overline{x_2} = x_1$. Condiția pentru a putea combina 2 termeni este ca aceștia să difere printr-o singură poziție binară. Pe poziția respectivă, în urma combinării, se va trage o linie $-$. Se bifează toți termenii care au fost comparați și care au participat la cel puțin o operație de combinare.
3. Termenii obținuți în pasul 2 se vor compara în aceeași manieră; un nou termen va fi generat prin combinarea a 2 termeni care diferă printr-o singură poziție binară și la care semnul $,$ $-$ este pe aceeași poziție. Procesul va continua până

Minimizing based on the Quine-McCluskey method

The minimization steps by this method are the following:

1. Build a three-column table: “canonical terms”, “binary representation” and “index”. Group the canonical terms in such manner that the terms from each group contain the same number of bits of 1, respectively 0. The number of bits of 1 from the group gives the index of that group.
2. Compare each term from the group of index i with all the terms from the group of index $i+1$. Each time it will be possible, combine two terms according to the relationship: $x_1 \bullet x_2 + x_1 \bullet \overline{x_2} = x_1$. The condition for combining two terms is that they differ by only one binary position. On that position, after the combining operation, a line will be drawn $-$. Mark all the terms that have been compared and participated in at least one combining operation.
3. The terms obtained in step 2 will be compared in the same manner; a new term will be generated by combining 2 terms that differ by only one binary position and for which the ‘ $-$ ’ sign is in the same position. The process will continue until all possible combinations are

când se epuizează toate combinațiile posibile. Termenii rămași nebifați constituie setul de *implicanți primi* ai funcției. Expresia lor algebrică se deduce cu ajutorul regulilor de la forma canonică disjunctivă. Unii dintre *implicanții primi* sunt *esențiali*: ei acoperă cel puțin un minterm al funcției care nu mai este acoperit de nici un alt implicant prim – prin urmare, ei nu trebuie să lipsească din expresia minimizată a funcției.

Vom exemplifica minimizarea prin metoda Quine-McCluskey pe următoarea funcție:

$$f = \sum (0,1,2,3,7,14,15,22,23,29,31)$$

Etapa 1

Crearea tabelului:

Versiunea 1

Termeni canonici (Canonical terms)	x ₄	x ₃	x ₂	x ₁	x ₀	Index
0 ✓	0	0	0	0	0	0
1 ✓	0	0	0	0	1	1
2 ✓	0	0	0	1	0	2
3 ✓	0	0	0	1	1	3
7 ✓	0	0	1	1	1	4
14 ✓	0	1	1	1	0	5
22 ✓	1	0	1	1	0	6
15 ✓	0	1	1	1	1	7
23 ✓	1	0	1	1	1	8
29 ✓	1	1	1	0	1	9
31 ✓	1	1	1	1	1	10

performed. The terms that remained unmarked constitute the set of *prime implicants* of the function. Their algebraic expression is deduced using the rules from the disjunctive canonical form. Some of the *prime implicants* are *essential*: they cover at least one minterm of the function that is not covered by any other prime implicant – as a consequence, they must not miss from the function's minimized expression.

We will show the minimization by the Quine-McCluskey method on the following function:

Step 1

Creating the table:

Version 1

Etapa 2

Efectuarea iterărilor, prin combinarea termenilor. Se obține o a 2-a versiune a tabelului:

*Versiunea 2*Step 2

Performing the iterations, by combining the terms. We obtain a second version of the table:

Version 2

Termeni canonici (Canonical terms)	x_4	x_3	x_2	x_1	x_0	Index
(0, 1)	✓	0	0	0	—	0
(0, 2)	✓	0	0	0	—	
(1, 3)	✓	0	0	—	1	1
(2, 3)	✓	0	0	0	1	
(3, 7)	0	0	—	1	1	2
(7, 15)	✓	0	—	1	1	3
(7, 23)	✓	—	0	1	1	
(14, 15)	0	1	1	1	—	
(22, 23)	1	0	1	1	—	
(15, 31) ✓	—	1	1	1	1	4
(23, 31) ✓	1	—	1	1	1	
(29, 31)	1	1	1	—	1	

Observație: Marcajul „✓” a fost adăugat în Etapa 2 în celulele primei versiuni a tabelului.

Iterările continuă până când nu mai putem combina nici o pereche de termeni canonici.

Versiunea 3

Remark: The “✓” mark was added in Step 2 in the cells of the first version of the table.

The iterations continue until we can not combine any other pair of canonical terms.

Version 3

Termeni canonici (Canonical terms)	x_4	x_3	x_2	x_1	x_0	Index
(0, 1, 2, 3)	0	0	0	—	—	0
(0, 2, 1, 3)	0	0	0	—	—	
(7, 15, 23, 31)	—	—	1	1	1	3
(7, 23, 15, 31)	—	—	1	1	1	

Observație: Marcajul „√” a fost adăugat în Etapa 2 în celulele celei de-a doua versiuni a tabelului, în timpul obținerii versiunii nr. 3.

Se observă că iterațiile se încheie aici.

Apare întrebarea: are rost să scriem termenii $(0, 1, 2, 3)$ și $(0, 2, 1, 3)$, respectiv $(7, 15, 23, 31)$ și $(7, 23, 15, 31)$, de vreme ce se vede că reprezintă același lucru?

Răspunsul este DA, din 2 motive:

1. Dacă am putut combina termenii $(0, 1)$ și $(2, 3)$, înseamnă că trebuie să putem combina și termenii $(0, 2)$ și $(1, 3)$. Aceasta este un *criteriu de verificare*. În plus, expresia din coloana a 2-a trebuie să fie identică.
2. Este necesar să luăm aceste perechi, aparent redundante, pentru a bifa toți termenii anteriori.

Vom nota acum termenii rămași nebifați *pe parcursul întregului algoritm*, care constituie setul de implicantă primă, astfel:

$$(3, 7)$$

$$(14, 15)$$

$$(22, 23)$$

$$(29, 31)$$

$$(0, 1, 2, 3)$$

$$(7, 15, 23, 31)$$

$$a = \overline{x_4} \bullet \overline{x_3} \bullet x_1 \bullet x_0$$

$$b = \overline{x_4} \bullet x_3 \bullet x_2 \bullet x_1$$

$$c = x_4 \bullet \overline{x_3} \bullet x_2 \bullet x_1$$

$$d = x_4 \bullet x_3 \bullet x_2 \bullet x_0$$

$$e = \overline{x_4} \bullet \overline{x_3} \bullet \overline{x_2}$$

$$f = x_2 \bullet x_1 \bullet x_0$$

Remark: The „√” mark was added in Step 2 in the cells of the second version of the table, while obtaining version 3.

One can notice that iterations are finishing here.

An important question arises: is it worthy to write the terms $(0, 1, 2, 3)$ and $(0, 2, 1, 3)$, and also $(7, 15, 23, 31)$ and $(7, 23, 15, 31)$, since it is obvious that they represent the same thing?

The answer is YES, for two reasons:

1. If we were able to combine the terms $(0, 1)$ and $(2, 3)$, this means we must be able to combine also the terms $(0, 2)$ and $(1, 3)$. This is a *verification criterion*. In addition, the expressions from the second column must be identical.
2. It is necessary to take these apparently redundant pairs in order to mark all the previous terms.

Now we will denote the terms that remained unmarked *during the whole algorithm*, which constitute the set of prime implicants, as follows:

Acum trebuie să construim *tabelul acoperirilor*, în care indicăm ce termen canonic (TC) este acoperit de către fiecare implicant prim (IP):

TC (CT)	0	1	2	3	7	14	15	22	23	29	31
IP (PI)											
a				*	*						
b						*	*				
c								*	*		
d										*	*
e	*	*	*	*							
f					*		*		*		*

Observăm că termenii canonici 0, 1 și 2 sunt acoperiți doar de către implicantul prim *e*, termenul canonic 14 este acoperit doar de către implicantul prim *b*, termenul canonic 22 este acoperit doar de către implicantul prim *c*, iar termenul canonic 29 este acoperit doar de către implicantul prim *d*. Așadar, *b*, *c*, *d* și *e* sunt *implicanți primi esențiali*.

Atunci, expresia funcției este:

$$f_1 = b + c + d + e + a, \text{ sau}$$

$$f_2 = b + c + d + e + f$$

Remarcând că expresia lui *f* este mai simplă decât a lui *a*, rezultă că soluția optimă este *f*₂.

Minimizarea prin metode grafice

Minimizarea pe baza diagramei Karnaugh

Această metodă de minimizare se

Now we must build the coverage table, in which we indicate which canonical term (CT) is covered by each prime implicant (PI):

We notice that the canonical terms 0, 1 and 2 are covered only by the prime implicant *e*, the canonical term 14 is covered only by the prime implicant *b*, the canonical term 22 is covered only by the prime implicant *c*, and the canonical term 29 is covered only by the prime implicant *d*. So, *b*, *c*, *d* and *e* are *essential prime implicants*.

Then, the function's expression is:

$$f_1 = b + c + d + e + a, \text{ or}$$

$$f_2 = b + c + d + e + f$$

Noticing that *f*'s expression is simpler than *a*'s expression, we deduce that the optimal solution is *f*₂.

Graphical minimization

Karnaugh map-based minimization

This minimization method is based on the adjacency property of the

bazează pe proprietatea de adiacență a codului Gray, cu ajutorul căruia se numerotează liniile și coloanele diagramei Karnaugh. În vederea minimizării se aleg suprafețele maxime (sub-cuburi) formate din constituentei ai unității, respectiv din constituentei ai lui 0, suprafețe (sub-cuburi) având ca dimensiune un număr de pătrate (compartimente) egal întotdeauna cu o putere a lui 2. Aceste suprafețe vor corespunde termenilor canonici, termenii vecini fiind adiacenți (diferă printr-un singur bit). Ca urmare, în urma grupării lor se vor reduce variabilele pe baza relației de absorbtie:

$$x_1 \bullet x_2 + x_1 \bullet \overline{x_2} = x_1.$$

Metoda propriu-zisă de minimizare este următoarea:

- se realizează grupări de compartimente (sub-cuburi) care sunt puteri ale lui 2. Un compartiment poate fi membru al mai multor suprafețe. O suprafață cu 2^m celule vecine va elimina 2^m variabile de intrare din expresia suprafeței respective.
- se scriu ecuațiile corespunzătoare fiecărei suprafețe, obținându-se astfel termenii elementari.
- se realizează forma disjunctivă minimă (FDM) prin însumarea termenilor elementari obținuți prin gruparea constituenteilor lui 1, sau forma conjunctivă minimă (FCM) prin înmulțirea termenilor

Gray code, which helps labeling the rows and columns of the Karnaugh map. In order to perform the minimization, one must choose the maximal surfaces (sub-cubes) formed by one's constituents, respectively by zero's constituents. These surfaces (sub-cubes) must contain a number of cells that is always equal to a power of 2. These surfaces correspond to the canonical terms, the neighbor terms being adjacent (they differ by only one bit). As a matter of consequence, after grouping them, the variables will be reduced thanks to the absorption relationship:

$$x_1 \bullet x_2 + x_1 \bullet \overline{x_2} = x_1.$$

The minimization method is as follows:

- make groups of cells (sub-cubes), in each group having a number of cells that is equal to a power of 2. A cell can be part of several surfaces. A surface with 2^m neighbor cells will eliminate 2^m input variables from the surface's expression.
- write the equations that correspond to each surface, thus obtaining the elementary terms.
- create the minimal disjunctive form (MDF) by summing the elementary terms obtained by grouping the one's constituents, or the minimal conjunctive form (MCF) by multiplying the elementary terms obtained by

elementari obținuți prin gruparea de constituente ai lui 0; funcțiile minime obținute sunt identice, ele diferind numai prin forma de prezentare.

Observație: Expresiile suprafețelor obținute în urma minimizării prin metoda bazată pe diagrame Karnaugh corespund implicantilor primi din cadrul metodei de minimizare Quine-McCluskey.

Exemplu: Să se minimizeze funcția

$$f = \sum(2,7,8,9,12,13,15)$$

a) Minimizarea în FCD

Urmărim să realizăm grupări maxime de constituente ai lui 1, care să conțină fiecare un număr de celule egal cu o putere a lui 2. Astfel se obțin suprafețele α , β , γ , δ și φ . Observăm că, pentru a acoperi toate celulele cu constituente ai lui 1, sunt necesarele suprafețele α , β , γ și δ , SAU α , β , γ și φ . Cele două variante sunt la fel de corecte, constituentul lui 1 din celula 13 (singurul rămas neacoperit după formarea suprafețelor α , β și γ) putând fi grupat fie cu constituentul lui 1 din celula 15, fie cu cel din celula 9.

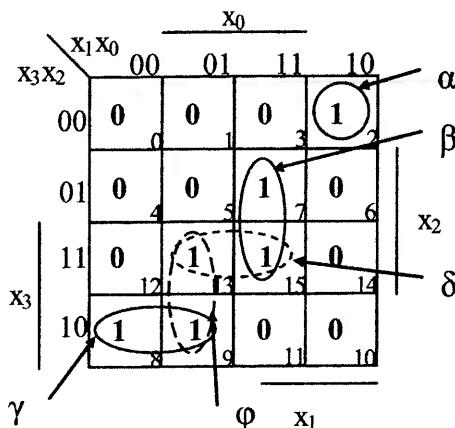
grouping the zero's constituents; the obtained minimal functions are identical, the difference between them being only their presentation form.

Remark: The expressions of the surfaces obtained after minimizing by the method based on Karnaugh maps correspond to the prime implicants from the Quine-McCluskey minimization method.

Example: Minimize the function

a) Minimizing in the DCF

We aim at making maximal groups of 1's constituents that contain each a number of cells equal to a power of 2. This way, we obtain the surfaces α , β , γ , δ and φ . We notice that, in order to cover all cells with 1's constituents, we need the surfaces α , β , γ and δ , OR α , β , γ and φ . The two variants are equally correct, the 1's constituent from cell 13 (the only one left uncovered after forming the surfaces α , β and γ) can be grouped either with the 1's constituent from cell 15 or with the 1's constituent from cell 9.



Pentru a determina expresia unei suprafețe, se pune întrebarea: „variabila x_i acoperă oare suprafața?”

Există doar trei variante:

- variabila x_i acoperă integral suprafața. În acest caz, variabila x_i apare adevărată în expresia suprafeței;

- variabila x_i nu acoperă deloc suprafața. În acest caz, variabila x_i apare negată în expresia suprafeței;

- variabila x_i acoperă doar jumătate din suprafață. În acest caz, variabila x_i dispare din expresia suprafeței.

În exemplul nostru, obținem următoarele expresii:

$$\alpha = \overline{x_3} \cdot \overline{x_2} \cdot x_1 \cdot \overline{x_0} \quad - \text{suprafața } \alpha \\ \text{conține } 2^0 \text{ celule (celula 2), deci} \\ \text{elimină 0 variabile de intrare din} \\ \text{expresie.}$$

$$\beta = x_2 \cdot x_1 \cdot x_0 \quad - \text{suprafața } \beta \text{ conține}$$

To determine the expression of a surface, we must ask the following question: "does the variable x_i cover the surface?" There are only three variants:

- variable x_i covers completely the surface. In this case, the variable x_i appears true in the surface's expression;

- variable x_i does not cover at all the surface. In this case, the variable x_i appears negated in the surface's expression;

- variable x_i covers only half of the surface. In this case, the variable x_i disappears from the surface's expression.

In our example, we obtain the following expressions:

$$\alpha = \overline{x_3} \cdot \overline{x_2} \cdot x_1 \cdot \overline{x_0} \quad - \text{surface } \alpha \\ \text{contains } 2^0 \text{ cells (cell 2), so it} \\ \text{eliminates 0 input variables from the} \\ \text{expression.}$$

2^1 celule (celulele 7 și 15), deci elimină 1 variabilă de intrare din expresie.

$$\gamma = x_3 \bullet \overline{x_2} \bullet \overline{x_1} \text{ -- suprafața } \gamma \text{ conține}$$

2^1 celule (celulele 8 și 9), deci elimină 1 variabilă de intrare din expresie.

$$\delta = x_3 \bullet x_2 \bullet x_0 \text{ -- suprafața } \delta \text{ conține}$$

2^1 celule (celulele 13 și 15), deci elimină 1 variabilă de intrare din expresie.

$$\varphi = x_3 \bullet \overline{x_1} \bullet x_0 \text{ -- suprafața } \varphi \text{ conține}$$

2^1 celule (celulele 9 și 13), deci elimină 1 variabilă de intrare din expresie.

Expresia finală a lui f este:

$$f' = \alpha + \beta + \gamma + \delta \text{ sau}$$

$$f'' = \alpha + \beta + \gamma + \varphi.$$

Variantele f' și f'' sunt egale din punct de vedere al costului implementării.

b) Minimizarea în FCC

Urmărим să realizăm grupări maxime de constituenți ai lui 0, care să conțină fiecare un număr de celule egal cu o putere a lui 2. Astfel se obțin suprafețele α , β , γ , δ și φ . Observăm că, pentru a acoperi toate celulele cu constituenți ai lui 0, sunt necesarele suprafețele α , β , γ și δ , SAU α , β , γ și φ . Cele două variante sunt la fel de corecte, constituentul

$\beta = x_2 \bullet x_1 \bullet x_0$ – surface β contains 2^1 cells (cells 7 and 15), so it eliminates 1 input variable from the expression.

$$\gamma = x_3 \bullet \overline{x_2} \bullet \overline{x_1} \text{ -- suprafața } \gamma \text{ contains}$$

2^1 cells (cells 8 and 9), so it eliminates 1 input variable from the expression.

$$\delta = x_3 \bullet x_2 \bullet x_0 \text{ -- suprafața } \delta \text{ contains}$$

2^1 cells (cells 13 and 15), so it eliminates 1 input variable from the expression.

$$\varphi = x_3 \bullet \overline{x_1} \bullet x_0 \text{ -- suprafața } \varphi \text{ contains}$$

2^1 cells (cells 9 and 13), so it eliminates 1 input variable from the expression.

The final expression of f is:

$$f' = \alpha + \beta + \gamma + \delta \text{ or}$$

$$f'' = \alpha + \beta + \gamma + \varphi.$$

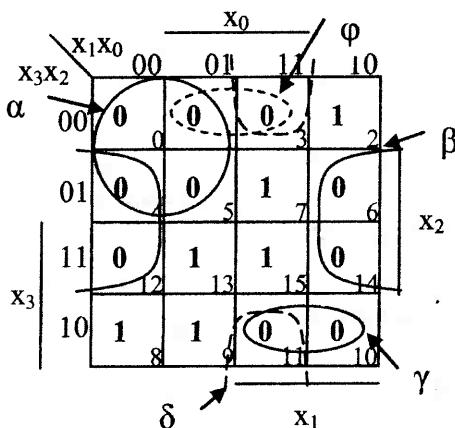
The two variants, f' and f'' , are similar as respect to the implementation cost.

b) Minimizing in the CCF

We aim at making maximal groups of 0's constituents that contain each a number of cells equal to a power of 2. This way, we obtain the surfaces α , β , γ , δ and φ . We notice that, in order to cover all cells with 1's constituents, we need the surfaces α , β , γ and δ , OR α , β , γ and φ . The two variants are equally correct, the 0's constituent from cell

lui 0 din celula 3 (singurul rămas neacoperit după formarea suprafețelor α , β și γ) putând fi grupat fie cu constituentul lui 0 din celula 1, fie cu cel din celula 11.

3 (the only one left uncovered after forming the surfaces α , β and γ) can be grouped either with the 0's constituent from cell 1 or with the 0's constituent from cell 11.



Pentru a determina expresia unei suprafețe, se pune întrebarea „variabila x_i acoperă oare suprafața?” Există doar trei variante:

- variabila x_i acoperă integral suprafața. În acest caz, variabila x_i apare negată în expresia suprafeței;
- variabila x_i nu acoperă deloc suprafața. În acest caz, variabila x_i apare adevărată în expresia suprafeței;
- variabila x_i acoperă doar jumătate din suprafață. În acest caz, variabila x_i dispare din expresia suprafeței.

În exemplul nostru, obținem expresiile:

To determine the expression of a surface, we must ask the following question: “does the variable x_i cover the surface?” There are only three variants:

- variable x_i covers completely the surface. In this case, the variable x_i appears negated in the surface's expression;
- variable x_i does not cover at all the surface. In this case, the variable x_i appears true in the surface's expression;
- variable x_i covers only half of the surface. In this case, the variable x_i disappears from the surface's expression.

In our example, we obtain the

$\alpha = x_3 + x_1$ – suprafața α conține 2^2 celule (celulele 0, 1, 4 și 5), deci elimină 2 variabile de intrare din expresie.

$\beta = \overline{x_2} + x_0$ – suprafața β conține 2^2 celule (celulele 4, 12, 6 și 14), deci elimină 2 variabile de intrare din expresie.

$\gamma = \overline{x_3} + x_2 + \overline{x_1}$ – suprafața γ conține 2^1 celule (celulele 10 și 11), deci elimină 1 variabilă de intrare din expresie.

$\delta = x_2 + \overline{x_1} + \overline{x_0}$ – suprafața δ conține 2^1 celule (celulele 3 și 11), deci elimină 1 variabilă de intrare din expresie.

$\varphi = x_3 + x_2 + \overline{x_0}$ – suprafața φ conține 2^1 celule (celulele 1 și 3), deci elimină 1 variabilă de intrare din expresie.

Expresia finală a lui f este:

$$f' = \alpha \bullet \beta \bullet \gamma \bullet \delta \text{ sau}$$

$$f'' = \alpha \bullet \beta \bullet \gamma \bullet \varphi.$$

Variantele f' și f'' sunt egale din punct de vedere al costului implementării.

Diagrame Karnaugh pentru funcții incomplet definite

Funcțiile incomplet definite sunt cele care în anumite puncte ale domeniului de definiție pot lua

expressions:

$\alpha = x_3 + x_1$ – surface α contains 2^2 cells (cells 0, 1, 4 and 5), so it eliminates 2 input variables from the expression.

$\beta = \overline{x_2} + x_0$ – surface β contains 2^2 cells (cells 4, 12, 6 and 14), so it eliminates 2 input variables from the expression.

$\gamma = \overline{x_3} + x_2 + \overline{x_1}$ – surface γ contains 2^1 cells (cells 10 and 11), so it eliminates 1 input variable from the expression.

$\delta = x_2 + \overline{x_1} + \overline{x_0}$ – surface δ contains 2^1 cells (cells 3 and 11), so it eliminates 1 input variable from the expression.

$\varphi = x_3 + x_2 + \overline{x_0}$ – surface φ contains 2^1 cells (cells 1 and 3), so it eliminates 1 input variable from the expression.

The final expression of f is:

$$f' = \alpha \bullet \beta \bullet \gamma \bullet \delta \text{ or}$$

$$f'' = \alpha \bullet \beta \bullet \gamma \bullet \varphi.$$

The two variants, f' and f'' , are similar as respect to the implementation cost.

Karnaugh maps for incompletely defined functions

Incompletely defined functions are those which in certain points of the definition points can take either the

valoarea 0 sau valoarea 1. Avem două posibilități:

- combinații de intrare pentru care funcția are valori indiferente (nedefinite);
- combinații ale variabilelor care nu pot să apară din punct de vedere fizic; în aceste situații trebuie studiat dacă combinațiile sunt susceptibile să se producă în urma unei manevre false sau în urma unui defect de funcționare; pentru a evita funcționarea greșită, în celulele corespunzătoare se impun valori pentru funcție, astfel încât să nu se perturbe funcționarea normală.

Valorile nespecificate precum și celulele corespunzătoare din diagrama Karnaugh se numesc „indiferente” sau „arbitrare” sau „redundante”. Ele se notează cu „X” și vor fi considerate în timpul minimizării ca având valoarea 1 sau 0, în funcție de situație, pentru a se obține o minimizare cât mai bună.

Diagrame Karnaugh pentru funcții cu expresii (variabile) înglobate

În general o diagramă Karnaugh cu n expresii (sau variabile) are 2^n celule. Prin înglobarea în diagramă a m expresii (variabile) dimensiunea diagramei se reduce, numărul de celule devenind 2^{n-m} .

value 0 or the value 1. There are two possibilities:

- input combinations for which the function has don't care (undefined) values;
- variables combinations that can not physically appear; in these situations one must study if the combinations could occur as a consequence of a false maneuver or a malfunctioning situation. To avoid a wrong behavior, in the corresponding locations, specific values are imposed for the function, so that the normal behavior is not disturbed.

Unspecified values and also their corresponding locations in the Karnaugh map are called “don't cares” or “arbitrary” or “redundant”. They are denoted by an “X” and will be considered during minimization as having the value 1 or 0, according to the situation, in order to obtain an optimal minimization.

Karnaugh maps for functions with embedded expressions (variables)

Generally speaking, a Karnaugh map of n expressions (or variables) has 2^n cells. By embedding m expressions (variables) in the map, the size of the map decreases, the number of cells becoming 2^{n-m} .

Pentru a minimiza o funcție prin diagrame Karnaugh cu expresii se fac următorii pași:

Etapa I: minimizare cu ignorarea variabilelor înglobate

1) se consideră toate variabilele înglobate din diagramă ca fiind 0 și se formează suprafețe (sub-cuburi) cu constituenții lui 1;

Etapa II: de aici încolo, se consideră toate celulele cu 1 indiferente

2) se consideră variabilele înglobate, rând pe rând, a fi 1, în timp ce toate celelalte variabile înglobate sunt considerate 0; se formează astfel suprafețe (sub-cuburi) cu variabilele înglobate;

3) se consideră intersecția (funcția SI) variabilelor înglobate cu grupările obținute în pasul 2;

4) se face reuniunea (funcția SAU) termenilor din pașii 1 și 3;

Etapa III: se scrie reuniunea tuturor termenilor obținuți, care constituie forma finală a funcției.

Exemplu: să se minimizeze funcția dată prin următoarea diagramă Karnaugh:

In order to minimize a function by Karnaugh maps with embedded expressions one must follow these steps:

Step I: minimizing by ignoring embedded variables

1) consider all the embedded variables from the map to be 0 and create surfaces (sub-cubes) of 1's constituents;

Step II: from this point forward, consider all the locations of 1 to be *don't care*.

2) consider the embedded variables, one by one, to be 1, while all the other embedded variables are considered 0; thus we will form surfaces (sub-cubes) containing embedded variables;

3) consider the intersection (AND function) of the embedded variables with the groups obtained in step 2;

4) perform the reunion (OR function) of the terms from steps 1 and 3;

Step III: write the reunion of all the obtained terms, which constitutes the final form of the function.

Example: minimize the function given by the following Karnaugh map:

		x_0			
		00	01	11	10
x_3x_2	00	a 0	1 1	x 3	1 2
	01	x 4	b 5	1 7	0 6
	11	1 12	1 13	1 15	c 14
	10	1 8	1 9	\bar{a} 11	0 10

Etapa I:

		x_0			
		00	01	11	10
x_3x_2	00	0 0	1 1	x 3	1 2
	01	x 4	0 5	1 7	0 6
	11	1 12	1 13	1 15	0 14
	10	1 8	1 9	0 11	0 10

Step I:

$$f_I = x_3 \cdot \overline{x_1} + x_2 \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_2} \cdot x_0 + \overline{x_3} \cdot \overline{x_2} \cdot x_1$$

Etapa II.a:

		x_0			
		00	01	11	10
x_3x_2	00	a 0	x 1	x 3	x 2
	01	x 4	0 5	x 7	0 6
	11	x 12	x 13	x 15	0 14
	10	x 8	x 9	0 11	0 10

Step II.a:

$$f_{II.a} = a \cdot \overline{x_3} \cdot \overline{x_2}$$

Etapa II.b:

		x_0				
		00	01	11	10	
x_3	x_2	00	0	x	x	x
		01	x	4	b	0
x_3	x_2	11	x	x	x	0
		10	x	x	0	0

x_1

Step II.b:

$$f_{II.b} = b \bullet x_2 \bullet x_0$$

Etapa II.c:

		x_0				
		00	01	11	10	
x_3	x_2	00	0	x	x	x
		01	x	0	x	0
x_3	x_2	11	x	x	x	c
		10	x	x	0	0

x_1

Step II.c:

$$f_{II.c} = c \bullet x_3 \bullet x_2$$

Etapa II.d:

		x_0				
		00	01	11	10	
x_3	x_2	00	0	x	x	x
		01	x	0	x	0
x_3	x_2	11	x	x	x	0
		10	x	x	0	0

x_1

Step II.d:

$$f_{II.d} = \bar{a} \bullet x_1 \bullet x_0$$

Observație: variabila \bar{a} se tratează ca o variabilă complet diferită (ca și cum am nota-o cu d).

Remark: the \bar{a} variable is treated as a completely different variable (like if it was denoted by d).

2.4. Probleme rezolvate

1. Demonstrați cu ajutorul tabelelor de adevăr, apoi folosind legile algebrei Booleene, validitatea următoarelor identități:

a) $\overline{x + y + z} = \overline{x} \bullet \overline{y} \bullet \overline{z}$

b) $\overline{x \bullet y \bullet z} = \overline{x} + \overline{y} + \overline{z}$

Soluție

a) Tabelul de adevăr este următorul:

x	y	z	$\overline{x + y + z}$	$\overline{x} \bullet \overline{y} \bullet \overline{z}$	Concluzie (conclusion)
0	0	0	1	1	
0	0	1	0	0	
0	1	0	0	0	
0	1	1	0	0	
1	0	0	0	0	
1	0	1	0	0	
1	1	0	0	0	
1	1	1	0	0	

Știm din teorema lui De Morgan că $\overline{x + y} = \overline{x} \bullet \overline{y}$. Notăm $x + y = \alpha$; evident, $\overline{x + y} = \overline{x} \bullet \overline{y} = \overline{\alpha}$. Rămâne deci de demonstrat că $\overline{\alpha + z} = \overline{\alpha} \bullet \overline{z}$, ceea ce constituie chiar enunțul teoremei inițiale a lui De Morgan. Q.E.D.

b) Se rezolvă analog – temă de casă!

2.4. Solved problems

1. Prove using truth tables, then using the laws of Boolean algebra, the validity of the following identities:

a) $\overline{x + y + z} = \overline{x} \bullet \overline{y} \bullet \overline{z}$

b) $\overline{x \bullet y \bullet z} = \overline{x} + \overline{y} + \overline{z}$

Solution

a) We have the following truth table:

x	y	z	$\overline{x + y + z}$	$\overline{x} \bullet \overline{y} \bullet \overline{z}$	Concluzie (conclusion)
0	0	0	1	1	
0	0	1	0	0	
0	1	0	0	0	
0	1	1	0	0	
1	0	0	0	0	
1	0	1	0	0	
1	1	0	0	0	
1	1	1	0	0	

We know from De Morgan's theorem that $\overline{x + y} = \overline{x} \bullet \overline{y}$. We denote $x + y = \alpha$; obviously, $\overline{x + y} = \overline{x} \bullet \overline{y} = \overline{\alpha}$. All we have to prove now is that $\overline{\alpha + z} = \overline{\alpha} \bullet \overline{z}$, which is exactly the initial De Morgan's theorem. Q.E.D.

b) The solution is similar – homework!



d) Vom scrie:

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

Deoarece $A + \overline{A} = 1$, avem:
 $\overline{C} + A \bullet B \bullet C$. Aici putem observa că dacă $C = 0$, rezultatul este 1, iar dacă $C = 1$, rezultatul este $A \bullet B$. Atunci, rezultatul final este $\overline{C} + A \bullet B$.

Observație

Există o teoremă a algebrei Booleene care arată că:

$$\overline{x} + x \bullet y = \overline{x} + y.$$

e) În

$$\overline{A} \bullet B \bullet (\overline{D} + \overline{C} \bullet D) + B \bullet (A + \overline{A} \bullet C \bullet D),$$

dacă desfacem parantezele, obținem:

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

ceea ce poate fi rescris astfel:

$$B \bullet (\overline{A} \bullet \overline{D} + A) + \overline{A} \bullet B \bullet D \bullet (\overline{C} + C).$$

Folosind rezultatul de la partea a 2-a a rezolvării punctului d), avem:

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

Dar $A + \overline{A} \bullet D = A + D$, deci avem:

$$B \bullet (\overline{D} + A + D) = B \bullet (1 + A) = B$$

Soluția 2

Vom construi diagrama Karnaugh a acestei expresii:

d) We will write:

Since $A + \overline{A} = 1$, we have:
 $\overline{C} + A \bullet B \bullet C$. Here we can notice that if $C = 0$, the result is 1, and if $C = 1$, the result is $A \bullet B$. Then, the final result is $\overline{C} + A \bullet B$.

Remark

There is a theorem of Boolean algebra that shows that:

$$\overline{x} + x \bullet y = \overline{x} + y.$$

e) In

if we open the brackets, we obtain:

which can be rewritten as follows:

$$B \bullet (\overline{A} \bullet \overline{D} + A) + \overline{A} \bullet B \bullet D \bullet (\overline{C} + C).$$

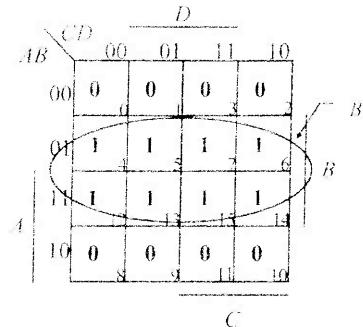
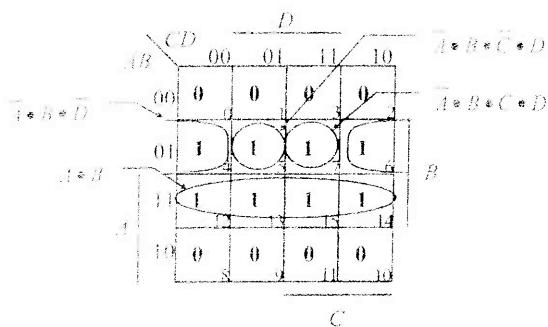
Using the result from the second part of the solution from point d), we have:

But $A + \overline{A} \bullet D = A + D$, so we have:

$$B \bullet (\overline{D} + A + D) = B \bullet (1 + A) = B.$$

Solution 2

We will build the Karnaugh map associated to this expression:



După cum se vede, termenii acoperă integral cele 8 biți de '1' din diagrama Karnaugh, deci expresia este egală cu B .

3. Aflați complementul următoarelor expresii:

- $x + y * z$
- $(x + \overline{y} + z) * (\overline{x} + \overline{z}) * (x + y)$

Soluție

a) Folosind teoremele lui De Morgan, obținem :

$$\overline{x + y * z} = \overline{x} * \overline{y * z} = \overline{x} * (\overline{y} + \overline{z})$$

b) Desfăcând parantezele și ținând cont că $x * \overline{x} = 0$ obținem :

$$x * \overline{z} + x * \overline{y} * \overline{z} + x * y * \overline{z} + \overline{x} * \overline{y} * z$$

Aici, știind că $a + \overline{a} = 1$ și că $a + a = a$, obținem forma finală: $x * \overline{z} + \overline{x} * y * z$.

As one can see, the terms entirely cover the 8 bits of '1' in the Karnaugh map, so the expression is equal to B .

3. Find the complement of the following expressions:

- $x + y * z$
- $(x + \overline{y} + z) * (\overline{x} + \overline{z}) * (x + y)$

Solution

a) Using De Morgan's laws, we obtain:

$$\overline{x + y * z} = \overline{x} * \overline{y * z} = \overline{x} * (\overline{y} + \overline{z}).$$

b) By opening the brackets and considering that $x * \overline{x} = 0$ we obtain:

$$x * \overline{z} + x * \overline{y} * \overline{z} + x * y * \overline{z} + \overline{x} * y * z$$

Here, knowing that $a + \overline{a} = 1$ and that $a + a = a$, we obtain the final form: $x * \overline{z} + \overline{x} * y * z$.

Acestei expresii îi corespunde următoarea diagramă Karnaugh:

To this expression corresponds the following Karnaugh map:

		z				
		00	01	11	10	
		0	0	1	3	2
x	0	1	0	0	1	1
x	1	4	5	7	6	
		y				

Este ușor să determinăm inversa funcției pe baza diagramei Karnaugh: celulele care sunt '1' devin '0', și invers.

It is easy to determine the complement of a function using the Karnaugh map: the locations that are '1' become '0', and vice-versa.

		z			
		00	01	11	10
		0	1	0	1
x	0	0	1	3	2
x	1	0	1	1	0
		y			

Minimizând funcția inversă folosind metoda expusă mai sus, obținem $x \bullet z + \bar{x} \bullet \bar{z} + \bar{x} \bullet \bar{y}$.

Același rezultat se obține și aplicând teoremele lui De Morgan.

4. Obțineți tabelul de adevăr al funcției $F = (\bar{A} + B) \bullet (\bar{B} + C)$ și exprimați-o sub formă de sumă de mintermi și apoi de produs de maxtermi.

By minimizing the complementary function using the above presented method, we obtain:

$$x \bullet z + \bar{x} \bullet \bar{z} + \bar{x} \bullet \bar{y}.$$

The same result is obtained also by applying De Morgan's laws.

4. Obtain the truth table of the function $F = (\bar{A} + B) \bullet (\bar{B} + C)$ and express it in the sum of minterms and product of maxterms forms.

Soluție

Tabelul de adevăr este următorul:

Solution

The truth table is the following one:

A	B	C	F	
0	0	0	1	minterm
0	0	1	1	minterm
0	1	0	0	maxterm
0	1	1	1	minterm
1	0	0	0	maxterm
1	0	1	0	maxterm
1	1	0	0	maxterm
1	1	1	1	minterm

În forma canonică disjunctivă (FCD), funcția se scrie ca o sumă de mintermi:

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

În forma canonică conjunctivă (FCC), funcția se scrie ca un produs de maxtermi:

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

5. Convertiți funcțiile de mai jos în cealaltă formă canonică:

a) $f(x, y, z) = \sum(1, 3, 7)$

b) $f(A, B, C, D) = \prod(0, 1, 2, 3, 4, 6, 12)$

In the disjunctive canonical form (DCF), the function is written as a sum of minterms:

In the conjunctive canonical form (CCF), the function is written as a product of maxterms:

5. Convert the following functions to the other canonical form:

a) $f(x, y, z) = \sum(1, 3, 7)$

b) $f(A, B, C, D) = \prod(0, 1, 2, 3, 4, 6, 12)$

Soluție

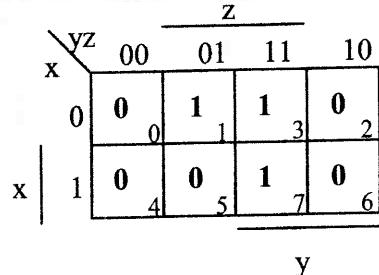
a) Tabelul de adevăr și diagrama Karnaugh pentru f sunt următoarele:

Solution

a) The truth table and the Karnaugh map for f are the following ones:

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

maxterm
minterm
maxterm
minterm
maxterm
maxterm
maxterm
minterm



Deci, f fiind exprimată inițial în FCD, în FCC avem:

$$f = (A + B + C) \bullet (A + \bar{B} + C) \bullet (\bar{A} + B + C) \bullet (\bar{A} + B + \bar{C}) \bullet (\bar{A} + \bar{B} + C)$$

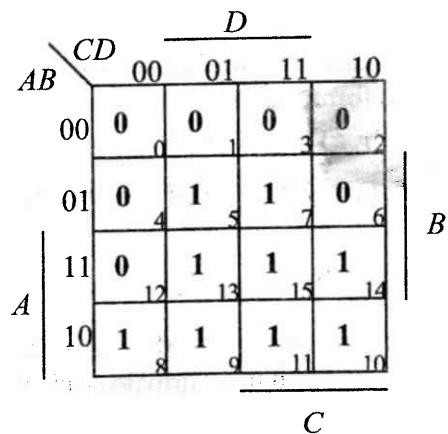
b) Tabelul de adevăr și diagrama Karnaugh pentru f sunt următoarele:

So, since f is initially expressed in the DCF, in the CCF we have:

b) The truth table and the Karnaugh map for f are the following ones:

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

maxterm
maxterm
maxterm
maxterm
maxterm
minterm
maxterm
minterm
minterm
minterm
minterm
minterm
maxterm
minterm
minterm



Dacă, în fiind exprimată inițial în FCC, în FCD avem:

$$\begin{aligned} f = & \overline{A} * B * \overline{C} * D + \overline{A} * B * C * D + A * \overline{B} * \overline{C} * \overline{D} + A * \overline{B} * \overline{C} * D + \\ & + A * \overline{B} * C * D + A * B * \overline{C} * D + A * B * C * \overline{D} + A * B * C * D \end{aligned}$$

b) Prietenul nostru, Ion, dorește să-și cumpere un cățel. Pentru a-l satisface pe Ion, care este foarte pretențios, cățelul trebuie să îndeplinească următoarele patru condiții:

- a) să fie de rasă, sau să fugă repede, sau să îi placă mângâierile;
- b) să fie de rasă, sau să nu-i placă mângâierile, sau să fie intelligent;
- c) să nu fugă repede, sau să îi placă mângâierile;
- d) să nu fie de rasă, sau să nu-i placă mângâierile.

Poate cățelul lui Ion să satisfacă toate aceste cerințe, și dacă da, ce calități va avea el?

Soluție

Această problemă se numește Problema sauzabilătății logice (SAT).

Noi sănă propozitiile astfel:

w = „este de rasă”

x = „fuge repede”

y = „îi plac mângâierile”

z = „este intelligent”.

Problema poate fi exprimată ca un

So, since f is initially expressed in the FCC, in the DCF we have:

6. Our friend, John, wants to buy himself a dog. In order to satisfy John, who is very exigent, the dog must fulfill the following four conditions:

- a) has pedigree, or runs fast, or likes caresses;
- b) has pedigree, or doesn't like caresses, or is intelligent;
- c) doesn't run fast, or likes caresses;
- d) doesn't have pedigree, or doesn't like caresses.

Can John's dog satisfy all these requirements and, if yes, which qualities will this dog have?

Solution

This problem is called the Boolean satisfiability problem (SAT).

We will denote the sentences as follows:

w = “has pedigree”

x = “runs fast”

y = “likes caresses”

z = “is intelligent”.

The problem can be expressed as a

sistem de ecuații Booleene:

$$\begin{cases} w+x+y=1 \\ w+\bar{y}+z=1 \\ \bar{x}+y=1 \\ \bar{w}+\bar{y}=1 \end{cases}$$

Acest sistem corespunde de fapt unei funcții Booleene exprimate în FCC:

$$f = (w+x+y) \cdot (w+\bar{y}+z) \cdot (\bar{x}+y) \cdot (\bar{w}+\bar{y})$$

Ne interesează deci acele puncte din domeniul de definiție al funcției în care ea ia valoarea 1.

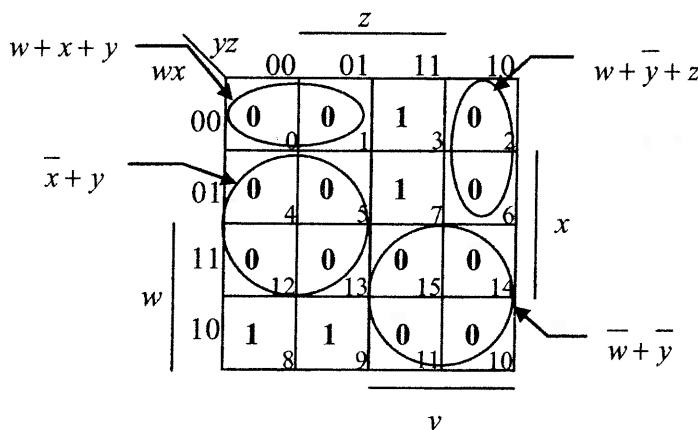
Desenând diagrama Karnaugh, obținem:

system of Boolean equations:

This system actually corresponds to a Boolean function expressed in the CCF:

We are interested by those points in the function's definition domain where it takes the value '1'.

By drawing its Karnaugh map, we obtain:



Observăm deci că soluția este dată de celulele 3, 7, 8 sau 9 (oricare dintre acestea constituie o soluție pentru problema noastră). De exemplu, celula 3 (0011) are sensul „nu este de

We notice that the solution is given by cells 3, 7, 8 or 9 (any of these constitutes a solution for our problem). For instance, cell 3 (0011) has the following meaning: „doesn't

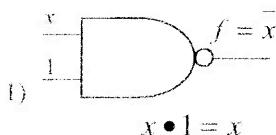
rasă”, „nu fugе repede”, „îi plac mânăierile” și „este intelligent”.

7. Demonstrați că toate funcțiile de comutație pot fi realizate folosind numai porți logice ȘI-NU.

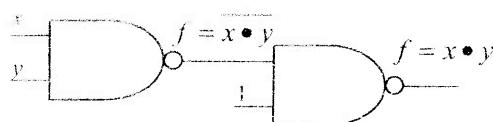
Soluție

Pentru aceasta, trebuie să demonstrăm că putem realiza funcțiile Booleene fundamentale ȘI, SAU și NU folosind *numai* porți logice ȘI-NU.

a) Implementarea porții logice NU se poate face într-o singură din următoarele două variante:



b) Implementarea porții logice ȘI se poate face astfel:



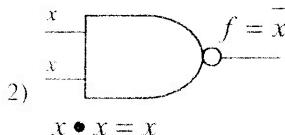
have pedigree”, „doesn't run fast”, „likes caresses” and „is intelligent”.

7. Show that all the switching functions can be made using only NAND gates.

Solution

In order to prove this, we have to show that we can make the fundamental Boolean functions AND, OR and NOT using *only* NAND logic gates.

a) The implementation of the NOT logic gate can be done in one of the following two variants:



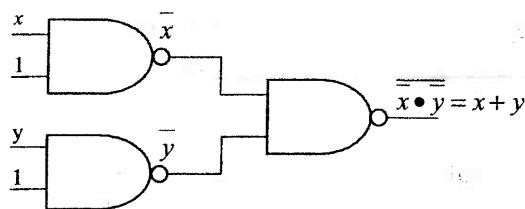
b) The implementation of the AND logic gate can be done as follows:

Găsiți și alte soluții!

c) Pentru implementarea porții logice SAU trebuie să folosim formulele lui De Morgan:

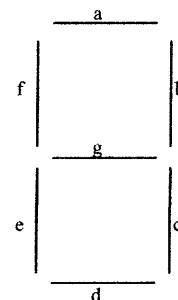
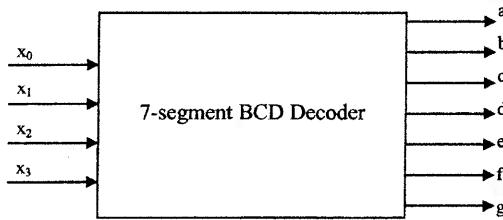
Other solutions exist; find them!

c) For the implementation of the OR logic gate we must use De Morgan's laws:



8. Minimizați funcțiile Booleene asociate unui decodificator BCD 7-segmente.

8. Minimize the Boolean functions associated to a 7-segment BCD Decoder.

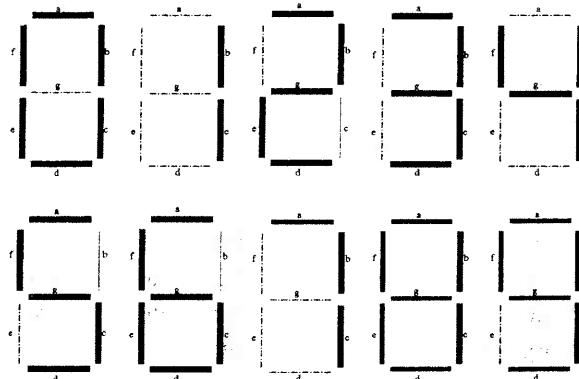


Soluție

Fiecare cifră zecimală codificată BCD care este primită pe intrările $x_0 - x_3$ va provoca aprinderea led-urilor în mod corespunzător:

Solution

Each BCD-encoded decimal digit that is received on the $x_0 - x_3$ inputs will trigger the leds to be turned on accordingly:



Se pune întrebarea: ce se va petrece dacă la intrare va fi primit un cod binar care nu reprezintă nici o cifră BCD? Există mai multe soluții de implementare posibile, care vor fi alese în funcție de contextul real în care va funcționa sistemul fizic:

- Dacă suntem absolut siguri că niciodată nu se va primi un cod diferit de cel al cifrelor zecimale, vom considera celulele respective din diagrama Karnaugh indiferente;
- Putem realiza o convenție: orice alt cod decât cel al cifrelor zecimale va avea ca efect stingerea tuturor led-urilor decodificatorului; prin urmare, vom considera celulele respective din diagrama Karnaugh a fi 0;
- Putem realiza o altă convenție: orice alt cod decât cel al cifrelor zecimale va avea ca efect afișarea unui simbol special pe led-urile decodificatorului (de exemplu, „E” de la „Eroare”); prin urmare, vom considera locațiile respective din diagrama Karnaugh a fi conform elementelor acelui simbol.

În continuare vom opta pentru prima variantă; prin urmare, vom construi tabelul de adevăr al celor 7 funcții Booleene și le vom minimiza conform metodei deja cunoscute:

One question arises: what happens if on the input we receive a binary code that doesn't represent any BCD digit? There are several implementation solutions, that will be chosen according to the real context in which the system will actually work:

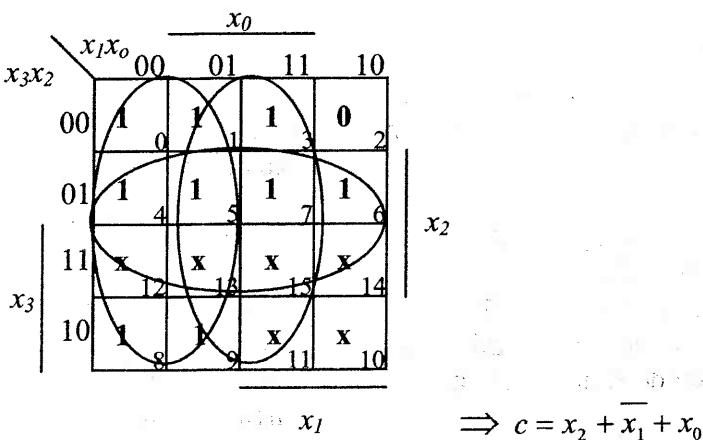
- If we are absolutely sure that we will never receive a code that is different from those of the decimal digits, we will consider those locations from the Karnaugh map to be *don't care*;
- We can make a convention: any other code than those of the decimal digits will trigger turning off all the decoder's leds; so, we will consider the corresponding locations from the Karnaugh map to be 0;
- We can make another convention: any other code than those of the decimal digits will trigger displaying a special symbol on the decoder's leds (for instance, „E” from „Error”); as a matter of consequence, we will consider the corresponding locations from the Karnaugh map to be according to the values of that symbol.

Hereinafter we will choose the first variant; as a matter of consequence, we will build the truth table of the 7 Boolean functions and we will then minimize them by the method we already know:

x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

Vom minimiza una dintre funcții, c , urmând ca celelalte să fie minizate în mod analog.

We will minimize one of the functions, c , the others being minimized similarly.



9. Pentru funcțiile:

$$f_1 = \sum_{\text{d.c.}} (0,1,2,3,5,8,10,14) + \sum_{\text{a.}} (4,6,7); f_2 = \sum_{\text{d.c.}} (0,3,4,5) + \sum_{\text{a.}} (6,7,9)$$

- a) Determinați $h = f_1 + f_2$ și $g = f_1 \cdot f_2$;
- b) Minimizați f_2 în forma canonicoă disjunctivă (PCD) și în forma canonicoă conjunctivă (PCC). Dacă rezultatele diferă, explicați de ce.
- c) Câte funcții f_1 există?

Cu „ $\sum_{\text{d.c.}}$ ” sunt marcat termenii indiferenți.

Soluție

a) Ne bazăm pe faptul că $a \cdot X = X$, $X \cdot X = X$, $X + X = X$ și $a + X = a$, unde X înseamnă indiferent.

Efectuând operațiile la nivel de termen canonic, obținem:

$$h = \sum_{\text{d.c.}} (0,1,2,3,4,5,8,10,14) + \sum_{\text{a.}} (6,7,9); g = \sum_{\text{d.c.}} (0,3,5) + \sum_{\text{a.}} (4,6,7)$$

Acest rezultat este mai ușor de observat dacă se construiește diagrama Karnaugh a fiecărei funcții (f_1 , f_2 , g și h).

b) Diagrama Karnaugh a lui f_2 este următoarea:

Q. For the following functions:

- a) Determine $h = f_1 + f_2$ and $g = f_1 \cdot f_2$;
- b) Minimize f_2 in the disjunctive canonical form (DCF) and in the conjunctive canonical form (CCF). If the results differ, explain why.
- c) How many f_1 functions exist?

The “ $\sum_{\text{d.c.}}$ ” notation indicates the don't care terms.

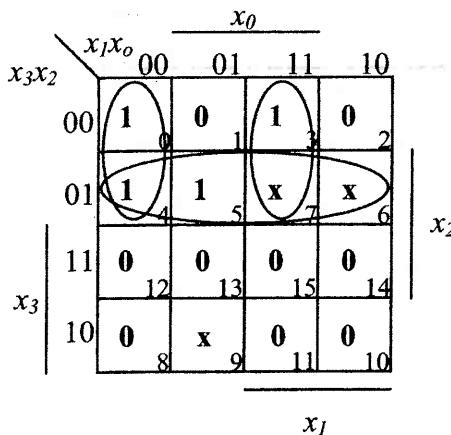
Solution

a) We use the relations: $a \cdot X = X$, $X \cdot X = X$, $X + X = X$ and $a + X = a$, where X means “don't care”.

By performing the operations at the canonical terms level, we obtain:

This result is easier to observe if we build the Karnaugh map of each function (f_1 , f_2 , g and h).

b) The Karnaugh map for f_2 is the following one:



Atunci, în FDM:

$$f_2 = \overline{x_3} \cdot x_2 + \overline{x_3} \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_1} \cdot \overline{x_0}$$

În FCM:

$$f_2 = \overline{x_3} \cdot (\overline{x_1} + x_0) \cdot (x_2 + x_1 + \overline{x_0})$$

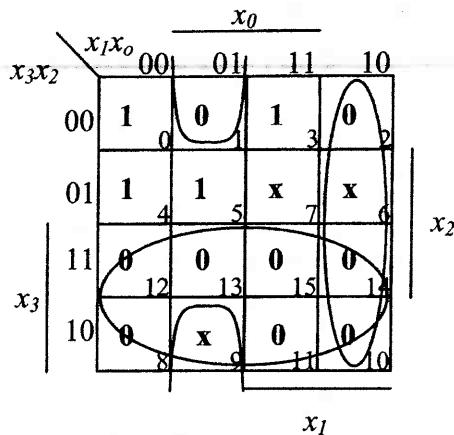
Rezultatele sunt *diferite* deoarece la minimizarea în FCD am considerat celula indiferentă 6 '1', pe când la minimizarea în FCC am considerat celula indiferentă 6 '0', ceea ce ne-a permis să obținem expresii mai simple.

c) Cei 3 termeni indiferenți pot lua valorile 0 sau 1: 000, 001, 010 etc. Prin urmare, avem $2^3 = 8$ funcții posibile, corespunzătoare numărului acestor combinații.

10. O poartă logică cu 4 intrări, numită LEMON, realizează următoarea funcție:

$$\text{LEMON}(a, b, c, d) = b \cdot c \cdot (a + d)$$

Presupunând că variabilele de



Then, in the MDF:

$$f_2 = \overline{x_3} \cdot x_2 + \overline{x_3} \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_1} \cdot \overline{x_0}$$

In the MCF:

$$f_2 = \overline{x_3} \cdot (\overline{x_1} + x_0) \cdot (x_2 + x_1 + \overline{x_0})$$

The results are *different* because when minimizing in the DCF we considered the don't care cell 6 '1', while when minimizing in the CCF we considered the don't care cell 6 '0', which allowed us to obtain simpler expressions.

c) The 3 don't care terms can take the values 0 or 1: 000, 001, 010 etc. So we have $2^3 = 8$ possible functions, corresponding to the number of these combinations.

10. A 4-input logic gate, named LEMON, realizes the following function:

$$\text{LEMON}(a, b, c, d) = b \cdot c \cdot (a + d)$$

Suppose the input variables are

intrare sunt disponibile atât în forma adevărată cât și în cea negată:

a) Arătați cum se poate realiza funcția:

$$f = \sum (0,1,6,9,10,11,14,15)$$

folosind 3 porți LEMON și o poartă SAU.

b) Demonstrați că toate funcțiile logice pot fi realizate cu porți LEMON și porți SAU.

Soluție

Construim mai întâi diagrama Karnaugh a funcției f :

		x_0				
		00	01	11	10	
x_1x_2		00	1 0	1 1	0 3	0 2
		01	0 4	0 5	0 7	1 6
		11	0 12	0 13	1 15	1 14
x_3		10	0 8	1 9	1 11	1 10

x_1

Numele LEMON vine de la aspectul funcției, atunci când o reprezentăm prin diagramă Karnaugh: observăm că arată ca un „L”.

available in both the true and the inverted (negated) form.

a) Show how to implement the function:

$$f = \sum (0,1,6,9,10,11,14,15)$$

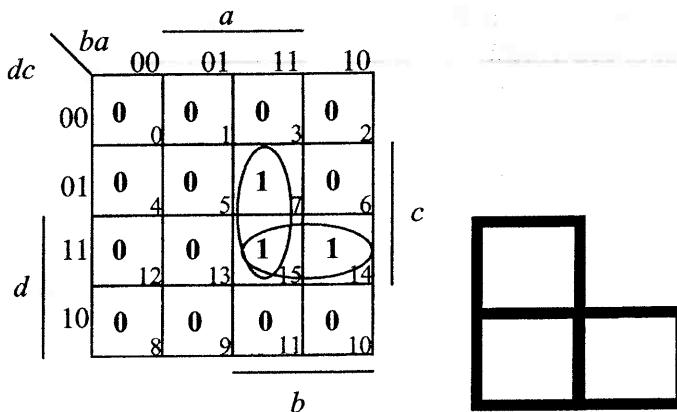
using 3 LEMON gates and an OR gate.

b) Prove that all logic functions can be realized with LEMON and OR gates.

Solution

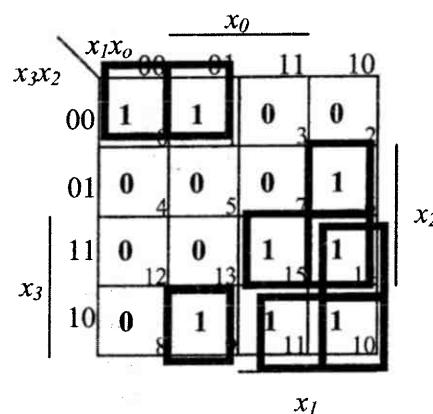
First we build the Karnaugh map of the function f :

The name LEMON comes from the function's aspect, when we represent it by a Karnaugh map: we notice that it looks like an „L”.



Putem acoperi toate celulele de '1' din funcția f prin 3 astfel de „L”-uri (al 3-lea, cel format din celulele 0, 1 și 9 fiind ceva mai greu de observat, dar totuși real – să nu uităm că în diagrama Karnaugh celulele de la extremități sunt adiacente!):

We can cover all the cells containing a '1' of the function f by 3 such „L”-s (the 3rd, the one composed of the cells 0, 1 and 9 is somehow harder to notice, but still real – don't forget that in the Karnaugh map the cells from its extremities are adjacent!):



Atunci funcția f se poate scrie (de sus în jos pe diagrama Karnaugh):

So we can write the function f (from top to bottom, in the Karnaugh map):

$$f = LEMON(\overline{x_3}, \overline{x_2}, x_0, \overline{x_1}) + LEMON(x_1, x_2, x_3, \overline{x_0}) + LEMON(\overline{x_2}, x_3, x_1, \overline{x_0})$$

b) De vreme ce avem disponibile variabilele de intrare negate, mai rămâne să arătăm implementarea funcțiilor **ȘI** și **SAU**.

$$x \bullet y = LEMON(1, x, y, 1) ;$$

11.

- a) Demonstrați teorema de expansiune a lui Shannon.
- b) Demonstrați că teorema de expansiune a lui Shannon este adevărată și dacă înlocuim operatorul **SAU** cu operatorul **XOR**.
- c) Minimizați funcția $f = \sum(0,1,2,4,6)$ în forma canonică disjunctivă și în forma canonică conjunctivă.
- d) Folosind rezultatul de la punctele b) și c), demonstrați că orice funcție Booleană poate fi exprimată folosind numai porți **XOR** și porți **ȘI**. Exemplificați această afirmație pe funcția de la punctul c).

Soluție

- a) Teorema de expansiune a lui Shannon:

$$f(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) = x_i \bullet f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + \overline{x_i} \bullet f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Pentru demonstrație vom lua cele 2 cazuri:

b) Since we have available the input variables in the negated form, we only must show the implementation of the AND and OR functions.

$$x + y = LEMON(x, 1, 1, y)$$

11.

- a) Prove Shannon's expansion theorem.
- b) Prove that Shannon's expansion theorem is true also if we replace the OR operator by the XOR operator.
- c) Minimize the function $f = \sum(0,1,2,4,6)$ in both the disjunctive canonical form and the conjunctive canonical form.
- d) Using the result from points b) and c), show that any Boolean function can be expressed using only XOR gates and AND gates. Exemplify this assertion on the function from point c).

Solution

- a) Shannon's expansion theorem:

For the demonstration we will take the two cases:

a1) $x_i = 0$. Atunci avem:

$$f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) = 0 \bullet f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + 1 \bullet f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Deci

a1) $x_i = 0$. Then we have:

$$f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \quad \text{Q.E.D.}$$

So

$$f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \quad \text{Q.E.D.}$$

a2) $x_i = 1$. Atunci avem:

a2) $x_i = 1$. Then we have:

$$f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) = 1 \bullet f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + 0 \bullet f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Deci

So

$$f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \quad \text{Q.E.D.}$$

b) Trebuie să demonstrăm că:

b) We have to prove that:

$$f(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) = x_i \bullet f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \oplus \overline{x_i} \bullet f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Demonstrația este analogă celei de la punctul a):

The demonstration is similar to the one made for point a):

b1) $x_i = 0$. Atunci avem:

b1) $x_i = 0$. Then we have:

$$f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) = 0 \bullet f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \oplus 1 \bullet f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Deci

So

$$f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \quad \text{Q.E.D.}$$

b2) $x_i = 1$. Atunci avem:

b2) $x_i = 1$. Then we have:

$$f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) = 1 \bullet f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \oplus 0 \bullet f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Deci

So

$$f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) \quad \text{Q.E.D.}$$

c) Diagrama Karnaugh pentru f este următoarea:

c) The Karnaugh map for f is the following:

		x_2			
		00	01	11	10
x_1	0	1 0	1 1	0 1	1 2
	1	1 1	0 1	0 1	1 0

$$f_{min} = \overline{x_0} + \overline{x_1} \bullet \overline{x_2}$$

$$f_{min} = (\overline{x_1} + x_0) \bullet (\overline{x_1} + \overline{x_0})$$

$$f_{min} = \overline{x_0} + \overline{x_1} \bullet \overline{x_2}$$

$$f_{min} = (\overline{x_1} + x_0) \bullet (\overline{x_1} + \overline{x_0})$$

d) Observăm că, potrivit teoremei lui Shannon, fiecare funcție poate fi descompusă după orice variabilă de intrare într-o formă care conține operatorii NU, ȘI și XOR. Această descompunere poate continua în mod recursiv pe cei doi operanzi ai lui XOR, până la nivelul variabilelor individuale. Așadar, mai trebuie demonstrată doar implementarea operatorului NU. Aceasta este foarte simplă folosind XOR: $a \oplus 1 = \bar{a}$. Este mai ușor să demonstrăm aceasta operând pe una din formele minime ale funcției Booleane. În cazul de față:

$$f(x_0, x_1, x_2) = \overline{x_0} \bullet /(\overline{x_1}, x_2, 0) \oplus x_0 \bullet /(\overline{x_1}, x_2, 1) = \overline{x_0} \bullet /(\overline{x_1} \bullet \overline{x_2} \bullet \overline{x_3}) = (x_0 \oplus 1) \bullet (x_0 \bullet (x_1 \oplus 1) \bullet (x_1 \oplus 1))$$

În această formă, expresia funcției conține numai porți XOR și ȘI.

2.1 Probleme propuse

1. Se dă funcția
 $f = \sum m(0,2,4,8,10,12) + \sum d(1,5,9,13)$

d) We notice that, according to Shannon's theorem, each function can be decomposed by any input variable in a form that contains the operators NOT, AND and XOR. This decomposition can continue recursively on the two operands of the XOR, until the level of the individual variables. Therefore, we only have to show the implementation of the NOT operator. This is very simple using XOR: $a \oplus 1 = \bar{a}$.

It is easier to prove that by operating on one of the minimized forms of the Boolean function. In our case:

In this form, the function's expression contains only XOR and AND gates.

2.3 Proposeci problems

1. Be given the function
 $f = \sum m(0,2,4,8,10,12) + \sum d(1,5,9,13)$

- a) Determinați expresia minimală a sumei de produse;
 b) Determinați expresia minimală a produsului de sume;
 c) Comparați rezultatele de la a) și b); dacă nu sunt identice, explicați de ce.

2. Se dă funcțiile:

$$f_1 = \sum (0,1,2,3,5,8,10,14);$$

$$f_2 = \sum (0,3,5);$$

$$f_3 = \sum (1,2,3,5,8,9,12).$$

Să se minimizeze funcțiile:

- a) fiecare funcție independent
 b) sistemul de funcții

Care variantă este mai avantajoasă?

Indicație: Toate funcțiile se consideră a avea același număr de variabile (cel al funcției celei mai mari) și se minimizează urmărind punerea în comun a termenilor canonici cu celelalte funcții din sistem.

3. Se dă funcția
 $f = \sum (1,2,4,7,8,11,13,14)$

- a) Să se exprime funcția în forma canonică disjunctivă
 b) Să se exprime funcția în forma canonică conjunctivă
 c) Să se implementeze cu porți SAU-EXCLUSIV.

4. Să se minimizeze prin metoda Quine-McCluskey

- a) Determine the minimal expression of the sum of products;
 b) Determine the minimal expression of the product of sums;
 c) Compare the results from a) and b); if they are not identical explain why.

2. Be given the functions:

$$f_1 = \sum (0,1,2,3,5,8,10,14);$$

$$f_2 = \sum (0,3,5);$$

$$f_3 = \sum (1,2,3,5,8,9,12).$$

Minimize the functions:

- a) each function independently
 b) the system of functions

Which variant has a greater advantage?

Hint: All functions are considered to have the same number of variables (the one of the biggest function) and are minimized trying to share the canonical terms with the other functions in the system.

3. Be given the function
 $f = \sum (1,2,4,7,8,11,13,14)$

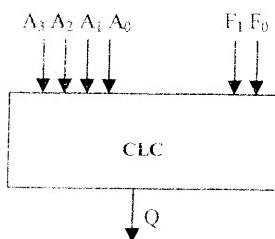
- a) Express the function in the disjunctive canonical form
 b) Express the function in the conjunctive canonical form
 c) Implement the function with XOR gates.

4. Minimize by the Quine-McCluskey method the function

5. Se dă funcția:
 $f = \sum(1,2,5,7,8,9,10,13,15)$ și apoi să se implementeze funcția minimizată folosind numai porți ȘI-NU.

5. Se dă funcția:
 $f = (\bar{A} + B) \bullet (\bar{A} + C + D) \bullet (A + \bar{C} + D)$
 Termenii indiferenți sunt: S_0, S_2, S_9 și S_{10} . Se cere:
- Să se scrie funcția f în forma canonica conjunctivă
 - Să se scrie expresia minimizată a lui f în forma canonica disjunctivă și în forma canonica conjunctivă
 - Dacă rezultatele diferă, să se explice *de ce*.

6. Implementați numai cu porți ȘI-NU și XOR circuitul logic combinațional descris în figura de mai jos:



F_0	F_1	
0	0	$Q = 1$ if $A_3A_2A_1A_0$ is a multiple of 2
0	1	$Q = 1$ if $A_3A_2A_1A_0$ is a multiple of 4
1	0	$Q = 1$ if $A_3A_2A_1A_0 = 13$
1	1	$Q = 1$ if $A > 7$

7. Se dă funcția
 $f = \sum(0,1,2,7,9,14,15) + \sum_{\Phi}(3,4,5)$
 – notația \sum_{Φ} indică termenii indiferenți.
 - Câte funcții f există?
 - Exprimați funcția f în forma

$f = \sum(1,2,5,7,8,9,10,13,15)$ and then implement the minimized function using only NAND gates.

5. Consider the function:
 $f = (\bar{A} + B) \bullet (\bar{A} + C + D) \bullet (A + \bar{C} + D)$
 The *don't care* terms are S_0, S_2, S_9 and S_{10} . Requirements:
 - Write the f function in the *conjunctive canonical form*;
 - Write the *minimized expression* of f in the *disjunctive canonical form* and in the *conjunctive canonical form*.
 - If the results differ, explain *why*.

6. Implement only with XOR and AND gates the combinational logic circuit defined in the figure below:

7. Be given the function
 $f = \sum(0,1,2,7,9,14,15) + \sum_{\Phi}(3,4,5)$
 – the \sum_{Φ} notation indicates the *don't care terms*.
- How many f functions exist?
 - Express the function f in the

canonică conjunctivă FCC și în forma canonică disjunctivă FCD;

c) Minimizați f în FCM și FDM. Dacă rezultatele diferă, explicați de ce.

8. Pentru circuitul din figură:

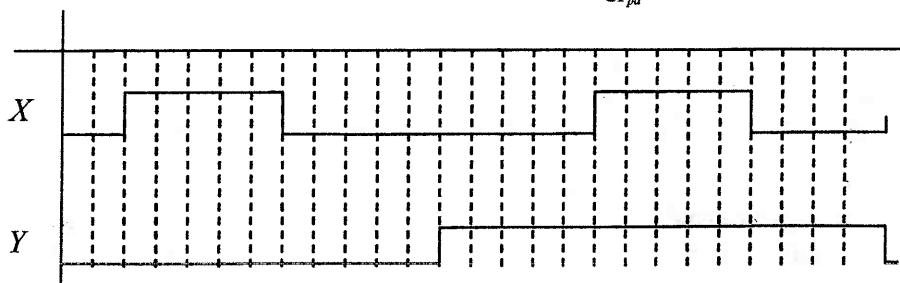
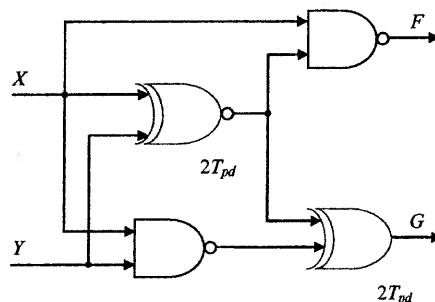
a) Scrieți ecuațiile funcțiilor F și G ;
 b) Știind că întârzierea de propagare a semnalelor prin porțile XOR și COINCIDENȚĂ este de 2 (două) ori mai mare decât întârzierea introdusă de către porțile ȘI-NU (T_{pd}), desenați răspunsul circuitului (forme de undă) la stimulii reprezentați pe cronoograma (diagrama cu forme de undă) din figura de mai jos. Fiecare interval reprezintă 1 (una) întârziere de poartă ȘI-NU.

conjunctive canonical form CCF and in the disjunctive canonical form DCF;

c) Minimize f in the MCF and in the MDF. If the results differ, explain why.

8. For the circuit in the next figure:

a) Write the equations of the F and G functions;
 b) Knowing that the signals propagation delay through the XOR and XNOR gates is 2 (two) times greater than the delay introduced by the NAND gates (T_{pd}), draw the circuit's response (the waveforms) at the stimuli presented as a waveform (chronogram) in the figure below. Each interval represents 1 (one) NAND gate delay.



9. Se dă funcția
 $f = \sum(0,1,5,7,9,12) + \sum_{\phi}(2,8,11,15)$

- a) Câte funcții f există?
- b) Exprimăți funcția f în forma canonică conjunctivă;
- c) Deduceți expresia minimală a lui f în forma canonică disjunctivă.

10. A spune: B minte.

B spune: C minte.

C spune: A și B mint.

Cine minte și cine spune adevărul?

9. Be given the function
 $f = \sum(0,1,5,7,9,12) + \sum_{\phi}(2,8,11,15)$

- a) How many functions f exist?
- b) Express the f function in the conjunctive canonical form;
- c) Deduce the minimal expression of f in the disjunctive canonical form.

10. A says: B lies.

B says: C lies.

C says: A and B lie.

Who lies and who is telling the truth?

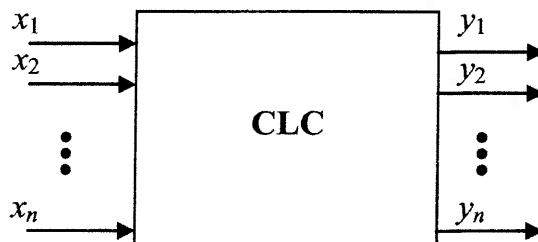
Capitolul 3. Circuite logice combinatoriale

3.1. Definiții

Circuitele logice combinatoriale, CLC, sunt un caz particular al sistemelor secvențiale finite (automate finite), numite automate de grad 0.

Circuitele logice combinatoriale se caracterizează prin faptul că variabilele de ieșire sunt independente de timp și de starea internă, fiind determinate numai de variabilele de intrare (starea variabilelor de intrare la momentul considerat).

Legătura dintre starea ieșirii și starea intrării unui CLC este realizată de funcția de transfer.



Oricare funcție de ieșire y (y_1, y_2, \dots, y_m) este funcție de toate variabilele de intrare (x_1, x_2, \dots, x_n). Un CLC se poate descrie astfel:

Chapter 3. Combinational logic circuits

3.1. Definitions

Combinational logic circuits, CLCs, are a particular case of finite sequential systems (finite automata), called 0-degree automata.

Combinational logic circuits are characterized by the fact that the output variables are independent of the time and the internal state, being determined only by the input variables (the input variables state at the moment considered).

The link between the state of the output and the state of the input of a CLC is realized by the transfer function.

Any output function y (y_1, y_2, \dots, y_m) is a function of all the input variables (x_1, x_2, \dots, x_n). A CLC can be described as follows:



$$\left\{ \begin{array}{l} Y_1 = \overline{A_1} \cdot \overline{A_2} \cdot \overline{A_3} \\ Y_2 = \overline{A_1} \cdot \overline{A_2} \cdot \overline{A_4} \\ Y_3 = \overline{A_1} \cdot A_3 \cdot A_4 \end{array} \right.$$

în care se vor exprima în formă canonica respectivă de la sau în forma canonica conjunctivă FCC.

Independență: Fără să se impună presupune că o dată cu modificarea conditiei de ieșire nu modifică condiția de stabilitate a rețelei de tranzistor. Datorită întârzierilor urmărite de circuitele logice și de conexiuni, modificarea simultană nu este posibilă.

În urmare, pe durata procesului finalizării de stabilire a variabilelor primare, vectorul ieșirilor poate lua valori intermedii diferite de valoarea finală, ceea ce conduce la fenomenul de hazard combinatorial. De aceea trebuie să se ia în considerare acest lucru în sistem numeric.

Analiza CLC

În cadrul analizei CLC se pleacă de la cunoașterea schemei logice a circuitului și se urmărește stabilirea funcționării acestuia. Stabilirea expresiei variabilelor de ieșire se face în trei etape: exprimare, confirmare și transformare a variabilelor de ieșire.

The functions will be expressed in the disjunctive canonical form DCF or in the conjunctive canonical form FCC.

The time independence means that simultaneously with the modification of the input variables the output variables are not modified. In reality, because of the delays introduced by the circuits and the connections, the simultaneous modification is not possible.

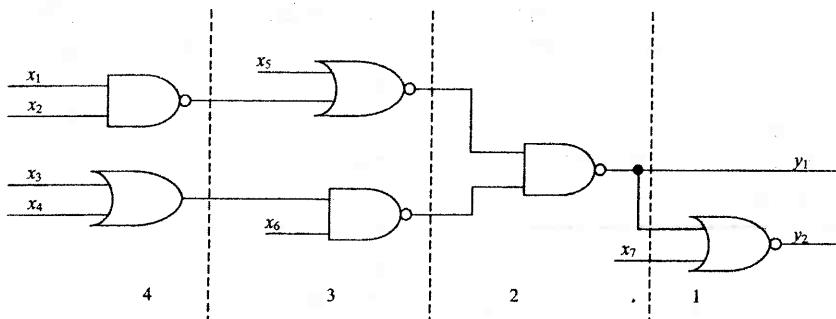
As a matter of consequence, during the transient process of the output variables' stabilization, the output vector can take intermediate values that are different from the final values. This leads to the combinational hazard phenomenon, which must be take into consideration when designing a digital system.

CLC analysis

In the CLC's analysis we start from the logic scheme of the circuit and we aim at determining its functioning. The process of determining the output variables' expressions is done from the inputs towards the outputs, by following the input variables' transformations.

Definim ca *număr de niveluri* al unui CLC numărul maxim de porți dintre intrările și ieșirile sale. Numerotarea nivelurilor se face de la ieșire înspre intrare.

We define as *number of levels* of a CLC the maximal number of gates between its inputs and outputs. The numbering of the levels is done from outputs towards the inputs.



Pentru a determina dacă un circuit este sau nu combinațional, se folosește următorul algoritm:

Algoritm de determinare a legăturilor inverse dintr-un CLC

1. Se numerotează toate porțile logice care au ca intrări o submulțime a mulțimii variabilelor de intrare ale circuitului logic (de la 1 la k);
2. Se numerotează de la $k + 1$ porțile logice care au ca intrări fie intrări ale circuitului, fie ieșiri ale porților numerotate la punctul 1. Dacă am reușit să numerotăm toate porțile circuitului logic, acesta este fără legături inverse și este circuit combinațional. Dacă nu am reușit numerotarea tuturor porților logice,

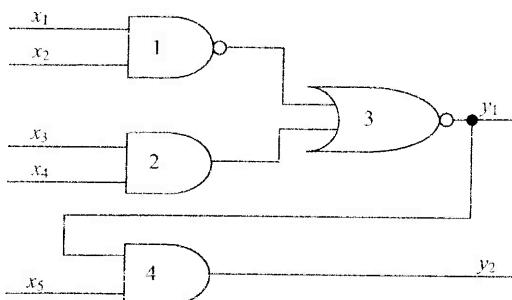
In order to determine if a circuit is combinational or not, the following algorithm must be used:

Algorithm for determining the reverse links inside a CLC

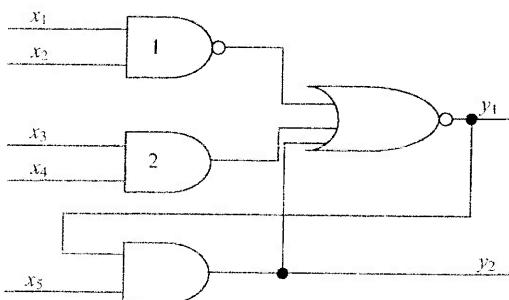
1. Number all the logic gates that have as inputs a subset of the logic circuit's input variables set (from 1 to k);
2. Number from $k + 1$ the gates that have as inputs either circuit's inputs or outputs of the gates that have already been numbered in step 1. If all the circuit's gates have been numbered, it has no reverse links and thus it is a combinational circuit. If the numbering of all the gates has not succeeded, then the

circuitul este de tip secvențial.

De pildă, circuitul de mai jos este combinațional (toate portile au putut fi numerotate conform algoritmului anterior):



Însă circuitul de mai jos NU este combinațional (nu s-au putut numera toate portile):



circuit is a sequential one.

For instance, the circuit below is combinational (all the gates could be numbered according to the previously defined algorithm):

But the circuit below is NOT combinational (it is impossible to number all the gates):

3.3. Sinteză CLC

În cadrul sintezei circuitelor logice combinaționale se cunoaște funcția pe care trebuie să o îndeplinească circuitul și se caută să se găsească structura acestuia.

3.3. CLC synthesis

In combinational logic circuits synthesis we know the function the circuit must fulfill and we aim at determining its structure.

The steps of the combinational logic

Etapele sintezei circuitelor logice combinaționale sunt:

1. Enunțul problemei;
2. Alcătuirea tabelului de adevăr, definirea funcției sau funcțiilor;
3. Minimizarea funcției sau funcțiilor;
4. Desenarea schemei circuitului.

Există mai multe metode de implementare a CLC, diferențiate după nivelul de complexitate al circuitelor integrate folosite.

Sinteză CLC cu circuite integrate SSI

Circuitele integrate de tip SSI – *small scale integration* – au până la 50 de tranzistoare integrate pe capsulă. Dintre aceste circuite fac parte porțile logice fundamentale: ȘI-NU, SAU-NU, NU, ȘI, SAU, SAU-EXCLUSIV, COINCIDENȚĂ. După parcurgerea etapelor de sinteză se face implementarea funcției sau funcțiilor logice cu ajutorul circuitelor integrate existente. CLC de tip SSI se folosesc mai mult pentru adaptarea la aplicație a circuitelor de tip MSI și LSI standardizate, care nu satisfac cu exactitate cerințele de proiectare. Ele vor fi utilizate acolo unde circuitele cu grad înalt de integrare nu pot fi folosite.

circuits synthesis are the following ones:

1. Problem specification;
2. Building the truth table, defining the function(s);
3. Minimizing the function(s);
4. Drawing the circuit's scheme.

There are several methods for implementing a CLC, differentiated by the degree of complexity of the integrated circuits used.

CLC synthesis using SSI integrated circuits

SSI – *small scale integration* – integrated circuits have less than 50 transistors integrated in the chip. Among these circuits we have the fundamental logic gates: NAND, NOR, NOT, AND, OR, XOR, XNOR.

After following the synthesis steps the logic function(s) implementation is realized by means of the existing integrated circuits. SSI CLCs are mostly used for adapting standardized MSI and LSI circuits to the application, if they don't satisfy exactly the design requirements. They will be used where the circuits with a higher degree of integration can not be used.

Sinteză CLC cu circuite integrate MSI

Circuitele integrate de tip MSI – *medium scale integration* – au până la 500 de tranzistoare integrate. Ele oferă structuri mai complexe, disponibile ca și structuri standard. Forma funcțiilor logice pe care dorim să le implementăm cu circuite de tip MSI trebuie să fie corelată cu circuitele disponibile. De obicei nu mai este necesară minimizarea funcțiilor.

Circuite combinaționale uzuale (specializate)

1. Convertoare de cod

Convertoarele de cod sunt CLC-uri care permit trecerea dintr-un cod binar în altul. La intrarea circuitului se aplică cuvintele unui cod și la ieșire se obține alt cod. Convertoarele de cod fac compatibilă funcționarea a 2 sisteme în care informația este codificată în mod diferit.

Exemplu: Conversia din cod Gray în cod binar.

Cuvintele de cod în cele două coduri sunt:

Gray:	g_n, g_{n-1}, \dots, g_0
Binar:	b_n, b_{n-1}, \dots, b_0

CLC synthesis using MSI integrated circuits

MSI – *medium scale integration* – integrated circuits have less than 500 integrated transistors. They offer more complex structures, which are available as standard structures.

The form of the logic functions that we wish to implement with MSI circuits must be correlated with the available circuits. Usually it is no longer necessary to minimize the functions.

Ordinary combinational circuits (specialized)

1. Code converters

Code converters are CLCs that allow the transformation from a binary code into another one. On the circuit's inputs are applied the words of a code and another code is obtained at its outputs. Code converters make compatible the functioning of two systems in which information is encoded differently.

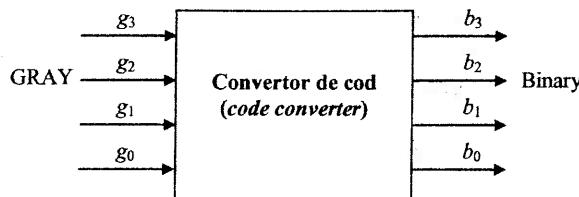
Example: The conversion from the Gray code into binary.

The code words in the two codes are:

Gray:	g_n, g_{n-1}, \dots, g_0
Binary:	b_n, b_{n-1}, \dots, b_0

Cutia neagră a circuitului este următoarea:

The circuit's black box is the following one:



Tabelul de adevăr al circuitului:

The circuit's truth table:

g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

În continuare se construiesc diagrame Karnaugh pentru determinarea funcțiilor minime pentru b_3, b_2, b_1, b_0 .

Now we will build the Karnaugh maps for determining the minimized functions for b_3, b_2, b_1, b_0 .

		g_0				
		00	01	11	10	
$g_3 g_2$		00	0	0	0	0
00	00	0	1	3	2	
01	01	0	0	7	6	
11	11	1	1	1	1	
10	10	1	1	1	1	

		g_0				
		00	01	11	10	
$g_3 g_2$		00	0	0	0	0
00	01	1	1	1	1	
01	11	0	0	0	0	
11	10	1	1	1	1	

 $b_3:$

		g_1				
		00	01	11	10	
$g_3 g_2$		00	0	0	1	1
00	01	1	1	0	0	
01	11	0	0	1	1	
11	10	1	1	0	0	

 $b_2:$

		g_1				
		00	01	11	10	
$g_3 g_2$		00	0	1	0	1
00	01	1	0	1	0	
01	11	0	1	0	1	
11	10	1	0	1	0	

 $b_1:$

Obținem:

 g_1 $b_0:$

We obtain:

$$\begin{cases} b_3 = g_3 \\ b_2 = g_3 \oplus g_2 \\ b_1 = g_3 \oplus g_2 \oplus g_1 \\ b_0 = g_3 \oplus g_2 \oplus g_1 \oplus g_0 \end{cases}$$

2. Codificatorul prioritatar

Codificatoarele sunt CLC la care activarea unei intrări conduce la apariția unui cuvânt de cod la ieșire. În practică, este posibil ca mai multe

2. Priority encoder

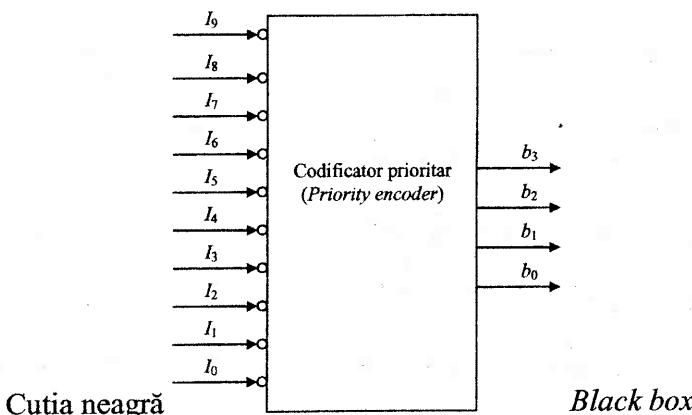
Encoders are CLCs at which an input activation leads to the apparition of a word code at the output. In practice, it is possible that several

intrări să fie active simultan. În acest caz, se stabilește o ierarhie a priorităților intrărilor și la ieșire va fi generat codul intrării celei mai prioritate.

Exemplu: Codificatorul priorităr de mai jos are intrările active pe 0. Intrarea care are cea mai mare prioritate este 9, iar cea mai mică prioritate o are intrarea 0.

inputs are simultaneously active. In this case, we will establish a hierarchy of the priorities and the circuit will generate at its outputs the code of the input with the highest priority.

Example: The priority encoder below has active-low inputs. The highest priority input is 9, and the lowest priority is assigned to input 0.



Intrări zecimale (decimal inputs)										BCD			
0	1	2	3	4	5	6	7	8	9	2^3	2^2	2^1	2^0
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	b_3	b_2	b_1	b_0
0	1	1	1	1	1	1	1	1	1	0	0	0	0
X	0	1	1	1	1	1	1	1	1	0	0	0	1
X	X	0	1	1	1	1	1	1	1	0	0	1	0
X	X	X	0	1	1	1	1	1	1	0	0	1	1
X	X	X	X	0	1	1	1	1	1	0	1	0	0
X	X	X	X	X	0	1	1	1	1	0	1	0	1
X	X	X	X	X	X	0	1	1	1	0	1	1	1
X	X	X	X	X	X	X	0	1	1	0	1	0	0
X	X	X	X	X	X	X	X	0	1	1	0	0	0
X	X	X	X	X	X	X	X	X	0	1	0	0	1

3. Decodificatorul

Decodificatoarele sunt CLC-uri la care se activează doar una dintre ieșiri, pentru combinația (codul) corespunzătoare a variabilelor de intrare. Ele au funcție inversă codificatoarelor. Ieșirile decodificatoarelor pot fi active pe 0 logic (funcționează în logică negativă) sau pe 1 logic.

În general, relația dintre numărul intrărilor și ieșirilor este următoarea: dacă avem m intrări, atunci vom avea $n \leq 2^m$ ieșiri. Cele mai frecvent folosite tipuri de decodificatoare sunt cele zecimale (cu 4 intrări și 10 ieșiri) și cele binare (cu 3 intrări și 8 ieșiri, cu 4 intrări și 16 ieșiri etc.).

Mai jos este prezentat un decodificator binar 3-la-8:

Cutia neagră

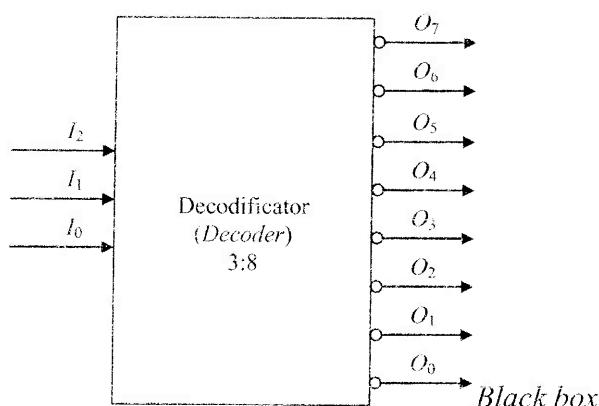
Tabelul de adevăr este următorul:

3. Decoder

Decoders are CLCs at which only one of the outputs is activated for a certain combination (code) of the input variables. Their function is complementary to the encoders' function. The decoders' outputs can be active-low (functions in negative logic) or active-high.

Generally speaking, the relationship between the number of inputs and the number of outputs is the following one: if we have m inputs, then we will have $n \leq 2^m$ outputs. The most widely used types of decoders are the decimal ones (with 4 inputs and 10 outputs) and the binary ones (with 3 inputs and 8 outputs, with 4 inputs and 16 outputs etc.).

Below is presented a 3-to-8 binary decoder:



The truth table is the following one:

I_2	I_1	I_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

4. Multiplexorul

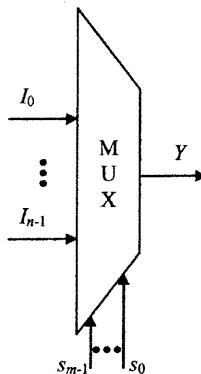
Multiplexoarele sunt CLC-uri care permit trecerea datelor de pe una din intrări la o ieșire unică. Selecția intrării se face printr-un cuvânt de cod de selecție numit și adresă.

Există deci 2 tipuri de intrări: intrări de date și intrări de selecție, și o singură ieșire.

4. Multiplexer

Multiplexers are CLCs that allow the passage of data from one of the inputs to an unique output. The selection of the input is done by means of a binary selection code also called address.

So there are 2 types of inputs: data inputs and selection inputs, and an unique output.



Dacă avem n intrări și m selecții, relația dintre m și n este în general următoarea: $n = 2^m$.

If we have n inputs and m selections, the relationship between m and n is generally: $n = 2^m$.

Din acest motiv, cele mai des întâlnite tipuri de multiplexoare sunt MUX 2:1, MUX 4:1, MUX 8:1, MUX 16:1.

Funcția realizată de ieșire este:

$Y = I_k$, unde k este numărul de combinație:

$$k = s_{m-1} \cdot 2^{m-1} + \dots + s_0 \cdot 2^0.$$

De exemplu, multiplexorul de tip 4:1 are 4 semnale de intrare, 2 semnale de selecție și un semnal de ieșire. Ecuarea ieșirii va fi:

$$Y = I_0 \cdot \overline{s_1} \cdot \overline{s_0} + I_1 \cdot \overline{s_1} \cdot s_0 + I_2 \cdot s_1 \cdot \overline{s_0} + I_3 \cdot s_1 \cdot s_0$$

Multiplexoarele integrate au disponibile atât ieșirea adeverată cât și cea negată, precum și o intrare de „Enable” pentru validare, care permite implementarea unei funcții SI suplimentare.

5. Demultiplexorul

Demultiplexoarele sunt circuitele complementare multiplexoarelor. Ele sunt CLC-uri care permit transmiterea datelor de pe o intrare de date comună pe una din ieșirile selectate. Selectarea ieșirii se face cu ajutorul unui cuvânt de cod de selecție numit și adresă.

Există deci 2 tipuri de intrări: intrarea de date unică și intrări de selecție, precum și mai multe ieșiri.

For this reason, the most popular types of multiplexers are 2:1 MUX, 4:1 MUX, 8:1 MUX, 16:1 MUX.

The function obtained at the output is:

$Y = I_k$, where k is the combination number:

$$k = s_{m-1} \cdot 2^{m-1} + \dots + s_0 \cdot 2^0.$$

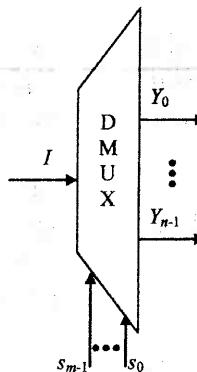
For instance, the 4:1 multiplexer has 4 input signals, 2 selection signals and one output signal. The output's equation will be:

Integrated multiplexers have available both the direct and the negated output, as well as a special input called “Enable” that allows to implement a supplemental AND function.

5. Demultiplexer

Demultiplexers are circuits that are complementary to the multiplexers. They are CLCs that allow data transmission from a common data input to one of the selected outputs. The output's selection is done by means of a binary selection code also called address.

So there are 2 types of inputs: the unique data input and selection inputs, as well as several outputs.



Dacă avem n intrări și m selecții, relația dintre m și n este în general următoarea: $n = 2^m$.

Din acest motiv, cele mai des întâlnite tipuri de demultiplexoare sunt DMUX 1:2, DMUX 1:4, DMUX 1:8, DMUX 1:16.

Funcția realizată de ieșire este:

$Y_k = I$, unde k este numărul de combinație:

$$k = s_{m-1} \cdot 2^{m-1} + \dots + s_0 \cdot 2^0.$$

Demultiplexoarele integrate au disponibilă și o intrare de „Enable” pentru validare, care permite implementarea unei funcții și suplimentare.

6. Comparatoare numerice

Comparatoarele numerice sunt CLC-uri care permit determinarea valorii relative a două numere binare. Comparatoarele pot fi de 1 bit sau de mai mulți biți.

Exemplu: Un comparator de 1 bit are următoarea structură:

If we have n inputs and m selections, the relationship between m and n is generally: $n = 2^m$.

For this reason, the most popular types of multiplexers are 1:2 DMUX, 1:4 DMUX, 1:8 DMUX, 1:16 DMUX.

The function realized by the output is:

$Y_k = I$, where k is the combination number:

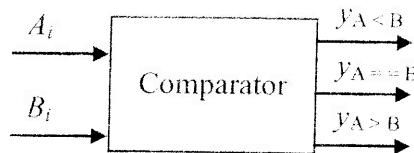
$$k = s_{m-1} \cdot 2^{m-1} + \dots + s_0 \cdot 2^0.$$

Integrated demultiplexers have also available a special input called “Enable” that allows to implement a supplemental AND function.

6. Digital comparators

Digital comparators are CLCs that allow determining the relative value of two binary numbers. Comparators can be of 1 bit or of several bits.

Example: A 1-bit comparator has the following structure:



Funcțiile de ieșire sunt:

The output functions are:

$$y_{A < B} = \overline{A_i} \bullet B_i \quad (A_i < B_i)$$

$$y_{A = B} = A_i \otimes B_i \quad (A_i = B_i)$$

$$y_{A > B} = A_i \bullet \overline{B_i} \quad (A_i > B_i)$$

7. Detectoare / generatoare de paritate

Detectoarele-generatoare de paritate sunt CLC-uri care au rolul de a determina și genera paritatea sau imparitatea numărului de variabile de intrare egale cu 1. Bitul de paritate este utilizat ca metodă de verificare a transferului de date. Sunt posibile 2 situații:

- numărul bițiilor de 1 + bitul de paritate = număr par
- numărul bițiilor de 1 + bitul de paritate = număr impar

Realizarea detectoarelor de paritate se bazează pe funcția logică SAU-EXCLUSIV (0 pentru par și 1 pentru impar).

8. Sumatoare / scăzătoare

Sumatoarele și scăzătoarele sunt CLC-uri care realizează adunarea, respectiv scăderea numerelor binare.

7. Parity detectors / checkers

Parity detectors / checkers are CLCs which have the role of determining and generating the parity or imparity of the number of input variables that are equal to 1. The parity bit is used as a method for verifying data transfers. Two situations can occur:

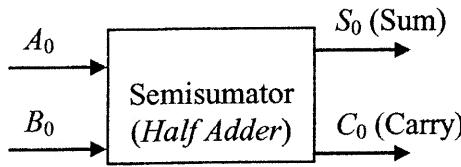
- the number of ones + the parity bit = even number
- the number of ones + the parity bit = odd number

The implementation of parity detectors is based on the XOR logic function (0 for even and 1 for odd).

8. Adders / subtracters

Adders and subtracters are CLCs that perform the addition and subtraction of binary numbers.

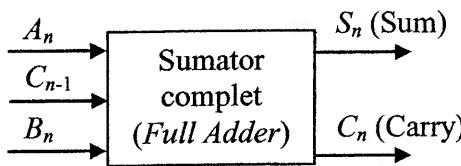
Semisumatorul este un CLC care efectuează suma a 2 numere binare de câte 1 bit, fără a ține cont de transportul de la bitul de semnificație imediat inferioară.



The *Half adder* is a CLC that computes the sum of two 1-bit binary numbers, without taking into consideration the carry from the previously significant bit.

$$\begin{aligned} S_0 &= A_0 \oplus B_0, \\ C_0 &= A_0 \bullet B_0 \end{aligned}$$

Sumatorul complet are în plus intrarea de transport de la bitul de rang imediat precedent:



The *Full adder* has one extra input, which represents the carry from the previously significant bit:

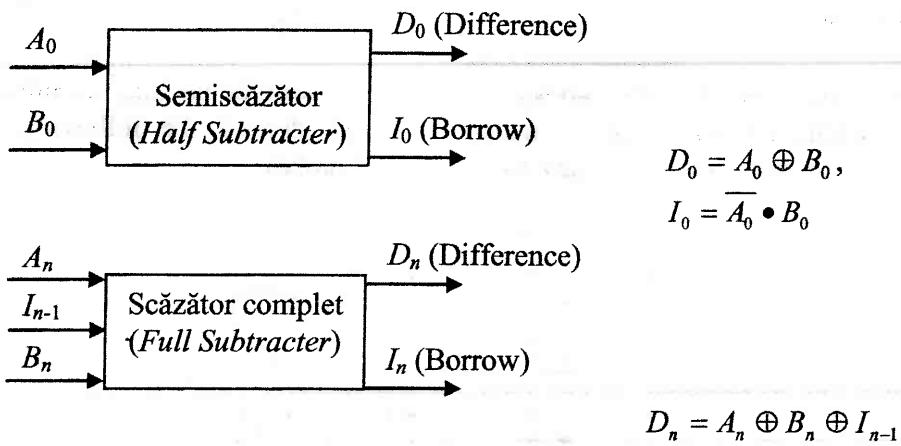
$$\begin{aligned} S_n &= A_n \oplus B_n \oplus C_{n-1} \\ C_n &= A_n \bullet B_n + C_{n-1} \bullet B_n + C_{n-1} \bullet A_n \end{aligned}$$

Sumatoarele pentru cuvinte binare de mai mulți biți se realizează prin interconectarea (cascadarea) sumatoarelor de 1 bit. Adunarea se efectuează în paralel, iar propagarea transportului în serie.

Semiscăzătorul de 1 bit are ieșirile D (Diferență) și I (Împrumut sau *Borrow*):

For binary words of more than 1 bit, the adders are made by interconnecting (cascading) 1-bit adders. The addition is performed in parallel, while carry propagation occurs serially.

The 1-bit half subtracter has the outputs D (Difference) and I (*Borrow*):



Scăzătorul complet de rangul n are ieșirile:

9. Unități aritmetico / logice

Unitățile aritmetico-logice sunt CLC-uri care realizează operații de tip aritmetic și operații de tip logic. Varietatea lor este foarte mare.

3.4. Probleme rezolvate

- Desenați structura internă a unui decodificator binar 3:8.
- Arătați că orice funcție Booleană poate fi implementată cu un decodificator și o poartă SAU. Cu ce poartă logică trebuie înlocuită poarta SAU dacă decodificatorul are ieșirile active pe 0?
- Exemplificați pentru funcția $f = \sum(0,1,2,4,6)$.

9. Arithmetic / logic units

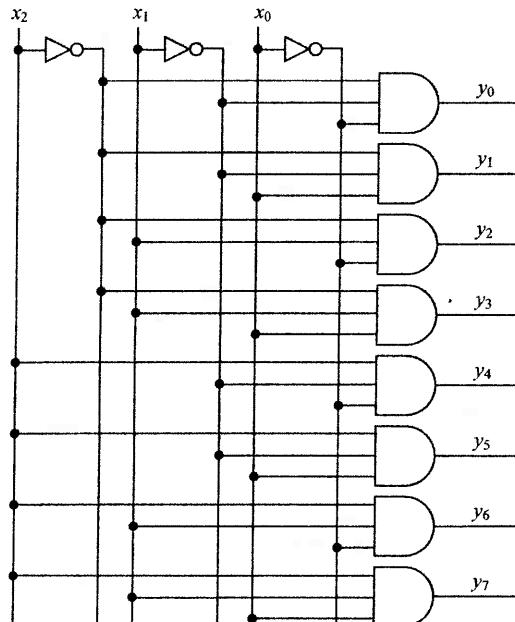
Arithmetic / logic units are CLCs that realize arithmetic operations and logic operations. There is a very large variety of such units.

3.4. Solved problems

- Draw the internal structure of a 3:8 binary decoder.
- Show that any Boolean function can be implemented with a decoder and an OR gate. Which other logic gate should be used instead of the OR gate if the decoder has active-low outputs?
- Exemplify for the function $f = \sum(0,1,2,4,6)$.

Solutie

a) Din tabelul de adevăr al decodificatorului se deduce următoarea structură internă a sa:



Observație: dacă porțile sunt řI-NU, atunci decodificatorul va avea ieșirile active pe 0.

b) După cum s-a văzut la punctul a), decodificatorul implementează toate produsele posibile ale variabilelor de intrare. În forma canonica disjunctivă, funcția Booleană este implementată ca o sumă-de-produse; de aceea, va fi suficient să utilizăm un SAU între termenii

Solution

a) From the decoder's truth table we can deduce the following internal structure:

Remark: if the logic gates are NAND gates, then the decoder will have active-low outputs.

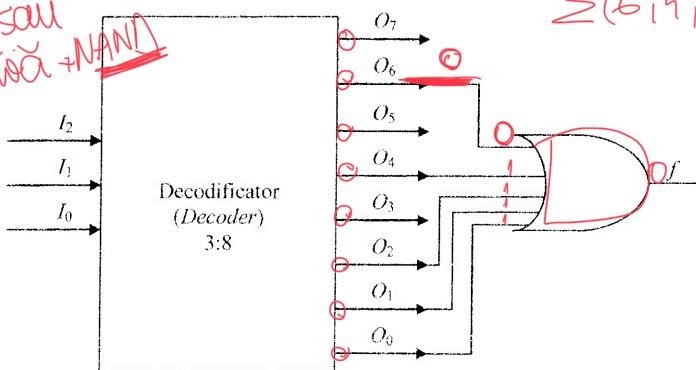
b) As we saw at point a), the decoder implements all the possible products of the input variables. In the disjunctive canonical form, the Boolean function is implemented as a sum-of-products; that is why, it will be sufficient to use an OR between those canonical terms of

canonici ai funcției care corespund ieșirilor decodificatorului.

Dacă decodificatorul are ieșirile active pe 0, conform teoremelor lui De Morgan, poarta SAU va trebui înlocuită cu o poartă ȘI-NU.

c) Dacă la intrare avem una din combinațiile 0, 1, 2, 4 sau 6, ieșirea corespunzătoare a decodificatorului se va activa (va avea valoarea 1), toate celelalte ieșiri ale sale fiind 0 (inactive). Funcția f va lua în acest caz valoarea 1.

*logice poz + sau
logică negativă + NAND*



Dacă la intrare avem una din celelalte combinații (3, 5 sau 7), ieșirea corespunzătoare a decodificatorului se va activa (va avea valoarea 1), toate celelalte ieșiri fiind 0 (inactive). Funcția f va lua în acest caz valoarea 0.

2. Desenați structura internă a unui codificator prioritar 4:2. Intrarea cea mai prioritată este 3.

the function that correspond to the decoder's outputs.

If the decoder has active-low outputs, according to De Morgan's laws, the OR gate will have to be replaced by a NAND gate.

c) If we have one of the combinations 0, 1, 2, 4 or 6 on the inputs, the corresponding decoder's output will be activated (it will have the value 1), while all its other outputs are 0 (inactive). The function f will have in this case the value 1.

$$\sum(0, 1, 2, 4, 6) \text{ OR}$$

A	B	f
1	0	0
0	1	1
1	1	0
0	0	0

NAND		
A	B	f
1	0	1
0	1	1
1	0	0
1	1	0

If we have one of the other combinations (3, 5 or 7), the corresponding decoder's output will be activated (it will have the value 1), while all its other outputs are 0 (inactive). The function f will have in this case the value 0.

2. Draw the internal structure of a 4:2 priority encoder, where input 3 has the greatest priority.

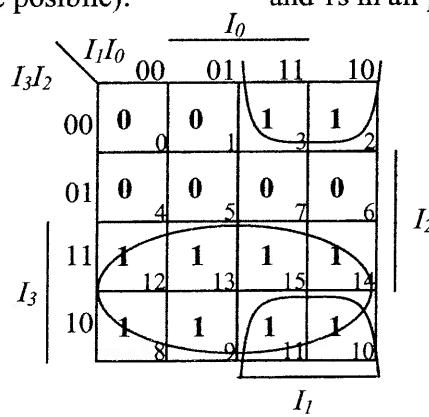
Soluție

Conform definiției codificatorului priorităr, avem următorul tabel de adevăr:

Intrări zecimale (decimal inputs)				BCD	
0	1	2	3	2^1	2^0
I_0	I_1	I_2	I_3	b_1	b_0
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

Pentru ieșirea b_1 putem observa direct expresia sa: $b_1 = I_3 + I_2 \cdot \overline{I_3}$.

Pentru ieșirea b_0 vom construi diagrama Karnaugh, ținând cont de faptul că celulele indiferente (X) din tabelul de adevăr, care apar în partea stângă a tabelului, trebuie expandate alcătuind toate combinațiile posibile (deci X-urile vor fi înlocuite cu 0 și 1 în toate combinațiile posibile):

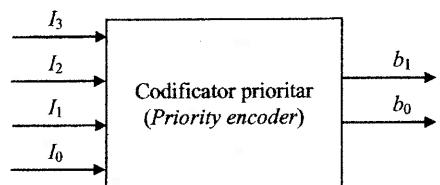


$$\text{Așadar: } b_0 = I_3 + \overline{I_2} \cdot I_1.$$

Solution

According to the priority encoder's definition, we have the following truth table:

Cutia neagră (black box)



For the b_1 output we can directly infer its expression: $b_1 = I_3 + I_2 \cdot \overline{I_3}$.

For the b_0 output we will build the Karnaugh map, considering that the don't care locations (X) in the truth table, that appear in the left side of the table, must be expanded by making all possible combinations (so the Xs will be replaced by 0s and 1s in all possible combinations):

$$\text{So: } b_0 = I_3 + \overline{I_2} \cdot I_1.$$

3. Folosind 2 multiplexoare 8:1, să se construiască (prin cascadare) un multiplexor 16:1.

Soluție

La acest gen de probleme este importantă corespondența dintre intrări și ieșiri, atât la componente cât și la modulul final. În acest caz:

- intrările de date: avem 16 la modulul final și câte 8 la cele două multiplexoare 8:1, deci este evident că există o corespondență 1:1 între ele.
- intrările de selecție: avem 4 la multiplexorul 16:1 și câte 3 la cele două multiplexoare 8:1, deci este evident că trebuie să facem o adaptare la acest nivel.
- ieșirile: avem o singură ieșire la multiplexorul 16:1 și câte 1 la cele două multiplexoare 8:1, deci este evident că trebuie să facem o adaptare la acest nivel.

Observăm că cele 16 intrări de date pot fi împărțite în 2 grupe egale: prima va conține intrările 0-7, repartizate fizic primului multiplexor 8:1, iar cealaltă intrările 8-15, repartizate fizic celui de-al doilea multiplexor 8:1. Dacă reprezentăm în binar numerele de la 0 la 15, avem nevoie de 4 biți s_3 , s_2 , s_1 și s_0 . Observăm că distincția dintre cele 2 grupe o face bitul cel mai semnificativ s_3 , care este 0

3. Using 2 8:1 multiplexers, build (by cascading them) a 16:1 multiplexer.

Solution

At this kind of problems it is important to match the inputs and outputs at the components and at the final module. In this case:

- data inputs: we have 16 of them at the final module and 8 of them at the level of each one of the two 8:1 multiplexers, so it is obvious that there is a 1:1 correspondence between them.
- selection inputs: there are 4 of them at the 16:1 multiplexer and 3 of them at each one of the two 8:1 multiplexers, so it is obvious that we must perform an adaptation at this level.
- outputs: there is only one output at the 16:1 multiplexer and 1 output at each of the two 8:1 multiplexers, so it is obvious that we must perform an adaptation at this level.

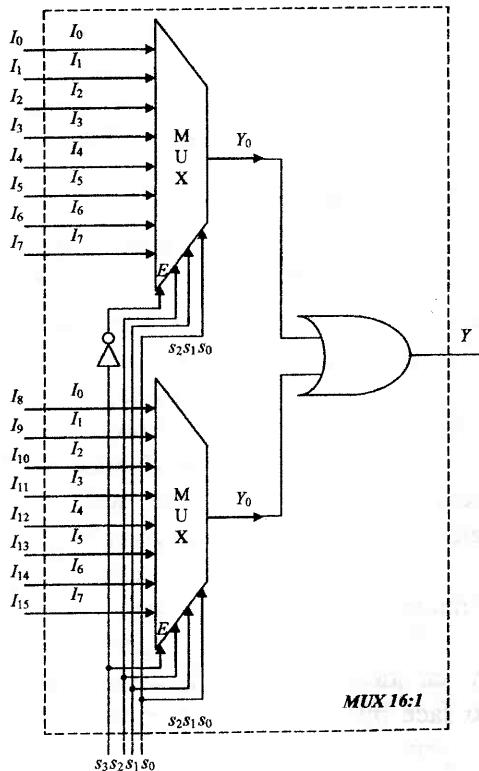
We observe that the 16 data inputs can be divided in 2 equal groups: the first one will contain the inputs 0-7, physically assigned to the first 8:1 multiplexer, while the other one will contain the inputs 8-15, physically assigned to the second 8:1 multiplexer. If we represent in binary the numbers from 0 to 15, we need 4 bits s_3 , s_2 , s_1 and s_0 . We observe that the distinction

pentru numerele din prima grupă și 1 pentru cele din a doua grupă. Trebuie deci să facem ca, atunci când se selectează o intrare de date dintre 0 și 7, să funcționeze primul multiplexor 8:1, iar atunci când se selectează o intrare de date dintre 8 și 15, să funcționeze al doilea multiplexor 8:1.

Această alternanță a multiplexorului activ o vom realiza folosind intrarea de selecție s_3 și intrarea suplimentară de *Enable* a fiecărui multiplexor 8:1. Schema finală este prezentată în figura următoare:

between the two groups is done by the most significant bit s_3 , which is 0 for the numbers from the first group and 1 for those from the second group. So, when a data input between 0 and 7 is selected, we must make the first 8:1 multiplexer work, and when a data input between 8 and 15 is selected, we must make the second 8:1 multiplexer work.

We alternate the active multiplexer using the selection input s_3 and the supplemental *Enable* input of each 8:1 multiplexer. The final scheme is presented in the next figure:



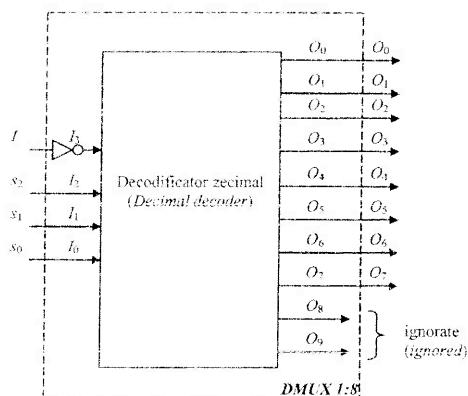
4. Să se realizeze un demultiplexor cu un decodificator zecimal.

Soluție

Pentru a rezolva această problemă trebuie să înțelegem foarte bine modul de funcționare al celor două componente. Prin simpla mapare a intrărilor și ieșirilor deducem că:

- decodificatorul zecimal are 4 intrări de date; demultiplexorul are 1 intrare de date și restul intrărilor sunt de selecție. Rezultă că putem implementa un demultiplexor 1:8 (care are 3 intrări de selecție).
- decodificatorul zecimal are 10 ieșiri; demultiplexorul are 8 ieșiri. Rezultă că ultimele două ieșiri ale decodificatorului vor fi ignorate.

Schema de corespondență este cea de mai jos:



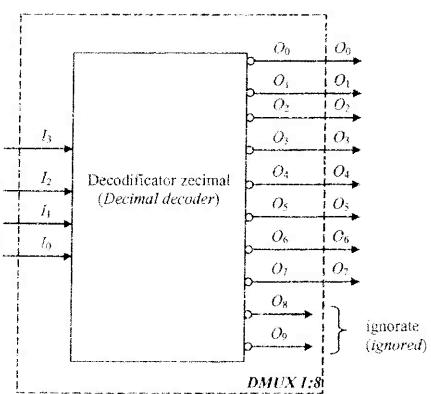
4. Design a demultiplexer using a decimal decoder.

Solution

To solve this problem we must understand very well the way these two components function. By simply mapping inputs and outputs we deduce that:

- the decimal decoder has 4 data inputs; the demultiplexer has 1 data input and the remaining inputs are selections. This means we can implement a 1:8 demultiplexer (which has 3 selection inputs).
- the decimal decoder has 10 outputs; the demultiplexer has 8 outputs. This means the last two outputs of the decoder will be ignored.

The corresponding scheme is given below:



Observăm că dacă ieșirile sunt active pe 1, este necesar un inversor la intrarea de date (I_3). Dacă ieșirile sunt active pe 0, decodificatorul este direct un demultiplexor: tot ce trebuie să facem este să considerăm intrările și ieșirile exact ca în schema de mai sus.

Pentru a înțelege mai bine schema, să luăm un exemplu:

- a) Presupunem că avem pe selecții „000” și pe intrarea de date ’0’.
- la nivelul demultiplexorului: trebuie ca intrarea de date să fie rutată pe ieșirea O_0 , care va avea atunci valoarea 0.
 - la nivelul decodificatorului: se va forma pe intrări combinația „0000”, care va avea ca efect activarea ieșirii O_0 , care va avea atunci valoarea 0.

- b) Presupunem că avem pe selecții „000” și pe intrarea de date ’1’.
- la nivelul demultiplexorului: trebuie ca intrarea de date să fie rutată pe ieșirea O_0 , care va avea atunci valoarea 1.
 - la nivelul decodificatorului: se va forma pe intrări combinația „1000”, care va avea ca efect activarea ieșirii O_8 , care va avea atunci valoarea 0, toate celelalte ieșiri fiind 1. Aceasta înseamnă că și O_0 va avea valoarea 1, exact aşa cum este necesar pentru ca decodificatorul să se comporte ca un demultiplexor.

We observe that if the outputs are active-high, an inverter is necessary on the data input (I_3). If the outputs are active-low, the decoder is directly a demultiplexer: all we have to do is to consider the inputs and outputs exactly as in the scheme above.

In order to better understand the scheme, let's take an example:

- a) Suppose we have “000” on the selections and ’0’ on the data input.
- at the level of the demultiplexer: the data input must be routed on the output O_0 , which will then have the value 0.
 - at the level of the decoder: the combination “0000” will be formed on the inputs; as an effect, the output O_0 will be activated (will have the value 0).
- b) Suppose we have “000” on the selections and ’1’ on the data input.
- at the level of the demultiplexer: the data input must be routed on the output O_0 , which will then have the value 1.
 - at the level of the decoder: the combination “1000” will be formed on the inputs; as an effect, the output O_8 will be activated (will have the value 0), all the other outputs being 1. This means that O_0 will also have the value 1, exactly as it is necessary for the decoder to behave as a demultiplexer.

5. Construiți un demultiplexor 1:40 folosind o rețea arborescentă de 6 decodificatoare zecimale. *Indicație:* se va folosi problema rezolvată 4.

Soluție

Cele 40 de ieșiri vor fi date de 5 demultiplexoare a către 8 ieșiri, implementate cu ajutorul unor decodificatoare zecimale exact aşa cum s-a prezentat la problema 4.

Numerele de la 0 la 39 se reprezintă în binar pe 6 biți: $s_5s_4s_3s_2s_1s_0$. Selectarea unei ieșiri va avea ca efect „scoaterea” semnalului de la intrarea de date pe unul din cele 5 demultiplexoare, celelalte 4 rămânând inactive (neselectate).

Primul demultiplexor primește selecțiile $s_5s_4s_3$, cu ajutorul cărora determinăm în care grup de 8 ieșiri se încadrează ieșirea selectată: 0-7, 8-15, 16-23, 24-31 sau 32-39. Apoi, intrarea de date e rutată la demultiplexorul corespunzător aceluia grup, urmând ca selecțiile $s_2s_1s_0$ să determine care ieșire din grupul respectiv să fie selectată (prima, a doua... etc.).

Putem vedea această organizare ca având o caracteristică multi-nivel: există o adresă „de bloc” (sau „de grup”, în cazul nostru) și o adresă de tip „deplasament în cadrul blocului” (sau „de offset”). Adresa de bloc este dată de selecțiile $s_5s_4s_3$, iar deplasamentul este dat de $s_2s_1s_0$.

5. Build a 1:40 demultiplexer using a tree of 6 decimal decoders. *Hint:* use the solved problem 4.

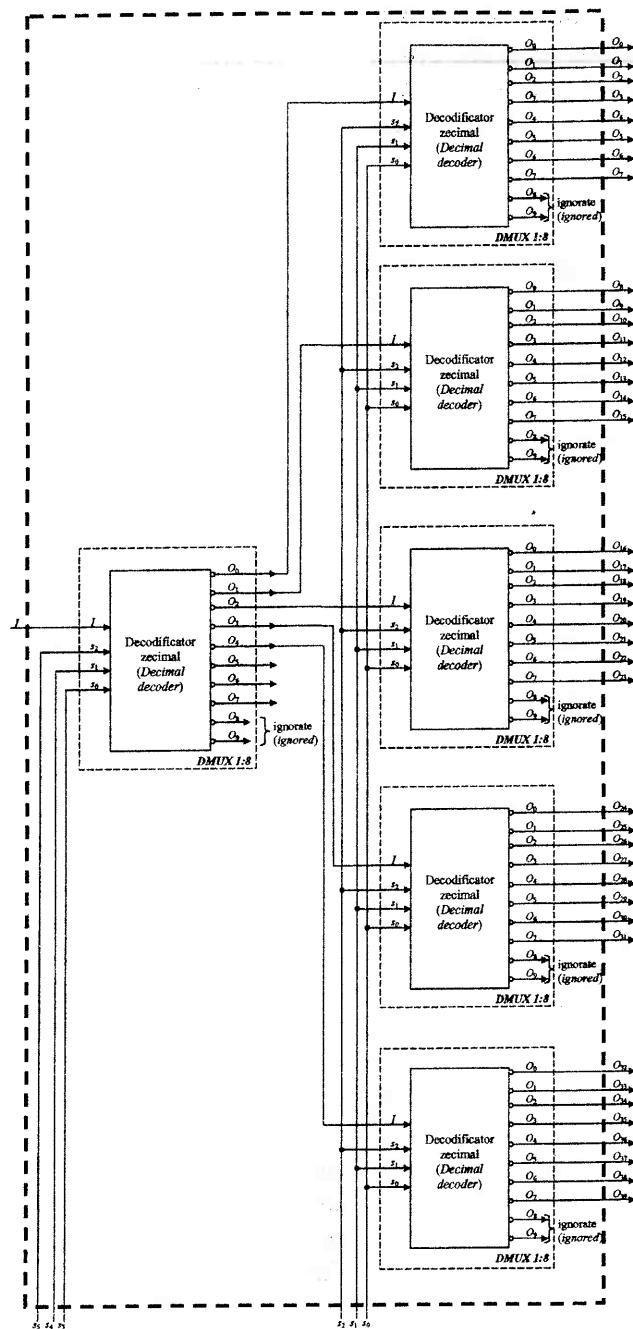
Solution

The 40 outputs will be given by 5 1:8 demultiplexers, implemented with decimal decoders exactly as presented at problem 4.

The numbers from 0 to 39 are represented in binary on 6 bits: $s_5s_4s_3s_2s_1s_0$. Selecting an output will have as an effect „getting out” the signal from the data input on one of the 5 demultiplexers, the four others remaining inactive (unselected).

The first demultiplexer receives the selections $s_5s_4s_3$, by means of whom it determines in which group of 8 outputs will range the selected output: 0-7, 8-15, 16-23, 24-31 or 32-39. Then, the data input is routed at the demultiplexer that corresponds to that particular group of 8, while the selections $s_2s_1s_0$ will determine which output from that group will be selected (the first one, the second one... etc.)

We can see this structure as having a multi-level characteristic: there is a “block address” (or „group address”, in our case) and an “offset inside the block”. The block address is given by the selections $s_5s_4s_3$, while the offset is given by $s_2s_1s_0$.



6. Implementați funcția $f = \sum (0,1,2,3,11,12,14,15)$ folosind:

- a) un multiplexor 16:1
- b) un multiplexor 8:1
- c) un multiplexor 4:1
- d) un multiplexor 2:1

Solutie

În general, atunci când avem de implementat o funcție Booleană cu un multiplexor, ideea de bază este să plasăm variabilele de intrare ale funcției pe selecțiile multiplexorului, în măsura în care numărul de selecții este suficient. Dacă multiplexorul nu dispune de suficiente intrări de selecție, este necesar să facem adaptări suplimentare.

În cazul nostru, observăm că funcția este de 4 variabile (cel mai mare termen canonic, 15, se reprezintă pe cel puțin 4 biți). Vom începe prin a construi diagrama Karnaugh:

		x_0			
		00	01	11	10
x_3x_2	00	1 0	1 1	1 3	1 2
	01	0 4	0 5	0 7	0 6
x_3	11	1 12	0 13	1 15	1 14
	10	0 8	0 9	1 11	0 10

x_1

6. Implement the function $f = \sum (0,1,2,3,11,12,14,15)$ using:

- a) a 16:1 multiplexer
- b) an 8:1 multiplexer
- c) a 4:1 multiplexer
- d) a 2:1 multiplexer

Solution

Usually, when we have to implement a Boolean function using a multiplexer, the basic idea is to place the function's input variables on the multiplexer's selections, if the number of selection inputs is sufficient. If the multiplexer does not have enough selection inputs, it is necessary to make supplemental adaptations.

In our case, we observe that this function is a four-variable one (the greatest canonical term, 15, is represented on at least 4 bits). We will start by building the Karnaugh map:

a) Implementarea funcției cu un multiplexor 16:1 este cea mai simplă. Vom plasa pe cele 4 selecții ale multiplexorului intrările $x_3x_2x_1x_0$ ale funcției, iar pe intrările de date – valorile funcției în fiecare punct al domeniului de definiție, conform diagramei Karnaugh (vezi figura următoare).

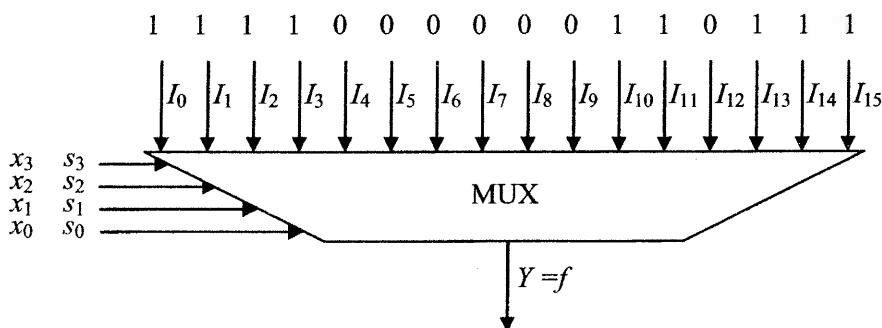
Pentru a înțelege modul de funcționare, să dăm niște valori pe intrările de selecție:

- dacă pe $s_3s_2s_1s_0$ avem „0000”, intrarea de date I_0 va fi rutată la ieșirea multiplexorului. Dar pe I_0 avem ‘1’, deci funcția va lua valoarea ‘1’.
- dacă pe $s_3s_2s_1s_0$ avem „0100”, intrarea de date I_4 va fi rutată la ieșirea multiplexorului. Dar pe I_4 avem ‘0’, deci funcția va lua valoarea ‘0’.
- etc. – se procedează în mod analog cu toate celelalte combinații ale variabilelor de intrare ale funcției.

a) The function's implementation with a 16:1 multiplexer is the simplest one. We will place on the four selections of the multiplexer the inputs $x_3x_2x_1x_0$ of the function, while on the data inputs – the function's values in each point of the definition domain, according to the Karnaugh map (see the next figure).

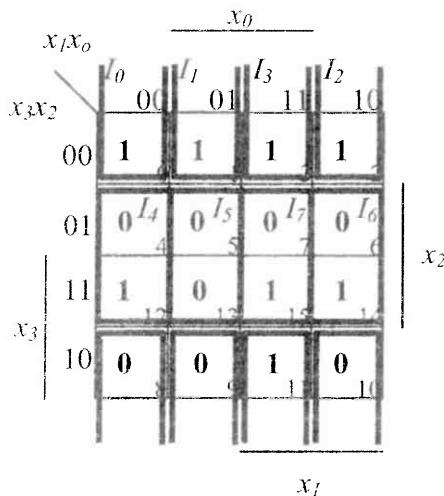
In order to understand the functioning way, let's give some values on the selection inputs:

- if on $s_3s_2s_1s_0$ we have “0000”, the I_0 data input will be routed out on the multiplexer's output. But on I_0 we have ‘1’, so the function will take the value ‘1’.
- if on $s_3s_2s_1s_0$ we have “1000”, the I_4 data input will be routed out on the multiplexer's output. But on I_4 we have ‘0’, so the function will take the value ‘0’.
- etc. – the same process is followed similarly with all the other combinations of the function's input variables.



b) Dacă nu avem la dispoziție decât un multiplexor 8:1, pe selecții vom plasa 3 variabile de intrare, iar cea de-a patra va influența formarea rezultatului participând pe intrările de date ale multiplexorului. Așadar, pe selecțiile $s_2s_1s_0$ ale multiplexorului vom plasa intrările $x_2x_1x_0$ ale funcției, iar toate intrările de date ale multiplexorului vor fi funcții de x_3 : $I_j = f(x_3)$, $j = \overline{0,7}$.

Examinând diagrama Karnaugh, vom descoperi zonele care corespund intrărilor de date $I_0, I_1 \dots I_7$:



b) If we only have an 8:1 multiplexer available, on the selections we will place 3 input variables, while the fourth one will influence the result by participating on the multiplexer's data inputs. So, on the multiplexer's selections $s_2s_1s_0$ we will place the function's inputs $x_2x_1x_0$, and all the multiplexer's data inputs will be functions of x_3 : $I_j = f(x_3)$, $j = \overline{0,7}$.

By examining the Karnaugh map, we will discover the regions that correspond to the data inputs $I_0, I_1 \dots I_7$:

- la nivelul multiplexorului: selecțiile $s_2s_1s_0 = „000”$ corespund intrării de date I_0 .
- la nivelul diagramei Karnaugh: deoarece pe $s_2s_1s_0$ avem intrările $x_2x_1x_0$ ale funcției, zona corespunzătoare din diagrama

- at the level of the multiplexer: the selections $s_2s_1s_0 = “000”$ correspond to the I_0 data input.
- at the level of the Karnaugh map: since on $s_2s_1s_0$ we have the function's inputs $x_2x_1x_0$, the corresponding region in the

Karnaugh este cea a celulelor ale căror coordonate se termină cu „000”. Acestea sunt celulele 0 și 8. În concluzie, intrările de date I_0 îi corespund celulele 0 și 8.

Analog, găsim că:

- intrările de date I_1 îi corespund celulele 1 și 9.
- intrările de date I_2 îi corespund celulele 2 și 10.
- intrările de date I_3 îi corespund celulele 3 și 11.
- intrările de date I_4 îi corespund celulele 4 și 12.
- intrările de date I_5 îi corespund celulele 5 și 13.
- intrările de date I_6 îi corespund celulele 6 și 14.
- intrările de date I_7 îi corespund celulele 7 și 15.

Știind că fiecare intrare de date este o funcție de x_3 , rămâne să stabilim expresia acestei funcții.

- pentru I_0 : observăm că atunci când $x_3 = 0$ (în celula 0), $I_0 = 1$, iar când $x_3 = 1$ (în celula 8), $I_0 = 0$. Atunci putem spune că $I_0 = x_3$.
- pentru I_1 : observăm că atunci când $x_3 = 0$ (în celula 1), $I_1 = 1$, iar când $x_3 = 1$ (în celula 9), $I_1 = 0$. Atunci putem spune că $I_1 = \overline{x_3}$.
- pentru I_2 : observăm că atunci când $x_3 = 0$ (în celula 2), $I_2 = 1$, iar când $x_3 = 1$ (în celula 10), $I_2 = 0$. Atunci putem spune că $I_2 = x_3$.
- pentru I_3 : observăm că atunci când

Karnaugh map is the group of cells whose coordinates end by „000”. These are the cells 0 and 8.

In conclusion, to the data input I_0 correspond the cells 0 and 8.

Similarly, we find that:

- to the data input I_1 correspond the cells 1 and 9.
- to the data input I_2 correspond the cells 2 and 10.
- to the data input I_3 correspond the cells 3 and 11.
- to the data input I_4 correspond the cells 4 and 12.
- to the data input I_5 correspond the cells 5 and 13.
- to the data input I_6 correspond the cells 6 and 14.
- to the data input I_7 correspond the cells 7 and 15.

Knowing that each data input is a function of x_3 , we still have to determine the function's expression.

- for I_0 : we observe that when $x_3 = 0$ (in cell 0), $I_0 = 1$, and when $x_3 = 1$ (in cell 8), $I_0 = 0$. Then we can say that $I_0 = x_3$.
- for I_1 : we observe that when $x_3 = 0$ (in cell 1), $I_1 = 1$, and when $x_3 = 1$ (in cell 9), $I_1 = 0$. Then we can say that $I_1 = \overline{x_3}$.

- for I_2 : we observe that when $x_3 = 0$ (in cell 2), $I_2 = 1$, and when $x_3 = 1$ (in cell 10), $I_2 = 0$. Then we can say that $I_2 = x_3$.

- for I_3 : we observe that when $x_3 = 0$

$x_3 = 0$ (în celula 3), $I_3 = 1$, iar când $x_3 = 1$ (în celula 11), $I_3 = 1$. Atunci putem spune că $I_3 = x_3$.

- pentru I_4 : observăm că atunci când $x_3 = 0$ (în celula 4), $I_4 = 0$, iar când $x_3 = 1$ (în celula 12), $I_4 = 1$. Atunci putem spune că $I_4 = x_3$.

- pentru I_5 : observăm că atunci când $x_3 = 0$ (în celula 5), $I_5 = 0$, iar când $x_3 = 1$ (în celula 13), $I_5 = 0$. Atunci putem spune că $I_5 = 0$.

- pentru I_6 : observăm că atunci când $x_3 = 0$ (în celula 6), $I_6 = 0$, iar când $x_3 = 1$ (în celula 14), $I_6 = 1$. Atunci putem spune că $I_6 = x_3$.

- pentru I_7 : observăm că atunci când $x_3 = 0$ (în celula 7), $I_7 = 0$, iar când $x_3 = 1$ (în celula 15), $I_7 = 1$. Atunci putem spune că $I_7 = x_3$.

Implementarea este redată în figura următoare:

(in cell 3), $I_3 = 1$, and when $x_3 = 1$ (in cell 11), $I_3 = 1$. Then we can say that $I_3 = 1$.

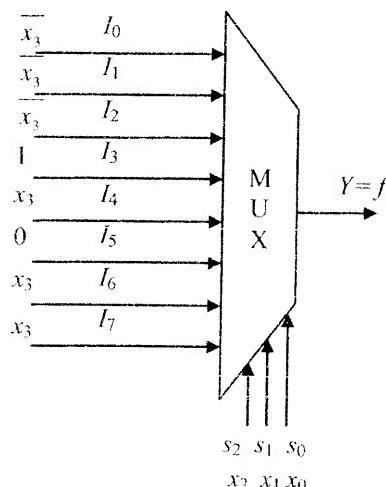
- for I_4 : we observe that when $x_3 = 0$ (in cell 4), $I_4 = 0$, and when $x_3 = 1$ (in cell 12), $I_4 = 1$. Then we can say that $I_4 = x_3$.

- for I_5 : we observe that when $x_3 = 0$ (in cell 5), $I_5 = 0$, and when $x_3 = 1$ (in cell 13), $I_5 = 0$. Then we can say that $I_5 = 0$.

- for I_6 : we observe that when $x_3 = 0$ (in cell 6), $I_6 = 0$, and when $x_3 = 1$ (in cell 14), $I_6 = 1$. Then we can say that $I_6 = x_3$.

- for I_7 : we observe that when $x_3 = 0$ (in cell 7), $I_7 = 0$, and when $x_3 = 1$ (in cell 15), $I_7 = 1$. Then we can say that $I_7 = x_3$.

The implementation is presented in the next figure:



c) Dacă nu avem la dispoziție decât un multiplexor 4:1, pe selecții vom plasa 2 variabile de intrare, iar cea de-a treia și cea de-a patra vor influența formarea rezultatului participând pe intrările de date ale multiplexorului. Așadar, pe selecțiile s_1s_0 ale multiplexorului vom plasa intrările x_1x_0 ale funcției, iar toate intrările de date ale multiplexorului vor fi funcții de x_3 și de x_2 : $I_j = f(x_3, x_2)$, $j = \overline{0,3}$.

Examinând diagrama Karnaugh, vom descoperi zonele care corespund intrărilor de date I_0 , I_1 , I_2 și I_3 :

c) If we only have a 4:1 multiplexer available, on the selections we will place 2 input variables, while the third and the fourth ones will influence the result by participating on the multiplexer's data inputs. So, on the multiplexer's selections s_1s_0 we will place the function's inputs x_1x_0 , and all the multiplexer's data inputs will be functions of x_3 and x_2 :

$$I_j = f(x_3, x_2), j = \overline{0,3}.$$

By examining the Karnaugh map, we will discover the regions that correspond to the data inputs I_0 , I_1 , I_2 and I_3 :

		x_0				
		00	01	11	10	
x_3x_2		00	1 I_0 0	1 I_1 1	1 I_3 3	1 I_2 2
		01	0 4	0 5	0 7	0 6
		11	1 12	0 13	1 15	1 14
		10	0 8	0 9	1 11	0 10

x_1

- la nivelul multiplexorului: selecțiile $s_1s_0 = „00”$ corespund intrării de date I_0 .
- la nivelul diagramei Karnaugh: deoarece pe s_1s_0 avem intrările x_1x_0 ale funcției, zona corespunzătoare

- at the level of the multiplexer: the selections $s_1s_0 = “00”$ correspond to the I_0 data input.
- at the level of the Karnaugh map: since on s_1s_0 we have the function's inputs x_1x_0 , the corresponding

din diagrama Karnaugh este cea a celulelor ale căror coordonate se termină cu „00”. Acestea sunt celulele 0, 4, 12 și 8.

În concluzie, intrările de date I_0 îi corespund celulele 0, 4, 12 și 8.

Analog, găsim că:

- intrările de date I_1 îi corespund celulele 1, 5, 13 și 9.

- intrările de date I_2 îi corespund celulele 2, 6, 14 și 10.

- intrările de date I_3 îi corespund celulele 3, 7, 15 și 11.

Știind că fiecare intrare de date este o funcție de x_3 și de x_2 , rămâne să stabilim expresia acestei funcții.

Putem observa direct expresia fiecărei funcții, sau vom construi tabelul de adevăr sau diagrama Karnaugh a funcțiilor de 2 variabile (x_3 și x_2).

Tabelul de adevăr al celor 4 funcții este următorul:

x_3	x_2	I_0	I_1	I_2	I_3
0	0	1	1	1	1
0	1	0	0	0	0
1	0	0	0	0	1
1	1	1	0	1	1

Deducem că:

region in the Karnaugh map is the group of cells whose coordinates end by “00”. These are the cells 0, 4, 12 and 8.

In conclusion, to the data input I_0 correspond the cells 0, 4, 12 and 8.

Similarly, we find that:

- to the data input I_1 correspond the cells 1, 5, 13 and 9.

- to the data input I_2 correspond the cells 2, 6, 14 and 10.

- to the data input I_3 correspond the cells 3, 7, 15 and 11.

Knowing that each data input is a function of x_3 and x_2 , we still have to determine this function's expression.

We can observe directly the expression of each function, or we will build the truth table or the Karnaugh map of the two-variable (x_3 and x_2) functions.

The truth table of the four functions is the following one:

We deduce that:

$$\begin{cases} I_0 = \overline{x_3} \otimes x_2 \\ I_1 = \overline{x_3} \bullet x_2 \\ I_2 = x_3 \otimes \overline{x_2} \\ I_3 = x_3 + \overline{x_2} \end{cases}$$

O altă metodă, alternativă, se bazează pe folosirea formei canonice disjunctive. Expresia funcției în FCD este următoarea:

$$f_{FCD} = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_3} \cdot \overline{x_2} \cdot x_1 \cdot \overline{x_0} + \overline{x_3} \cdot \overline{x_2} \cdot x_1 \cdot x_0 + \\ + x_3 \cdot \overline{x_2} \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x_0} + x_3 \cdot x_2 \cdot x_1 \cdot x_0 +$$

- intrarea de date I_0 corespunde termenilor care conțin expresia $\overline{x_1} \cdot x_0$
- intrarea de date I_1 corespunde termenilor care conțin expresia $x_1 \cdot \overline{x_0}$
- intrarea de date I_2 corespunde termenilor care conțin expresia $x_1 \cdot \overline{x_0}$
- intrarea de date I_3 corespunde termenilor care conțin expresia $x_1 \cdot x_0$.

Rescriind funcția f și grupând termenii dând factor comun aceste expresii, obținem:

$$f = \overline{x_1} \cdot \overline{x_0} \cdot (\overline{x_3} \cdot \overline{x_2} + x_3 \cdot x_2) + \overline{x_1} \cdot x_0 \cdot \overline{x_3} \cdot \overline{x_2} + x_1 \cdot \overline{x_0} \cdot (\overline{x_3} \cdot \overline{x_2} + x_3 \cdot x_2) + \\ + x_1 \cdot x_0 \cdot (\overline{x_3} \cdot \overline{x_2} + x_3 \cdot x_2 + x_3 \cdot x_2)$$

Se observă că ultima expresie din paranteze se simplifică la expresia $x_3 + \overline{x_2}$.

Implementarea este redată în figura următoare:

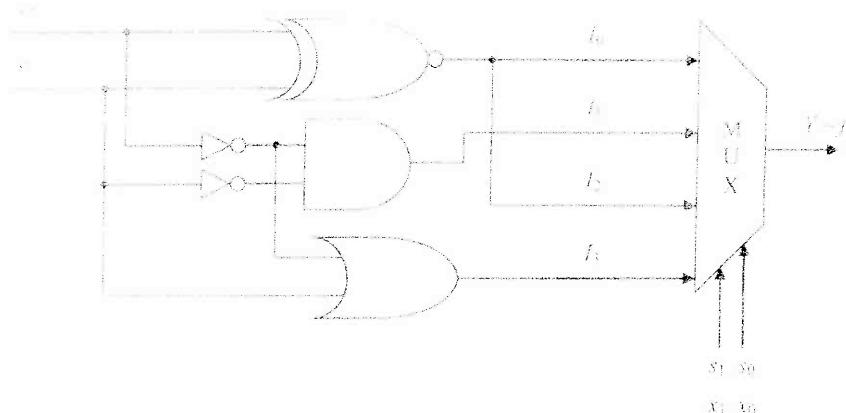
Another method (an alternative one), is based on using the disjunctive canonical form. The function's expression in the DCF is the following one:

- the I_0 data input corresponds to the terms that contain the expression $\overline{x_1} \cdot x_0$
- the I_1 data input corresponds to the terms that contain the expression $x_1 \cdot \overline{x_0}$
- the I_2 data input corresponds to the terms that contain the expression $x_1 \cdot \overline{x_0}$
- the I_3 data input corresponds to the terms that contain the expression $x_1 \cdot x_0$.

By rewriting the function f and by grouping the terms by factorizing these expressions, we obtain:

We notice that the last expression between brackets is simplified to the expression $x_3 + \overline{x_2}$.

The implementation is shown in the next figure:



d) Dacă nu avem la dispoziție decât un multiplexor 2:1, pe selecții vom păsa 1 variabilă de intrare, iar cea de-a doua, a treia și a patra vor influența formarea rezultatului participând pe intrările de date ale multiplexorului. Așadar, pe selecția s_0 ale multiplexorului vom păsa intrarea x_3 a funcției (pentru a varia puțin, nu vom pune x_0 de data această), iar toate intrările de date ale multiplexorului vor fi funcții de x_2 , x_1 și x_0 .

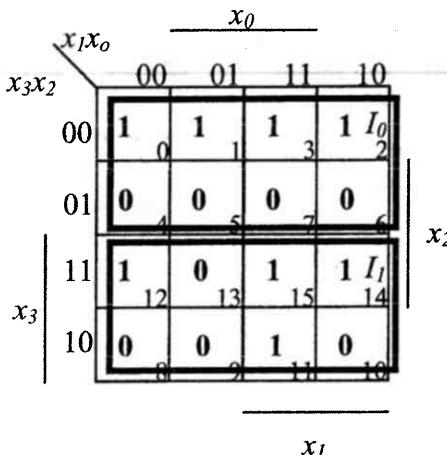
$$I_0 = f(x_2, x_1, x_0), f = \overline{0,1}.$$

Examinând diagrama Karnaugh, vom descoperi zonele care corespund intrărilor de date I_0 și I_1 :

c) If we only have a 2:1 multiplexer available, on the selections we will place 1 input variable, while the second, the third and the fourth ones will influence the result by participating on the multiplexer's data inputs. So, on the multiplexer's selection s_0 we will place the function's input x_3 (to make a little variation, this time we will not put x_0 !), and all the multiplexer's data inputs will be functions of x_2 , x_1 and x_0 .

$$I_0 = f(x_2, x_1, x_0), f = \overline{0,1}.$$

By examining the Karnaugh map, we will discover the regions that correspond to the data inputs I_0 and I_1 :



- la nivelul multiplexorului: selecția $s_0 = „0”$ corespunde intrării de date I_0 .

- la nivelul diagramei Karnaugh: deoarece pe s_0 avem intrarea x_3 a funcției, zona corespunzătoare din diagrama Karnaugh este cea a celulelor ale căror coordonate încep cu „0”. Acestea sunt celulele 0, 1, 2, 3, 4, 5, 6 și 7.

În concluzie, intrării de date I_0 îi corespund celulele 0, 1, 2, 3, 4, 5, 6 și 7.

Analog, găsim că intrării de date I_1 îi corespund celulele 8, 9, 10, 11, 12, 13, 14 și 15.

Știind că fiecare intrare de date este o funcție de x_2 , x_1 și de x_0 , rămâne să stabilim expresia acestei funcții.

Vom construi diagramele Karnaugh a celor două funcții de 3 variabile (x_2 , x_1 și x_0).

- at the level of the multiplexer: the selection $s_0 = “0”$ corresponds to the I_0 data input.

- at the level of the Karnaugh map: since on s_0 we have the function's input x_3 , the corresponding region in the Karnaugh map is the group of cells whose coordinates start by “0”. These are the cells 0, 1, 2, 3, 4, 5, 6 and 7.

In conclusion, to the data input I_0 correspond the cells 0, 1, 2, 3, 4, 5, 6 and 7.

Similarly, we find that to the data input I_1 correspond the cells 8, 9, 10, 11, 12, 13, 14 and 15.

Knowing that each data input is a function of x_2 , x_1 and x_0 , we still have to determine the function's expression.

We will build the Karnaugh maps of the two three-variable (x_2 , x_1 and x_0) functions.

		X ₀				
		00	01	11	10	
X ₂		0	1 0	1 1	1 3	1 2
X ₁	0	0 4	0 5	0 7	0 6	
	1					

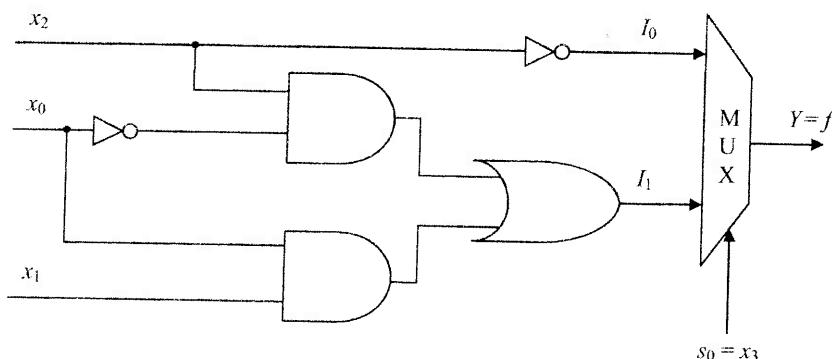
		X ₀				
		00	01	11	10	
X ₂		0	0 0	0 1	1 3	0 2
X ₁	0	1 4	0 5	1 7	1 6	
	1					

$$I_0 = \overline{x_2}$$

Implementarea este redată în figura de mai jos:

$$I_1 = x_2 \cdot \overline{x_0} + x_1 \cdot x_0$$

The implementation is shown in the figure below:



7. Implementați funcția

$$f = \sum(0, 2, 3, 4, 7, 8, 9) + \sum_{\Phi}(6, 10, 11, 12, 13, 14, 15)$$

folosind numai un decodificator zecimal (fără nici o altă poartă sau componentă logică). *Observație:* notația \sum_{Φ} indică termenii *indiferenți*.

Soluție

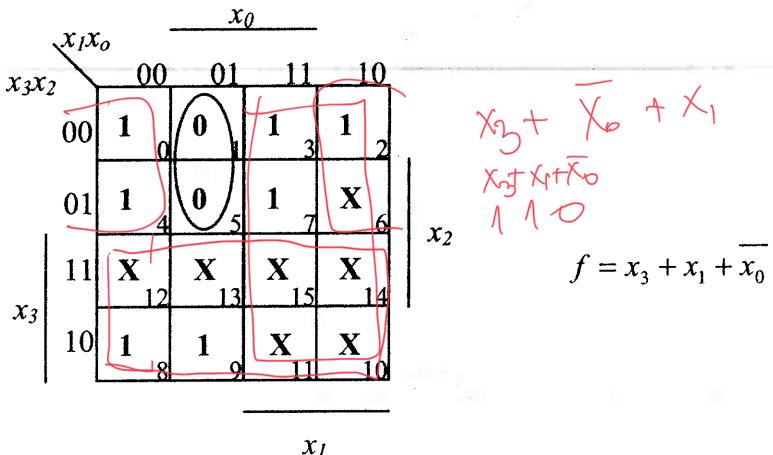
Vom folosi diagrama Karnaugh și vom minimiza funcția:

7. Implement the function

using only one decimal decoder (without any other logic gate or component). *Remark:* the \sum_{Φ} notation indicates the *don't care* terms.

Solution

We will use the Karnaugh map and we will minimize the function:

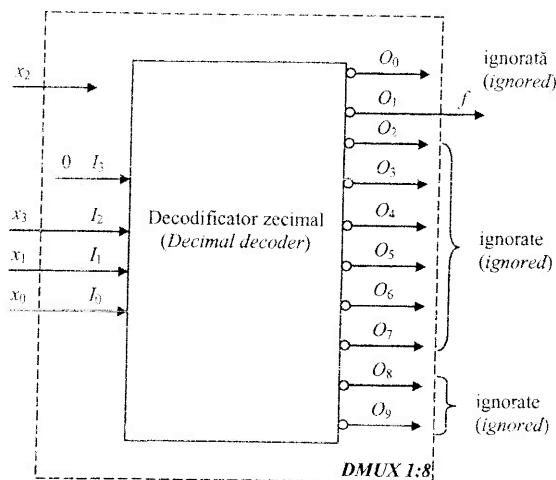


Observăm că funcția se poate scrie ca depinzând numai de 3 variabile. Vom folosi decodificatorul zecimal (cu ieșirile active pe 0!) în configurație de demultiplexor (ca la problemele 4 și 5), pe selecții plasând variabilele de intrare x_3 , x_1 și x_0 .

Din diagrama Karnaugh remarcăm că ieșirea funcției este „0” atunci când pe $x_3x_1x_0$ avem „001”. În consecință, pe intrarea de date I_3 a demultiplexorului vom pune mereu „0”, iar ieșirea O_1 a decodificatorului zecimal este chiar ieșirea funcției.

We notice that the function can be written as depending only on 3 variables. We will use the decimal decoder (with active-low outputs!) in the demultiplexer configuration (as in problems 4 and 5). On the selections we will place the x_3 , x_1 and x_0 input variables.

From the Karnaugh map we observe that the function's output is '0' when on $x_3x_1x_0$ we have "001". As a matter of consequence, on the demultiplexer's I_3 data input we will put always '0', while the output O_1 of the decimal decoder is the function's output.



Se observă că \$x_2\$ nu are nici o influență asupra generării valorii funcției.

8. Implementați funcția:

$$f(A, B, C, D, E) = A + \bar{C} \cdot D + B \cdot \bar{D} + \bar{B} \cdot D + \bar{B} \cdot C \cdot E$$

folosind *numai* un multiplexor. Sunt disponibile semnalele ‘,0’, ‘,1’ și variabilele numai în forma adevărată, nu și negată.

Solutie

Vom construi diagrama Karnaugh a funcției considerând variabilele de intrare \$A\$, \$B\$, \$C\$ și \$D\$, iar variabila \$E\$ va fi înglobată (am reprezentat și grupările corespunzătoare termenilor din expresia funcției, aşa cum este ea dată în enunțul problemei):

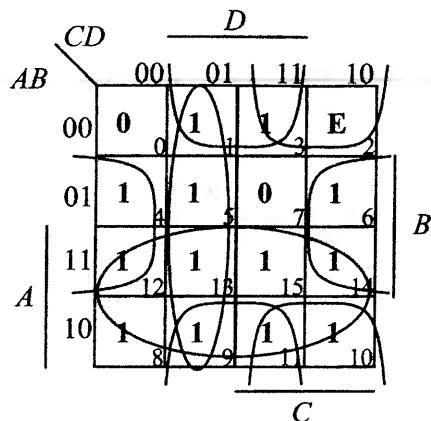
We can notice that \$x_2\$ has no influence on the generation of the function's value.

8. Implement the function:

using *only* one multiplexer. You have available the signals ‘,0’, ‘,1’ and the variables only in their true form (not negated).

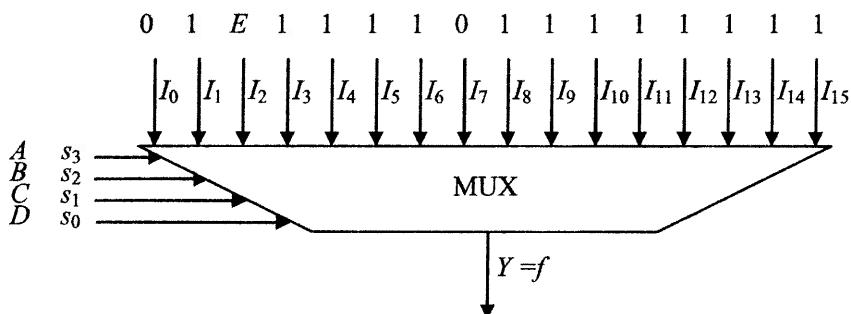
Solution

We will build the Karnaugh map of the function, considering the input variables \$A\$, \$B\$, \$C\$ and \$D\$, while the variable \$E\$ will be embedded (we have also represented the surfaces corresponding to the terms from the function's expression, as given in the problem's text):



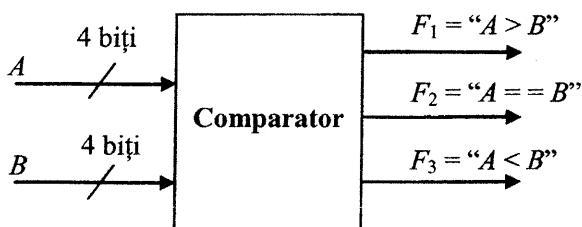
În consecință, cel mai potrivit multiplexor este de tipul 16:1. Pe selecții vom plasa A , B , C și D , iar pe intrările de date vom avea valorile '0', '1' și E .

So, the best suited multiplexer is a 16:1 one. On the selections we will place A , B , C and D , while on the data inputs we will have the values '0', '1' and E .



9. Proiectați următorul comparator de numere pe 4 biți:

9. Design the following 4-bit number comparator:



Soluție

Problema este mult mai simplă decât pare la o primă abordare. În nici un caz nu vom construi tabelele de adevăr și / sau diagramele Karnaugh ale funcțiilor F_1 , F_2 și F_3 , deoarece am avea de minimizat funcții de 8 variabile!

Vom considera numerele A și B ca fiind formate din biții $a_3a_2a_1a_0$, respectiv $b_3b_2b_1b_0$.

Vom aborda problema de o manieră logică, întrebându-ne: „când spunem că numărul A este mai mare decât numărul B ”?

Răspunsul îl putem formula în limbaj natural astfel: „ A este mai mare decât B dacă este îndeplinită una din condițiile de mai jos:

- dacă a_3 este mai mare decât b_3 , SAU
- dacă a_3 este egal cu b_3 și a_2 este mai mare decât b_2 , SAU
- dacă a_3 este egal cu b_3 , a_2 este egal cu b_2 și a_1 este mai mare decât b_1 , SAU
- dacă a_3 este egal cu b_3 , a_2 este egal cu b_2 , a_1 este egal cu b_1 și a_0 este mai mare decât b_0 ."

În caz contrar, evident, A nu este mai mare decât B (este mai mic sau egal cu B).

Nu mai rămâne decât să transpunem acest text din limbaj natural într-o formă algebraică:

Solution

The problem is much simpler than it looks at first sight. We will definitely not build the truth tables and / or the Karnaugh maps for the functions F_1 , F_2 and F_3 , because we would have to minimize 8-variable functions!

We will consider the numbers A and B as being composed of the bits $a_3a_2a_1a_0$, and $b_3b_2b_1b_0$ respectively.

We will approach the problem in a logical manner, by asking ourselves: “when do we say that the number A is greater than the number B ”?

We can formulate the answer in natural language as follows: „ A is greater than B if one of the following conditions is met:

- if a_3 is greater than b_3 , OR
- if a_3 is equal to b_3 AND a_2 is greater than b_2 , OR
- if a_3 is equal to b_3 , a_2 is equal to b_2 AND a_1 is greater than b_1 , OR
- if a_3 is equal to b_3 , a_2 is equal to b_2 , a_1 is equal to b_1 AND a_0 is greater than b_0 .”

Otherwise, obviously, A is not greater than B (it is less or equal to B).

All we have to do now is to transpose this from the natural language into an algebraic form:

$$F_1 = (a_3 \bullet \bar{b}_3) + [(a_3 \otimes b_3) \bullet (a_2 \bullet \bar{b}_2)] + [(a_3 \otimes b_3) \bullet (a_2 \otimes b_2) \bullet (a_1 \bullet \bar{b}_1)] + \\ + [(a_3 \otimes b_3) \bullet (a_2 \otimes b_2) \bullet (a_1 \otimes b_1) \bullet (a_0 \bullet \bar{b}_0)]$$

Expresia lui F_3 se obține raționând
în mod similar:

$$F_3 = (\bar{a}_3 \bullet b_3) + [(a_3 \otimes b_3) \bullet (\bar{a}_2 \bullet b_2)] + [(a_3 \otimes b_3) \bullet (a_2 \otimes b_2) \bullet (\bar{a}_1 \bullet b_1)] + \\ + [(a_3 \otimes b_3) \bullet (a_2 \otimes b_2) \bullet (a_1 \otimes b_1) \bullet (\bar{a}_0 \bullet b_0)]$$

În sfârșit, expresia lui F_2 se obține
cel mai simplu:

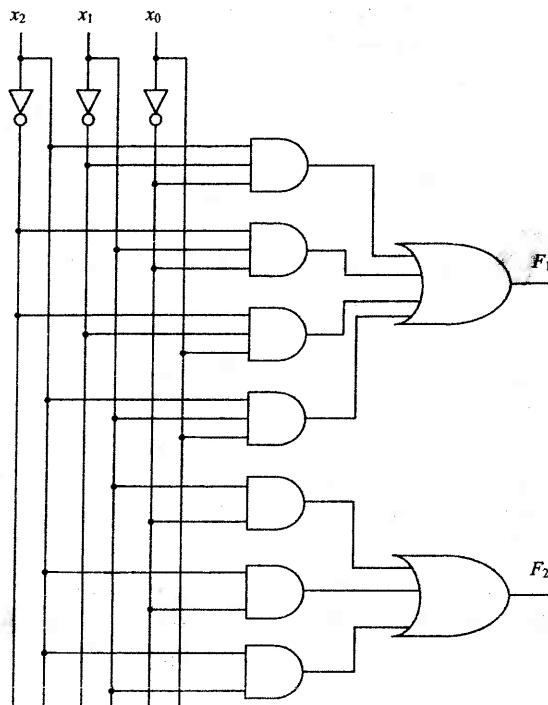
$$F_2 = (a_3 \otimes b_3) \bullet (a_2 \otimes b_2) \bullet (a_1 \otimes b_1) \bullet (a_0 \otimes b_0)$$

F_3 's expression is obtained by
reasoning in a similar manner:

Finally, F_2 's expression is the
simplest one to obtain:

10. Redesenați circuitul din schema
următoare cu multiplexoare 4:1.

9. Redraw the circuit from the next
scheme with 4:1 multiplexers:



Soluție

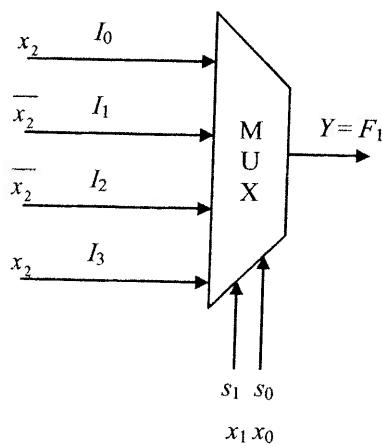
Mai întâi determinăm expresiile celor două funcții, F_1 și F_2 :

$$F_1 = x_2 \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_2} \cdot x_1 \cdot \overline{x_0} + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_2 \cdot x_1 \cdot x_0$$

$$F_2 = x_1 \cdot \overline{x_0} + x_2 \cdot \overline{x_0} + x_2 \cdot x_1$$

Pe baza ecuațiilor, construim diagramele Karnaugh ale celor două funcții:

		x_0			
		00	01	11	10
x ₂		0	0 1	0 3	1 2
x ₂	1	1 4	0 5	1 7	0 6
		x_1			



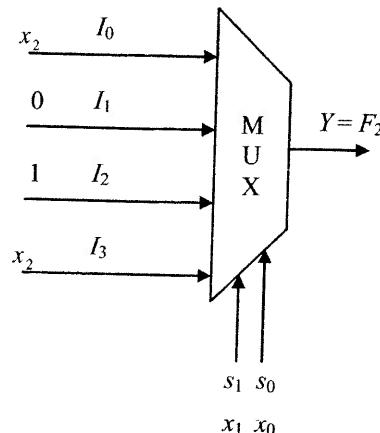
Aceste structuri se obțin folosind același tip de raționament ca la problema 5.

Solution

First we determine the expressions of the two functions, F_1 and F_2 :

Based on the equations, we build the Karnaugh maps of the two functions:

		x_0				
		00	01	11	10	
x ₂		0	0 0	0 1	0 3	1 2
x ₂	1	1 4	0 5	1 7	0 6	
		x_1				



These structures are obtained using the same type of reasoning as for problem 5.

11. Să se implementeze funcția Booleană cu variabile înglobate definită prin următoarea diagramă Karnaugh folosind un decodificator și porți logice:

		x_0				
		00	01	11	10	
x_3x_2		00	a	0	0	X
	01	00	0	0	1	X
	11	X	X	\bar{b}	X	X
	10	1	X	X	X	X

Soluție

Vom scrie mai întâi funcția în forma canonicoă disjunctivă:

$$f = a \cdot \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{x_3} \cdot x_2 \cdot x_1 \cdot x_0 + x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + \overline{b} \cdot x_3 \cdot x_2 \cdot x_1 \cdot x_0$$

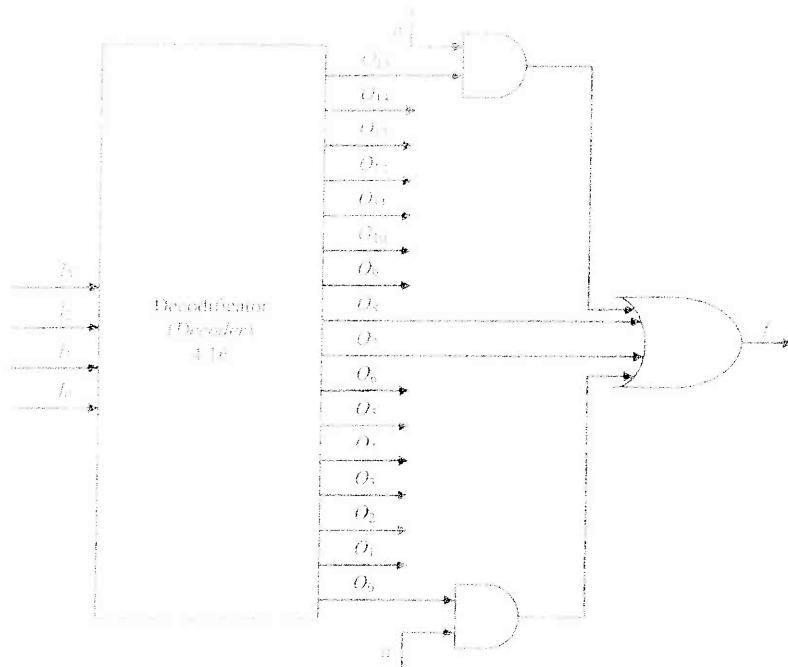
În felul acesta, ținând cont și de explicațiile oferite în partea de prezentare teoretică, este foarte ușor să deducem schema de implementare a funcției:

11. Implement the Boolean function with embedded variables that is defined by the next Karnaugh map using a decoder and logic gates:

Solution

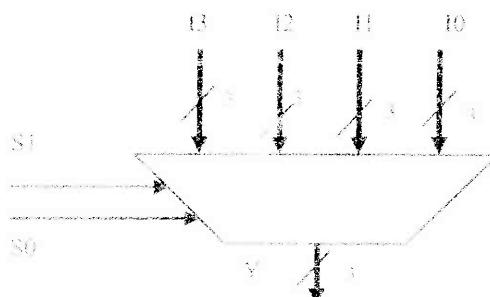
First we will write the function in the disjunctive canonical form:

This way, taking also into account the explanations offered in the theoretical presentation section, it is very easy to deduce the function's implementation scheme:



12. Proiectați un MULTIPLEXOR 4:1 cu lățimea căii de date de 3 biți.

12. Design a 4:1 MULTIPLEXER with the data path of 3 bits wide.

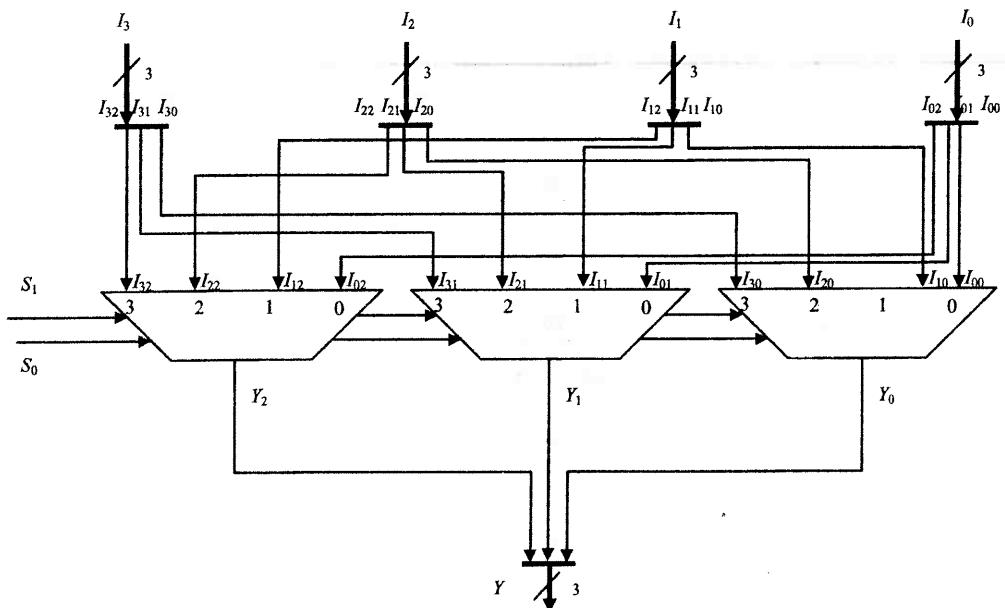


Soluție

Schema se construiește relativ simplu prin cascadarea a 3 multiplexoare 4:1.

Solution

The scheme is relatively easy to build by cascading 3 4:1 multiplexers.



13. Implementați un sumator complet de numere binare pe 4 biți.

Solutie

Sumatorul complet de numere binare pe 4 biți se obține prin cascadarea a 4 sumatoare complete elementare de 1 bit. Schema bloc a unui astfel de sumator, precum și ecuațiile ieșirilor sale au fost prezentate deja în partea teoretică a acestui capitol.

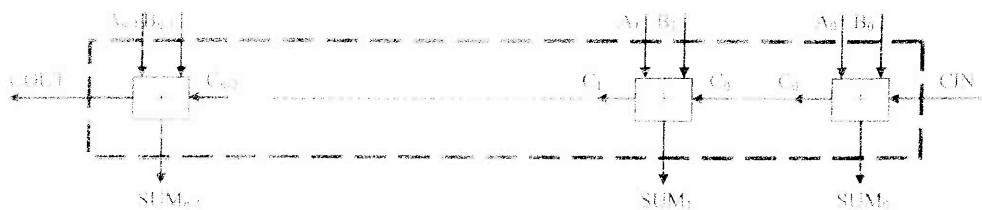
Schema de cascadare pentru n biți este cea următoare, unde fiecare modul reprezintă un sumator complet elementar de 1 bit:

13. Implement a 4-bit full adder.

Solution

The 4-bit full adder is obtained by cascading 4 1-bit elementary full adders. The block diagram of such a full adder, as well as the equations of its outputs have already been presented in the theoretical part of this chapter.

The cascading scheme for n bits is the next one, where each module represents an elementary 1-bit full adder:



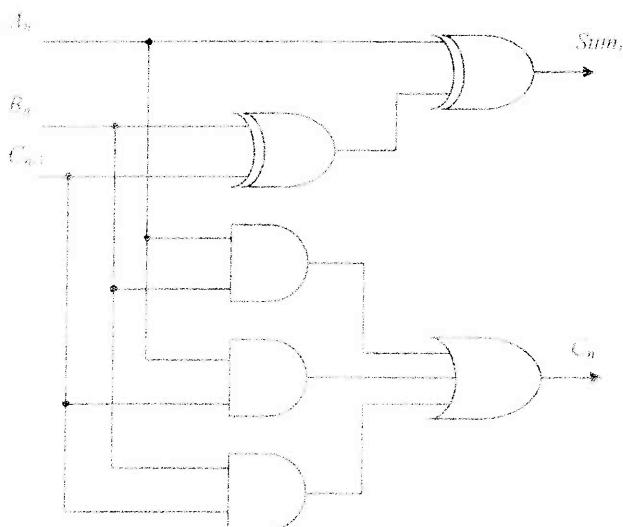
Fiecare sumator complet elementar de 1 bit are deci următoarea structură (conform ecuațiilor

So each elementary 1-bit full adder has the following structure (according to the equations

$$\begin{aligned} \delta_n &= A_n \oplus B_n \oplus C_{n-1} \\ C_n &= A_n \cdot B_n + C_{n-1} \cdot B_n + C_{n-1} \cdot A_n \end{aligned}$$

prezentate în introducerea teoretică a acestui capitol):

presented in the theoretical introduction of this chapter):



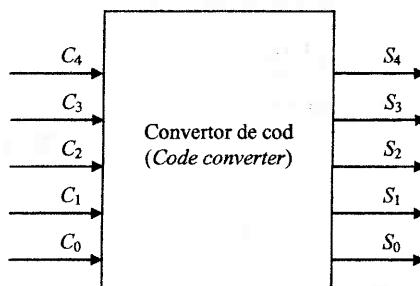
Astfel, dat fiind că este descrisă atât structura de ansamblu cât și structura internă a fiecărui modul din componentă acesteia, problema este complet rezolvată.

So, since we described both the general structure and the internal structure of each module inside it, the problem is solved completely.

14. Implementați un dispozitiv pentru conversia numerelor de 5 biți din reprezentarea *Complementul față de 2* în reprezentarea *Întregi cu Semn*.

Soluție

Pentru început vom desena cutia neagră a sistemului:



Notăm numărul de la intrare cu C , alcătuit din biții $C_4C_3C_2C_1C_0$, iar numărul de la ieșire cu S , alcătuit din biții $S_4S_3S_2S_1S_0$.

Trebuie să începem prin a studia ce numere pot fi reprezentate în cele două sisteme de numerație, pe 5 biți:

- în *Complementul față de 2*, pe 5 biți putem reprezenta numerele de la -16 la +15.

- în *Întregi cu Semn*, pe 5 biți putem reprezenta numerele de la -15 la +15.

După cum se vede, există o diferență de un număr, care este dată de faptul că în *Întregi cu Semn* există 2 reprezentări ale lui 0: „+ 0”

14. Implement a device for converting 5-bit numbers from the *Two's Complement Integer* representation to the *Signed Magnitude Integer* representation.

Solution

For beginning, we will draw the system's black box:

We denote the input number by C , composed of the bits $C_4C_3C_2C_1C_0$, and the output number by S , composed of the bits $S_4S_3S_2S_1S_0$.

We must start by studying which numbers can be represented in the two numbering systems, on 5 bits:

- in *Two's Complement*, on 5 bits we can represent numbers ranging from -16 to +15.

- in the *Signed Magnitude* system, on 5 bits we can represent numbers ranging from -15 to +15.

As it can be seen, there is a difference of one number, which is given by the fact that in the *Signed Magnitude* system there are 2

și „-0”.

Așadar, când pe intrările circuitului avem „10000” (adică „-16”), nu există un număr corespondent în *Înregi cu Semn*, aşa că vom „converti” acest număr generând la ieșire același cuvânt de cod binar „10000” (cu semnificația „-0”).

Odată stabilit acest cadru general, să trecem la analiza detaliată a problemei. După cum știm, există două cazuri:

a) $C_4 = 0$. În acest caz, $S = C$.

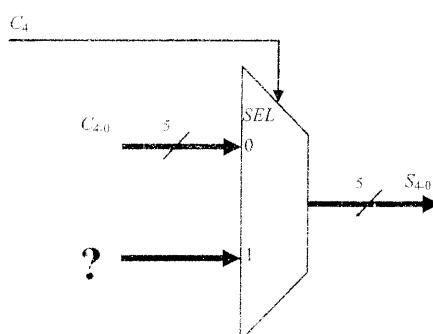
b) $C_4 = 1$. Distingem două subcazuri:

b1) $C_4C_3C_2C_1C_0 = \text{„}10000\text{”}$. În acest caz, S este egal cu C : $S_4S_3S_2S_1S_0 = \text{„}10000\text{”}$.

b2) În acest caz, S se obține conform regulii de conversie între cele 2 reprezentări: se inversează toți biții lui C și se adună „1”.

Pe baza acestei analize, vom construi gradat schema circuitului.

Începem cu etapa a):



representations of 0: “+0” and “-0”. So, when on the circuit’s inputs we have “10000” (which means “-16”), there is no corresponding number in the *Signed Magnitude* system, so we will “convert” this number by generating at the output the same binary word “10000” (having the meaning: “-0”).

Once settled this general framework, let’s move on to the detailed analysis of this problem. As we know, there are 2 cases:

a) $C_4 = 0$. In this case, $S = C$.

b) $C_4 = 1$. We distinguish two subcases:

b1) $C_4C_3C_2C_1C_0 = \text{“}10000\text{”}$. In this case, S equals C : $S_4S_3S_2S_1S_0 = \text{“}10000\text{”}$.

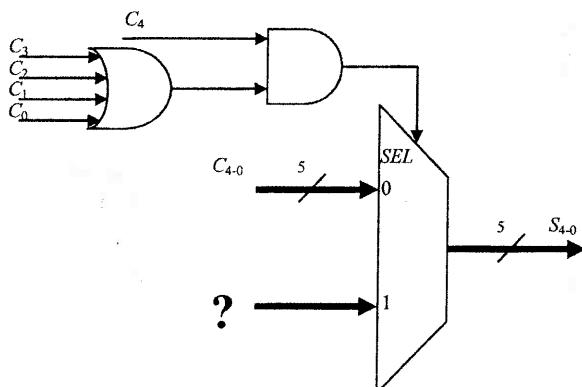
b2) In this case, S is obtained according to the conversion rule between the two numbering systems: all C ’s bits are inverted and then we add ‘1’.

Based on this analysis, we will gradually build the circuit’s scheme. We start by step a):

În locul semnului de întrebare trebuie construită arhitectura corespunzătoare punctului b). Simultan, trebuie actualizată (mai precis rafinată) și partea deja construită. În limbaj natural:

- dacă ($C_4 = 0$) SAU ($C_4 = 1$ ȘI $C_{3-0} = „0000”$), atunci $S = C$. În consecință, selecția multiplexorului din figură va trebui să aibă valoarea „0”. Transpunând această frază într-o ecuație Booleană și apoi într-o schemă de circuit, se obține, ținând cont că :

$$\begin{aligned} SEL &= \overline{\overline{C}_4 + C_4} \cdot \overline{\overline{C}_3} \cdot \overline{\overline{C}_2} \cdot \overline{\overline{C}_1} \cdot \overline{\overline{C}_0} = C_4 \cdot \overline{C_4} \cdot \overline{C_3} \cdot \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0} \\ &= C_4 \cdot (\overline{C_4} + C_3 + C_2 + C_1 + C_0) = C_4 \cdot (C_3 + C_2 + C_1 + C_0) \end{aligned}$$



- dacă $C_4 = 1$ ȘI $C_{3-0} \neq „0000”$, atunci S se calculează cum am prezentat mai sus. În schemă se remarcă un bloc de inversoare (NU) și un sumator complet pe 4 biți, realizat conform problemei 13, la

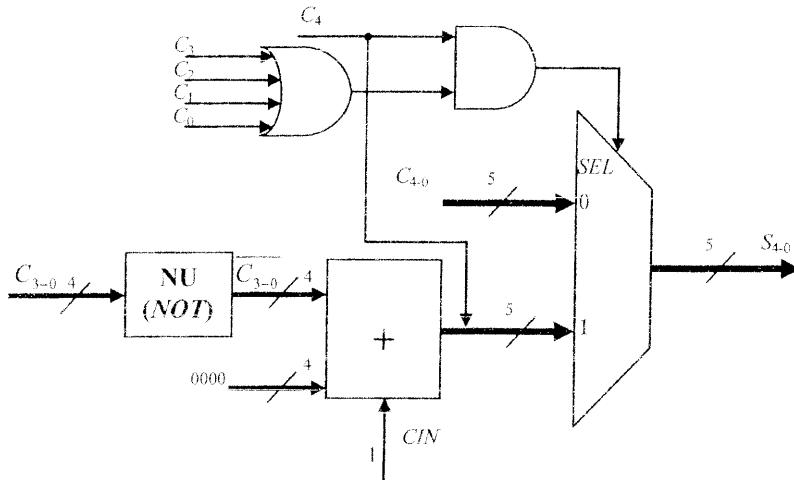
Instead of the question mark we have to build the architecture for point b). In the same time, we have to update (or, more precisely, to refine) the part that is already built. In natural language:

- if ($C_4 = 0$) OR ($C_4 = 1$ AND $C_{3-0} = „0000”$), then $S = C$. As a matter of consequence, the multiplexer's selection from this figure will have to carry the value '0'. By transposing this phrase into a Boolean equation and then in circuitry, we obtain, taking into account the fact that:

- if $C_4 = 1$ AND $C_{3-0} \neq „0000”$, then S is computed as we have shown above. In the scheme we notice a block of inverters (NOT) and a 4-bit full adder, made according to the problem 13, where one of the

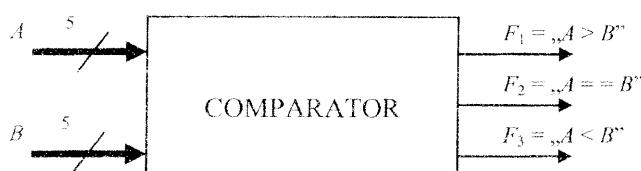
care unul dintre operanzi este „0000” iar *Carry In* este „1”.

operands is “0000” and *Carry In* is ‘1’.



14. Implementați un COMPARATOR de numere pe 5 biți în reprezentarea *Complementul față de 2*:

14. Implement a COMPARATOR for 5-bit numbers in the Two's Complement Integer representation:



Soluție

Formulăm rezolvarea în limbaj natural:

- $A > B$ dacă $(A_4 = 1 \text{ și } B_4 = 0)$ SAU $[(A_4 = B_4) \text{ și } (A_{3..0} > B_{3..0})]$.
- $A < B$ dacă $(A_4 = 0 \text{ și } B_4 = 1)$ SAU $[(A_4 = B_4) \text{ și } (A_{3..0} < B_{3..0})]$.
- $A == B$ dacă $(A_4 = B_4) \text{ și }$

Solution

We formulate the solution in natural language:

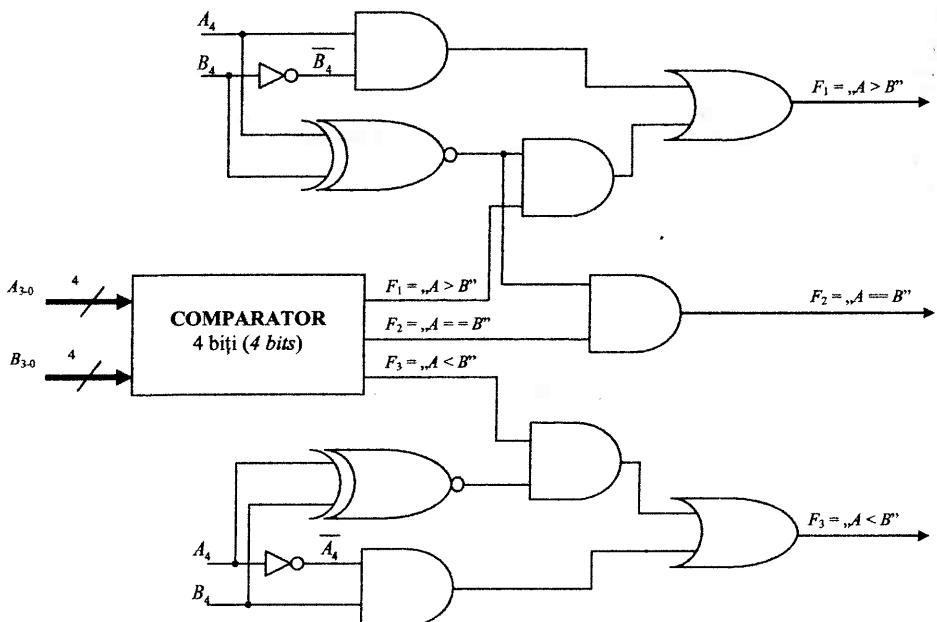
- $A > B$ if $(A_4 = 1 \text{ și } B_4 = 0)$ SAU $[(A_4 = B_4) \text{ AND } (A_{3..0} > B_{3..0})]$.
- $A < B$ if $(A_4 = 0 \text{ și } B_4 = 1)$ SAU $[(A_4 = B_4) \text{ AND } (A_{3..0} < B_{3..0})]$.
- $A == B$ if $(A_4 = B_4) \text{ AND }$

$(A_{3..0} == B_{3..0})$.

Reiese clar că se impune folosirea unui comparator de 4 biți ca modul component al acestui circuit. Realizarea comparatorului de 4 biți a fost deja prezentată la problema rezolvată 8.

$(A_{3..0} == B_{3..0})$.

It is obvious that we must use a 4-bit comparator as a component module of this circuit. The realization of the 4-bit comparator has already been presented at the solved problem no. 8.



14. Implementați cu buffer-e tri-state și cu inversoare:
- funcția logică SI
 - funcția logică SAU
 - funcția logică SAU-EXCLUSIV

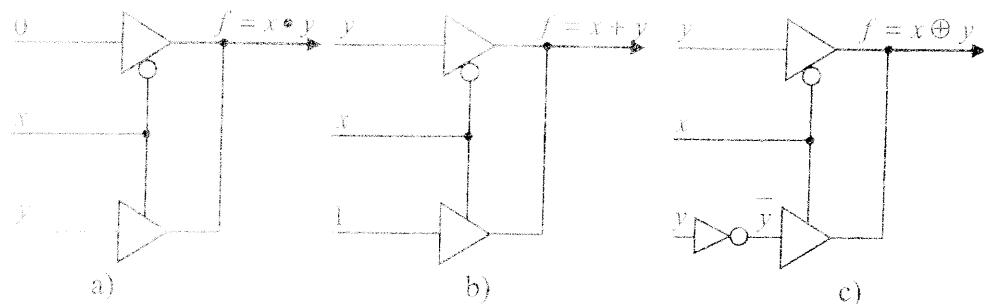
Solutie

În figurile următoare sunt prezentate schemele de implementare ale celor trei funcții logice cerute:

14. Implement using tri-state buffers and inverters:
- the AND logic function
 - the OR logic function
 - the XOR logic function

Solution

In the next figures are presented the implementation schemes of the three logic functions requested:

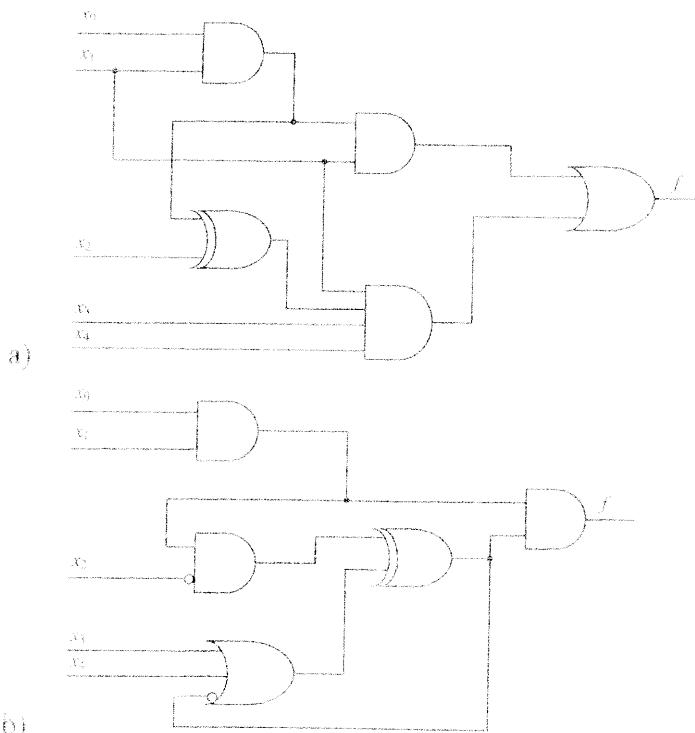


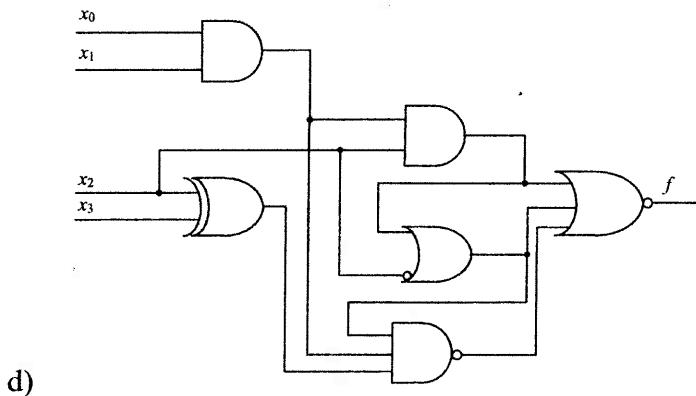
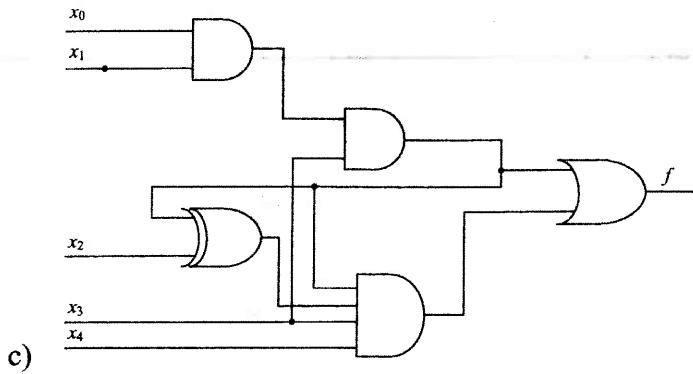
3.5. Probleme propuse

1. Determinați dacă circuitele de mai jos sunt sau nu combinaționale:

3.5. Proposed problems

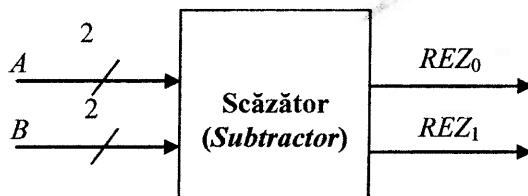
1. Determine if the circuits below are combinational or not :





2. Proiectați următorul circuit scăzător de numere pe 2 biți:

2. Design the following 2-bit numbers subtraction device



Se presupune că niciodată numărul A nu va fi mai mic decât B (acele cazuri se vor considera condiții indiferente).

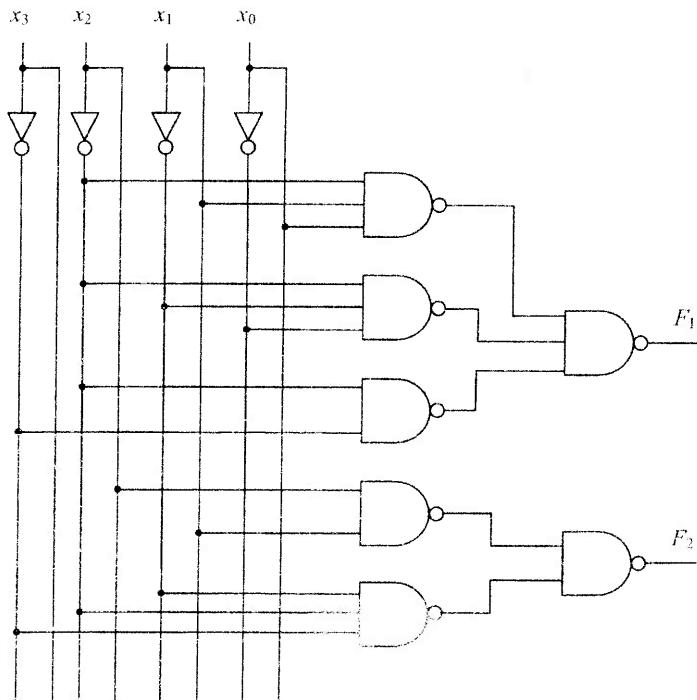
It is assumed that the number A will never be less than B (those cases will be considered don't care conditions).

3. Să se implementeze cu 2 etaje de MULTIPLEXOARE o funcție Booleană care ia valoarea logică 1 pentru numerele prime cuprinse în intervalul 0-63.

4. Să se exprime funcțiile implementate în figura de mai jos prin forma canonica disjunctivă și diagramă Karnaugh. Implementarea din figură este minimă? Justificați.

3. Implement with two stages of MULTIPLEXERS a Boolean function that takes the logic value 1 for the prime numbers ranging in the 0-63 interval.

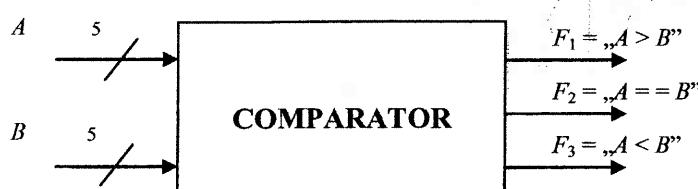
4. Express the functions implemented in the figure below by the disjunctive canonical form and Karnaugh map. Is the implementation from the figure minimal? Justify your answer.



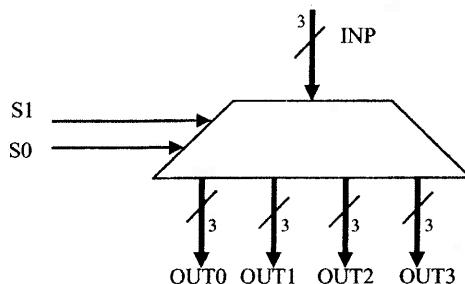
5. Implementați un *convertor de cod* din cod binar de 4 biți în cod Gray de 4 biți.

5. Implement a code converter from 4-bit binary code to 4-bit Gray code.

6. Implementați un SUMATOR COMPLET de numere întregi pe 5 biți în reprezentarea în *Complementul față de 2*.
7. Implementați un dispozitiv de conversie a numerelor de 5 biți din reprezentarea *Întregi cu Semn* în reprezentarea în *Complementul față de 2*.
8. Proiectați un comparator de numere reprezentate în *Întregi cu Semn*, pe 5 biți:
6. Implement a FULL ADDER for 5-bit numbers in the *Two's Complement Integer* representation.
7. Implement a device for converting 5-bit numbers from the *Signed Magnitude Integer* representation to the *Two's Complement Integer* representation.
8. Implement a COMPARATOR for 5-bit numbers in the *Signed Magnitude Integer* representation:



9. Proiectați un DEMULTIPLEXOR 1:4 cu lățimea căii de date de 3 biți.
9. Design a 1:4 DEMULTIPLEXER with the data path of 3 bits wide:



10. Proiectați un Sumator complet de numere *Fracționare cu semn* în *Complementul față de 2*, de 8 biți,
10. Build a Full Adder for two 8-bit numbers, N_1 and N_2 , represented in the *Two's Complement Signed*

N_1 și N_2 , unde N_1 este un număr de tipul (5,3), iar N_2 este un număr de tipul (4,4).

11. Implementați un SUMATOR COMPLET de numere întregi pe 5 biți, la care primul operand este reprezentat în *Complementul față de 2*, iar al doilea operand este reprezentat în *Întregi cu Semn*, rezultatul fiind reprezentat în *Complementul față de 2*.

12. a) Cum se aplică algoritmul Quine-McCluskey pentru minimizarea funcțiilor Booleene în forma canonica conjunctivă?
 b) Exemplificați pe următoarea funcție Booleană:

$$f = \prod_{\Phi} (0,2,3,4,7,8,9) + \prod_{\Phi} (6,10,11,12,13,14,15)$$

Fractional numbering system, where N_1 is a (5,3) number and N_2 is a (4,4) number.

11. Implement a FULL ADDER for 5-bit numbers, where the first operand is represented in *Two's Complement Integer*, while the second operand is represented in the *Signed Magnitude Integer* numbering system. The result is represented in *Two's Complement Integer*.

12. a) How can the Quine-McCluskey algorithm be applied for minimizing Boolean functions in the conjunctive canonical form?
 b) Apply this algorithm on the following Boolean function:

Capitolul 4. **Circuite logice** **secvențiale**

4.1. Definiții

Circuitele logice secvențiale, CLS, sunt automate de ordinul 1. Ele se obțin din automatele de ordinul 0 (CLC) prin introducerea unor reacții (legături inverse). Sunt alcătuite din circuite logice combinaționale și elemente de memorare binară.

Semnalele de ieșire ale CLS depind atât de combinația semnalelor aplicate pe intrări cât și de *starea* circuitului. Dacă la CLC-uri ieșirile depindeau strict numai de intrări, aici apare un element suplimentar: *starea internă a circuitului*, care influențează la rândul ei ieșirile ce vor fi generate.

Un CLS este caracterizat deci atât de secvența semnalelor de ieșire cât și de secvența stărilor sale interne, stocate în elementele de memorie, pentru fiecare secvență de intrări aplicate circuitului.

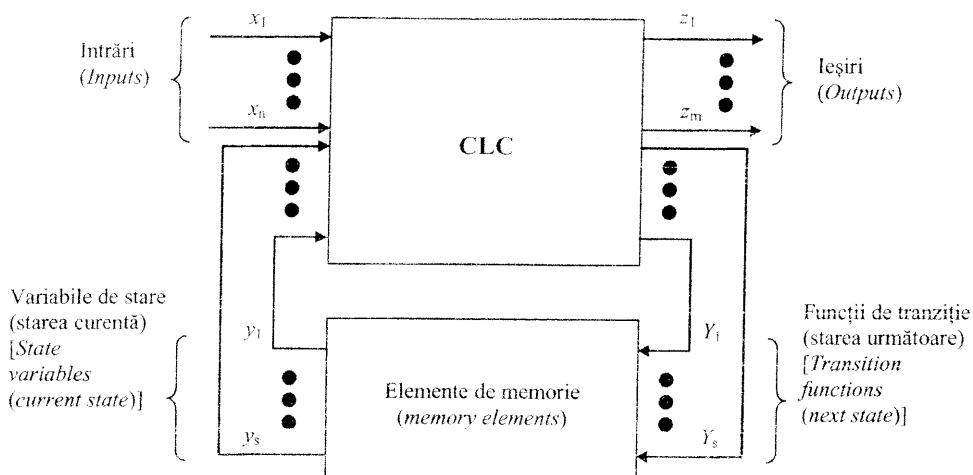
Chapter 4. **Sequential logic** **circuits**

4.1. Definitions

Sequential logic circuits, SLCs, are automata of the first degree. They are obtained from automata of degree 0 (CLCs) by introducing reactions (feedback loops). They are composed of combinational logic circuits and binary memory elements.

The SLC's output signals depend both on the combination of the signals applied on the inputs and on the circuit's *state*. While at the CLCs the outputs were depending strictly on the inputs, here a supplemental element appears: the circuit's *internal state*, which also influences the outputs that will be generated.

An SLC is thus characterized both by the input signals sequence and by its internal states sequence, the later being stored in the memory elements, for each sequence of inputs that are applied to the circuit.



După modul de funcționare (modul de transmitere a semnalelor) există două categorii principale de CLS-uri:

1. *asincrone* – comportarea este determinată de aplicarea pe intrări a semnalelor la momente de timp arbitrară; starea circuitului depinde de ordinea în care se schimbă semnalele de intrare;
2. *sincrone* – comportarea este determinată de aplicarea pe intrări a semnalelor în momente discrete, bine determinate în timp; sincronizarea se realizează cu ajutorul unor impulsuri date de un generator de tact (ceas sau *clock*).

Principala componentă cu ajutorul căreia se construiesc CLS-urile este *circuitul basculant bistabil*.

With respect to the functioning mode (the way signals are transmitted) there are two main categories of SLCs:

1. *asynchronous* – their behavior is determined by applying the signals on the inputs at arbitrary moments of time; the circuit's state depends on the order in which the input signals change;
2. *synchronous* – their behavior is determined by applying the signals on the inputs at discrete, well determined moments of time; synchronization is done by means of impulses given by a *clock generator*.

The main component used for building SLCs is the *Latch* or the *Flip-Flop*.

4.2. Circuite basculante bistabile

Circuitele basculante bistabile (CBB-uri sau bistabile) sunt circuite logice secvențiale care au două stări stabile distincte. Trecerea dintr-o stare în alta se face la aplicarea unei comenzi din exterior.

Caracteristica principală a CBB este că sunt *sisteme cu memorie* (elemente de memorie binară). Bistabilul este cel mai mic element de memorie, care stochează un singur bit.

Un bistabil poate păstra un timp nedefinit informația binară și în același timp starea sa poate fi citită în orice moment. Se asociază uneia dintre cele 2 stări ale bistabilului funcția de memorare a cifrei binare „1” și celei de a doua stări funcția de memorare a cifrei binare „0”. Bistabilul are 2 ieșiri, una care punе în evidență cifra binară memorată, numită *ieșire adevărată* (notată de regulă cu Q) și a doua, care punе în evidență valoarea negată a cifrei binare memorate, denumită *ieșire negată* (notată de regulă cu \bar{Q}).

Principalele tipuri de CBB-uri sunt următoarele:

4.2. Latches and Flip-Flops

The *Latches* or *Flip-Flops* are sequential logic circuits having two distinct stable states. The transition from a state into another one is done when an external command is applied.

The main characteristic of these circuits is the fact that they are *memory-based systems* (they are binary memory elements). The Latch (or Flip-Flop) is the smallest memory element, that stores one single bit.

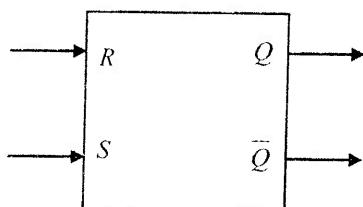
A Latch can store an undefined lap of time the binary information and in the same time its state can be read at any moment. We associate to one of its two states the function of memorizing the binary digit ‘1’, and to its other state we associate the function of memorizing the binary digit ‘0’. The Latch has two outputs, one that yields the binary digit stored inside the circuit, called *true output* (usually denoted by Q) and the other one yields the negated value of the binary digit stored, called *negated output* (usually denoted by \bar{Q}).

The main types of Latches and Flip-Flops are the following ones:

Bistabilul RS asincron (latch)

Bistabilul RS asincron este cel mai simplu element de memorare care poate fi realizat cu circuite logice elementare. El are 2 intrări de comandă (de date): S (Set) și R (Reset) și două ieșiri Q și \bar{Q} (complementare).

Simbolul și tabelul de adevăr ale bistabilului RS asincron sunt următoarele (am notat cu Q_n și Q_{n+1} starea bistabilului la momentul „ n ”, respectiv la momentul următor):



Se observă că ultima combinație din tabelul de adevăr nu este posibilă (nu putem seta și reseta simultan bistabilul).

Semnalul ‚1’ logic aplicat pe intrarea S (setare) aduce bistabilul în starea $Q = 1$ și $\bar{Q} = 0$, iar semnalul ‚1’ logic aplicat pe intrarea R (resetare) aduce bistabilul în starea $Q = 0$ și $\bar{Q} = 1$.

Circuitul este de tip asincron, deoarece comutarea bistabilului se

Asynchronous RS Latch

The asynchronous RS Latch is the simplest memory element that can be built with elementary logic circuits. It has two command (data) inputs: S (Set) and R (Reset) and two complementary outputs Q and \bar{Q} .

The asynchronous RS Latch's symbol and truth table are the ones shown below (we denoted by Q_n and Q_{n+1} its internal state at the moment “ n ” and at the next moment of the discrete time, respectively):

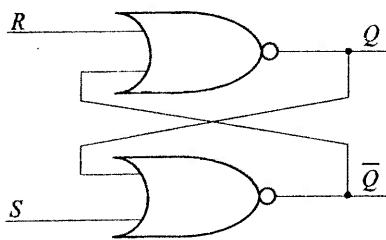
S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	*

One can notice that the last combination in the truth table is not possible (we can not set and simultaneously reset the Latch).

The signal ‘1’ applied on the S input (set) brings the Latch in state $Q = 1$ and $\bar{Q} = 0$, while the signal ‘1’ applied on the R input (reset) brings the Latch in state $Q = 0$ and $\bar{Q} = 1$.

The circuit is of asynchronous type, because the Latch switches

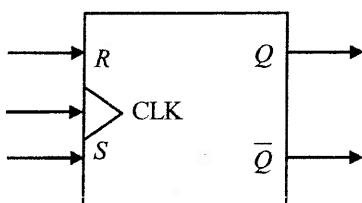
face imediat după aplicarea semnalului pe una din intrări. Pentru bistabilul RS asincron condiția de funcționare normală este $S \bullet R = 0$. Schema internă a bistabilului este următoarea:



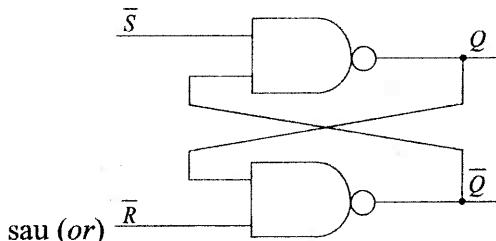
Bistabilul RS sincron (latch)

Bistabilul RS sincron se obține din bistabilul RS asincron prin adăugarea unor porți logice suplimentare cu scopul de a răspunde la semnalele de intrare R și S numai sub acțiunea unui semnal de comandă extern numit impuls de tact (ceas).

Ieșirile bistabilului RS sincron se modifică doar când semnalul de tact (ceas) CLK este activ. Simbolul și tabelul de adevăr ale bistabilului RS sincron sunt următoarele:



immediately after applying the signal on one of the inputs. For the asynchronous RS Latch, the normal functioning condition is $S \bullet R = 0$. The internal scheme of this Latch is the following one:



Synchronous RS Latch

The synchronous RS Latch is obtained from the asynchronous RS Latch by adding some extra logic gates, with the aim of responding to the R and S input signals only when an external command signal (called 'clock') is applied.

The synchronous RS Latch's outputs are modified only when the clock (CLK) signal is active. The synchronous RS Latch's symbol and truth table are the ones shown below:

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	*

Și la acest bistabil situația intrărilor în care $S = R = 1$ introduce o nedeterminare, de aceea ea trebuie evitată.

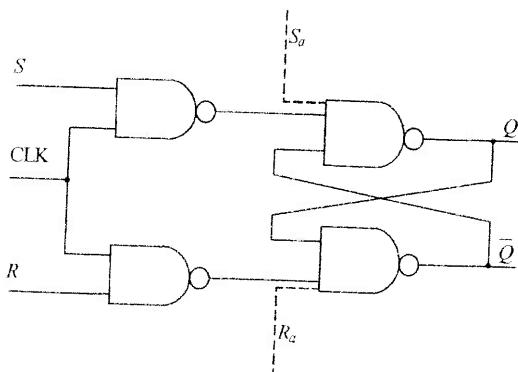
Cât timp CLK este 0, intrările de date nu influențează bistabilul. Când CLK = 1 bistabilul urmărește modificările intrărilor de date. Când CLK redevine '0' bistabilul se zăvorăște (de aceea se numește latch), păstrează informația avută anterior.

Schema internă a bistabilului este următoarea:

At this Latch too, in case on the inputs we have $S = R = 1$, this introduces a nondeterministic situation which must be avoided.

As long as CLK is 0, the data inputs do not influence the Latch. When CLK = 1, the Latch follows the modifications of the data inputs. When CLK falls back to '0', the Latch is closed (this is the source of its name), keeps the information previously present.

The Latch's internal scheme is the following one:



Noțiunea de *funcție de excitație* este caracteristică pentru fiecare bistabil. Ea pune în evidență ce valori trebuie să fie puse pe intrările bistabilului pentru a se realiza o tranziție specifică.

Tabelul de excitație pentru bistabilul RS sincron este următorul:

The *excitation function* concept is characteristic for each Latch. It highlights which values should be present on the Latch's inputs in order to make a specific transition.

The excitation table for the synchronous RS Latch is the following one:

Q_n	Q_{n+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

Observație. În afara intrărilor sincrone, la bistabilul RS sincron se introduc și intrări asincrone, R_a și S_a , la nivelul bistabilului RS asincron. Aceste intrări sunt utilizate cu scopul forțării la 0, prin R_a , sau la 1, prin S_a , a ieșirii bistabilului.

Apariția unor comenzi pe aceste intrări se realizează independent de prezența sau absența tactului. Din acest motiv intrările asincrone ale unui bistabil sunt prioritare în raport cu intrările sincrone.

Bistabilul JK sincron

Un bistabil de tip JK sincron se obține din bistabilul RS sincron prin efectuarea legăturilor care permit eliminarea condiției $R \bullet S = 0$. Bistabilul JK sincron elimină situația de nedeterminare pe ieșiri, prezentă la bistabilul RS, la combinația $S = R = 1$ pe intrări. Se folosesc reacții (legături inverse) suplimentare.

\bar{S} și \bar{R} sunt intrările asincrone, care

Remark. Besides the synchronous inputs, in the synchronous RS Latch we also introduce asynchronous inputs, R_a and S_a , at the level of the asynchronous RS Latch. These inputs are used to force to '0', by means of R_a , or to '1', by means of S_a , of the Latch's output. The apparition of some commands on these inputs is done independently of the presence or the absence of the clock. For this reason, the asynchronous inputs of a Latch are prioritary compared to the synchronous inputs.

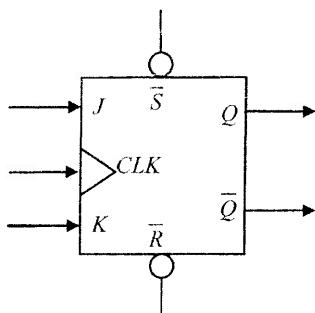
Synchronous JK Latch

A synchronous JK Latch is obtained from the synchronous RS Latch by making the connections that care allow to eliminate the condition $R \bullet S = 0$. The synchronous JK Latch eliminates the nondeterministic situation on the outputs, which was present at the RS Latch, for the combination $S = R = 1$ on the inputs. Supplemental feedback loops are used for this purpose.

\bar{S} and \bar{R} are the asynchronous

acționează la ultimul nivel de porți logice, nu depind de semnalul de tact și sunt prioritare față de intrările sincrone J și K (adică în momentul în care una dintre ele se activează, bistabilul va funcționa în regim asincron).

Simbolul și tabelul de adevăr ale bistabilului JK sincron sunt următoarele:



Există două tipuri de bistabile de tip JK sincron, unul care comută pe front (atunci când se schimbă tactul, din 0 în 1 – *frontul ascendent, ridicător sau pozitiv* – sau din 1 în 0 – *frontul descendente, coborâtor sau negativ*) și altele care comută pe nivel (atunci când tactul este pe nivel stabil 1). Primele sunt de tip Stăpân-Sclav sau *Master-Slave (Flip-Flop)* – după cum vom vedea mai jos –, celelalte de tip *Latch* (zăvor).

Schema internă a bistabilului JK sincron de tip zăvor este următoarea:

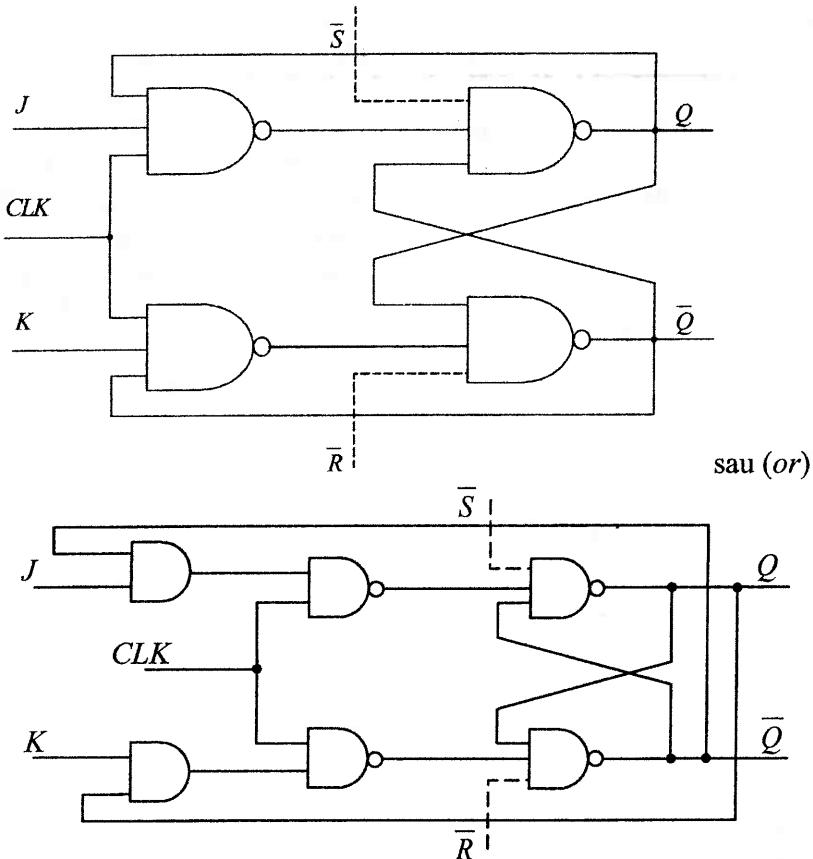
inputs, acting at the last level of logic gates, and they do not depend on the clock signal and are priority compared to the synchronous inputs J and K (i.e. at the moment when one of them is activated, the Latch will function in an asynchronous working mode).

The synchronous JK Latch's symbol and truth table are the ones shown below:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

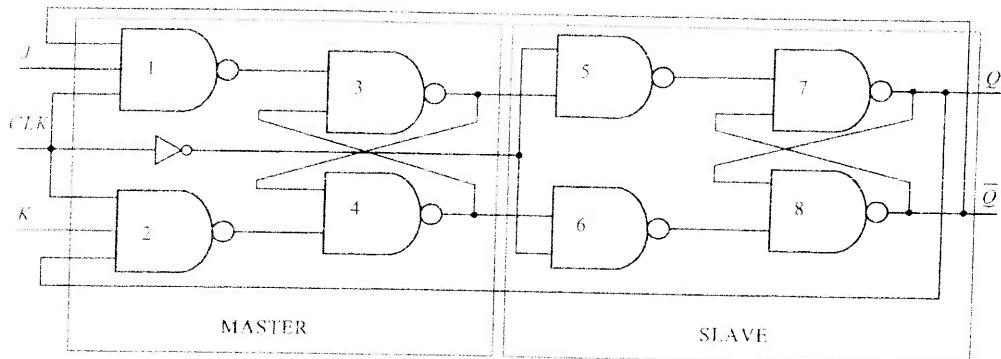
There are two types of synchronous JK memory cell: an edge-triggered one (when the clock signal is changing from 0 to 1 – *raising, ascending or positive edge* – or from 1 to 0 – *falling, descending or negative edge*) and a level-triggered one (when the clock signal is stable on level 1). The first type is called *Master-Slave (Flip-Flop)* – as we will see below – the other one is called *Latch*.

The internal scheme of the synchronous JK Latch is the following one:



Schema internă a bistabilului JK sincron de tip Stăpân-Sclav sau *Master-Slave* este următoarea:

The internal scheme of the synchronous JK Flip-Flop is the following one:



Tabelul de excitare pentru bistabilul JK sincron este următorul:

The excitation table for the synchronous JK Latch is the following one:

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Bistabilul D sincron (delay)

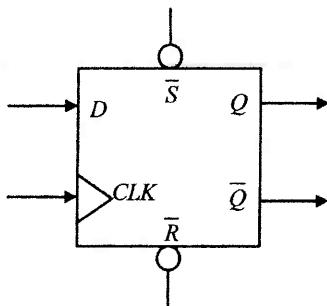
Bistabilul D sincron are o singură intrare, D și cele 2 ieșiri complementare, Q și \bar{Q} . Starea următoare a bistabilului D este determinată de modificarea intrării D . El întârzie cu un tact informația pe care o primește pe intrare (circuit elementar de întârziere). Sunt cele mai răspândite bistabile în registrele de date.

Simbolul și tabelul de adevăr ale bistabilului D sincron sunt următoarele:

Synchronous D (delay) Latch

The synchronous D Latch has a single input, D and the two complementary outputs, Q and \bar{Q} . The next state of the D Latch is determined by the modification of the D input. It delays by one clock cycle the information that it receives on the input (elementary delay circuit). These are the most widely spread Latches in data registers.

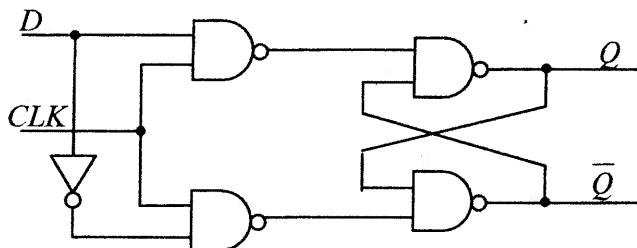
The synchronous D Latch's symbol and truth table are the ones shown below:



D	Q_{n+1}
0	0
1	1

Bistabilul D sincron se obține din bistabilul RS sincron astfel:

The synchronous D Latch is obtained from the synchronous RS Latch as follows:



Starea următoare a bistabilului de tip D sincron este dependentă doar de semnalul aplicat pe intrare, ea fiind independentă de starea actuală a bistabilului.

Tabelul de excitare pentru bistabilul D sincron este următorul:

The next state of the synchronous D Latch depends only on the signal applied on the input, because it is independent on the Latch's current state.

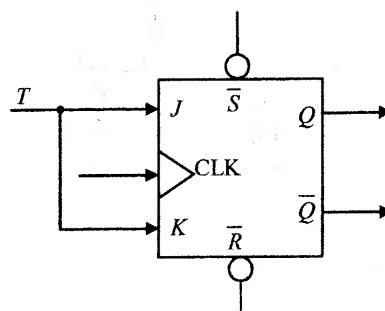
The excitation table of the synchronous D Latch is the following one:

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1



Bistabilul T sincron se obține din bistabilul JK sincron astfel:

The synchronous T Latch is obtained from the synchronous JK Latch as follows:



Există două tipuri de bistabile de tip T sincron, unele care comută pe front (atunci când se schimbă tactul, din 0 în 1 – *frontul ascendent, ridicător sau pozitiv* – sau din 1 în 0 – *frontul descendente, coborâtor sau negativ*) și altele care comută pe nivel (atunci când tactul este pe nivel stabil 1). Primele sunt de tip *Flip-Flop*, celelalte de tip *Latch* (zăvor).

Tabelul de excitație pentru bistabilul T sincron este următorul:

There are two types of synchronous T memory cell: an edge-triggered one (when the clock signal is changing from 0 to 1 – *raising edge* – or from 1 to 0 – *falling edge*) and a level-triggered one (when the clock signal is stable on level 1). The first type is called *Master-Slave (Flip-Flop)*, the other one is called *Latch*.

The excitation table for the synchronous T Latch is the following one:

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

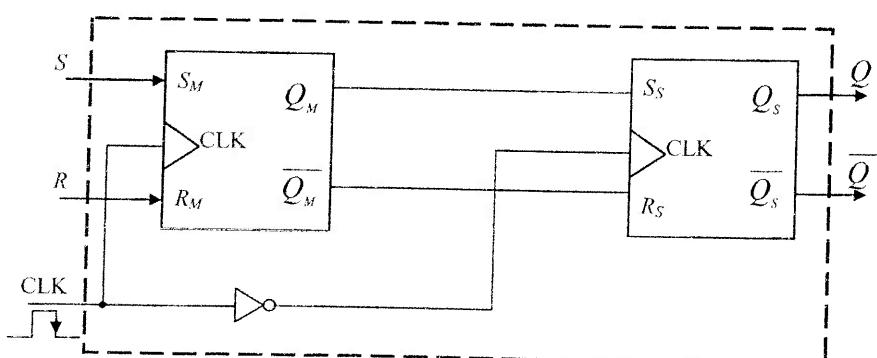
Bistabilul Stăpân-Sclav sau Master-Slave (Flip-Flop)

La bistabilele de acest tip, comutarea se efectuează pe frontul posterior (descendent) al impulsului de ceas. În esență un bistabil *master-slave* este alcătuit din doi bistabili sincroni și eventuale reacții suplimentare.

Vom ilustra funcționarea acestui tip de bistabil pentru un bistabil RS.

Primul bistabil, denumit *master* (stăpân), citește informația după frontul anterior al impulsului de comandă (de tact sau de clock), iar cel de-al doilea denumit *slave* (sclav) – ale cărui ieșiri corespund cu ieșirile bistabilului luat în ansamblu – comută pe frontul posterior al impulsului de comandă (de tact).

Schema de principiu este următoarea:



De reținut că principiul *master-slave* poate fi aplicat oricărui tip de bistabil.

The Master-Slave Flip-Flop

For Flip-Flops, the switching is done on the negative (descending) edge of the clock impulse. Basically, a *master-slave* Flip-Flop is composed of two synchronous Latches and possibly supplemental feedbacks.

We will illustrate the functioning of this type of elementary memory cell for an RS Flip-Flop.

The first Latch, called *master*, reads the information after the raising edge of the command (clock) impulse, and the second one, called *slave* – whose outputs match the outputs of the Flip-Flop taken in its totality – switches on the falling edge of the command (clock) impulse.

The general scheme is the following one:

Notice that the *master-slave* principle can be applied to any type of Latch.

4.3. Aplicații ale circuitelor basculante bistabile

Bistabilele sunt la baza realizării majorității circuitelor logice secvențiale, de aceea este esențial să cunoaștem principalele tipuri de blocuri constructive care au la bază bistabile.

Numărătoare

Numărătoarele sunt circuite logice secvențiale care contorizează numărul de impulsuri de tact aplicate la intrare. Ele se realizează prin asocierea circuitelor basculante bistabile (având rol de celule de memorie binară), cu circuite logice combinaționale, care determină modul corect în care urmează ca numărătorul să-și schimbe starea la fiecare nou impuls de tact aplicat la intrare.

Clasificarea numărătoarelor se face după anumite criterii:

1. *modul de funcționare* (comutare a bistabililor):
 - asincrone – celulele de memorie din care este construit numărătorul nu comută simultan;
 - sincrone – celulele de memorie din care este construit numărătorul comută simultan sub acțiunea unui impuls de tact aplicat simultan tuturor celulelor.
2. *modul de modificare a stărilor (conținutului)*:

4.3. Applications of Latches and Flip-Flops

Most sequential logic circuits rely on Latches and Flip-Flops, thus it is essential to know the main types of building blocks that use Latches and Flip-Flops as basic “bricks” of their architecture.

Counters

Counters are sequential logic circuits that count the number of clock impulses applied on their input. They are realized by associating Flip-Flops and Latches (which play the role of binary memory cells), with combinational logic circuits, which determine the correct way in the counter will change its state at each new clock impulse applied on the input.

Counters classification is done after certain criteria:

1. *functioning mode* (the way the Flip-Flops switch their state):
 - asynchronous – the memory cells from the counter's structure do not switch simultaneously;
 - synchronous – the memory cells from the counter's structure switch simultaneously when triggered by a clock impulse applied simultaneously at all cells.
2. *state (content) modification mode*:

- directe – își crește conținutul cu o unitate la fiecare impuls aplicat la intrare;
- inverse – conținutul scade cu o unitate la fiecare impuls aplicat la intrare;
- reversibile – numără direct sau invers, în funcție de o comandă aplicată din exterior.

3. modul de codificare a informației:

- binare;
- binar-zecimale,
- modulo „ p ” etc.

Numărătoarele se pot realiza cu celule de memorie de tip T care realizează o divizare cu 2. Prin interconectarea a „ n ” celule de memorie se obține un numărător cu un număr de 2^n stări distincte. Fiecărei stări îi vom asocia câte un cuvânt de cod binar de lungime n , reprezentând conținutul celor n celule binare pentru starea dată a numărătorului. Codul în care numără un numărător va fi dat de succesiunea cuvintelor de cod binar asociate stărilor sale.

Dacă din cele 2^n stări ale numărătorului se elimină k stări, rezultă un numărător cu $p = 2^n - k$ stări distincte. Matematic, operația realizată de numărător este o operație modulo p .

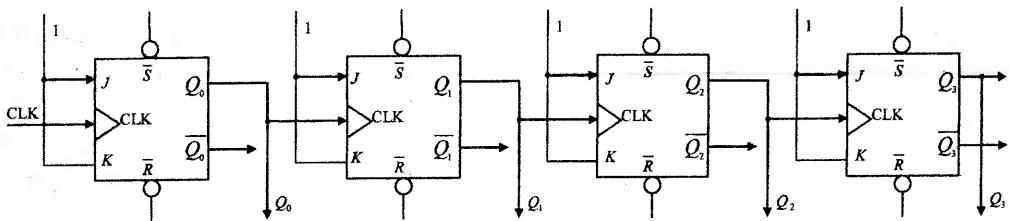
Iată un exemplu de numărător binar, asincron și direct, pe 4 biți:

- direct – their content increases with a unit at each impulse applied at the input;
 - inverse – their content decreases with a unit at each impulse applied at the input;
 - reversible – they count upwards or downwards, according to an external command.
3. *information encoding mode:*
- binary;
 - binary-decimal;
 - modulo “ p ” etc.

Counters can be realized with T type memory cells, which make a division by 2. By interconnecting “ n ” memory cells we obtain a counter with 2^n distinct states. We will associate to each state a binary code of length n , representing the content of the n binary cells for the given state of the counter. The code in which a counter counts will be given by the sequence of the binary words associated to its states.

If from the 2^n states of the counter we eliminate k states, we obtain a counter having $p = 2^n - k$ distinct states. Mathematically, the operation realized by the counter is a modulo p operation.

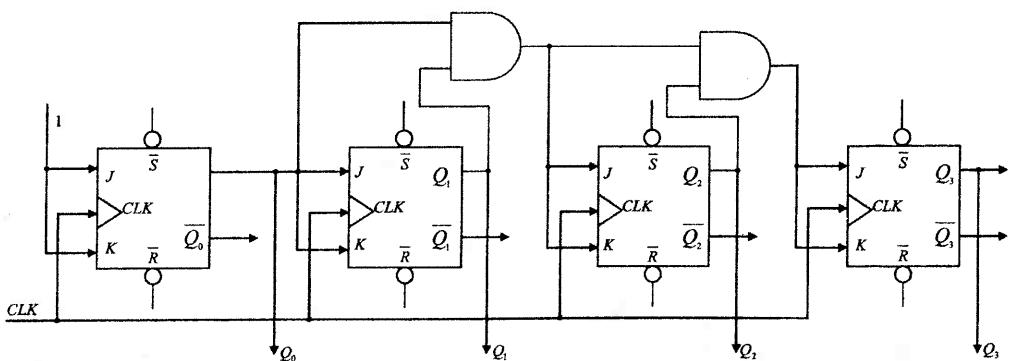
Here is an example of a 4-bit asynchronous direct binary counter:



Funcționarea acestui tip de numărător este asincronă în raport cu impulsul de numărare recepționat la intrare. Schema prezintă dezavantajul că numărătorul urmărește impulsul de intrare cu o întârziere variabilă. Atunci când procesul trebuie oprit într-o stare, dificultatea care apare este că următorul impuls de numărare e generat de procesul care comandă înainte ca numărătorul să poată genera impulsul de oprire. Aceasta este un serios factor de limitare în utilizarea numărătoarelor asincrone. Iată în continuare un alt exemplu: un numărător binar, sincron și direct, pe 4 biți:

The functioning of this type of counter is asynchronous with respect to the counting impulse received on the input. The scheme has a major drawback: the counter follows the input impulse with variable delay. When the process must be stopped in a certain state, the difficulty that appears is that the next counting impulse is generated by the driving process before the counter can generate the stopping impulse. This is a serious limiting factor in the usage of asynchronous counters.

Here is another example: a 4-bit synchronous direct binary counter:



Intrările J și K ale primului bistabil sunt legate la „1” și vor comuta bistabilul la fiecare tact (conform tabelului său de adevăr). Așadar bistabilul comută doar din 2 în 2 impulsuri de tact, adică atunci când Q_0 trece din 1 în 0, deci intrările sale J și K pot fi legate la ieșirea primului bistabil. Așadar treilea bistabil basculează din 4 în 4 impulsuri și va fi comandat de funcția SI între ieșirile Q_1 și Q_2 , iar al patrulea bistabil comută din 8 în 8 impulsuri și va fi comandat de funcția SI între ieșirile Q_5 , Q_6 și Q_0 .

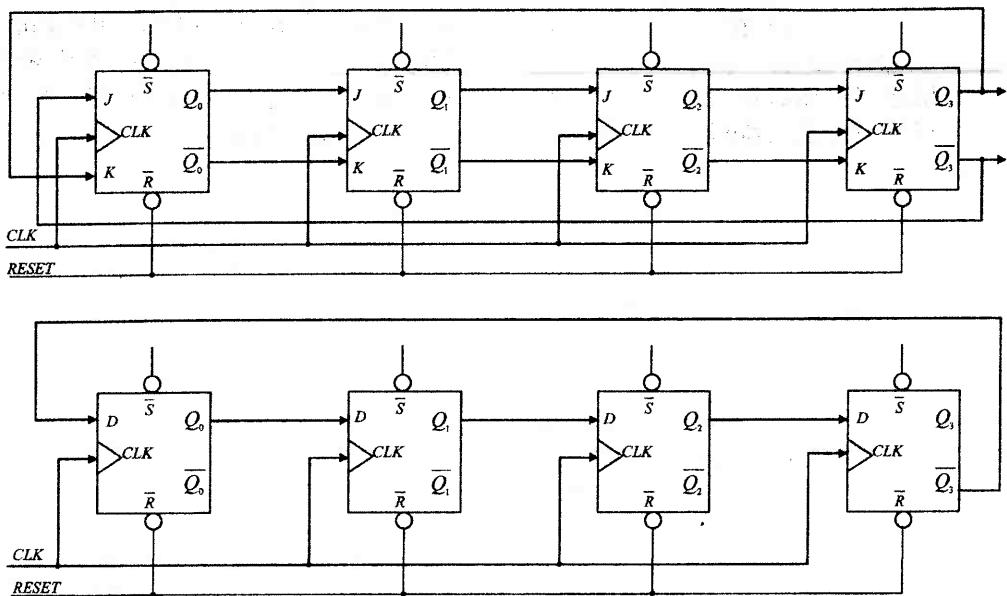
Număraoare Moebius

Un caz particular de număraoare sunt număraoarele Moebius. Acestea au structura analogă celei din figura următoare (unde avem un exemplu de realizare pe 4 biți, cu bistabili JK și mai jos cu bistabili D), putând fi extinse pe un număr de biți oricât de mare:

The J and K inputs of the first Flip-Flop are connected to ‘1’ and will switch the Flip-Flop on each clock impulse (according to its truth table). The second Flip-Flop switches only every 2 clock impulses, i.e. when Q_0 goes from 1 to 0, so its inputs J and K can be connected to the output of the first flip-flop. The third Flip-Flop switches every 4 clock impulses and will be driven by the AND function between the outputs Q_1 and Q_2 , and the fourth Flip-Flop switches every 8 clock impulses and will be driven by the AND function between the outputs Q_5 , Q_6 and Q_0 .

Moebius counters

A particular case of counters are the Moebius counters. They have a structure that is analogous to the one in the next figure (where we have an example of a 4-bit Moebius counter, implemented first with JK Flip-Flops, then with D Flip-Flops). They can be extended on any number of bits:



Succesiunea stărilor este:
 $0 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 0$.

Numele Moebius vine de la asemănarea cu faimoasa *panglică* sau *bandă a lui Moebius*, care spre deosebire de alte panglici, în sensul clasic al cuvântului, nu are decât o singură parte (și o singură margine)! O astfel de bandă se poate realiza deosebit de simplu dintr-o foaie de hârtie format A4. Se taie foaia pe lung, iar cele două bucăți de hârtie rezultate se unesc la un capăt. Capetele panglicii astfel formate se unesc și ele, dar unul din capete se răsușește cu o jumătate de rotație. Va rezulta astfel o panglică aparent normală dar care a fost lipită „invers”.

The states sequence is:
 $0 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 0$.

The name Moebius comes from its similarity with the famous *Moebius ribbon* which, unlike other ribbons (in the classical sense of this word) has only one side (and one edge)! Such a ribbon can be made extremely easily from a piece of paper size A4. We cut the paper on its long edge and the two resulting pieces of paper are united at one end. The ends of the ribbon that is obtained this way are also united, but one of the ends is twisted by a 180° rotation. The resulting ribbon apparently looks normal, but it has been glued “the opposite way”.



Banda lui Moebius

Moebius ribbon

Registre

Registrele sunt circuite logice secvențiale care permit stocarea și / sau deplasarea informației codificate binar. Ele se realizează din celule de memorie binară (CBB-uri) și din circuite logice combinaționale (CLC-uri), care permit înscrerea, citirea și transferul informației. Capacitatea unui registru este dată de numărul celulelor sale de memorie.

Orice informație binară, care nu depășește capacitatea registrului, poate fi înscrisă prin acționarea corespunzătoare a intrărilor (care depinde și ea de natura bistabilelor). Registrele pot să fie de mai multe tipuri: de memorie; de deplasare; combinate; universale.

Registers

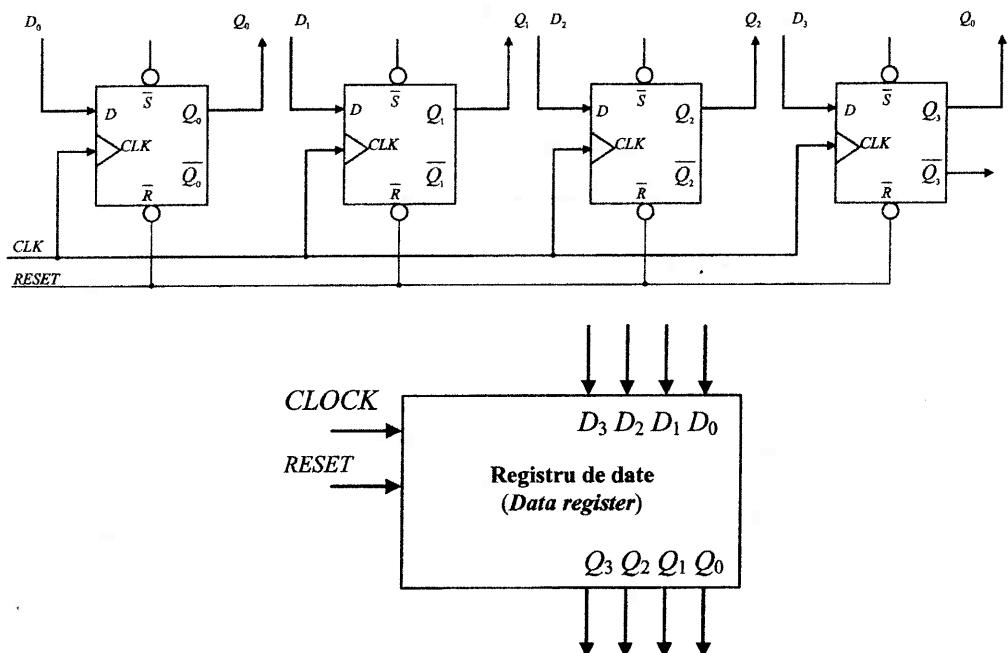
Registers are sequential logic circuits that allow storing and / or shifting the binary encoded information. They are made from binary memory cells (usually, Flip-Flops) and from combinational logic circuits (CLCs), which allow writing, reading and transferring information. A register's capacity is given by the number of its memory cells.

Any binary information that does not exceed the register's capacity can be written by properly setting the input values (this also depends on the type of the Flip-Flops).

There are several types of registers: memory registers; shift registers; combined registers; universal registers.

Registrele de memorie memorează informația binară în celule de memorie binară. În fiecare celulă de memorie se memorează un bit de informație. Încărcarea se poate face paralel, prin intrările asincrone *Set* și *Reset*.

Memory registers store binary information in binary memory cells. In each memory cell is stored one bit of information. The loading can be done in parallel, by means of the asynchronous inputs *Set* and *Reset*.



Registrele de deplasare realizează transferul informației. Transferul se poate face: de la stânga la dreapta; de la dreapta la stânga; în ambele sensuri.

La fiecare impuls de tact conținutul registrului se deplasează cu câte o celulă (în sensul stabilit). Semnalul de ieșire este identic cu cel de intrare, dar întârziat cu un număr de impulsuri de tact egal cu numărul de

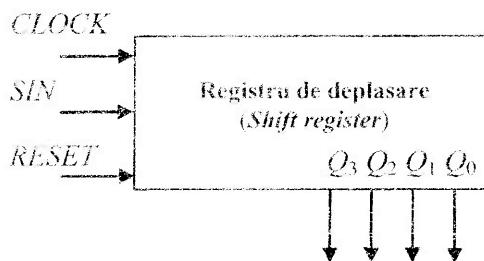
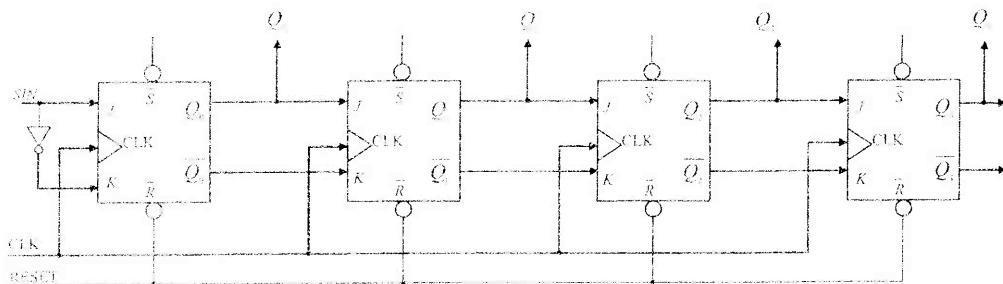
Shift registers perform the information transfer. The transfer can be done: from left to right; from right to left; in both senses.

At each clock impulse the register's content is shifted by one cell (in the pre-established direction). The output signal is identical to the input one, but delayed by a number of clock impulses that is equal to the

celule de memorie (bistabile) din care este format registrul.

Exceptând primul bistabil, ecuația de stare a unui registru de deplasare stânga-dreapta este dată de relația: $Q_i(t+1) = Q_{i-1}(t) \bullet CLK$ (unde CLK este impulsul de tact).

Exemplu Registrul de deplasare stânga-dreapta implementat cu bistabile JK.



La fiecare impuls de tact conținutul bistabilului Q_i se transferă în bistabilul Q_{i+1} . În bistabilul Q_0 se introduce informația din exterior, iar conținutul ultimului bistabil se

number of memory cells (usually, Flip-Flops) which constitute the register.

Except for the first Flip-Flop, the state equation of a left-to-right shift register is given by the relationship: $Q_i(t+1) = Q_{i-1}(t) \bullet CLK$ (where CLK is the clock impulse).

Example Left-to-right shift register implemented with JK Flip-Flops.

At each clock impulse the content of the Flip-Flop Q_i is transferred into the Flip-Flop Q_{i+1} . In the Flip-Flop Q_0 we introduce the information from the outside world, and the

pierde. Încărcarea registrului se realizează deci în mod serie.

Inițializarea registrului se face prin semnalul de *Reset*, care forțează toate ieșirile registrului în starea '0'.

Registrele de deplasare dreapta-stânga și reversibile (bidirectionale) se realizează folosind circuite logice combinaționale suplimentare.

Registrele combine sunt cele care au și funcția de memorare și cea de deplasare.

Registrele universale cumulează toate funcțiile: deplasare stânga-dreapta, deplasare dreapta-stânga, încărcare serie sau paralelă a informației.

4.4. Sinteza circuitelor secvențiale sincrone

Circuitele secvențiale sincrone trec dintr-o stare în alta la momente distințe de timp, determinate de impulsurile de tact. Între două impulsuri de tact starea circuitului nu se modifică.

content of the last Flip-Flop is lost. In conclusion, the register is loaded serially.

The register's initialization is done by means of the *Reset* signal, which forces all the register's outputs into state '0'.

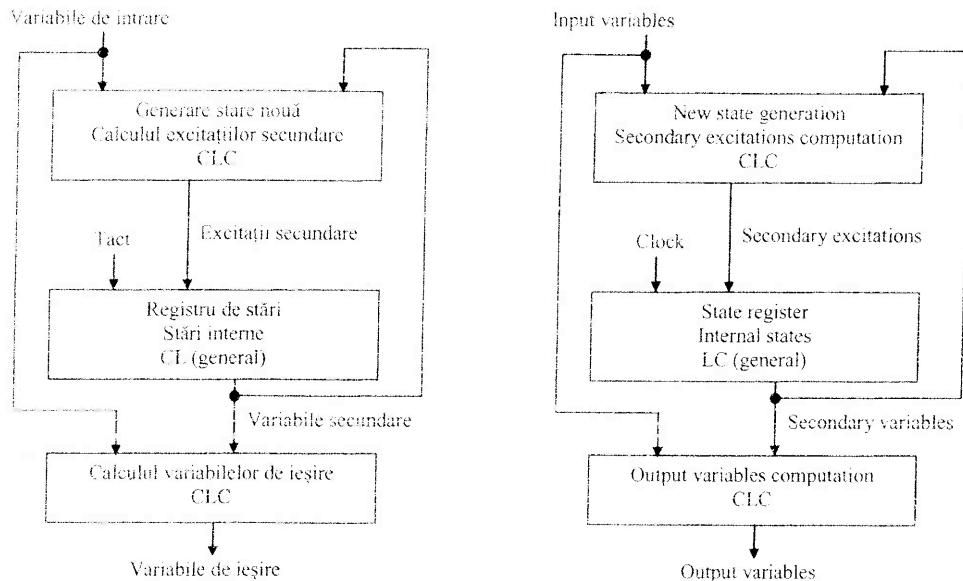
Right-to-left and reversible (bidirectional) shift register are made using supplemental combinational logic circuits.

Combined registers are those that have both the memory and the shift functions.

Universal registers cumulate all the functions: left-to-right shift, right-to-left shift, serial or parallel load of information.

4.4. Sequential logic circuits synthesis

Sequential synchronous circuits pass from a state into another one at distinct moments of time, which are determined by the clock impulses. Between two clock impulses the circuit's state is not modified.



CL este un circuit logic general, în care se păstrează starea internă a circuitului logic secvențial. Este deci numit și *Registru de stări*, implementat cu bistabili RS, D, JK, T, registre, memorii, dar care poate fi implementat și ca un circuit logic general cu buclă de reacție (deci nu conține bistabile, ci numai porți logice sau alte componente combinaționale, starea fiind memorată cu ajutorul buclelor de reacție – acesta este cazul automatelor asincrone).

Primul CLC determină funcțiile de excitație secundare (numite și funcții de tranziție) care în prezența tactului determină trecerea circuitului din starea curentă în altă stare. El poate fi numit *Generatorul*

LC is a general logic circuit, which stores the internal state of the sequential logic circuits. Therefore, it is also called *State register*, implemented with RS, D, JK, T Flip-Flops, registers, memories, but which can also be implemented as a general logic circuit with feedback loop (so it doesn't contain Flip-Flops, but only logic gates or other combinational components, the state being memorized by means of the feedback loops – this is the case of asynchronous automata).

The first CLC determines the secondary excitation functions (also called transition functions) which, in presence of the clock, determine the circuit's passage from the current state into another state. It can be

noii stări; se poate realiza cu porți logice sau cu circuite specializate (multiplexoare, decodificatoare etc.).

Variabilele de intrare sunt în general sincrone cu impulsul de tact, dar pot fi și de tip asincron.

Al doilea CLC determină funcțiile de ieșire, în funcție de starea curentă și de intrări (acesta este cazul cel mai general, al așa-ziselor *automate Mealy*; dacă ieșirea depinde numai de starea curentă, avem *automate Moore*).

Etapele de sinteză

În sinteza circuitelor secvențiale sincrone trebuie urmări 7 pași sau etape:

1. Exponerea condițiilor de funcționare (descrierea comportării circuitului).

Se stabilește modalitatea de definire a circuitului care trebuie sintetizat prin unul din următoarele formalisme:

- tabel de tranziții;
- graf de tranziții;
- organigramă;
- forme de undă.

Trebuie evidențiate:

- stările prin care trece circuitul;
- valorile variabilelor de intrare care declanșează schimbarea stării;
- valorile rezultante ale variabilelor de ieșire.

called the *New state generator*; it can be made with logic gates or with specialized circuits (multiplexers, decoders, etc.).

The input variables are generally synchronous with the clock impulse, but can also be asynchronous.

The second CLC determines the output functions, according to the current state and the inputs (this is the most general case, of the so-called *Mealy automata*; if the output only depends on the current state, we have *Moore automata*).

Synthesis steps

In the synthesis of sequential synchronous circuits one must follow 7 main steps:

1. Exposing the functioning conditions (describing the circuit's behavior).

One must establish the way of defining the circuit to be synthesized by one of the following formalisms:

- transition table;
- transition graph;
- state diagram;
- waveform.

One must outline:

- the states through which the circuit passes;
- the values of the input variables that trigger a change of state;
- the resulting values of the output variables.

Evoluția circuitului începe într-o stare inițială și de obicei se revine la această stare, după sfârșitul ciclului său de funcționare.

2. Se codifică stările circuitului.
3. Se urmărește reducerea numărului de stări (simplificarea) circuitului, dacă este posibil, astfel încât să se păstreze funcționarea lui corectă.
4. Se decide asupra modului de implementare (CLC-urile și registrul de stări interne).
5. Se determină funcțiile de excitație și cele de ieșire.
6. Se studiază problemele legate de eventualele ieșiri false (hazard) sau tranziții false.
7. Se desenează circuitul.

4.5. Probleme rezolvate

1. Realizați un bistabil JK folosind un bistabil D.

Soluție

Există 6 conversii posibile între principalele 3 tipuri de bistabili (JK, D și T), aceasta fiind poate cea mai dificilă. Problema poate fi ilustrată foarte bine de figura următoare, în care vedem modulul „cel mare” care trebuie să se comporte ca un bistabil JK, în timp ce în interiorul său acesta conține un bistabil D.

The circuit's evolution starts from an initial state and usually the system gets back in this state, after the end of its functioning cycle.

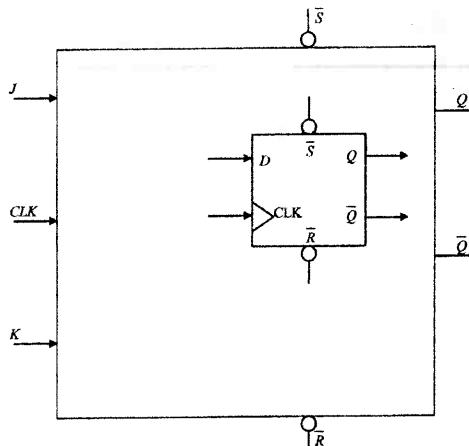
2. Encoding the circuit's states.
3. Aiming at reducing the number of states of the circuit (simplifying), if possible, so that its correct behavior is preserved.
4. Deciding upon the implementation mode (the CLCs and the internal states register).
5. Determining the excitation functions and the output functions.
6. Studying problems related to the possible false outputs (hazard) or false transitions.
7. Drawing the circuit.

4.5. Solved problems

1. Realize a JK Flip-Flop using a D Flip-Flop.

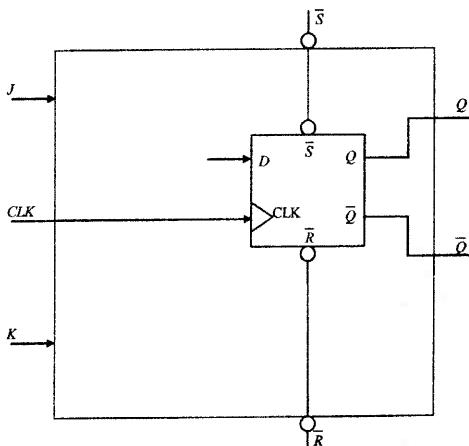
Solution

There are six possible conversions between the main three types of Flip-Flops (JK, D and T), and this one is maybe the most difficult. The problem can be well illustrated by the next figure, in which we see the “big” module which has to act as a JK Flip-Flop, while inside it contains a D Flip-Flop.



Trebuie să realizăm pentru început corespondența dintre intrări și ieșiri. Primele legături sunt evidente:

For starting we must make the correspondence between inputs and outputs. The first connections are straightforward:



Rămâne de determinat funcția Booleană a cărei ieșire este intrarea D a bistabilului D. Pentru aceasta, stim că la bistabilul JK starea

We still have to determine the Boolean function whose output is the Flip-Flop's D input. In order to do this, we know that in the JK Flip-

următoare depinde de J , K și Q . La bistabilul D, starea următoare Q^* este egală cu intrarea sincronă D .

Așadar: $D = Q^* = f(J, K, Q)$.

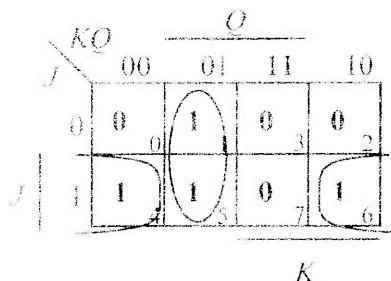
Construim tabelul de adevăr al acestei funcții Booleene, ținând cont de tabelul de adevăr al celor doi bistabili:

J	K	Q	$D = Q^*$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Rezultă următoarea diagramă Karnaugh.

diagramă

We obtain the following Karnaugh map:



Deci $D = J \cdot \bar{Q} + \bar{K} \cdot Q$.

Atunci schema finală este:

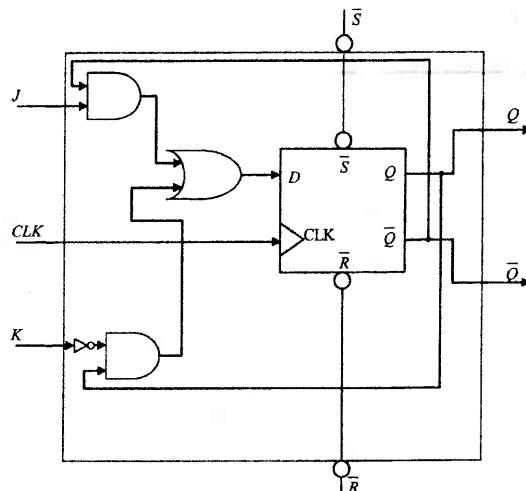
So $D = J \cdot \bar{Q} + \bar{K} \cdot Q$.

Then, the final scheme is:

Flop the next state depends on J , K and Q . In the D Flip-Flop, the next state Q^* equals the synchronous input D .

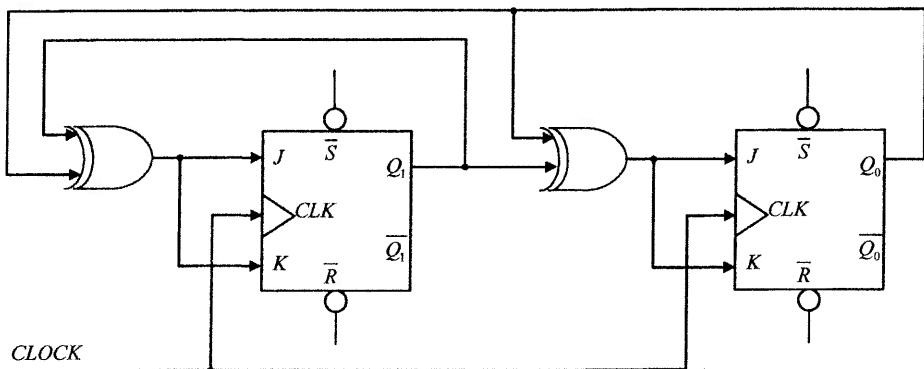
So: $D = Q^* = f(J, K, Q)$.

We build the truth table of this Boolean function, taking into account the two Flip-Flops' truth tables:



2. Determinați graful de tranziții al circuitului:

2. Determine the transition graph of the following circuit:



Soluție

Întrucât avem două bistabile în circuit, rezultă că există 4 stări distincte. În lipsa unui circuit de inițializare, starea initială a sistemului poate fi oricare din cele 4 posibile: 00, 01, 10 sau 11. Așadar,

Solution

Since we have two Flip-Flops in the circuit, there are 4 distinct states. In the absence of an initialization circuit, the system's initial state can be any of the four possible states: 00, 01, 10 or 11. So, we will have to

va trebui să desenăm un graf cu 4 noduri, iar din fiecare nod va exista *exact* o tranziție, notată printr-un arc orientat. Problema se consideră complet rezolvată în momentul în care vom fi determinat toate arcele din graf.

Pentru început, vom obține ecuațiile intrărilor sincrone (J și K) ale celor două bistabile, pe care le vom nota, de la stânga la dreapta, Q_1 și Q_0 :

$$J_1 = K_1 = Q_1 \oplus \overline{Q_0}; \quad J_0 = K_0 = Q_1 \oplus Q_0.$$

Vom crea apoi un tabel de adevăr:

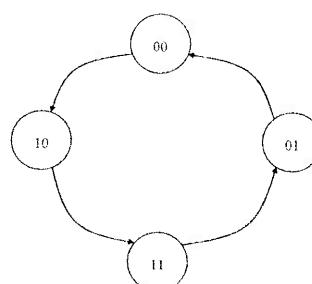
Q_1	Q_0	J_1	K_1	J_0	K_0	Q_1^+	Q_0^+
0	0	1	1	0	0	1	0
0	1	0	0	1	1	0	0
1	0	0	0	1	1	1	1
1	1	1	1	0	0	0	1

Atunci, graful de tranziții se obține foarte ușor din tabelul de adevăr:

draw a graph containing 4 vertices, and from each vertex there will be *exactly* one transition, denoted by an oriented edge. The problem is considered to be completely solved when we will have determined all the edges from the graph.

For beginning, we will obtain the equations of the synchronous inputs (J and K) of the two Flip-Flops, that we will denote, from left to right, Q_1 and Q_0 :

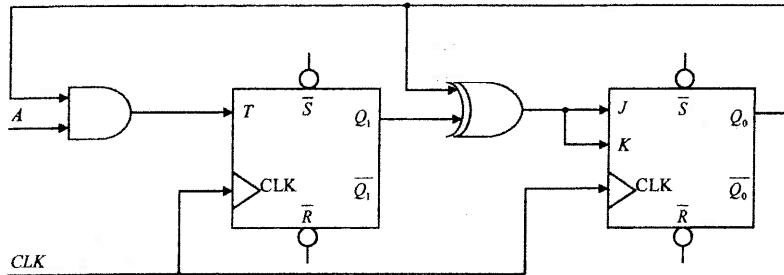
Then, we will create a truth table:



3. Să se determine graful de tranziții al circuitului din figura următoare.

The transition graph is then very easy to obtain from the truth table:

3. Determine the transition graph of the circuit in the next figure.

Discuție.Discussion.Soluție

Deoarece avem o intrare (A), rezultă că din fiecare nod vor pleca exact 2 arce: unul pentru cazul când $A = 0$ și unul pentru cazul când $A = 1$. În rest, se procedează la fel ca la problema anterioară.

Obținem ecuațiile intrărilor sincrone (T , J și K) ale celor două bistabile, pe care le vom nota, de la stânga la dreapta, Q_1 și Q_0 :

$$T = A \bullet Q_0; \quad J = K = Q_1 \oplus Q_0$$

Solution

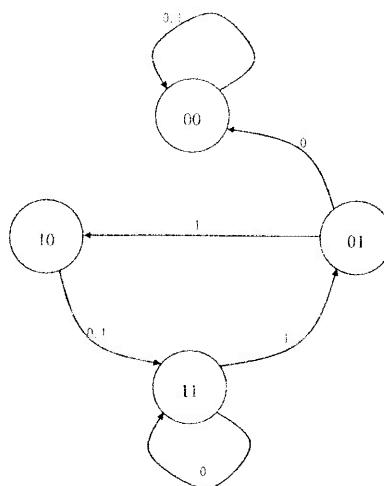
Since we have an input (A), it results that from each node we will have exactly two outgoing edges: one for the case when $A = 0$ and one for the case when $A = 1$. Otherwise, we proceed the same way as for the previous problem.

We obtain the equations of the synchronous inputs (T , J and K) of the two Flip-Flops, that we will denote, from left to right, Q_1 and Q_0 :

A	Q_1	Q_0	T_1	J_0	K_0	Q_1^+	Q_0^+
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0
1	0	1	1	1	1	1	0
0	1	0	0	1	1	1	1
1	1	0	0	1	1	1	1
0	1	1	0	0	0	1	1
1	1	1	1	0	0	0	1

Graful de tranzitii se obtine foarte usor din tabelul de adevar:

The transition graph is very easily obtained from the truth table:



4. Să se proiecteze cu toate cele 3 tipuri de bistabile (JK, D și T) un circuit logic sevențial (CLS) cu următoarele proprietăți:

- pentru intrarea $DIR = 0$ generează la ieșire numerele de la 0 la 6 în ordine crescătoare;
- pentru intrarea $DIR = 1$ generează la ieșire numerele de la 6 la 0 în ordine descrescătoare;
- ieșirea $z = 1$ pentru fiecare configurație care este multiplu de 3.

Soluție

Vom desena pentru început cutia neagră a sistemului (vezi figura următoare). Stările interne nu reprezintă ieșiri propriu-zise, dar în

4. Design using all the three types of Flip-Flops (JK, D and T) a sequential logic circuit (SLC) with the following properties:

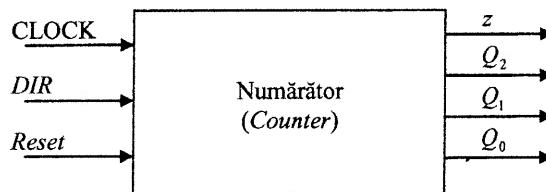
- when the input $DIR = 0$ it generates at its output the numbers from 0 to 6 in ascending order;
- when the input $DIR = 1$ it generates at its output the numbers from 6 to 0 in descending order;
- the output $z = 1$ for each configuration multiple of 3.

Solution

First, we will draw the system's black box (see the next figure). The internal states do not represent proper outputs, but in this case it is

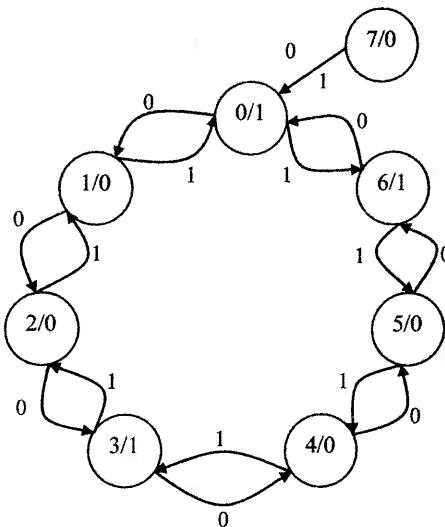
acest caz este necesar să fie „scoase” și ele în exterior și făcute accesibile întocmai precum ieșirile. Se observă din analiza problemei că circuitul secvențial sincron se comportă ca un numărător reversibil modulo 7. *DIR* este intrarea care dă sensul de numărare (ascendent sau descendenter), iar *z* este ieșirea.

necessary to “pull them out” and make them accessible exactly as the regular outputs are. We notice from the problem’s analysis that the sequential logic circuits behaves like a reversible modulo 7 counter. *DIR* is the input that gives the counting direction (ascending or descending), while *z* is the output.



Desenăm apoi graful de tranziții:

Then, we draw the transition graph:



Se observă că ieșirea sistemului depinde numai de starea internă, așa că ea poate fi reprezentată în

One can notice that the system’s output only depends on its internal state, so it can be represented inside

interiorul nodului (semnificația este „stare internă / ieșire”).

Din specificația problemei, observăm că starea 7 este în afara buclei de numărare. Trebuie precizat comportamentul din starea 7, în care se poate ajunge numai în mod accidental. Operațiunea de reducere a numărătorului în bucla de numărare prestabilită, în cazul în care ajunge în mod accidental într-o altă stare care nu face parte din bucla de numărare, se numește „auto-corecție”.

Operațiunea de aducere a numărătorului în starea inițială (în acest caz, aceasta o vom considera starea 0), pentru a nu ajunge în mod accidental într-o altă stare care nu face parte din bucla de numărare, se numește „auto-inițializare”.

Așadar, prin specificarea tranzitiei din starea 7 în starea 0, indiferent dacă pe intrarea *DIR* avem 0 sau 1, am realizat simultan auto-corecția și auto-inițializarea numărătorului.

Dată fiind că avem 7 stări în sistem, ele vor fi reprezentate pe 3 biți: Q_2 , Q_1 și Q_0 (reprezentând ieșirile a 3 bistabile care alcătuiesc împreună registrul de stări interne ale sistemului).

Urmează să identificăm dependențele de date. Ne punem întrebările:

a) De cine depind ieșirile? După

the node (the meaning is „internal state / output”).

From the problem's specification, we observe that state 7 is outside the counting loop. We must specify the behavior from state 7, which can be reached only accidentally. The process of bringing back the counter inside its predefined counting loop, in case it accidentally gets into another state that does not belong to the counting loop, is called „self-correction”.

The process of bringing back the counter in the initial state (in the present case, we will consider state 0 to be the initial one), to prevent it to accidentally reach another state that does not belong to the counting loop, is called „self-initialization”.

So, by specifying the transition from state 7 to state 0, regardless of the value that we have on the *DIR* input (0 or 1), we have simultaneously made the self-correction and the self-initialization of this counter.

Since we have 7 states in our system, they will be represented on 3 bits: Q_2 , Q_1 and Q_0 (representing the outputs of 3 Flip-Flops that together make up the system's internal states register).

Then we must identify the data dependencies. We ask ourselves the following questions:

a) On whom do the outputs depend?

cum am arătat, ele depind numai de starea prezentă.

b) De cine depinde starea următoare? Cu alte cuvinte, dacă furnizăm un nou impuls de tact sistemului, care va fi starea următoare? Răspunsul este: depinde în ce stare ne aflăm în momentul actual și ce valoare are intrarea *DIR*.

Așadar:

$$\begin{aligned} Q^+ &= f(Q_2, Q_1, Q_0, \text{DIR}) \\ z &= g(Q_2, Q_1, Q_0) \end{aligned}$$

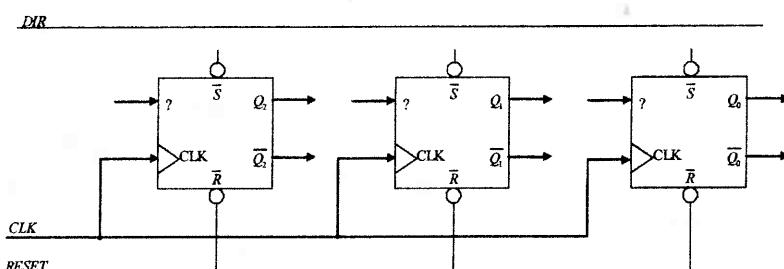
În această etapă, avem deja o idee aproximativă despre aspectul sistemului:

As we have shown, they only depend on the current state.

b) On whom does the next state depend? In other words, if we provide a new clock impulse to the system, which will be the next state? The answer is: it depends in which state we are at the current moment of time and what is the value present on the *DIR* input.

Therefore:

At this stage, we already have an approximate idea about the system's aspect:



Încă nu am ales tipul de bistabile cu care vom implementa sistemul, dar știm că acesta este sincron (toate bistabilele primesc același semnal de tact) și conține 3 bistabile. Am introdus și o intrare de *Reset* asincronă.

Pe baza tabelului de excitație al fiecărui bistabil, urmează să creăm

We didn't choose yet the type of Flip-Flops that we will use to implement the system, but we know that it is a synchronous one (all the Flip-Flops receive the same clock signal) and that it contains 3 Flip-Flops. We have also introduced an asynchronous *Reset* input.

Based on each Flip-Flop's excitation table, we will create the

tabelul de adevăr al sistemului. Tabelele de excitație sunt reunite în tabelul de mai jos:

system's truth table. The excitation tables are reunited in the table below:

Q	Q^+	J	K	D	T
0	0	0	X	0	0
0	1	1	X	1	1
1	0	X	1	0	1
1	1	X	0	1	0

Tabelul de adevăr al sistemului este cel de mai jos:

The system's truth table is the following one:

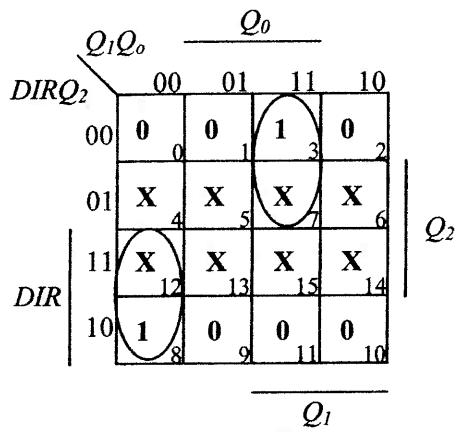
Starea curentă (current state)			Starea viitoare (next state)			Implementare cu bistabile JK (implementation with JK FFs)						Implementare cu bistabile D (implementation with D FFs)			Implementare cu bistabile T (implementation with T FFs)			Ieșire (Output)		
DIR	Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2	K_2	J_1	K_1	J_0	K_0	D_2	D_1	D_0	T_2	T_1	T_0	z	
0	0	0	0	0	0	1	0	X	0	X	1	X	0	0	1	0	0	1	1	
1	0	0	0	1	1	0	1	X	1	X	0	X	1	1	0	1	1	0	1	
0	0	0	1	0	1	0	0	X	1	X	X	1	0	1	0	0	1	1	0	
1	0	0	1	0	0	0	0	X	0	X	X	1	0	0	0	0	0	1	0	
0	0	1	0	0	1	1	0	X	X	0	1	X	0	1	1	0	0	0	1	0
1	0	1	0	0	0	1	0	X	X	1	1	X	0	0	1	0	1	1	0	
0	0	1	1	1	0	0	1	X	X	1	X	1	1	0	0	1	1	1	1	
1	0	1	1	0	1	0	0	X	X	0	X	!	0	1	0	0	0	1	1	
0	1	0	0	1	0	1	X	0	0	X	1	X	1	0	1	0	0	1	0	
1	1	0	0	0	1	1	X	1	1	X	1	X	0	1	1	1	1	1	0	
0	1	0	1	1	1	0	X	0	1	X	X	1	1	1	0	0	1	1	0	
1	1	0	1	1	0	0	X	0	0	X	X	1	1	0	0	0	0	1	0	
0	1	1	0	0	0	0	X	1	X	1	0	X	0	0	0	1	1	0	1	
1	1	1	0	1	0	1	X	0	X	1	1	X	1	0	1	0	1	1	1	
0	1	1	1	1	0	0	X	1	X	1	X	1	0	0	0	1	1	1	0	
1	1	1	1	0	0	0	X	1	X	1	X	1	0	0	0	1	1	1	0	

Pe baza acestui tabel, vom obține ecuațiile în cele 3 cazuri (ale implementării cu bistabile de tip JK, D și T).

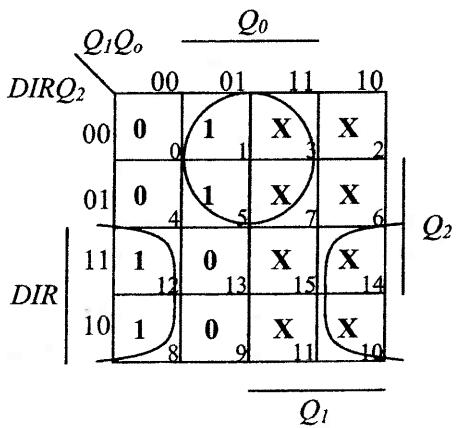
Based on this table, we will obtain the equations in the three cases (i.e. when implementing with JK, D and T Flop-Flops).

Implementarea cu bistabile JK

Este necesar să trasăm 6 diagrame Karnaugh. Știm că toate intrările J și K sunt funcții de intrarea DIR și de starea prezentă ($Q_2Q_1Q_0$); atunci, pe baza tabelului de mai sus, obținem:



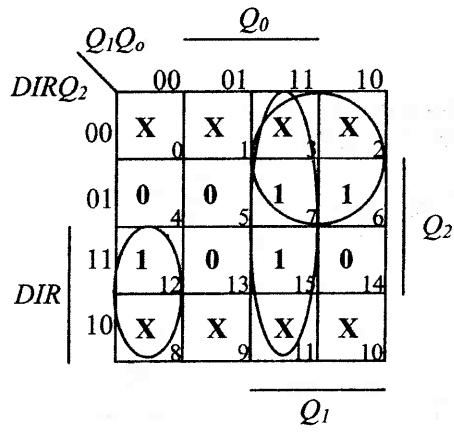
$$J_2 = DIR \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{DIR} \cdot Q_1 \cdot Q_0$$



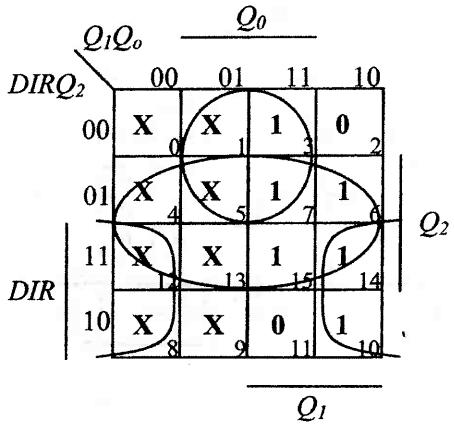
$$J_1 = DIR \cdot \overline{Q_0} + \overline{DIR} \cdot Q_0$$

Implementing with JK Flip-Flops

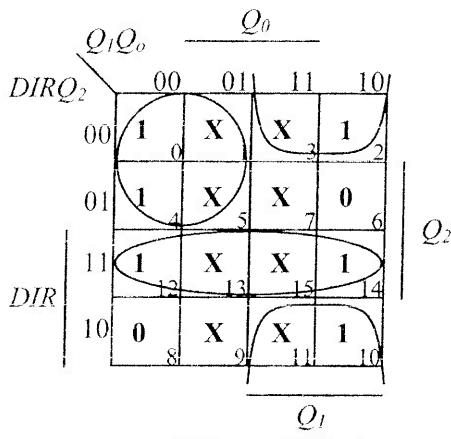
We have to draw 6 Karnaugh maps. We know that all J and K inputs are functions of the DIR input and the current state ($Q_2Q_1Q_0$); then, based on the table above, we obtain:



$$K_2 = DIR \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{DIR} \cdot Q_1 + Q_1 \cdot Q_0$$



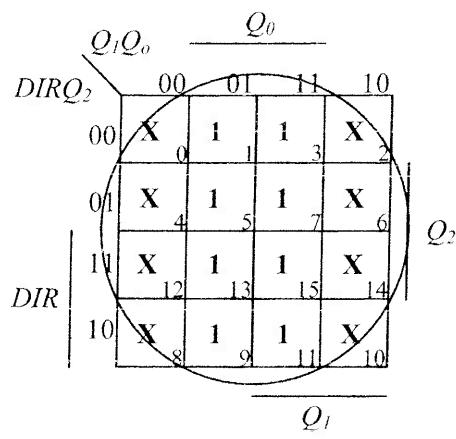
$$K_1 = DIR \cdot \overline{Q_0} + \overline{DIR} \cdot Q_0 + Q_2$$



$$J_0 = DIR \cdot Q_2 + \overline{Q_2} \cdot Q_1 + \overline{DIR} \cdot \overline{Q_1}$$

Implementarea cu bistabile D

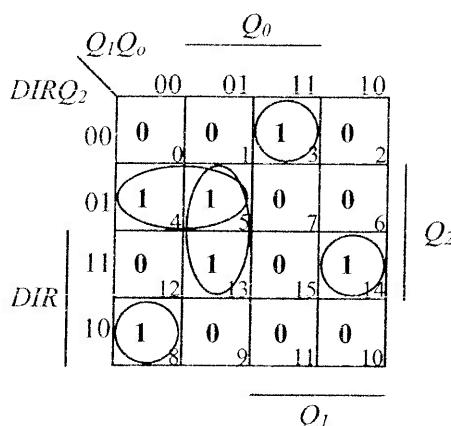
Este necesar să trasăm 3 diagrame Karnaugh. Știm că toate intrările D sunt funcții de intrarea DIR și de starea prezentă ($Q_2Q_1Q_0$); atunci, pe baza tabelului anterior, obținem:



$$K_0 = 1$$

Implementing with D Flip-Flops

We have to draw 3 Karnaugh maps. We know that all D inputs are functions of the DIR input and the current state ($Q_2Q_1Q_0$); then, based on the previous table, we obtain:



$$D_2 = \overline{DIR} \cdot \overline{Q_2} \cdot Q_1 \cdot Q_0 + DIR \cdot Q_2 \cdot \overline{Q_1} + Q_2 \cdot \overline{Q_1} \cdot Q_0 + DIR \cdot Q_2 \cdot Q_1 \cdot \overline{Q_0} + DIR \cdot \overline{Q_2} \cdot Q_1 \cdot \overline{Q_0}$$

		Q_0				
		00	01	11	10	
$DIR Q_2$		00	0	1	0	1
		01	0	1	0	0
DIR		11	1	0	0	0
		10	1	0	1	0

 Q_1

$$D_1 = DIR \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{DIR} \cdot \overline{Q_1} \cdot Q_0 + \overline{DIR} \cdot \overline{Q_2} \cdot Q_1 \cdot \overline{Q_0} + DIR \cdot \overline{Q_2} \cdot Q_1 \cdot Q_0$$

		Q_0				
		00	01	11	10	
$DIR Q_2$		00	1	0	0	1
		01	1	0	0	0
DIR		11	1	0	0	1
		10	0	0	0	1

 Q_1

$$D_0 = DIR \cdot Q_1 \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{DIR} \cdot \overline{Q_2} \cdot \overline{Q_0}$$

Implementarea cu bistabile T

Este necesar să trasăm 3 diagrame Karnaugh. Știm că toate intrările T sunt funcții de intrarea DIR și de starea prezentă ($Q_2Q_1Q_0$); atunci, pe baza tabelului anterior, obținem:

Implementing with T Flip-Flops

We have to draw 3 Karnaugh maps. We know that all T inputs are functions of the DIR input and the current state ($Q_2Q_1Q_0$); then, based on the previous table, we obtain:

		Q_0				Q_1				
		00	01	11	10	00	01	11	10	
		DIR	00	01	11	10	DIR	00	01	11
00	00	0	0	0	1	0	0	1	0	
00	01	0	0	1	0	0	1	1	0	
01	00	1	0	0	0	1	0	0	1	
01	01	1	1	1	1	1	1	1	1	
11	00	0	1	0	0	0	1	0	1	
11	01	0	1	1	0	0	1	1	0	
10	00	1	0	1	0	1	0	1	0	
10	01	0	0	0	0	0	0	0	0	

$$T_0 = \overline{DIR} * Q_0 * Q_1 + Q_0 * Q_1 * Q_0 + \overline{DIR} * Q_0 * Q_1 + DIR * Q_1 * Q_0$$

		Q_0				Q_1				
		00	01	11	10	00	01	11	10	
		DIR	00	01	11	10	DIR	00	01	11
00	00	0	0	0	1	0	0	1	0	
00	01	0	0	1	0	0	1	1	0	
01	00	1	0	0	0	1	0	0	1	
01	01	1	1	1	1	1	1	1	1	
11	00	0	1	0	0	0	1	0	1	
11	01	0	1	1	0	0	1	1	0	
10	00	1	0	1	0	1	0	1	0	
10	01	0	0	0	0	0	0	0	0	

$$T_0 = \overline{DIR} * \overline{Q_0} * Q_1 + Q_0 * \overline{Q_1} + \overline{DIR} * Q_0$$

$$T_1 = \overline{DIR} * Q_0 + Q_1 * \overline{Q_0} + \overline{DIR} * \overline{Q_1}$$

în funcție de tipul bistabilelor impuse pentru implementare și pe baza ecuațiilor obținute, se desenază schema circuitului.

Dacă nu se impune utilizarea unui anumit tip de bistabil, putem realiza o comparație între cele 3

		Q_0				Q_1				
		00	01	11	10	00	01	11	10	
		DIR	00	01	11	10	DIR	00	01	11
00	00	0	0	0	1	0	0	1	0	
00	01	0	0	1	0	0	1	1	0	
01	00	1	0	0	0	1	0	0	1	
01	01	1	1	1	1	1	1	1	1	
11	00	0	1	0	0	0	1	0	1	
11	01	0	1	1	0	0	1	1	0	
10	00	1	0	1	0	1	0	1	0	
10	01	0	0	0	0	0	0	0	0	

$$T_0 = Q_0 + \overline{DIR} * Q_1 + Q_0 * \overline{Q_1} + \overline{DIR} * \overline{Q_1}$$

According to the type of Flip-Flops that are imposed for the implementation and based on the obtained equations, we draw the circuit's scheme.

If the use of a certain type of Flip-Flop is not imposed, we can make a

implementări pentru a o alege pe cea mai avantajoasă într-un anumit context. Se observă că:

- la implementarea cu bistabile JK, tabelul de adevăr se construiește mai anevoieos, este necesar să minimizăm un număr dublu de funcții (vom avea deci de 2 ori mai multe diagrame Karnaugh), dar expresiile finale sunt mai simple (datorită faptului că diagramele Karnaugh conțin multe X-uri);
- la implementarea cu bistabile D, tabelul de adevăr se construiește cel mai ușor (dat fiind că practic coloanele corespunzătoare D -urilor sunt identice cu cele ale stării următoare Q^+), însă expresiile rezultate în urma minimizării sunt mai complicate;
- la implementarea cu bistabile T, suntem oarecum la mijloc: tabelul de adevăr se construiește ceva mai greu decât la implementarea cu bistabile D, dar mai ușor decât la implementarea cu bistabile JK, iar expresiile au o lungime intermediară între cele ale implementărilor cu bistabile JK și D.

Alegerea tipului de bistabil folosit pentru implementare se realizează în funcție de context și de disponibilitatea pieselor pentru aplicația dată.

Aici vom exemplifica realizarea sistemului nostru cu bistabile de tip

comparison between the three implementations in order to choose the most appropriate in a given context. One can notice that:

- when implementing with JK Flip-Flops, the truth table is more difficult to construct, it is necessary to minimize twice the number of functions (so we will have two times more Karnaugh maps), but the final expressions are simpler (because of the fact that the Karnaugh maps contain many Xs);
- when implementing with D Flip-Flops, the truth table is the easiest to construct (since the D columns are practically identical to the next state Q^+ columns), but the expressions that result after the minimization are more complex;
- when implementing with T Flip-Flops, we are somehow in the middle: the truth table is a little bit harder to construct than when implementing with D Flip-Flops, but still easier than when implementing with JK Flip-Flops, and the expressions have an intermediate length between those of the JK and D Flip-Flops implementations.

The choice of the type of Flip-Flop used for the implementation is made according to the context and the components' availability for the given application.

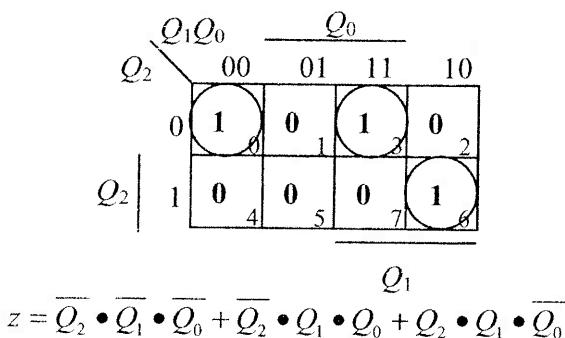
Here we will show the implementation of our system with

T.

Vom prezenta și implementarea ieșirii z . După cum am mai arătat, ieșirea z este o funcție de starea prezentă: $z = g(Q_2, Q_1, Q_0)$. Putem exprima z ca o funcție Booleană astfel:

$$z = \Sigma (0, 3, 6).$$

Din tabelul de adevăr obținem diagrama Karnaugh pentru ieșirea z :



Implementarea funcției de ieșire z este prezentată în figura următoare, încapsulată în schema întregului sistem.

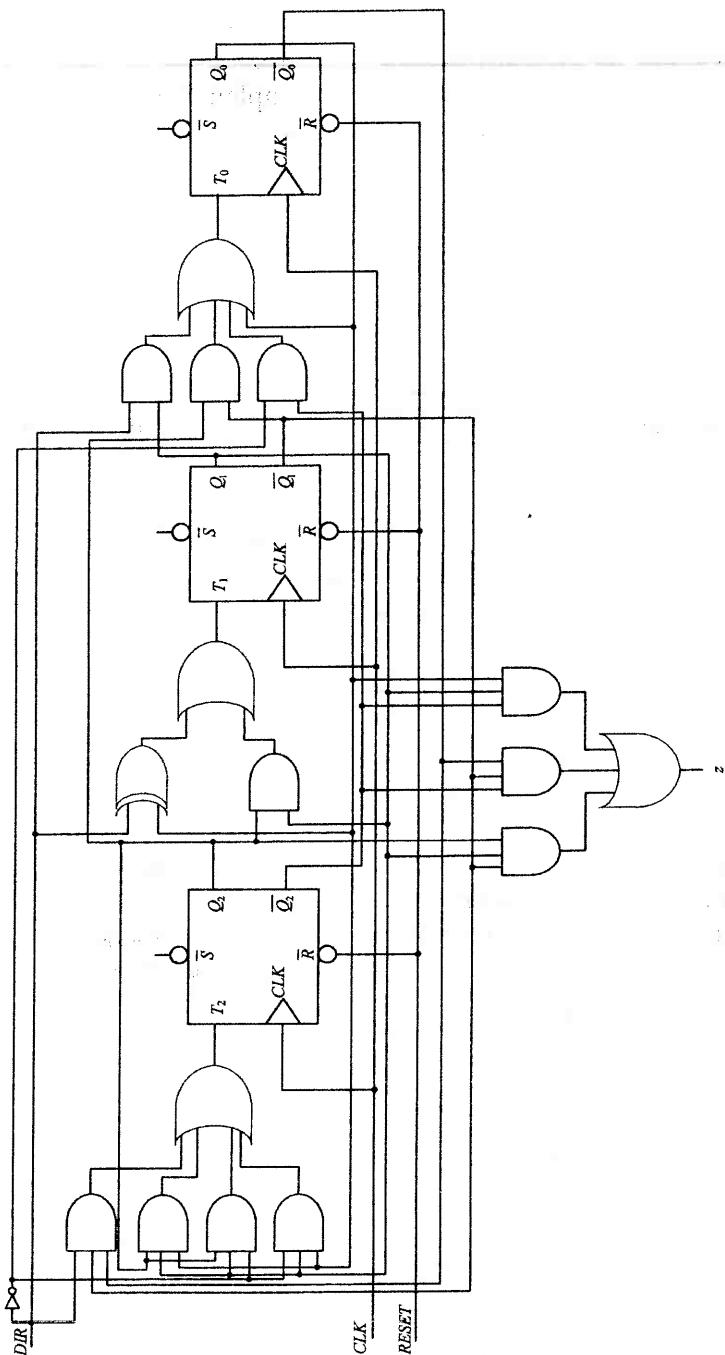
T Flip-Flops.

We will also present the implementation of the output z . As we have explained above, the z output is a function of the current state: $z = g(Q_2, Q_1, Q_0)$. We can express z as a Boolean function as follows:

$$z = \Sigma (0, 3, 6).$$

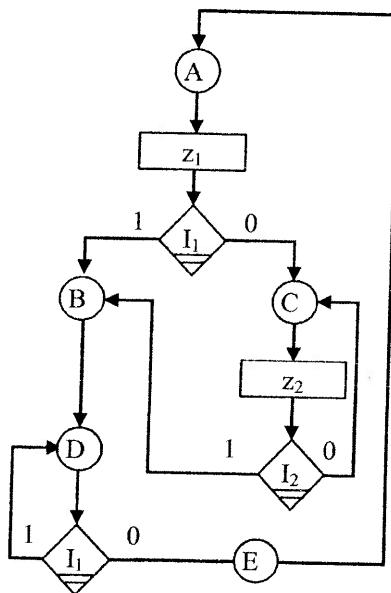
From the truth table we obtain the Karnaugh map for the output z :

The implementation of the output function z is shown in the next figure, embedded in the overall system's scheme.



5. Implementați sistemul sevențial sincron descris în organigrama de mai jos:

5. Implement the synchronous sequential system described in the following state-diagram:



Soluție

Vom implementa sistemul prin următoarele metode:

I. Din punct de vedere al implementării registrului de stări:

a) cu bistabile D, folosind metoda bazată pe diagrama Karnaugh a stării următoare;

b) cu bistabile JK, folosind metoda bazată pe diagrama Karnaugh a stării următoare;

II. Din punct de vedere al implementării generatorului noii stări:

c) cu multiplexoare;

Solution

We will implement the system by the following methods:

I. With respect to the implementation of the states register:

a) with D Flip-Flops, using the method based on the next state Karnaugh map;

b) with JK Flip-Flops, using the method based on the next state Karnaugh map;

II. With respect to the implementation of the next state generator:

c) multiplexers-based;

- d) cu decodificatoare;
e) cu memorii și multiplexoare;

III. Din punct de vedere al implementării registrului de stări și a generatorului noii stări:
f) cu numărător MSI.

a) Întrucât variabilele de intrare I_1 și I_2 sunt asincrone, vom aplica regula de codificare a stărilor: „calea de ieșire a unei variabile de intrare asincrone trebuie să conducă la stări adiacente”.

Aplicând această regulă, găsim perechile de stări care trebuie să fie codificate adjacente (codul lor să difere printr-un singur bit): B și C, D și E.

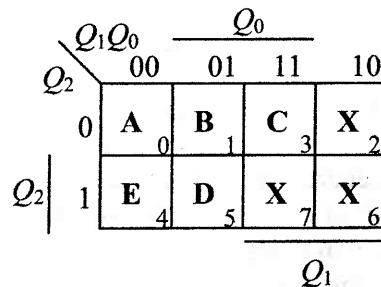
Vom plasa stăriile în diagrama Karnaugh astfel încât să respectăm aceste reguli:

- d) decoders-based;
e) with memories and multiplexers;
III. With respect to the implementation of the states register and of the next state generator:
f) MSI counter-based.

a) Since the input variables I_1 and I_2 are asynchronous, we will apply the rule for states encoding: „the output path from an asynchronous input variable must lead to adjacent states”.

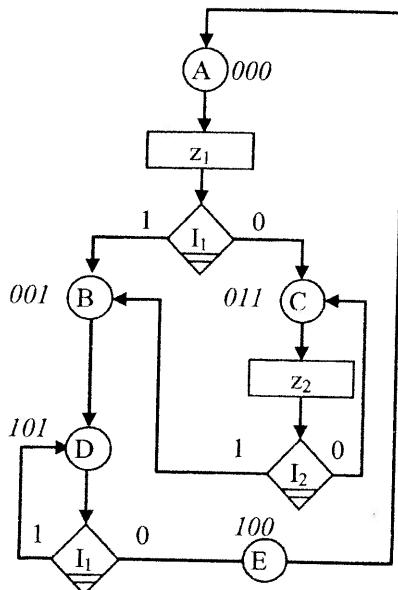
Applying this rule, we find the pairs of states that must be encoded adjacently (their code must differ by only one bit): B and C, D and E.

We will place the states in the Karnaugh map to respect these rules:



Avem atunci următoarea codificare a stărilor: A = 000, B = 001, C = 011, D = 101, E = 100. Celelalte combinații ale variabilelor de stare Q_2 , Q_1 și Q_0 sunt neutilizate, deci le vom considera indiferente (X).

Then we have the following states encoding: A = 000, B = 001, C = 011, D = 101, E = 100. The other combinations of the state variables Q_2 , Q_1 and Q_0 are unused, so we will consider them don't care (X).



Construim acum diagrama Karnaugh a stării următoare. În celula corespunzătoare stării A, vom scrie codul stării care urmează după starea A.

Observăm că:

- dacă $I_1 = 0$, starea următoare este C (011)
- dacă $I_1 = 1$, starea următoare este B (001)

În sinteză, putem afirma că expresia stării următoare stării A este $0\bar{I}_11$.

În mod analog se completează diagrama Karnaugh a stării următoare pentru toate stările circuitului:

We build now the next state Karnaugh map. In the cell that corresponds to state A, we will write the code of the state following after state A.

We notice that:

- if $I_1 = 0$, the next state is C (011)
- if $I_1 = 1$, the next state is B (001)

In conclusion, we can say that the expression of the state following after state A is $0\bar{I}_11$.

Analogously we fill the next state Karnaugh map for all the states of the circuit:

		Q_0	$Q_2^+ Q_1^+ Q_0^+$		
		00	01	11	10
Q_2	0	$0I_1$	101	$0I_2$	XXX
	1	000	$10I_1$	XXX	XXX
		4	5	7	6
					Q_1

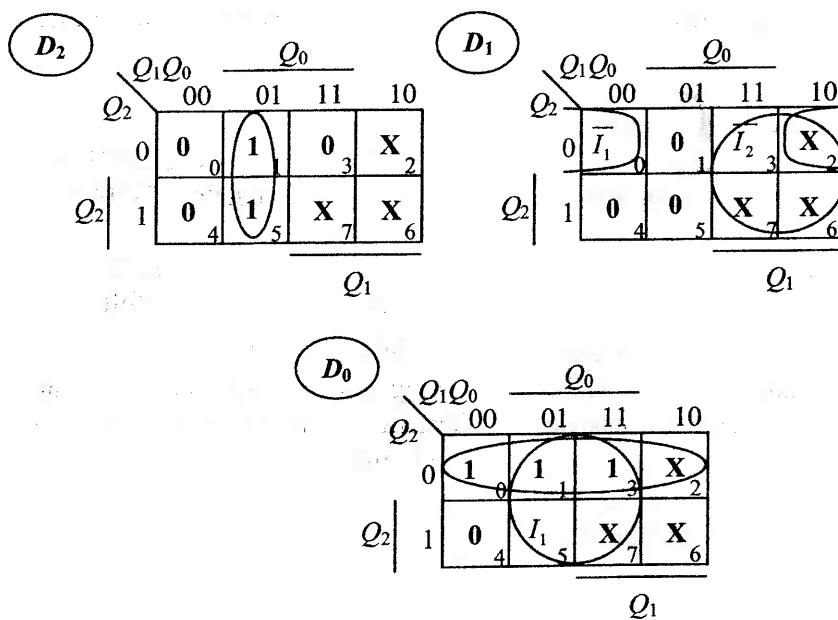
Diagrama Karnaugh a stării următoare (*next state Karnaugh map*)

Diagrama Karnaugh a stării următoare conține acum Q_2^+, Q_1^+ și Q_0^+ , care știm că, în cazul implementării cu bistabile de tip D, sunt egale cu D_2, D_1 și D_0 .

Ea se va „sparge” în 3 diagrame Karnaugh, câte una pentru fiecare intrare de date a bistabilelor:

The next state Karnaugh map contains now Q_2^+, Q_1^+ and Q_0^+ , which we know that, in case of the D Flip-Flops implementation, are equal to D_2, D_1 and D_0 .

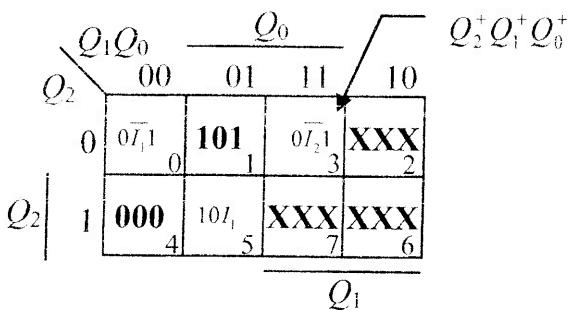
It will be „split” into three Karnaugh maps, one for each Flip-Flop data input:



$$D_2 = \overline{Q_1} \bullet Q_0; D_1 = \overline{I_1} \bullet Q_2 \bullet Q_0 + \overline{I_2} \bullet Q_1; D_0 = \overline{Q_2} + I_1 \bullet Q_0$$

Schema circuitului se construiește pe baza ecuațiilor de mai sus, în mod analog problemei rezolvate 4.

b) Pornim de la diagrama Karnaugh a stării următoare:



Pentru a găsi ecuațiile intrărilor J și K ale bistabilelor JK, trebuie să ținem cont și de coordonatele celulei.

De exemplu, starea A este codificată 000, iar starea următoare stării A este $0\overline{I}_11$. Așadar

- la nivelul bistabilului Q_2 , trebuie să se treacă din 0 în 0. Pentru aceasta, $J_2K_2 = 0X$ (conform tabelului de excitare al bistabilului JK).

- la nivelul bistabilului Q_1 , trebuie să se treacă din 0 în \overline{I}_1 . Trebuie să ne întrebăm „ce trebuie să avem pe J_1K_1 pentru a obține această tranziție?” Știm, din tabelul de excitare al bistabilului JK, că pentru a trece din 0 în 0, $J_1K_1 = 0X$, iar

The circuit's scheme is built based on the equations above, analogously to the solved problem 4.

b) We start from the next state Karnaugh map.

To find the equations of the J and K inputs of the JK Flip-Flops, we must take into account the cell's coordinates, too.

For instance, state A is encoded 000, and the state following after state A is $0\overline{I}_11$. So:

- at the level of the Q_2 Flip-Flop, the transition must be from 0 to 0. To achieve this, $J_2K_2 = 0X$ (according to the JK Flip-Flop's excitation table).

- at the level of the Q_1 Flip-Flop, the transition must be from 0 to \overline{I}_1 . We must ask ourselves „what should we put on J_1K_1 to obtain this transition?” We know, from the JK Flip-Flop's excitation table, that in order to pass from 0 to 0,

pentru a trece din 0 în 1, $J_1K_1 = 1X$. Putem sintetiza spunând că pentru a trece din 0 în α (unde α este o variabilă Booleană), $J_1K_1 = \alpha X$. Atunci, în cazul nostru, $J_1K_1 = \overline{I_1}X$. În mod analog, se determină că pentru a trece din 1 în α (unde α este o variabilă Booleană), $JK = X\overline{\alpha}$.

- la nivelul bistabilului Q_0 , trebuie să se treacă din 0 în 1. Pentru aceasta, $J_0K_0 = 1X$ (conform tabelului de excitație al bistabilului JK).

Similar, se determină următoarele diagrame Karnaugh pentru J_2, J_1, J_0, K_2, K_1 și K_0 :

		Q_0			
		00	01	11	10
J_2	0	0	1	0	X ₂
	1	X ₄	X ₅	X ₇	X ₆
		Q_2	Q_1		

$$J_2 = \overline{Q_1} \cdot Q_0$$

		Q_0			
		00	01	11	10
J_1	0	0	0	X ₃	X ₂
	1	0	0	X ₇	X ₆
		Q_2	Q_1		

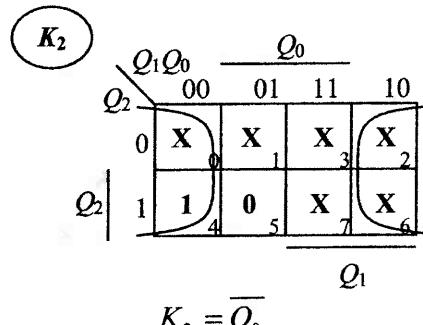
$$J_1 = 0$$

$J_1K_1 = 0X$, and for passing from 0 to 1, $J_1K_1 = 1X$. We can conclude by saying that in order to pass from 0 to α (where α is a Boolean variable), $J_1K_1 = \alpha X$. Then, in our case, $J_1K_1 = \overline{I_1}X$.

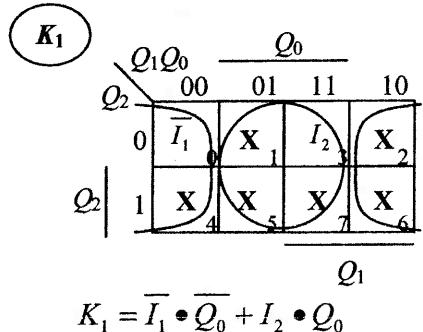
Analogously, we determine that in order to pass from 1 to α (where α is a Boolean variable), $JK = X\overline{\alpha}$.

- at the level of the Q_0 Flip-Flop, the transition must be from 0 to 1. To achieve this, $J_0K_0 = 1X$ (according to the JK Flip-Flop's excitation table).

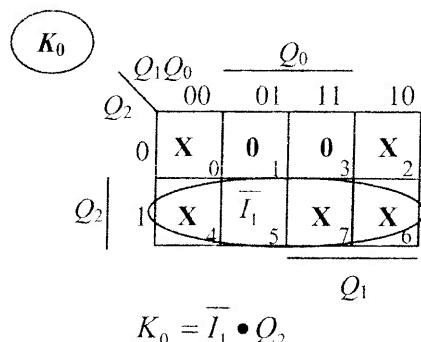
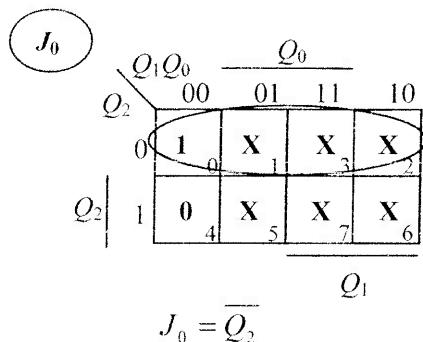
Similarly, we determine the following Karnaugh maps for J_2, J_1, J_0, K_2, K_1 and K_0 :



$$J_2 = \overline{Q_0}$$



$$J_1 = \overline{I_1} \cdot \overline{Q_0} + I_2 \cdot Q_0$$



Schema circuitului se construiește pe baza ecuațiilor de mai sus, în mod analog problemei rezolvate 4.

- c) Implementarea bazată pe multiplexoare se bazează tot pe diagrama Karnaugh a stării următoare.

Implementarea CLC-urilor care determină intrarea bistabilelor D se face cu ajutorul multiplexoarelor (folosim diagramele Karnaugh pentru D_2 , D_1 și D_0 de la punctul a)).

Stim că toate intrările de date D sunt funcții de variabilele de stare Q_2 , Q_1 și Q_0 și au expresiile:

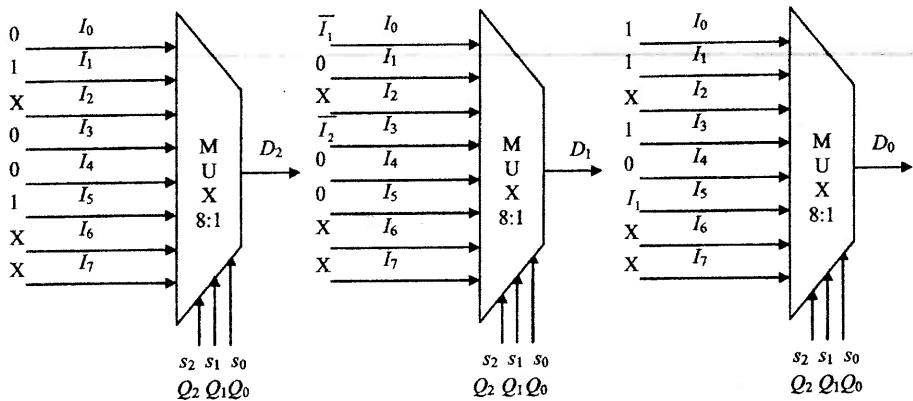
$$D_2 = \overline{Q_1} \bullet Q_0; D_1 = \overline{I_1} \bullet \overline{Q_2} \bullet \overline{Q_0} + \overline{I_2} \bullet Q_1; D_0 = \overline{Q_2} + I_1 \bullet Q_0$$

The circuit's scheme is built based on the equations above, analogously to the solved problem 4.

- c) The multiplexers-based implementation relies on the next state Karnaugh maps too.

The implementation of the CLCs that determine the input of each D Flip-Flop is done with multiplexers (we use the Karnaugh maps for D_2 , D_1 and D_0 from point a)).

We know that all the data inputs D are functions of the state variables Q_2 , Q_1 and Q_0 and they have the following expressions:



d) Implementarea bazată pe decodificatoare se bazează tot pe diagraama Karnaugh a stării următoare.

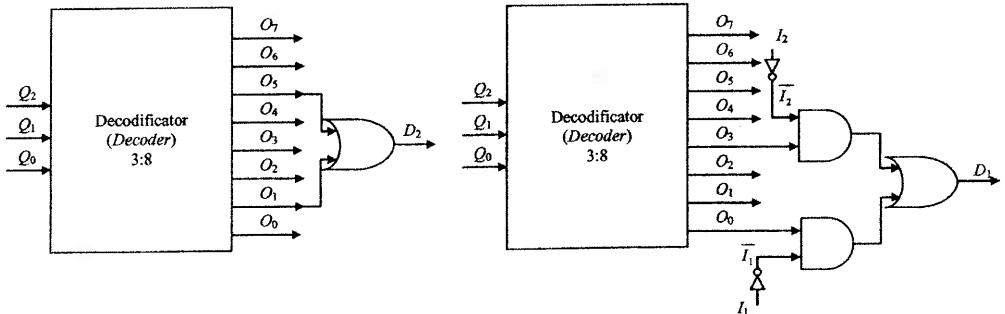
Implementarea CLC-urilor care determină intrarea bistabilelor D se face cu ajutorul decodificatoarelor (folosim diagramele Karnaugh pentru D_2, D_1 și D_0 de la punctul a)). Știm că toate intrările de date D sunt funcții de variabilele de stare Q_2, Q_1 și Q_0 și au expresiile:

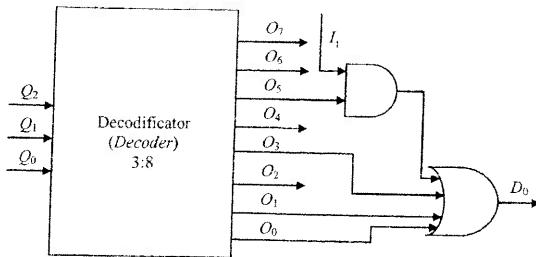
d) The decoders-based implementation relies on the next state Karnaugh map too.

The implementation of the CLCs that determine the input of each D Flip-Flop is done with decoders (we use the Karnaugh maps for D_2, D_1 and D_0 from point a)).

We know that all the data inputs D are functions of the state variables Q_2, Q_1 and Q_0 and they have the following expressions:

$$D_2 = \overline{Q}_1 \cdot Q_0; D_1 = \overline{I}_1 \cdot \overline{Q}_2 \cdot \overline{Q}_0 + \overline{I}_2 \cdot Q_1; D_0 = \overline{Q}_2 + I_1 \cdot Q_0$$



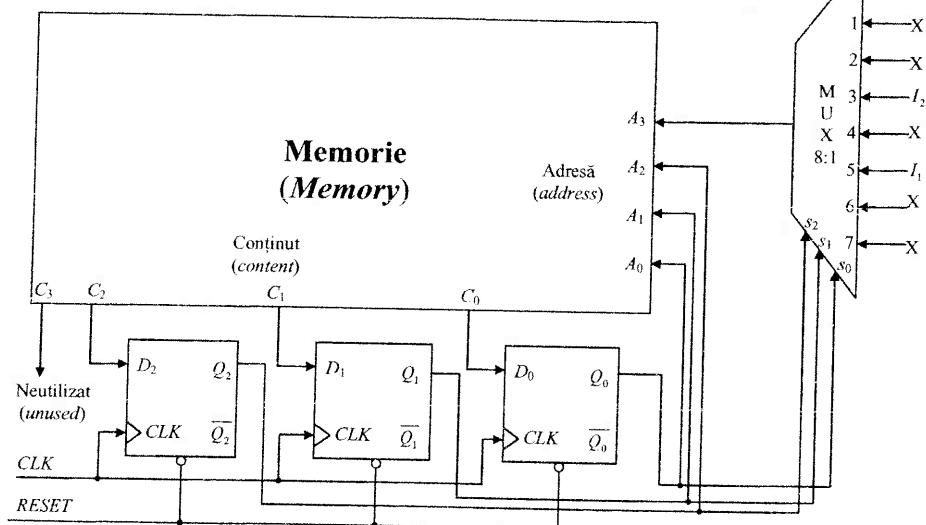


e) La implementarea cu memorie și multiplexor, nu mai este necesară codificarea stărilor după o anumită regulă, deoarece accesul la memorie se face în mod egal.

Schema fundamentală este cea următoare:

e) For the memory and multiplexer-based implementation, it is no longer necessary to encode the states by a certain rule, because access to the memory is done equally.

The fundamental scheme is the following one:



Cele trei bistabile formează *registrul de stări* al sistemului (aici sunt stocate stările). În momentul

The three Flip-Flops constitute the system's states register (here are stored the states). When the current

când starea actuală este una din care se ia o decizie (A, C sau D), variabila de intrare testată în starea respectivă participă la construirea adresei stării următoare (stării din care se va continua execuția organigramei) pe linia A_3 .

În acest sistem este foarte importantă și harta memoriei, care se construiește pe baza schemei bloc și a organigramei. Remarcăm că nu se folosește bitul C_3 al conținutului (blocurile de memorie disponibile pe piață au conținutul pe un număr de biți egal cu o putere a lui 2). Valoarea indiferentă (X) se alege de obicei 0.

state is a state in which a decision is taken (A, C or D), the input variable tested in that state participates to the construction of the next state (the state from which the state diagram's execution will continue) on the A_3 line.

In this system, the memory map is also very important. This is built based on the block schematic and the state diagram. Notice that we don't use the C_3 bit from the content (the memory blocks that are available on the market have a content width equal to a power of 2). The don't care value (X) is usually chosen to be 0.

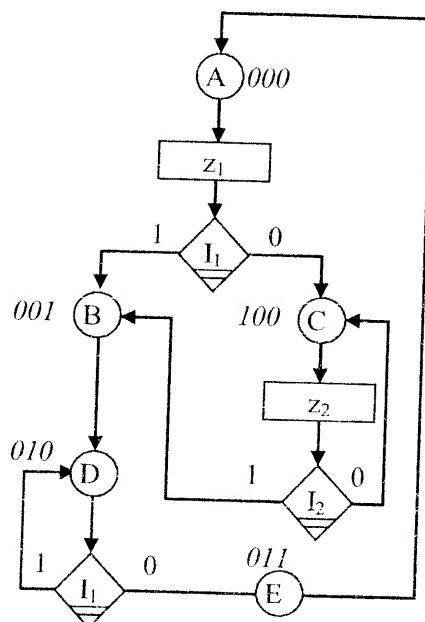
Intrare + Starea curentă (Input + current state)				Starea următoare (next state)				Ieșiri (outputs)	
A_3	$A_2 = Q_2$	$A_1 = Q_1$	$A_0 = Q_0$	C_3	$C_2 = Q_2^+$	$C_1 = Q_1^+$	$C_0 = Q_0^+$	z_1	z_2
0	0	0	0	X	0	1	1	1	0
1	0	0	0	X	0	0	1	1	0
0	0	0	1	X	1	0	1	0	0
1	0	0	1	X	1	0	1	0	0
0	0	1	0	X	X	X	X	X	X
1	0	1	0	X	X	X	X	X	X
0	0	1	1	X	0	1	1	0	1
1	0	1	1	X	0	0	1	0	1
0	1	0	0	X	0	0	0	0	0
1	1	0	0	X	0	0	0	0	0
0	1	0	1	X	1	0	0	0	0
1	1	0	1	X	1	0	1	0	0
0	1	1	0	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

f) În cadrul acestei metode, ideea este de a codifica stările în secvență binară normală și a implementa registrul de stări printr-un numărător MSI, care face în mod firesc tranziția de la o stare N la starea $N + 1$. Această metodă este adecvată mai ales implementării sistemelor care prezintă o formă liniară a organigramei (cu relativ puține decizii), însă cu ajutorul ei putem implementa orice sistem.

Așadar, vom re-codifica stările organigramei conform acestui principiu:

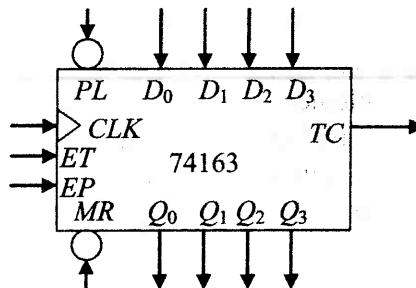
f) In this method, the idea is to encode the states in a normal binary sequence and to implement the states register by an MSI counter, which naturally realizes the transition from a state N to the state $N + 1$. This method is appropriate especially for the implementation of systems that have a linear profile of their states diagram (with relatively few decisions), but using it we can implement any system.

So, we will have to re-encode the states from the states diagram according to this principle:



Vom folosi un numărător MSI cu încărcare paralelă sincronă, similar cu circuitul integrat 74163:

We will use an MSI counter with synchronous parallel load, similar to the 74163 chip:



În mod firesc, numărătorul își incrementează conținutul la fiecare impuls de tact, trecând astfel dintr-o stare a organigramei în starea următoare. În anumite cazuri însă, este necesar să folosim intrarea de încărcare paralelă și să încărcăm codul altrei stări, în funcție de decizia care se ia. Situațiile când trebuie activată intrarea de încărcare paralelă (*PL*) sunt următoarele:

The counter naturally increments its content on each clock impulse, thus passing from a given state of the state diagram to the next one. But in some cases, we must use the parallel load input and load another state's code, according to the decision that is taken. The situations when we must activate the parallel load (*PL*) input are the following ones:

Starea sistemului (system's state)	Intrarea (input)	Codul stării de încărcat pe D_{3-0} (State code to be loaded on D_{3-0})
A (000)	$I_1 = 0$	100 (C)
C (100)	$I_2 = 0$	100 (C)
C (100)	$I_2 = 1$	001 (B)
D (010)	$I_1 = 1$	010 (D)
E (011)	X	000 (A)

Deci, dacă (starea este A ȘI $I_1 = 0$) SAU (starea este C) SAU (starea este D ȘI $I_1 = 1$) SAU (starea este E), vom activa intrarea *PL* a numărătorului. Pentru aceasta folosim un decodificator, care ne va indica starea curentă, și portii logice. Pentru a determina ce coduri trebuie

So, if (the state is A AND $I_1 = 0$) OR (the state is C) OR (the state is D AND $I_1 = 1$) OR (the state is E), we will activate the counter's *PL* input. In order to do that we use a decoder, that will indicate us the current state, and logic gates. To determine which codes must be

încărcate simultan cu activarea PL , vom folosi un multiplexor. Observăm că există 4 posibilități diferite: încărcarea codurilor stărilor C, B, D sau A; prin urmare, avem nevoie de un multiplexor 4:1 cu lățimea căii de date de 3 biți.

Vom plasa pe intrările de date ale acestui multiplexor cele 4 coduri distincte care trebuie încărcate, astfel: pe I_0 plasăm codul lui C (100), pe I_1 plasăm codul lui B (001), pe I_2 plasăm codul lui D (010), iar pe I_3 plasăm codul lui A (000). Deci:

- Când selecțiile multiplexorului vor fi $s_1s_0=00$, trebuie încărcată în numărător starea C, atunci când sistemul este în starea A și $I_1 = 0$ sau când sistemul este în starea C și $I_2 = 0$.
- Când selecțiile multiplexorului vor fi $s_1s_0=01$, trebuie încărcată în numărător starea B, atunci când sistemul este în starea C și $I_2 = 1$.
- Când selecțiile multiplexorului vor fi $s_1s_0=10$, trebuie încărcată în numărător starea D, atunci când sistemul este în starea D și $I_1 = 1$.
- Când selecțiile multiplexorului vor fi $s_1s_0=11$, trebuie încărcată în numărător starea A, atunci când sistemul este în starea E.

Putem sintetiza situația în tabelul următor:

loaded simultaneously with PL 's activation, we will use a multiplexer. We notice there are 4 different possibilities: loading the codes that correspond to states C, B, D or A; thus, we need a 4:1 multiplexer with a data path width of 3 bits.

We will place on this multiplexer's data inputs the 4 distinct codes that are to be loaded, as follows: on I_0 we place C's code (100), on I_1 we place B's code (001), on I_2 we place D's code (010), and on I_3 we place A's code (000). So:

- When the multiplexer's selections will be $s_1s_0=00$, we must load into the counter the state C, when the system is in state A and $I_1 = 0$ or when the system is in state C and $I_2 = 0$.
- When the multiplexer's selections will be $s_1s_0=01$, we must load into the counter the state B, when the system is in state C and $I_2 = 1$.
- When the multiplexer's selections will be $s_1s_0=10$, we must load into the counter the state D, when the system is in state D and $I_1 = 1$.
- When the multiplexer's selections will be $s_1s_0=11$, we must load into the counter the state A, when the system is in state E.

We can summarize the situation in the table below:

Codul stării de încărcat în numărător (Code of the state to be loaded into the counter)	Condiția (Condition)	Selecțiile multiplexorului (multiplexer's selections)
		$s_1 s_0$
C (100)	$A \bullet \overline{I_1} + C \bullet \overline{I_2}$	00
B (001)	$C \bullet I_2$	01
D (010)	$D \bullet I_1$	10
A (000)	E	11

Din acest tabel deducem că s_1 este 1 când starea numărătorului este D și intrarea I_1 este 1, SAU când starea numărătorului este E. Exprimând acest lucru printr-o ecuație, se obține:

$$s_1 = D \bullet I_1 + E.$$

Analog, deducem că s_0 este 1 când starea numărătorului este C și intrarea I_2 este 1 SAU când starea numărătorului este E.

Exprimând acest lucru printr-o ecuație, se obține:

$$s_0 = C \bullet I_2 + E.$$

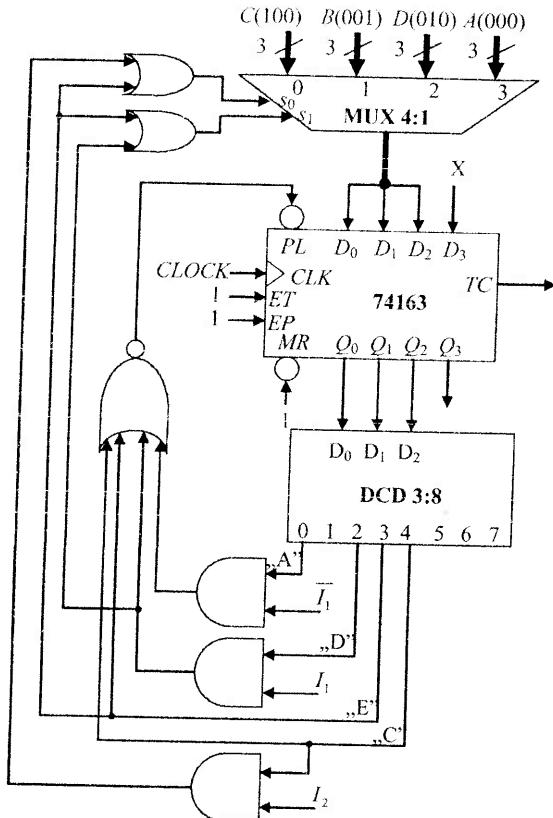
Schema finală a sistemului este prezentată în figura următoare:

From this table we deduce that s_1 is 1 when the counter's state is D and the input I_1 is 1, OR when the counter's state is E. If we express this by an equation, we obtain:

Analogously, we deduce that s_0 is 1 when the counter's state is C and the input I_2 is 1 OR when the counter's state is E.

If we express this by an equation, we obtain:

The system's final scheme is presented in the next figure:



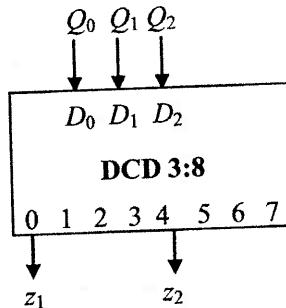
Am lăsat pentru final implementarea ieșirilor acestui sistem. Dat fiind că ieșirile depend întotdeauna numai de starea curentă (aspect care se deduce din organigramă), pentru toate metodele prezentate anterior se poate folosi implementarea propusă aici.

Se observă că ieșirea z_1 se generează numai când sistemul este în starea A, iar ieșirea z_2 se generează numai când sistemul este în starea C. Păstrăm ultima codificare a stărilor

We left for the end the implementation of this system's outputs. Since the outputs always depend only on the current state (this aspect can be deduced from the states diagram), for all the methods presented above we can use the implementation proposed hereby.

We observe that the output z_1 is generated only when the system is in state A, and the output z_2 is generated only when the system is in state C. We keep the last states

(cea de la punctul e)). Atunci, pentru implementarea ieșirilor vom folosi un decodificator:



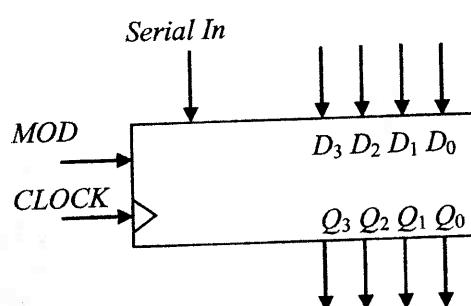
encoding (the one from point e)). Then, for implementing the outputs we will use a decoder:

6. Proiectați cu bistabili D sistemul numeric din figură, cu următoarele proprietăți:

- a) pentru MOD = 0, pe următorul ceas se realizează deplasarea datelor
Serial_In $\rightarrow Q_3 \rightarrow Q_2 \rightarrow Q_1 \rightarrow Q_0$.
- b) pentru MOD = 1, se încarcă pe următorul tact codul de pe liniile de date D₃-D₂-D₁-D₀.

6. Design with D Flip-Flops the digital system in the figure below, having the following properties:

- a) for MOD = 0, on the next clock impulse data are shifted
Serial_In $\rightarrow Q_3 \rightarrow Q_2 \rightarrow Q_1 \rightarrow Q_0$.
- b) for MOD = 1, the code present on the data inputs D₃-D₂-D₁-D₀ is loaded on the next clock impulse.



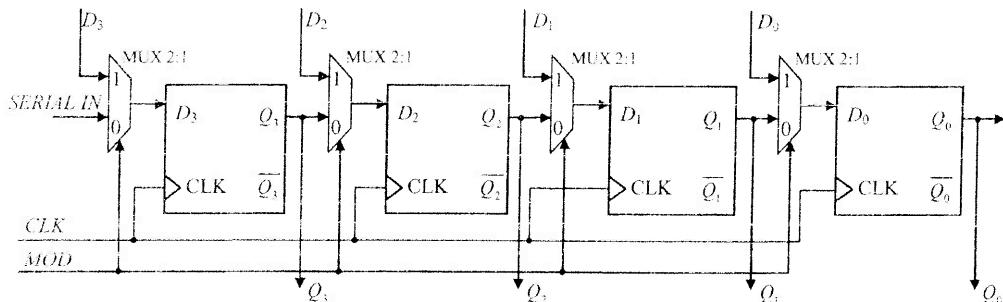
Soluție

Acest circuit este un registru de deplasare clasic cu funcția

Solution

This circuit is a classical shift register which has in addition the

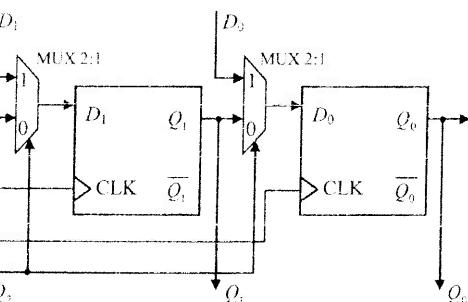
suplimentară de încărcare paralelă sincronă. Schema sa internă se bazează pe bistabili D și multiplexoare 2:1; ea este prezentată în figura de mai jos:



7. Proiectați cu bistabili D sistemul numeric din figură, cu următoarele proprietăți:

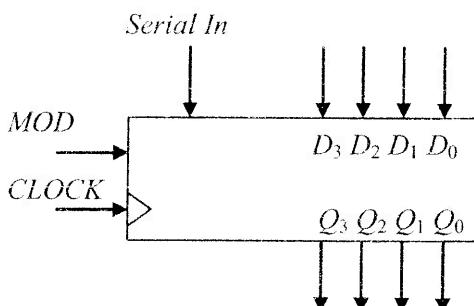
- pentru $MOD = 0$, pe următorul ceas se realizează deplasarea datelor $Serial\ In \rightarrow Q_3 \rightarrow Q_2 \rightarrow Q_1 \rightarrow Q_0$.
- pentru $MOD = 1$, se încarcă imediat în paralel codul de pe liniile de date $D_3-D_2-D_1-D_0$.

synchronous parallel load function. Its internal scheme is based on D Flip-Flops and 2:1 multiplexers; it is shown in the figure below:



7. Design with D Flip-Flops the digital system in the figure below, having the following properties:

- for $MOD = 0$, on the next clock impulse data are shifted $Serial_In \rightarrow Q_3 \rightarrow Q_2 \rightarrow Q_1 \rightarrow Q_0$.
- for $MOD = 1$, the code present on the data inputs $D_3-D_2-D_1-D_0$ is loaded immediately, in parallel.



Soluție

Și acest circuit este tot un registru

Solution

This circuit too is a classical shift

de deplasare clasic, însă aici funcția suplimentară de încărcare paralelă este *asincronă*. Întrucât intrările asincrone sunt întotdeauna prioritare față de cele sincrone, structura sa va fi cea a unui registru de deplasare la care se folosesc și intrările asincrone *Set* și *Reset* pentru realizarea operației de încărcare paralelă.

Pentru a determina schema sa internă, vom face o evaluare a situației la nivelul bistabilului de indice i :

register, but here the parallel load function is *asynchronous*. Since asynchronous inputs always have a greater priority than the synchronous ones, its structure will be of a shift register at which we will use the asynchronous *Set* and *Reset* inputs to perform the parallel load operation.

In order to determine its internal scheme, we will make an evaluation of the situation at the level of the i^{th} Flip-Flop:

Intrarea asincronă (asynchronous input)	Condiție (condition)
$\overline{Set}_i = 0$ (activ)	$(MOD = 1) \bullet (D_i = 1)$
$\overline{Set}_i = 1$ (inactiv)	$(MOD = 0) + [(MOD = 1) \bullet (D_i = 0)]$
$\overline{Reset}_i = 0$ (activ)	$(MOD = 1) \bullet (D_i = 0)$
$\overline{Reset}_i = 1$ (inactiv)	$(MOD = 0) + [(MOD = 1) \bullet (D_i = 1)]$

Pe baza acestei analize, ecuația intrării *Reset* este $\overline{MOD} + MOD \bullet D_i$. Aplicând proprietățile algebrei Booleene, ecuația se simplifică la forma următoare:

$$\overline{Reset} = \overline{MOD} + MOD \bullet D_i = \overline{MOD} \bullet (D_i + 1) + MOD \bullet D_i = D_i \bullet (\overline{MOD} + MOD) + \overline{MOD} = \overline{MOD} + D_i$$

Deci $\overline{Reset} = \overline{MOD} + D_i$. Procedând în mod analog, găsim ecuația: $Set = \overline{MOD} + \overline{D}_i$.

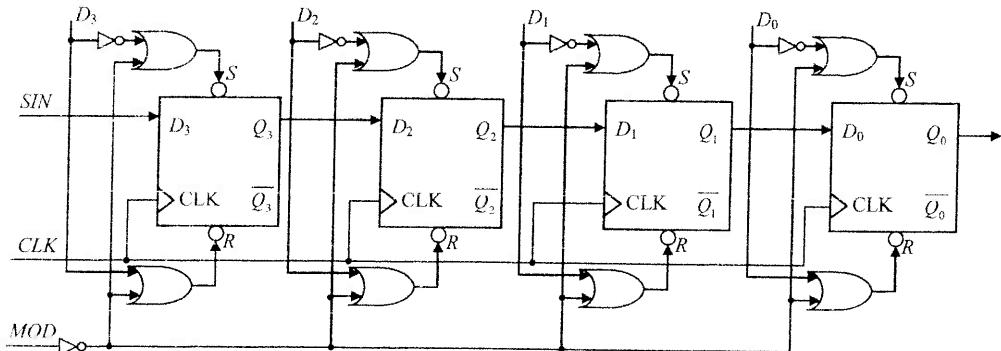
Schema internă se bazează pe

Based on this analysis, the equation of the *Reset* input is $\overline{MOD} + MOD \bullet D_i$. By applying Boolean algebra's properties, the equation is simplified to the following form:

So $\overline{Reset} = \overline{MOD} + D_i$. By an analogous procedure, we find the equation: $Set = \overline{MOD} + \overline{D}_i$.

The internal scheme is based on D

bistabile D și porți; ea este prezentată în figura următoare:



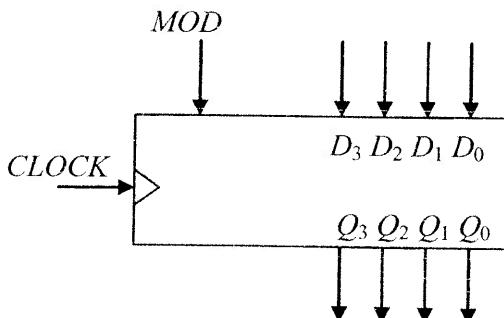
8. Circuitul integrat din figură numără în bucla $0 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 1$, dacă $MOD = 0$, și încarcă în paralel, pe următorul tact după activarea lui MOD ($MOD = 1$), datele de pe liniile $D_3 - D_0$. Se presupune că pe liniile $D_3 - D_0$ nu va fi niciodată un cod diferit de $0, 8, 12, 14, 15, 7, 3$ sau 1 .

Să se proiecteze schema internă a acestui circuit, cu bistabile D.

Flip-Flops and gates; it is shown in the next figure:

8. The circuit in the figure below counts in the loop $0 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 1$, if $MOD = 0$, and loads in parallel, on the following clock impulse after the activation of MOD ($MOD = 1$), data from lines $D_3 - D_0$. Assume that on the $D_3 - D_0$ lines there will never be a code different from $0, 8, 12, 14, 15, 7, 3$ or 1 .

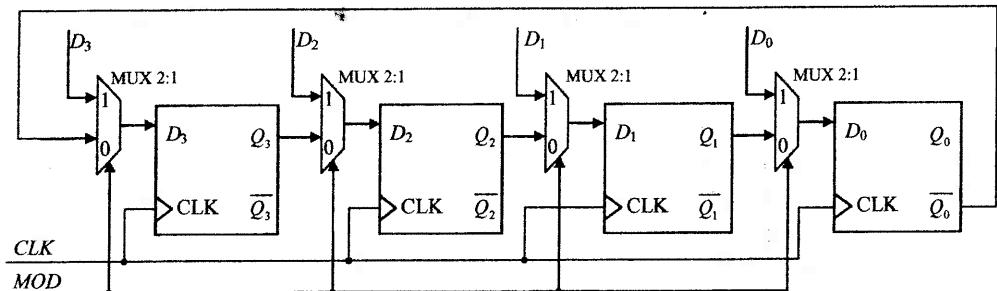
Design the internal schematic of this circuit, with D Flip-Flops.



Soluție

Observăm că acest circuit este un hibrid între un numărător Moebius (deducem acest lucru examinând bucla sa de numărare) și un registru de date.

Schema sa internă se bazează pe bistabili D și multiplexoare 2:1; ea a fost dedusă făcând un raționament analog celui de la problema rezolvată 6 și este prezentată în figura următoare:



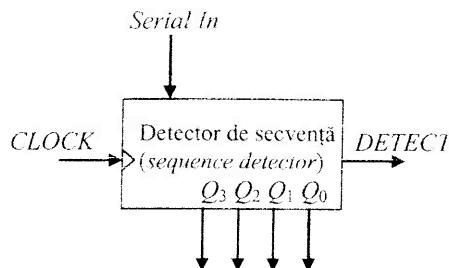
9. Proiectați cu circuite MSI și / sau SSI un sistem logic secvențial care citește date de pe o linie serială și detectează apariția secvenței „0110” din sirul de intrare, afișând și numărul de asemenea secvențe detectate.

Solution

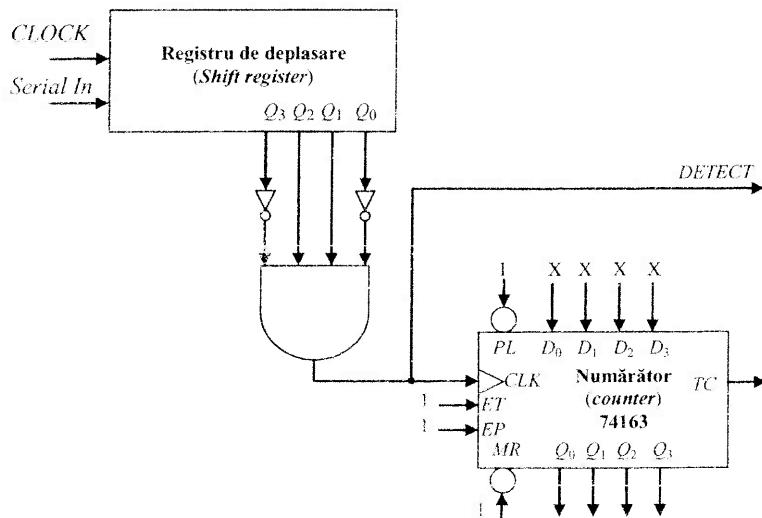
We notice that this circuit is a hybrid between a Moebius counter (we deduce that by examining its counting loop) and a data register.

Its internal scheme is based on D Flip-Flops and 2:1 multiplexers; it has been deduced by reasoning analogously to the inference process we did at the solved problem 6. It is presented in the next figure:

9. Design using MSI and / or SSI components a sequential logic system that reads data from a serial input line and detects the appearance of the “0110” sequence in the input stream, also displaying the number of such detected sequences.

Solutie

Această problemă se rezolvă relativ ușor dacă știm să alegem componentele cele mai potrivite. Vom capta biți care sosesc pe linia serială într-un registru de deplasare, care va stoca ultimii 4 biți sosiți. Apoi, vom detecta şablonul „0110” cu ajutorul unei porți SI cu 4 intrări, iar fiecare detecție reușită va furniza un impuls de ceas unui numărător care va contoriza secvențele detectate. Schema este prezentată în figura următoare:

Solution

This problem is relatively easy to solve if we know how to choose the most appropriate components. We will capture the bits arriving on the serial input line in a shift register, which will store the last 4 bits received. Then we will detect the template “0110” using a 4-bit AND gate, and each successful match will provide a clock impulse to a counter that will count the number of detected sequences. The scheme is presented in the next figure:

10. O reclamă luminoasă funcționează astfel:

- a) reclama rămâne aprinsă o perioadă de timp T_A ;
- b) reclama "pâlpâie" (ledul este aprins și, alternativ, stins) o perioadă $T_B = T_A$ (deci ledul clipește de 5 ori în 5 secunde);
- c) procesul se reia de la a).

Proiectați (folosind circuite MSI și / sau SSI) sistemul numeric care comandă funcționarea reclamei luminoase.

Soluție

Prima problemă o constituie împărțirea perioadei de funcționare în două intervale temporale egale. Acest lucru se realizează cu ajutorul unui numărător zecimal care primește un semnal de tact cu perioada de 1 secundă. Presupunând că acest semnal are factorul de umplere de 50%, remarcăm că el este foarte potrivit pentru a implementa regimul de funcționare „clipire”.

În schema următoare, ieșirile decodificatorului ne indică în care jumătate a intervalului temporal ne aflăm, iar blocul de logica combinațională implementează cele două regimuri de funcționare:

10. An advertising panel has the following behavior:

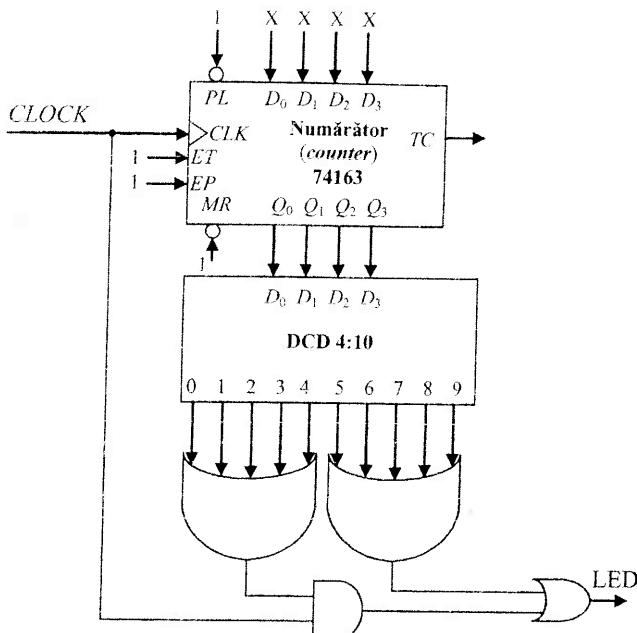
- a) The LEDs stay ON for a period of time $T_A = 5$ seconds;
- b) The LEDs "blink" (the LEDs are ON and, alternatively, OFF) a period of time $T_B = T_A$ (so, the LEDs blink 5 times in 5 seconds);
- c) goto a).

Design (using MSI and / or SSI components) the digital system that commands the functioning of this advertising panel.

Solution

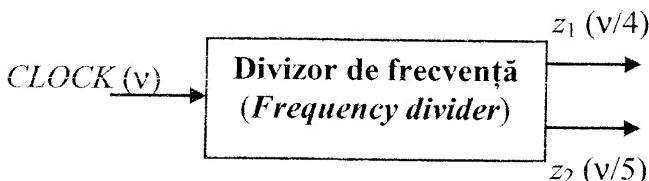
The first problem is to divide the functioning period in two equal temporal intervals. This is done with a decimal counter that receives a clock signal having a 1-second cycle. Assuming that this signal has a duty cycle of 50%, we notice that it is very well suited for implementing the "blinking" working mode.

In the next scheme, the decoder's outputs indicate in which half of the temporal interval we are currently, and the combinational logic block implements the two working regimes:



11. Proiectați un dispozitiv divizor de frecvență conform schemei:

11. Design a frequency divider with the following structure:



Soluție

Pentru realizarea divizoarelor de frecvență se folosesc numărătoare. Remarcăm că se cere obținerea unor semnale periodice de frecvențe $v/4$ și respectiv $v/5$, dar nu se impune ca factorul lor de umplere să fie de 50%.

Solution

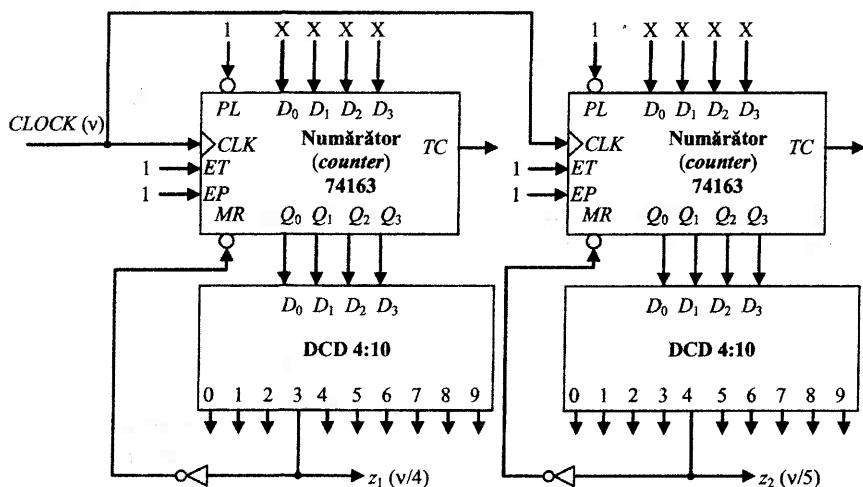
To make frequency dividers, we will use counters. Notice that it is required to obtain periodical signals with frequencies of $v/4$ and $v/5$, respectively, but it is not required that their duty cycle is of 50%. So we need a counter and a

Avem deci nevoie de un numărător și de un bloc de logică combinațională care să genereze câte un impuls la fiecare 5 perioade de tact. Primul numărător este modulo 4 (în starea 3 se activează intrarea sa sincronă de *MR*, ceea ce-l va face să revină în starea 0 pe următorul impuls de tact). Al doilea numărător este modulo 5 (în starea 4 se activează intrarea sa sincronă de *MR*, ceea ce-l va face să revină în starea 0 pe următorul impuls de tact).

combinational logic block that generates an impulse each 5 clock cycles.

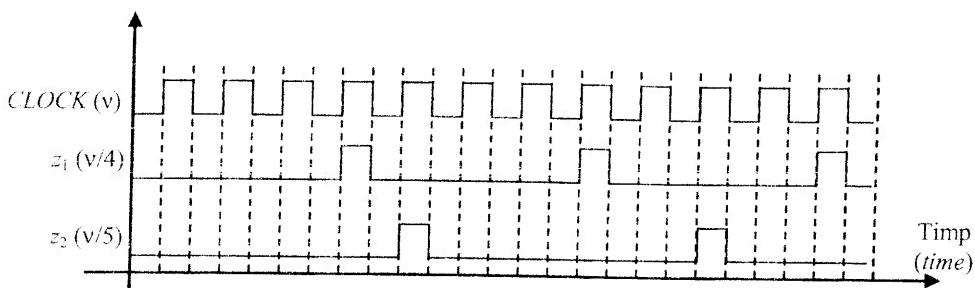
The first counter is a modulo 4 counter (in state 3 we will activate its *MR* input, which will make it revert to state 0 on the next clock impulse).

The second counter is a modulo 5 counter (in state 4 we will activate its *MR* input, which will make it revert to state 0 on the next clock impulse).



În figura de mai jos sunt prezentate formele de undă generate, din care se poate observa că semnalele *z₁* și *z₂* sunt periodice, conform cerințelor problemei.

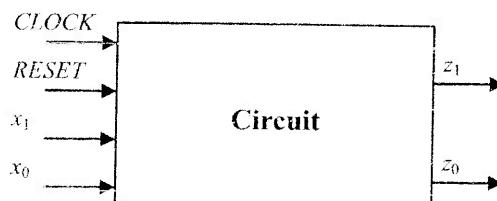
In the figure below are shown the generated waveforms, from which one can notice that the *z₁* and *z₂* signals are periodical, according to the problem's requirements.



12. Un circuit sevențial are două intrări sincrone și două ieșiri. Intrările (x_1 și x_0) reprezintă un număr pe doi biți, N . Dacă valoarea actuală a lui N (cea care sosește pe x_1x_0) este mai mare decât valoarea anterioară, atunci ieșirea $z_1 = 1$. Dacă valoarea actuală a lui N este mai mică decât valoarea anterioară, atunci ieșirea $z_0 = 1$. Altfel, z_1 și z_0 sunt 0. Proiectați circuitul logic sevențial care realizează acest comportament (știind că are 5 stări – una dintre ele fiind de inițializare).

Soluție

În primul rând vom desena cutia neagră a sistemului:



Vom reprezenta formal comportamentul sistemului folosind

12. A sequential circuit has two synchronous inputs and two outputs. The inputs (x_1 and x_0) represent a 2-bit binary number, N . If the present value of N (the one that arrives on x_1x_0) is greater than the previous value, then the output $z_1 = 1$. If the present value of N is less than the previous value, then the output $z_0 = 1$. Otherwise, z_1 and z_0 are 0. Derive a sequential logic circuit with this behavior (*Hint*: The machine needs only five states – one of them is the initialization state).

Solution

First we will draw the system's black box:

We will represent formally the system's behavior, this time using

de data aceasta metoda bazată pe *tabelul de tranziție* (tabelul conține, pentru fiecare celulă, starea următoare / ieșirile):

$x_1 \ x_0$	00	01	11	10
Starea curentă (current state)				
A [$N = \text{nedefinit} (\text{undefined})$]	B / 00	C / 00	D / 00	E / 00
B ($N = 00$)	B / 00	C / 10	D / 10	E / 10
C ($N = 01$)	B / 01	C / 00	D / 10	E / 10
D ($N = 10$)	B / 01	C / 01	D / 10	E / 00
E ($N = 11$)	B / 01	C / 01	D / 00	E / 01

Observăm că, întrucât intrările sunt sincrone, nu este necesară aplicarea vreunei reguli de codificare a stărilor.

Întrucât în starea A nu există o valoare anterioară a lui N , toate tranzițiile care pornesc din starea A vor fi cu ieșiri 00 (nu se poate spune că nouă N este mai mare sau mai mic decât N -ul anterior, deoarece nu există vreun N anterior).

Vom codifica apoi stările și le vom înlocui în tabelul de tranziție cu codurile lor respective:

the method based on a *transition table* (the table contains, for each cell, next_state / outputs):

We observe that, since inputs are synchronous, it is not necessary to apply any rule for encoding the states.

Since in state A there is no “previous value” of N , all transitions starting from state A will have 00 as outputs (we can not say that the new N is greater or equal than the previous N , because there is no previous N).

Then we will encode the states and will replace them in the transition table with their codes:

		Q_0			
		00	01	11	10
Q_2	0	A ₀	X ₁	X ₃	X ₂
	1	B ₄	C ₅	D ₇	E ₆

Q_1

Cod stare (state code)	$x_1 \ x_0$	00	01	11	10
	Starea curentă (current state)				
000	A [$N = \text{nedefinit (undefined)}$]	100 / 00	101 / 00	111 / 00	110 / 00
100	B ($N = 00$)	100 / 00	101 / 10	111 / 10	110 / 10
101	C ($N = 01$)	100 / 01	101 / 00	111 / 10	110 / 10
111	D ($N = 11$)	100 / 01	101 / 01	111 / 00	110 / 01
110	E ($N = 10$)	100 / 01	101 / 01	111 / 10	110 / 00

Observăm că atât starea viitoare, cât și ieșirile depind atât de intrări cât și de starea curentă, ceea ce ne arată că avem de-a face cu un automat Mealy. Presupunând că dorim să implementăm sistemul cu bistabile de tip D, aceasta se poate exprima astfel:

$$D_i = f(x_1, x_0, Q_2, Q_1, Q_0), i = \overline{0,2}$$

$$z_j = g(x_1, x_0, Q_2, Q_1, Q_0), j = \overline{0,1}$$

În mod normal, ar trebui acum să „spargem” diagrama Karnaugh globală în 5 diagrame Karnaugh particolare (câte una pentru D_2 , D_1 , D_0 , z_1 și z_0). Întrucât minimizarea unor diagrame Karnaugh de 5 variabile este foarte anevoieasă, căutăm o soluție mai simplă.

Codificarea stărilor a fost special aleasă încât Q_1Q_0 să fie egal cu numărul anterior recepționat pe intrările x_1x_0 . Atunci intrările x_1 și x_0 vor fi egale cu D_1D_0 și astfel se va determina starea următoare a sistemului:

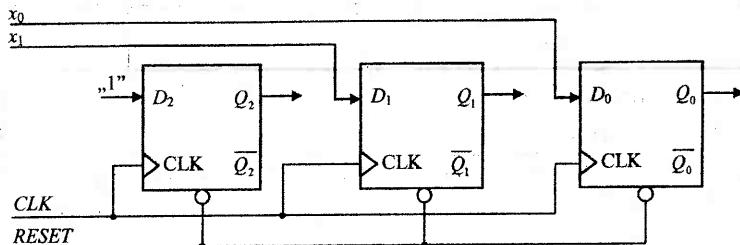
We notice that both the next state and the outputs depend on the inputs and on the current state, which indicates us that we are dealing with a Mealy automaton. Assuming we want to implement the system with D Flip-Flops, we can express this as follows:

$$D_i = f(x_1, x_0, Q_2, Q_1, Q_0), i = \overline{0,2}$$

$$z_j = g(x_1, x_0, Q_2, Q_1, Q_0), j = \overline{0,1}$$

Normally we should now “break” the global Karnaugh map in 5 particular Karnaugh maps (one for each function: D_2 , D_1 , D_0 , z_1 and z_0). Since minimizing 5-variable Karnaugh maps is very laborious, we seek a simpler solution.

The states encoding was done in a special manner, to make sure Q_1Q_0 is equal to the previously received number (the one received on the x_1x_0 inputs). Then the inputs x_1 and x_0 will be equal to D_1D_0 and this way will be determined the system's next state:



Pentru calculul ieșirilor, observăm că:

- dacă starea sistemului este A, atunci ieșirile z_1 și z_0 sunt 00.
- dacă starea sistemului NU este A, atunci ieșirile z_1 și z_0 pot fi date de un comparator între x_1x_0 (noul N) și Q_1Q_0 (vechiul N). Putem detecta faptul că starea curentă nu este A cu ajutorul unei porți SI-NU cu 3 intrări, ale cărei intrări sunt Q_2 , Q_1 și Q_0 .

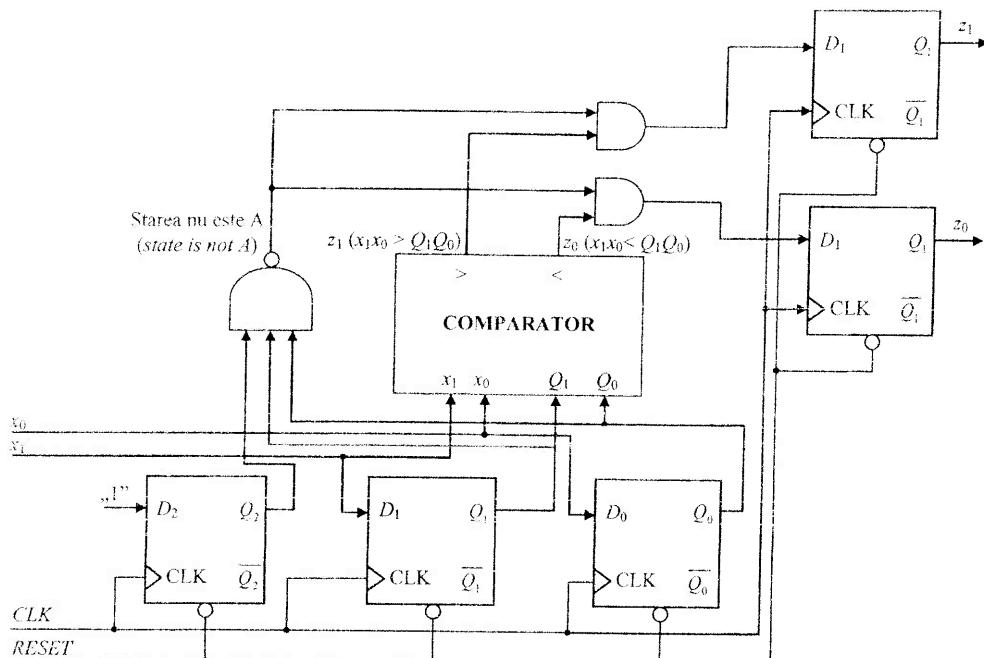
Un aspect important este acela că sistemul este o mașină Mealy. Trebuie să înțelegem că aici, atunci când spunem că „se realizează tranziția din starea A în starea B cu ieșirea z ”, ieșirea z este generată de-abia în starea B, deci la sfârșitul tranziției. De aceea, pe fiecare semnal de ieșire al unui automat Mealy trebuie inserat un bistabil D. Modelând și combinând toate aceste observații, obținem schema finală a sistemului:

To compute the outputs, we notice that:

- if the system's state is A, then the z_1 and z_0 outputs are 00.
- if the system's state is NOT A, the z_1 and z_0 outputs can be given by a comparator between x_1x_0 (the new N) and Q_1Q_0 (the old N). We can detect the fact that the current state is not A by means of a 3-input NAND gate whose inputs are Q_2 , Q_1 and Q_0 .

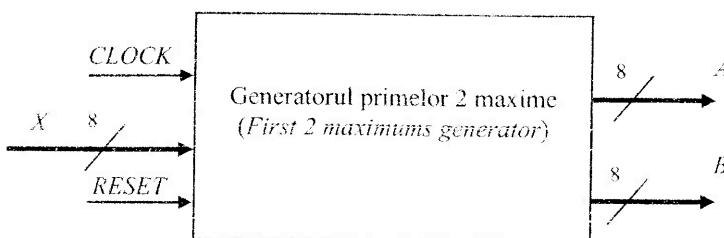
An important aspect is that the system is a Mealy machine. We have to understand that in this case, when we say „a transition is realized from state A to state B with output z ”, the output z is generated only in state B, i.e. at the end of the transition. This is why, on each output signal of a Mealy automaton a D Flip-Flop must be inserted.

By modeling and combining all these observations, we obtain the system's final scheme:



13. Pe o linie de date sosesc cuvinte de 8 biți în mod serial, sincron. Proiectați un sistem numeric care returnează primele două cele mai mari numere din *stream*-ul de intrare [A este numărul cel mai mare, B este numărul imediat următor după A ($A > B >$ oricare alt număr din stream-ul de intrare)].

13. On a data line, 8-bit words come in a serial manner, synchronously. Design a digital system that yields the two greatest numbers from the input stream [A is the maximal number, B is the number following immediately after A ($A > B >$ any other number from the input stream)].



Soluție

Natura problemei este astfel încât în acest caz este foarte adecvată o implementare de tip *pipeline*. Variabilele de intrare și de ieșire X , A și B le vom stoca în registre de date, care vor fi inițializate cu cel mai mic număr întreg fără semn ce poate fi reprezentat pe 8 biți (evident, acest număr este 0).

Vom compara noul X atât cu actualul A cât și cu actualul B .

Numărul din registrul A va fi înlocuit doar dacă noul X este mai mare decât actualul A .

În registrul B se pot încărca mai multe valori, în funcție de caz:

Solution

The problem's nature is such that in this case it is very appropriate to realize a pipelined implementation. The input and output variables, X , A and B , will be stored in data registers, that will be initialized with the smallest possible unsigned integer that can be represented on 8 bits (obviously, this number is 0).

We will compare the new X both with the current A and with the current B .

The number in the A register will only be replaced if the new X is greater than the current A .

In the B register we can load several values, according to the situation:

$X > A$	$X > B$	Valoarea de încărcat în registrul B (value to be loaded in the B register)	Valoarea de încărcat în registrul A (value to be loaded in the A register)
Adevărat (true)	Adevărat (true)	A	X
Fals (false)	Adevărat (true)	X	A – valoare neschimbată (value unchanged)
Fals (false)	Fals (false)	B – valoare neschimbată (value unchanged)	A – valoare neschimbată (value unchanged)
Adevărat (true)	Fals (false)	Imposibil (impossible)	Imposibil (impossible)

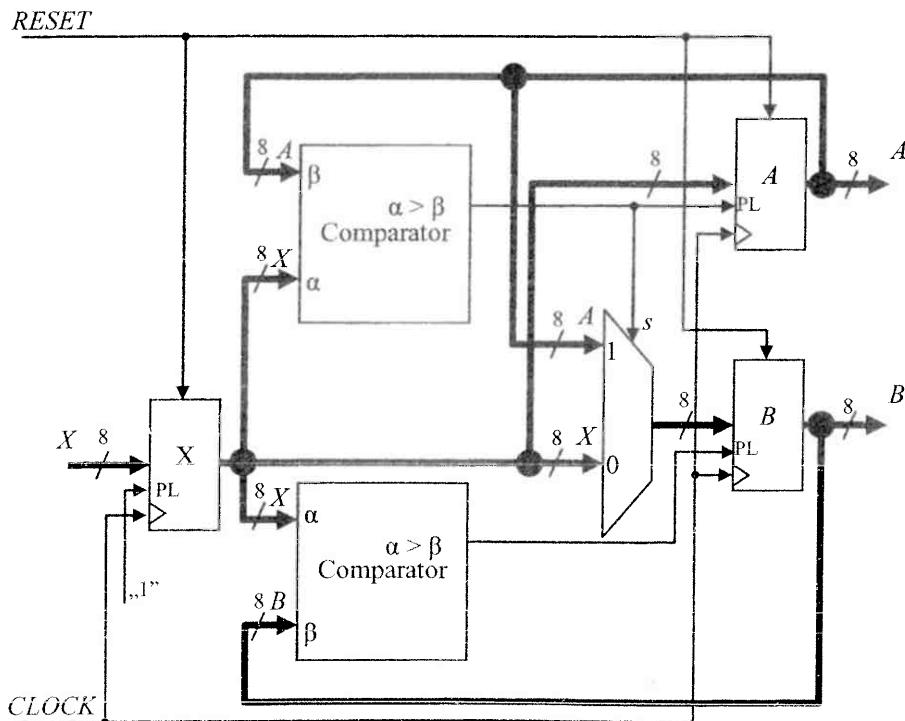
Evident, implementarea se va baza, pe lângă registrele de date deja

Obviously, the implementation will be based, apart the already

menționate, pe două comparatoare de numere pe 8 biți și pe un multiplexor 2:1 cu calea de date de 8 biți (la care selecția s va fi ieșirea „ $X > A$ ” a primului comparator).

Observăm că registrul A va fi încărcat cu o nouă valoare doar dacă $X > A$, iar registrul B va fi încărcat cu o nouă valoare doar dacă $X > B$. Vom folosi deci intrarea PL (*parallel load*) pentru a controla modificarea valorilor stocate în registre. Aceasta ne permite ca, deși avem 4 cazuri, să folosim doar un multiplexor 2:1, conform schemei finale prezentate mai jos:

mentioned data registers, on two 8-bit comparators and a 2:1 multiplexer with an 8-bit data path (for which the selection s will be the first comparator's “ $X > A$ ” output). We notice that the A register will be loaded with a new value only if $X > A$, and the B register will be loaded with a new value only if $X > B$. Therefore, we will use PL (*parallel load*) input to control the change of the values stored inside them. This allows us to use a smaller 2:1 multiplexer, even though we have 4 cases, according to the final scheme presented below:



Recapitulăm, pentru a facilita înțelegerea schemei:

Let's recapitulate, for a better understanding of this scheme:

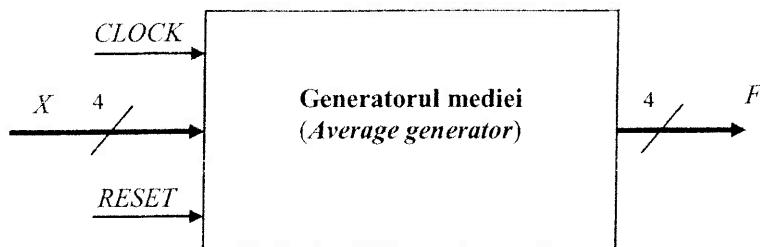
$X > A$	$X > B$	PL_A	PL_B	Valoarea de încărcat în registrul B (value to be loaded in the B register)	Valoarea de încărcat în registrul A (value to be loaded in the A register)
Adevărat (true)	Adevărat (true)	1	1	A	X
Fals (false)	Adevărat (true)	0	1	X	A – valoare neschimbată (value unchanged)
Fals (false)	Fals (false)	0	0	B – valoare neschimbată (value unchanged)	A – valoare neschimbată (value unchanged)
Adevărat (true)	Fals (false)	1	0	Imposibil (impossible)	Imposibil (impossible)

O ultimă observație: datorită caracterului de arhitectură *pipeline*, rezultatul corect va ieși pe A și B cu o întârziere de o perioadă a semnalului de tact. Dacă dorim să obținem rezultatele într-o singură perioadă de tact, vom elibera registru X de la intrare.

14. Pe o linie de date sosesc cuvinte de 4 biți în mod serial, sincron. Proiectați un sistem numeric care returnează MEDIA ultimelor 4 numere din *stream*-ul de intrare ($F = 0,25 \times \text{Suma_ultimelor_patru_numere}$). MEDIA va trebui să fie rotunjită (de exemplu, $18 / 4 = 4$).

One last remark: because of the *pipelined* architecture style, the correct result will appear on A and B with one clock signal cycle delay. If we want to get the results in a single clock cycle, we will eliminate the X register from the input.

14. On a data line, 4-bit words come in a serial manner, synchronously. Design a digital system that yields the AVERAGE of the last 4 numbers from the input stream ($F = 0.25 \times \text{Sum_of_the_last_four_numbers}$). The AVERAGE must be rounded (for instance, $18 / 4 = 4$).

Soluție

Pentru a rezolva această problemă trebuie să realizăm următoarele:

- să stocăm ultimele 4 cuvinte (semi-octeți). Vom realiza aceasta conectând în serie 4 registre de date pe câte 4 biți.
- să adunăm valorile stocate în aceste registre. Vom realiza aceasta cu ajutorul unor sumatoare complete.
- să împărțim rezultatul adunării la 4. Vom realiza aceasta printr-o deplasare la dreapta cu 2 poziții binare a rezultatului adunării. Aici se face și rotunjirea, dacă ținem cont de faptul că neglijăm cei 2 biți care se pierd.

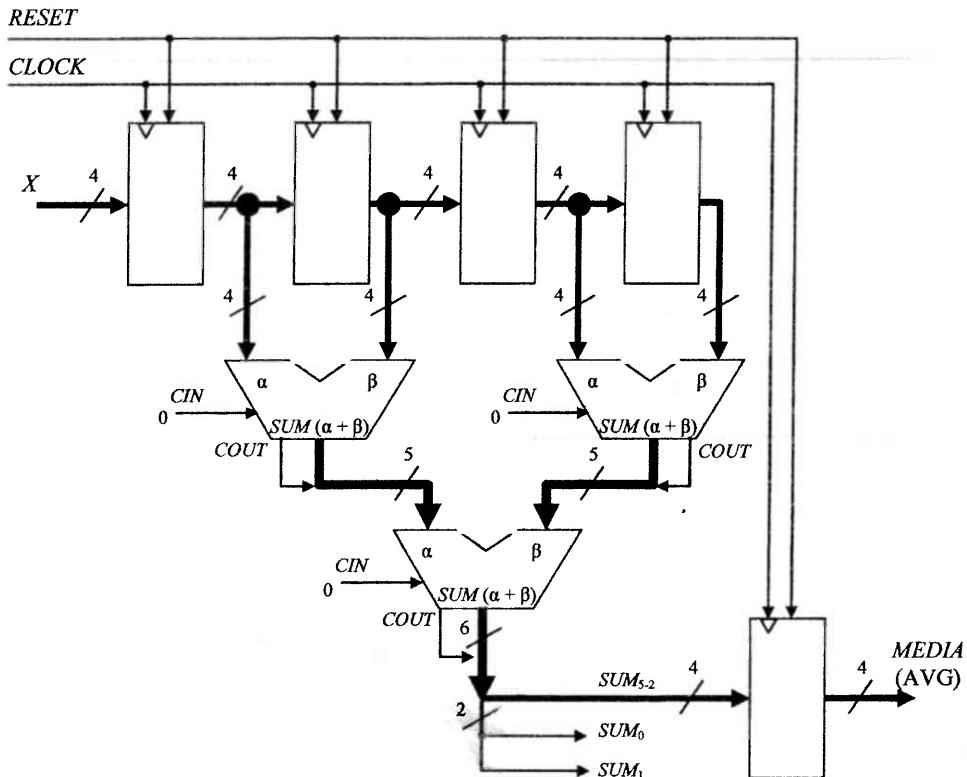
Schema este foarte ușor de determinat:

Solution

To solve this problem we must perform the following actions:

- store the last 4 words (half bytes). We will do this by connecting serially four 4-bit data registers.
- add the values stored in these registers. We will do this by means of some full adders.
- divide by 4 the result of the addition. We will realize this by a right shift with 2 binary positions of the result of the addition. Here we also perform the rounding, if we take into consideration the fact that we neglect the 2 bits that are lost.

The scheme is very easy to determine:



Câteva observații:

- Când adunăm 2 numere pe N biți, rezultatul va fi tot pe N biți plus un semnal de *Carry Out*. Prin urmare, primele două sumatoare complete sunt de 4 biți.
- Al treilea sumator complet va fi de 5 biți, iar rezultatul produs de el (*Carry Out* concatenat cu *SUM*) va fi pe 6 biți.
- Media va fi tot pe 4 biți: vom prelua doar primii 4 biți cei mai semnificativi din cei 6 biți produși de al treilea sumator complet, iar

A few remarks:

- When adding two N -bit numbers, the result will be also on N bits plus a *Carry Out* signal. Therefore, the first two full adders are 4-bit full adders.
- The third full adder will be a 5-bit one, and the result generated by it (*Carry Out* concatenated with *SUM*) will be on 6 bits.
- The average will be on 4 bits, too: we will take only the first four most significant bits from the six bits produced by the third full adder, and

bîții SUM_1 și SUM_0 se vor pierde.

- Rezultatul se va obține cu o întârziere de o perioadă a semnalului de tact, la fel ca la problema precedentă.

Exerciții suplimentare:

a. Cum se poate crea în mod eficient un pipeline pe baza acestei arhitecturi?

b. Propuneți și alte soluții ale acestei probleme, cu un număr mai mic de resurse! (*Indicație:* numerele noi se adună, iar ultimul număr din lanțul de registre de date se scade din total).

15. Proiectați cu componente MSI și / sau SSI standard un bloc de memorie de capacitate 4×2 bîți, cu un port de SCRİERE (A) și două porturi de CITIRE (B și C).

Soluție

Memoriile multi-port se pot implementa cu bistabile D și multiplexoare / demultiplexoare. În cazul de față, soluția este relativ simplă și se poate modela conform schemei din figura următoare:

the bits SUM_1 and SUM_0 will be lost.

- The result will be obtained with one clock cycle delay, like in the previous problem.

Supplemental exercises:

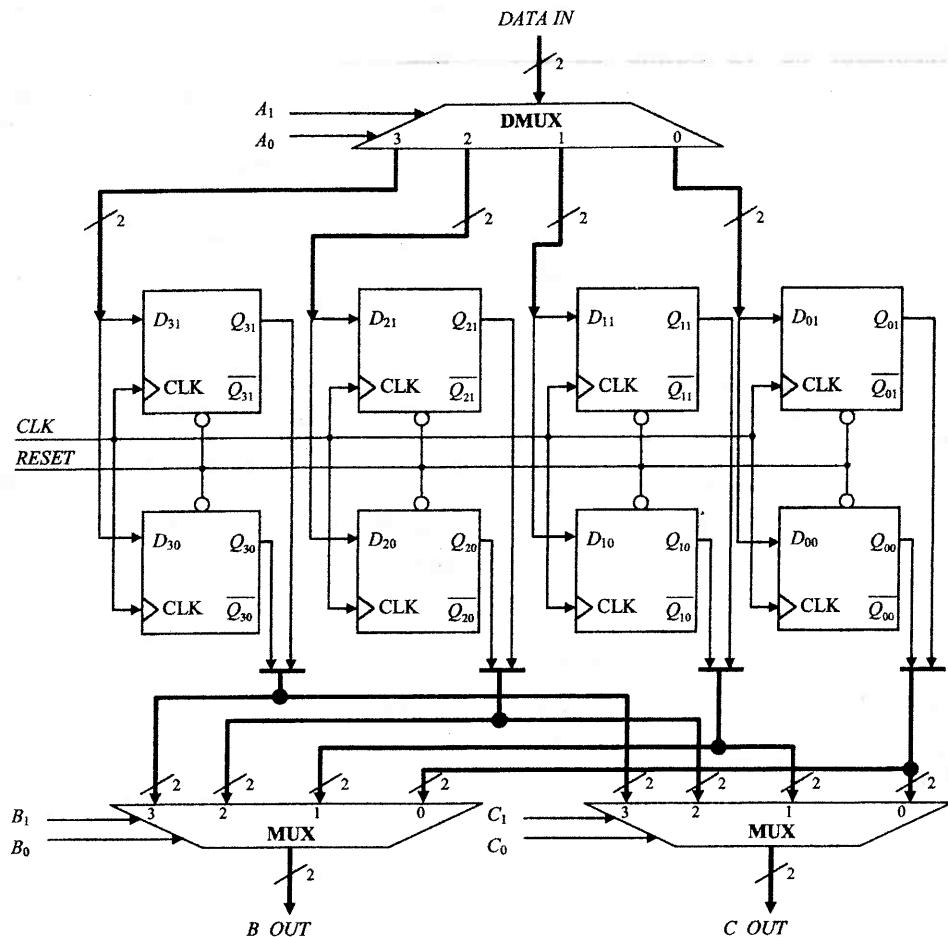
a. How could be efficiently created a pipeline, based on this architecture?

b. Propose other solutions to this problem, with a smaller amount of resources! (*Hint:* the new numbers are added, and the last number from the data registers chain is subtracted from the total).

15. Design using standard MSI and / or SSI components a memory block having a capacity of 4×2 bits, having a WRITE port (A) and two READ ports (B and C).

Solution

Multi-port memories can be implemented with D Flip-Flops and multiplexers / demultiplexers. In this case, the solution is relatively simple and can be modeled according to the scheme in the next figure:

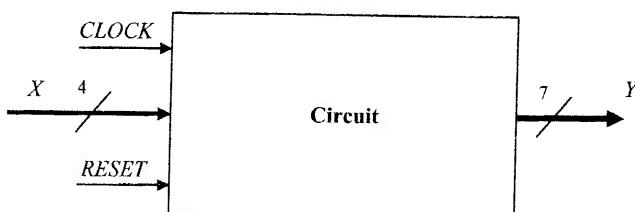


16. Proiectați cu componente MSI și / sau SSI standard un sistem logic care citește date de pe o linie de intrare cu lățimea de 4 biți numită X și returnează la ieșirea pe 7 biți numită Y numărul $Y = X * 5$.

16. Design using standard MSI and / or SSI components a logic system that reads data from a 4-bit input line called X and yields at the 7-bit output called Y the number $Y = X * 5$.

Soluție

Mai întâi vom desena cutia neagră a sistemului.



Vom utiliza o soluție oarecum asemănătoare celei adoptate pentru rezolvarea problemei 14.

Stim că $X * 5 = X * 4 + X$. Avem deci nevoie de un dispozitiv care să realizeze înmulțirea cu 4 și de un sumator complet. Însă stim că înmulțirea cu 4 se face deplasând la stânga numărul cu 2 poziții binare. Nu avem nevoie de un registru de deplasare pentru aceasta, ci vom lega pur și simplu firele astfel încât să obținem acest efect, pe cei 2 biți cei mai puțin semnificativi punând 0.

Sumatorul complet va fi deci pe 6 biți, după cum se poate vedea în schema din figura următoare.

La nivelul sumatorului complet, avem:

- operandul α va fi compus astfel: $\alpha_{5-2} = X_{3-0}$, $\alpha_{1-0} = „00”$ (deci $\alpha = X * 4$)
- operandul β va fi compus astfel: $\beta_{5-4} = „00”$, $\beta_{3-0} = X_{3-0}$ (deci $\beta = X$).

Solution

First, we will draw the black box of the device.

We will adopt a solution that is somehow similar to the one we have chosen for solving problem 14.

We know that $X * 5 = X * 4 + X$. Therefore, we need a device that performs the multiplication by 4 and a full adder. But we know that multiplication by 4 is done by shifting left the number by 2 binary positions. We don't need a shift register for that, but we will just connect the wires in such a way that we obtain this effect, putting 0 on the two least significant bits.

So the full adder will be on 6 bits, as one can see in the scheme from the next figure.

At the level of the full adder, we have:

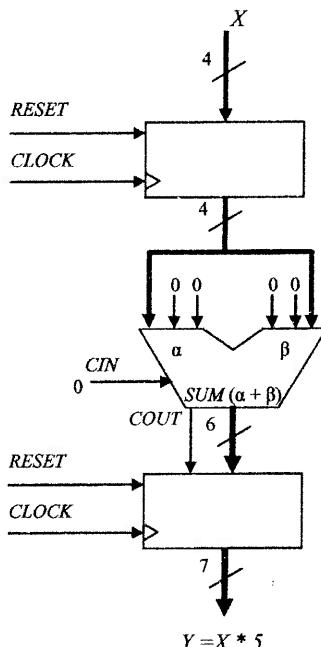
- the α operand will be composed as follows: $\alpha_{5-2} = X_{3-0}$, $\alpha_{1-0} = „00”$ (so $\alpha = X * 4$)
- the β operand will be composed as follows: $\beta_{5-4} = „00”$, $\beta_{3-0} = X_{3-0}$ (so $\beta = X$).

Ieșirea este pe 7 biți, deoarece în fața celor 6 biți produși de sumator se concatenează bitul de *Carry out*.

Se observă că și această arhitectură este de tip *pipeline*, ieșirea apărând cu o întârziere (numită *latență*) de o perioadă a semnalului de tact.

The output is on 7 bits, because in front of the 6 bits produced by the full adder we will concatenate the *Carry out* bit.

One can notice that this architecture too is a *pipelined* one, the output appearing with a delay of one clock cycle (this is called *latency*).



17. Un controlor de semafor pentru pietoni funcționează după cum urmează: ledul ROȘU este APRINS și ledul VERDE este STINS. Dacă cineva dorește să traverseze strada, el / ea apasă un buton. Imediat după apăsarea butonului, ledul ROȘU clipește de 5 (cinci) ori în 5 secunde, apoi se stinge. Simultan,

17. A traffic light controller for pedestrians functions as follows: The RED light is ON and the GREEN light is OFF. If somebody wants to cross the street, he / she presses a button. Immediately after the button has been pressed, the RED light blinks 5 (five) times in 5 seconds, then it shuts down.

ledul VERDE se APRINDE timp de 27 de secunde. După aceste 27 de secunde, din nou, ledul ROȘU este APRINS și ledul VERDE este STINS. Timp de 32 de secunde, apăsarea butonului de către pietoni nu are nici un efect (ledul ROȘU va fi APRINS și ledul VERDE STINS), după care apăsarea butonului de către pietoni are din nou efect. Implementați un sistem numeric având comportamentul descris mai sus. Există disponibil un semnal de TACT cu frecvență de 1 Hz și un factor de umplere de 50%. *Indicație:* Implementați mai întâi sistemul care controlează ledul ROȘU și apoi exprimați ledul VERDE ca o funcție de cel ROȘU și de regimul de funcționare „clipire”.

Soluție

Sistemele numerice complexe pot fi văzute ca fiind compuse dintr-o *Unitate de comandă* UC (sistemul logic sevențial care implementează algoritmul propriu-zis) și o *Unitate de execuție* UE (colecția de resurse care realizează operațiunile elementare). Unitatea de execuție se mai numește și *Unitatea căii de date*.

Unitatea de comandă (UC) este cea care controlează în mod individual fiecare resursă din Unitatea de execuție (UE), furnizându-i acesteia

Simultaneously, the GREEN light goes ON for 27 seconds. After those 27 seconds, again, the RED light is ON and the GREEN light is OFF. For 32 seconds, pressing the button by pedestrians has no effect (the RED light will be ON and the GREEN light will be OFF), then pressing the button by pedestrians is effective again. Implement a digital system with the behavior described above. There is a CLOCK signal with a frequency of 1 Hz and a duty cycle of 50% available. *Indication:* Implement first the system that controls the RED light and then express the GREEN light as a function of the RED light and the blinking working mode.

Solution

Complex digital systems can be seen as being composed of a *Command unit* CU (the sequential logic system that implements the algorithm itself) and an *Execution unit* EU (the collection of resources that execute the elementary operations). The Execution unit is also called *Datapath unit*.

The Command unit (CU) is the one that individually controls each resource in the Execution unit (EU), by providing to the latter the command signals for all its

din urmă semnalele de comandă pentru toate resursele sale în configurația potrivită și la momentul potrivit, astfel încât algoritmul să fie dus la bun sfârșit.

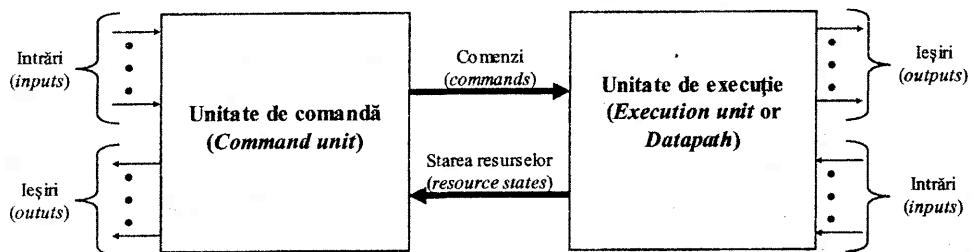
Pentru a lua decizii, UC are nevoie să citească atât variabilele de intrare ale sistemului, cât și starea resurselor din UE.

Atât intrările cât și ieșirile sistemului pot fi citite / generate, atât în / din UC cât și în / din UE.

resources, in the right configuration and at the right moment of time, such as the algorithm is executed properly.

In order to take decisions, the CU needs to read both the system's input variables and the state of the resources from the EU.

Both the inputs and the outputs of this system can be read / generated, both in / from the UC and in / from the UE.



În cazul de față, UE va fi alcătuită dintr-un modul de control al semaforului și dintr-un timer capabil să semnaleze cele 3 perioade specificate în enunțul problemei. UC va fi un circuit logic secvențial care va comanda modulul de control al semaforului și va lua decizii atât în funcție de starea *timer*-ului cât și în funcție de Butonul pus la dispoziția pietonilor.

În cazul nostru, întregul sistem este sincron: UC și UE funcționează pe același tact. Nu este necesară impunerea unui semnal de tact separat pentru *timer*.

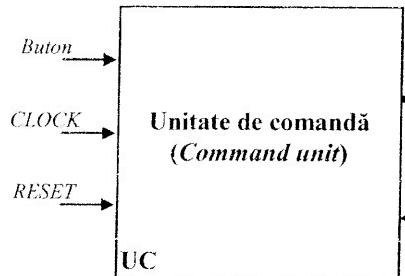
In this case, the EU will be composed of a traffic light controller module and a timer able to signal the three periods specified in the problem's text. The CU will be a sequential logic circuit that will command the traffic light controller module and will take decisions both according to the timer's state and to the Button that is available for the pedestrians.

In our case, the whole system is synchronous: the CU and the EU work on the same clock signal. It is not necessary to impose a separated clock signal for the timer.

Deși le vom prezenta secvențial, în realitate construirea organigramei și a schemei bloc se face în paralel.

Semnalele primite de UC din partea *timer-ului* sunt cele care semnalează trecerea perioadelor menționate: de 5, 27 și 32 de secunde. UC trebuie să pornească *timer-ul* la momentul potrivit și să-l re-initializeze după ce și-a încheiat misiunea.

Modulul de control al semaforului este integrat în UE și nu transmite nimic către UC, ci doar controlează aprinderea / stingerea led-urilor ROȘU și VERDE.



Unitatea de comandă

UC stabilește regimurile de funcționare, care rezultă din specificația sistemului:

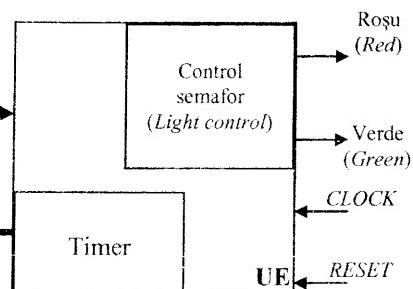
- 1) Aprindere ROȘU (z_R)
- 2) Clipire ROȘU (z_{CLIP})
- 3) VERDE APRINS (z_{VERDE})
- 4) Buton BLOCAT (z_{BUT})

Fiecare regim de lucru îi va corespunde o ieșire din UC,

Even though we will present them sequentially, in fact the construction of the state diagram and of the block scheme is done in parallel.

The signals received by the CU from the timer are those that signal the expiration of the above mentioned periods: 5, 27 and 32 seconds. The CU must start the timer at the right moment and re-initialize it after its task was done.

The traffic light controller module is integrated in the EU and transmits nothing towards the CU; it just controls turning ON / OFF of the RED and GREEN lights.



The Command unit

The CU establishes the working regimes, which result from the system's specification:

- 1) RED ON (z_R)
- 2) RED blinking (z_{CLIP})
- 3) GREEN ON (z_{VERDE})
- 4) Button LOCKED (z_{BUT})

To each working mode corresponds a CU's output, respectively: z_R ,

respectiv: z_R , z_{CLIP} , z_{VERDE} și z_{BUT} .

În plus mai există o ieșire:

5) Validare funcționare (*Clock enable*) timer (z_{CET})

Organograma sistemului se bazează pe specificația acestuia în limbaj natural. În ea vom avea următoarele intrări:

1) Buton

2) T_5 (provine de la timer-ul din UE; arată că au trecut 5 secunde)

3) T_{27-32} (provine de la timer-ul din UE; arată că au trecut încă 27 de secunde după regimul de clipire, respectiv 32 de secunde din regimul de blocare a butonului).

Organograma sistemului este prezentată în figura următoare:

z_{CLIP} , z_{VERDE} and z_{BUT} .

In addition, there is another output:

5) Timer enable (*Clock enable*) (z_{CET})

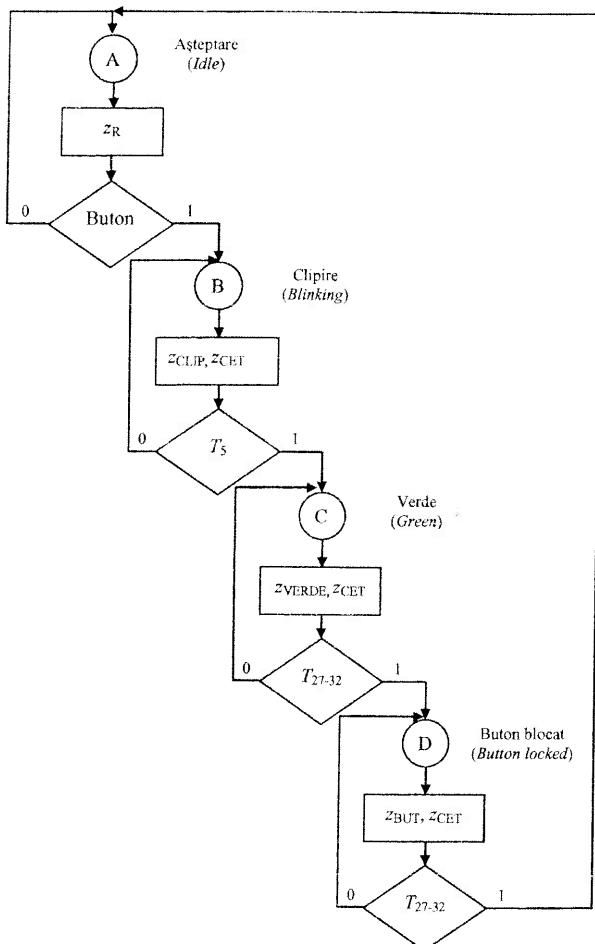
The system's state diagram is based on its specification in natural language. We will have the following inputs:

1) Buton

2) T_5 (comes from the EU's timer; it shows that 5 seconds have passed)

3) T_{27-32} (comes from the EU's timer; it shows that another 27 seconds have passed after the blinking régime, respectively 32 seconds from the button locking regime).

The system's state diagram is presented in the next figure:



Unitatea de execuție

UE este alcătuită din cele 2 module principale, *timer*-ul și modulul de control al semaforului, care vor fi detaliiate în cele ce urmează:

a) *Timer*-ul

Putem implementa cele 3 perioade temporale din specificație cu

The Execution unit

The UE is composed of the two main modules: the timer and the traffic light controller module, which will be detailed below:

a) The timer

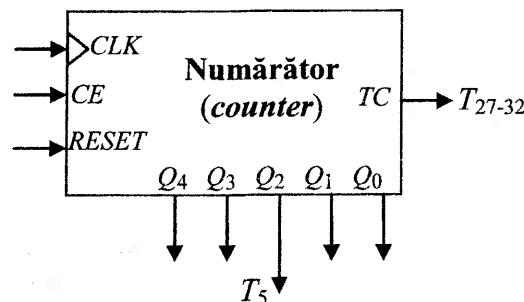
We can implement the three temporal periods from the

ajutorul unui singur numărător pe 5 biți, astfel:

- Perioada 1 („clipire ROȘU”): numărătorul pornește de la 0 și ajunge până la 4. Când se atinge starea 4 se generează semnalul T_5 către UC.
- Perioada 2 („VERDE”): numărătorul continuă din starea 4 și ajunge până la 31, numărând astfel 27 de secunde. Când se atinge starea 31 se generează semnalul T_{27-32} către UC.
- Perioada 3 („Buton blocat”): numărătorul continuă să numere, în bucla sa specifică, de la 0 și ajunge din nou în starea 31, numărând astfel 32 de secunde. Când se atinge starea 31 se generează semnalul T_{27-32} către UC.

În continuare, numărătorul va rămâne blocat (nu va primi *Clock Enable* de la UC) până la următoarea apăsare a Butonului de către un pieton.

Schema *timer*-ului este redată în figura următoare (se știe că ieșirea *TC* – *terminal count* – se obține intern ca un *ȘI* între toate ieșirile Q_{4-0} ale numărătorului):



specification with a single 5-bit counter, as follows:

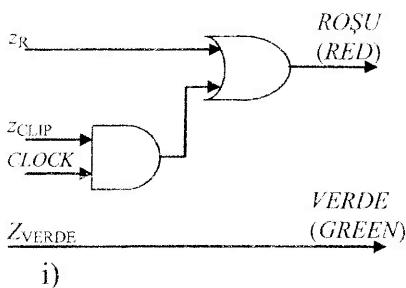
- Period 1 (“RED blinking”): the counter starts from 0 and reaches state 4. When state 4 is reached, the T_5 signal is generated towards the CU.
- Period 2 (“GREEN”): the counter continues from state 4 and reaches state 31, thus counting 27 seconds. When the state 31 is reached, the T_{27-32} signal is generated towards the CU.
- Period 3 (“Button locked”): the counter continues to count, in its specific loop, from 0 and thus it gets again in state 31, thus counting 32 seconds. When state 31 is reached, the T_{27-32} signal is generated towards the CU.

Then, the counter will remain locked (it will not receive *Clock Enable* from the CU) until the next time the Button is pressed by a pedestrian.

The timer's scheme is shown in the next figure (we know that the *TC* – *terminal count* – output is obtained internally as an AND between all the counter's outputs, Q_{4-0}):

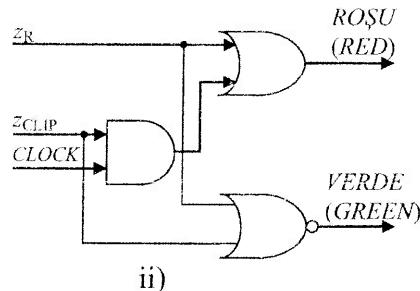
b) Modulul de control al semaforului

Structura acestui modul este asemănătoare schemei de la problema rezolvată nr. 10. Regimul de „clipire” este implementat printr-un SI cu semnalul de tact, iar pentru implementarea ieșirii VERDE avem două posibilități: i) fie o comandă direct cu ieșirea z_{VERDE} a UC, fie ii) observăm că ledul VERDE este aprins doar dacă ledul ROȘU nu este *nici* aprins, *nici* în regim de clipire. În limba engleză, NICI = NOR, deci putem folosi o poartă SAU-NU.



b) The traffic light controller module

The structure of this module is similar to the scheme from the solved problem no. 10. The “blinking” regime is implemented by an AND with the clock signal, and for the implementation of the GREEN output we have two possibilities: i) either we command it directly with the CU’s z_{VERDE} output, or ii) we observe that the GREEN light is ON only if the RED light is neither ON nor blinking. In English, NOR has the same meaning as in Boolean algebra, so we can use a NOR gate.



În concluzie, în urma acestor rafinări ale procesului de proiectare, interfața UC este formată din următoarele semnale:

- Intrări: *Buton*, T_5 , T_{27-32}
- Ieșiri: z_R , z_{CLIP} , z_{BUT} , z_{CET} . Observăm că z_{BUT} nu face altceva decât să indice pietonilor că deocamdată nu pot (nu are rost să)

In conclusion, after these refinings of the design process, the CU’s interface is composed of the following signals:

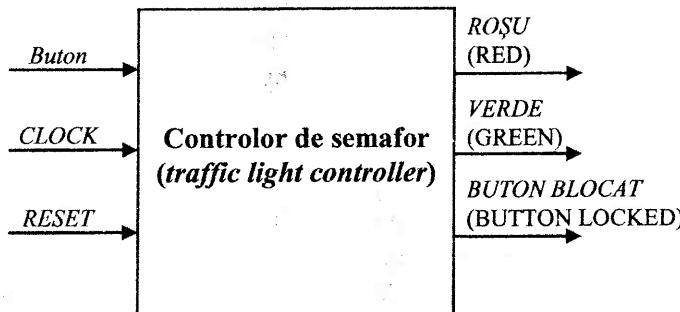
- Inputs: *Buton*, T_5 , T_{27-32}
- Outputs: z_R , z_{CLIP} , z_{BUT} , z_{CET} . Notice that z_{BUT} does not do anything but indicates the pedestrians that for the moment they

apese Butonul.

Cutia neagră a întregului sistem este următoarea:

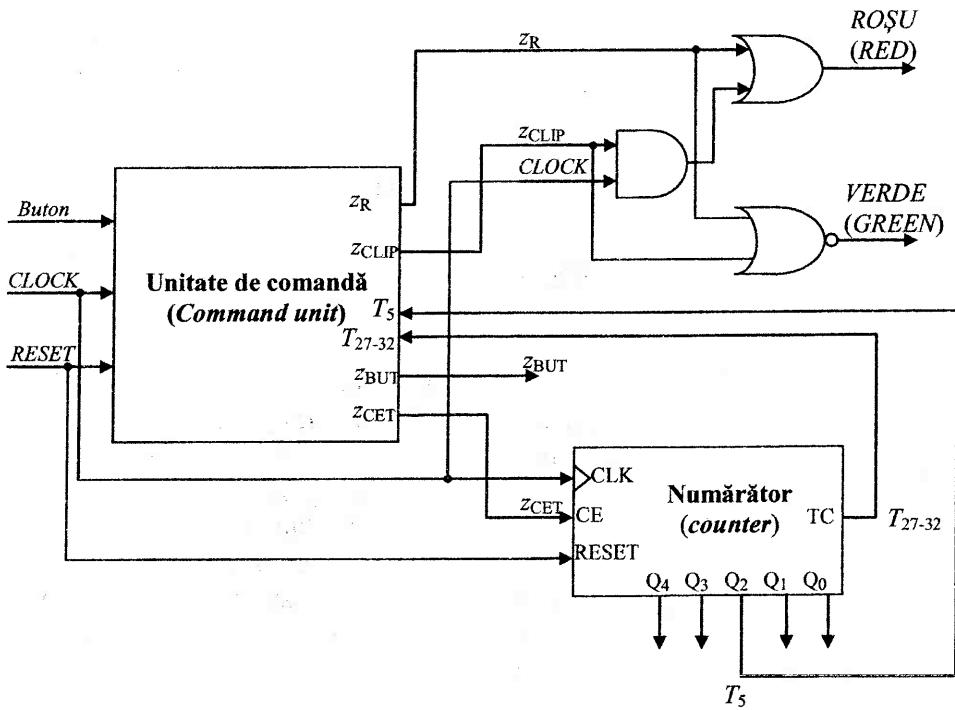
can not (it is useless to) press the Button.

The whole system's black box is the following one:



Schema bloc finală a sistemului este deci următoarea:

The system's final block scheme is thus the following one:



Stocarea și reading componentelor standard nu ne mai permite decât să proiectăm un IC cu lățimea următoarelor sale:

Ulatorul former urmărișmei, de genul metoda bazată pe numărător MSI (vezi problema rezolvată 5). Vom codifica stările astfel:

$$A: 00, B: 01, C: 10, D: 11.$$

Intrarea PL a numărătorului MSI trebuie activată în următoarele condiții:

- a) dacă starea curentă este A și $Buton = 0$;
- b) dacă starea curentă este B și $T_k = 0$;
- c) dacă starea curentă este C și $T_{2T_k} = 0$;
- d) dacă starea curentă este D și $T_{3T_k} = 0$.

Remarcăm că după starea D urmează în mod firesc starea A, dacă folosim un numărător modulo 4. Asadar, suntem când nu activăm PL , număratorul va trece firesc din starea curentă în starea următoare. Când activăm PL , în numărător se încarcă întotdeauna starea curentă.

Observăm că automatul este o mașină Moore (ieșirile depind numai de starea internă, nu și de intrări). Vom putea implementa ieșările cu ajutorul unui decodificator, astfel:

- a) ieșirea z_1 se generează în starea A;

- b) ieșirea z_2 se generează în

since the IC contains standard components, we only still have to design the CLD based on its state diagram.

Because of the state diagram's shape, we choose the method based on an MSI counter (see the solved problem no. 5). We will encode the states as follows:

$$A: 00, B: 01, C: 10, D: 11.$$

The MSI counter's PL input must be activated in the following conditions:

- a) if the current state is A and $Buton = 0$;
- b) if the current state is B and $T_k = 0$;
- c) if the current state is C and $T_{2T_k} = 0$;
- d) if the current state is D and $T_{3T_k} = 0$.

We notice that after state D follows naturally the state A, if we use a module 4 counter. Therefore, when we do not activate PL , the counter will naturally go from the current state in the next state. When we activate PL , in the counter we will always load the current state.

We notice that the automaton is a Moore machine (the outputs only depend on the internal state, not on the inputs). We will be able to implement the outputs with a decoder, as follows:

- a) the z_1 output is generated in state A;
- b) the z_2 output is generated in

starea B

c) ieșirea z_{CET} se generează în stările B, C sau D

d) ieșirea z_{BUT} se generează în starea D

e) la ieșirea z_{VERDE} putem renunța, după cum am arătat mai sus (o generăm din z_R și z_{CLIP}).

Numărătorul MSI este un numărător cu *PL* sincron și *Reset* asincron. El va fi resetat fie din exterior (cu semnalul *RESET* distribuit întregului sistem) fie din interior, în momentul când se atinge starea 4 (când $Q_2 = 1$ – astfel bucla de numărare va conține stările 0, 1 2 și 3, fiind un numărător modulo 4).

Implementarea finală a UC este prezentată în figura următoare:

state B

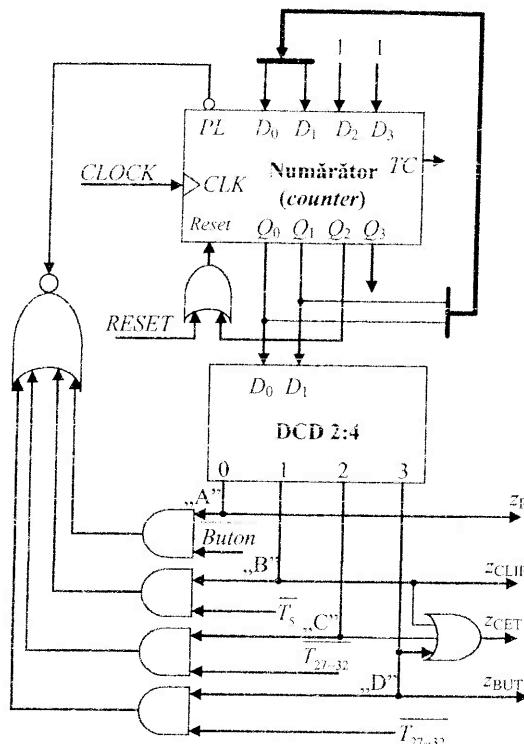
c) the z_{CET} output is generated in states B, C or D

d) the z_{BUT} output is generated in state D

e) we can give up the z_{VERDE} output, as we have shown above (we generate it from z_R and z_{CLIP}).

The MSI counter is a counter with synchronous *PL* and asynchronous *Reset*. It will be reset either from the external environment (by the global *RESET* signal which is distributed to the whole system) or from the inside, when state 4 is reached (when $Q_2 = 1$ – this way, the counting loop will contain states 0, 1, 2 and 3, being a modulo 4 counter).

The CU's final implementation is presented in the next figure:



18. Faceți parte dintr-o echipă care are de proiectat un cuptor electric compact și ieftin, conceput special pentru camere de cămin studențesc. Cuptorul trebuie să aibă un comportament deosebit de simplu: el va găti întotdeauna la aceeași temperatură (190°C) și timp de 25 de minute (mereu aceeași perioadă). Cuptorul trebuie să funcționeze după cum urmează. Starea inițială este una de așteptare, iar încălzitorul este stins. Când se apasă butonul de „Start” al cuptorului, încălzitorul trebuie pornit pentru a pre-încălzi

18. You are part of a team designing a compact and inexpensive electronic oven, especially designed for very small flats or student halls of residence. The oven is to have a particularly simple operation: it will always cook to the same temperature (190°C) and for the same time (25 minutes).

The oven is to operate as follows. The oven is initially in an idle state and the heating element is switched off. When the “Start” button of the oven is pressed, the heating element

cuptorul (deci pentru a-l aduce la temperatura de gătit). În timp ce se realizează aceasta, ledul „Preheating” („PH”) trebuie aprins pe panoul frontal al cuptorului. În interiorul cuptorului există un senzor de temperatură („TS”) care îi semnalează unității de control a cuptorului momentul când cuptorul este suficient de bine încălzit (190°C) pentru a se putea începe gătitul. În acest moment se va aprinde un led „Insert Food” („IF”), pentru ca utilizatorul să știe că acum cuptorul este pregătit pentru a găti mâncarea.

După ce pune mâncarea în cuptor, utilizatorul apasă din nou butonul „Start”, iar încălzitorul va rămâne pornit pentru a găti mâncarea un timp pre-stabilit, de 25 de minute. Pe durata acestui interval de timp, ledul „Cooking” trebuie să fie aprins. După încheierea procesului de gătit, încălzitorul trebuie oprit și ledul „Cooking” trebuie stins. În plus față de regimul de funcționare normal descris mai sus, cuptorul trebuie să aibă următoarea funcționalitate de siguranță: după terminarea etapei de pre-încălzire, dacă utilizatorul nu introduce mâncarea și nu apasă „Start” în 5 minute, cuptorul se va închide automat și va reveni în starea de aşteptare.

Senzorul de temperatură există deja.

should turn on to pre-heat the oven (i.e. to bring it up to its cooking temperature). While it is doing this, a “Preheating” (“PH”) lamp should light up on the front of the cooker. A temperature sensor (“TS”) inside the oven indicates to the oven controller when the oven is hot enough (190°C) to begin cooking. At this point, an “Insert Food” (“IF”) lamp turns on to tell the users that the oven is now ready to cook their meal.

After putting the item to be cooked into the oven, the user presses “Start” again, and the heating element will remain on to cook the food for the pre-set time of 25 minutes. During this time, the “Cooking” (“CK”) lamp should light. After cooking has completed, the heating element should turn off and the “Cooking” lamp should go out. In addition to the above normal operation, the oven is to have the following safety feature: once pre-heating is complete, if the users do not insert their item and press “Start” within 5 mins, the oven will switch off automatically and return to its idle state.

The temperature sensor is already designed. The 5 min and 25 min timers will be built by you using MSI counters. A clock signal with

Timer-ele de 5 și de 25 de minute vor fi construite de dumneavoastră folosind numărătoare MSI. Este disponibil un semnal de tact cu perioada de 1 minut. Senzorul de temperatură funcționează non-stop, iar când se atinge temperatura de 190°C , el generează ieșirea „TS”. *Timer*-ul de 25 de minute este pornit când cuptorul este în starea în care gătește, și generează ieșirea $T_{25} = 1$ după trecerea a 25 de minute. *Timer*-ul de 5 de minute este pornit de îndată ce cuptorul este suficient de încălzit pentru a începe să gătească și așteaptă ca utilizatorul să introducă mâncarea. După trecerea a 5 minute, *timer*-ul generează ieșirea $T_5 = 1$.

- Desenați schema bloc a sistemului. Evidențiați *Unitatea de execuție* și *Unitatea de control*. Indicați ce ieșiri sunt vizibile pe panoul frontal (cel pe care-l vede utilizatorul) și ieșirile care sunt interne sistemului.
- Trasați organograma unității de control a cuptorului descrise mai sus. Determinați cu atenție câte stări sunt necesare, inclusiv starea de așteptare. Determinați de asemenea care sunt intrările și ieșirile sistemului, știind că anumite ieșiri aprind ledurile, pe când altele pornesc *timer*-ele. Ieșirile senzorului de temperatură

the period of 1 minute is available. The temperature sensor is always enabled, and once the temperature reaches the required value of 190°C , it produces the output “TS”. The 25 min timer is started when the oven is in its cooking state, and sets its output, T_{25} , to ‘1’ when the 25 mins are over. The 5 min timer is started as soon as the oven is hot enough to start cooking and waiting for the user to insert the item to be cooked. When the 5 min are up, the timer sets its output, T_5 , to ‘1’.

- Draw the system's block diagram. Highlight the *Execution Unit* and the *Control Unit*. Show the outputs that are visible on the front panel (the one that the user sees) and the outputs that are internal to the system.
- Draw a state diagram for the simple oven controller described above. Think carefully about how many states you will require, including the idle state. Also consider what are the inputs and outputs of the system, bearing in mind that some outputs light up the lamps, while others start the timers. The outputs of the temperature sensor and timers will act as inputs to your system.

- și ale *timer*-elor vor constitui intrări în UC a sistemului.
- Precizați care intrări sunt sincrone și / sau asincrone.
 - Implementați sistemul numeric care controlează cuptorul.

Remarcă: Nu trebuie modelat senzorul de temperatură. Nu trebuie proiectate numărătoarele MSI necesare.

Soluție

Am revăzut deja la problema rezolvată anteroară noțiunile fundamentale legate de *Unitatea de comandă* UC și *Unitatea de execuție* UE. Vom trece direct la proiectarea schemei bloc a sistemului.

UC și UE vor avea fiecare alt semnal de tact: UE semnalul de tact *CLOCK*₂ cu perioada de 1 minut, iar UC un semnal de tact *CLOCK*₁ mult mai rapid, cu frecvența de 10 kHz (aceasta este orientativă, poate avea și altă valoare).

Unitatea de execuție

În UE avem următoarele resurse:

- Două *timer*-e. Ele vor fi implementate prin numărătoare MSI cu *Clock enable*, intrare prin care sunt controlate de către UC; aceste

- Think carefully and specify which inputs are synchronous and / or asynchronous.
- Implement the digital system that controls the oven.

Remark: You do not need to model the temperature sensor. You do not need to design the necessary MSI counters.

Solution

We have already overviewed at the previous solved problem the basic concepts related to the *Command unit* CU and the *Execution unit* EU. We will pass directly to the design of the system's block scheme.

The CU and the EU will have each another clock signal: the EU the *CLOCK*₂ signal with a period of 1 minute, and the CU the *CLOCK*₁ signal, which is much faster, having a frequency of 10 kHz (this value is orientative; any other one could be used).

The Execution unit

In the EU we have the following resources:

- Two timers. They will be implemented by MSI counters with *Clock enable*. By this input they are

numărătoare vor funcționa pe $CLOCK_2$. Cele două numărătoare au și câte o intrare de *Reset* sincron. *Timer*-ele vor genera semnalul T_5 , respectiv T_{25} atunci când se atinge valoarea prevăzută.

- Senzorul de temperatură nu are nevoie de nici un semnal logic ca intrare; el va genera doar ieșirea TS .
- Încălzitorul va avea doar o singură intrare, *Heat*, primită de la UC. Când $Heat = 1$, încălzitorul funcționează; când $Heat = 0$, încălzitorul este oprit.

Unitatea de comandă

În UC avem următoarele intrări:

- *Start* – buton extern acționat de către utilizator. Întrucât utilizatorul poate acționa acest buton oricând (nu există nici o condiționare de vreun semnal de tact), este o intrare asincronă.
- semnalul T_5 , primit de la *timer*-ul de 5 minute din UE. Putem considera că cele două semnale de tact, $CLOCK_1$ și $CLOCK_2$ sunt sincronizate la începutul funcționării (de fapt, $CLOCK_2$ poate fi obținut prin divizarea semnalului $CLOCK_1$). Atunci T_5 este o intrare sincronă.
- semnalul T_{25} , primit de la *timer*-ul de 25 de minute din UE. Din aceleși considerente ca mai sus, și T_{25} este o intrare sincronă.
- semnalul TS , primit de la senzorul

controlled by the CU; these counters will work on $CLOCK_2$. The two counters also have a synchronous *Reset* input. The timers will generate the signals T_5 and T_{25} respectively, when the expected value is reached.

- The temperature sensor needs no logic signal as input; it will only generate the TS output.
- The heating element will only have one input, *Heat*, received from the CU. When $Heat = 1$, the heating element functions; when $Heat = 0$, the heating element is off.

The Command unit

We have the following inputs in the CU:

- *Start* – an external button pressed by the user. Since the user can press this button any time (there is no conditioning by any clock signal), this is an asynchronous input.
- the T_5 signal, received from the 5 minutes timer in the EU. We can consider the two signals, $CLOCK_1$ and $CLOCK_2$, to be synchronized at the beginning of the functioning period (in fact, $CLOCK_2$ can be obtained by dividing the $CLOCK_1$ signal). Then T_5 is a synchronous input.
- the T_{25} signal, received from the 25 minutes timer in the EU. For the same reasons as above, T_{25} is a synchronous input too.

de temperatură din UE. Întrucât acest semnal poate apărea independent de semnalele de tact, este o intrare asincronă.

Din UC avem următoarele ieșiri:

- z_{CET5} – ieșire care furnizează *Clock enable* (permite funcționarea) *timer-ului* de 5 minute din UE.
- $Reset_{T5}$ – ieșire care resetează *timer-ul* de 5 minute din UE.
- z_{CET25} – ieșire care furnizează *Clock enable* (permite funcționarea) *timer-ului* de 25 de minute din UE.
- $Reset_{T25}$ – ieșire care resetează *timer-ul* de 25 de minute din UE.
- *Heat* – ieșire care pornește încălzitorul din UE.
- *PH* – ieșire scoasă pe panoul frontal, vizibilă utilizatorului; indică faptul că ne aflăm în etapa de pre-încălzire a cuptorului.
- *IF* – ieșire scoasă pe panoul frontal, vizibilă utilizatorului; indică faptul că utilizatorul trebuie să introducă mâncarea de gătit în cuptor.
- *CK* – ieșire scoasă pe panoul frontal, vizibilă utilizatorului; indică faptul că mâncarea este în curs de a fi gătită în cuptor.

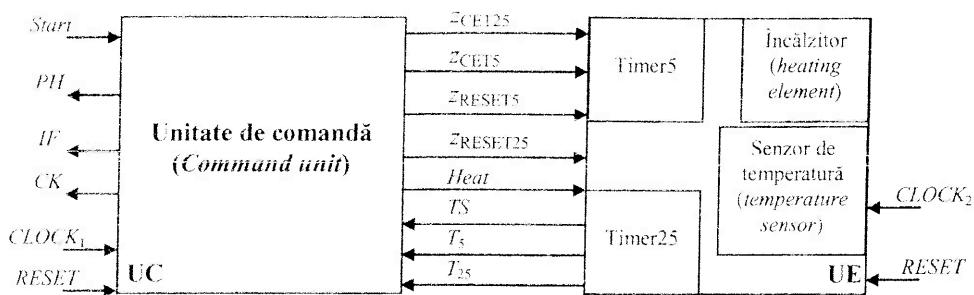
Schema bloc este redată în figura următoare:

- the TS signal, received from the temperature sensor from the EU. Since this signal can appear independently from the clock signals, it is an asynchronous input.

We have the following outputs from the CU:

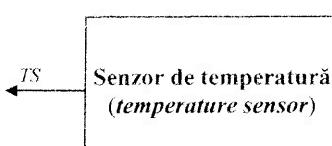
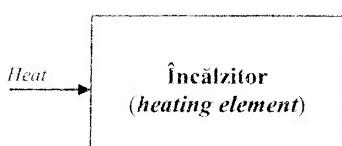
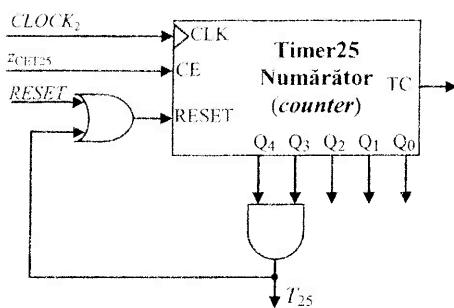
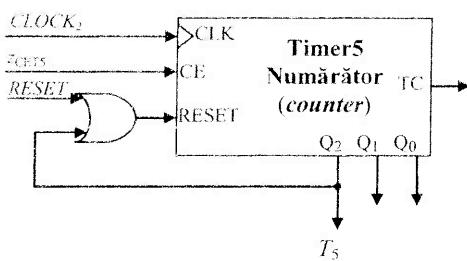
- z_{CET5} – output that provides *Clock enable* to the 5 minutes timer in the EU.
- $Reset_{T5}$ – output that resets the 5 minutes timer in the EU.
- z_{CET25} – output that provides *Clock enable* to the 25 minutes timer in the EU.
- $Reset_{T25}$ – output that resets the 25 minutes timer in the EU.
- *Heat* – output that starts the heating element in the EU.
- *PH* – output brought out on the frontal panel, visible to the user; indicates that we are in the oven pre-heating stage.
- *IF* – output brought out on the frontal panel, visible to the user; indicates that the user must insert the food to be cooked in the oven.
- *CK* – output brought out on the frontal panel, visible to the user; indicates that the food is now cooked in the oven.

The block scheme is shown in the next figure:



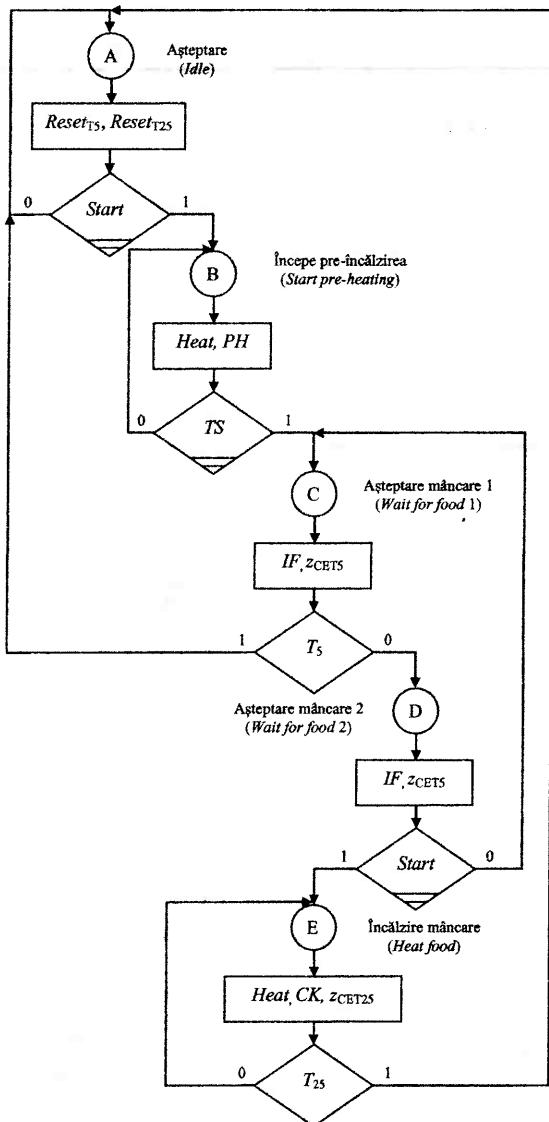
În UE trebuie să implementăm doar timer-ele. Timer5 este un numărător pe 3 biți, modulo 5, iar Timer25 este un numărător pe 5 biți, modulo 25.

In the EU we only need to implement the timers. Timer5 is a modulo 5 3-bit counter; Timer25 is a modulo 25 5-bit counter.



Organigrama UC este prezentată în figura următoare.

The CU's state diagram is presented in the next figure.



Observăm că ieșirile depend numai de starea internă, nu și de intrări, deci acest automat este o mașină Moore.

Dacă dorim să realizăm

We notice that the outputs depend only on the internal state, not on the inputs, so this automaton is a Moore machine.

If we wish to make the

implementarea cu ajutorul unei metode bazate pe bistabile, va trebui să avem adiacente următoarele stări: A și B; B și C; D și A; E și C; E și A.

Stările pot fi codificate astfel: A = 000, B = 001, C = 101; D = 010; E = 100.

implementation with a Flip-Flops-based method, we will need to encode adjacently the following states: A and B; B and C; D and A; E and C; E and A.

The states can be encoded as follows: A = 000, B = 001, C = 101; D = 010; E = 100.

		Q_0			
		00	01	11	10
		A ₀	B ₁	X ₃	D ₂
Q_2	0	A ₀	B ₁	X ₃	D ₂
	1	E ₄	C ₅	X ₇	X ₆

Q_1

Totuși, date fiind complexitatea sa, pentru implementarea UC alegem de data aceasta metoda bazată pe memorie și multiplexor. Pentru implementarea ieșirilor vom folosi un decodificator suplimentar, ținând cont că:

- ieșirile $reset_{T5}$ și $reset_{T25}$ sunt generate în starea A (000);
- ieșirea $Heat$ este generată în stările B (001) și E (100);
- ieșirea PH este generată în starea B (001);
- ieșirile IF și z_{CET5} sunt generate în stările C (101) și D (010);
- ieșirile CK și z_{CET25} sunt generate în starea E (100).

Harta memoriei este cea din tabelul următor:

However, because of its complexity, for implementing the CU we choose this time the method based on a memory and a multiplexer. To implement the outputs we will use a supplemental decoder, taking into account the following facts:

- outputs $reset_{T5}$ and $reset_{T25}$ are generated in state A (000);
- output $Heat$ is generated in states B (001) and E (100);
- output PH is generated in state B (001);
- outputs IF and z_{CET5} are generated in states C (101) and D (010);
- outputs CK and z_{CET25} are generated in state E (100).

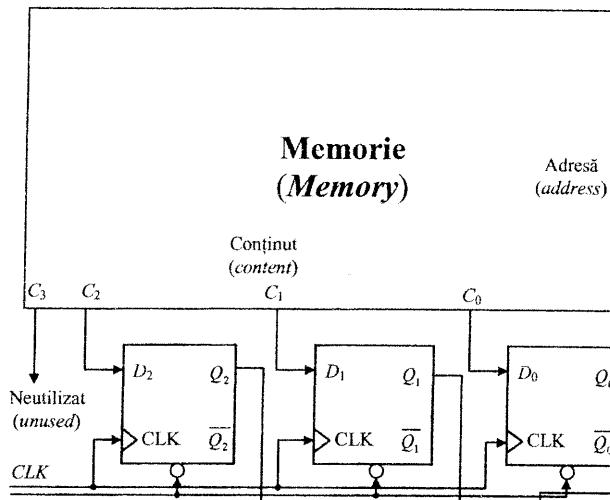
The memory map is the one from the next table:

Intrare + Starea curentă (Input + current state)				Starea următoare (next state)				
A_3	$A_2 = Q_2$	$A_1 = Q_1$	$A_0 = Q_0$	C_3	$C_2 = Q_2^+$	$C_1 = Q_1^+$	$C_0 = Q_0^+$	
0	0	0	0	X	0	0	0	
1	0	0	0	X	0	0	1	
0	0	0	1	X	0	0	1	
1	0	0	1	X	1	0	1	
0	0	1	0	X	1	0	1	
1	0	1	0	X	1	0	0	
0	0	1	1	X	X	X	X	
1	0	1	1	X	X	X	X	
0	1	0	0	X	1	0	0	
1	1	0	0	X	0	0	0	
0	1	0	1	X	1	0	0	
1	1	0	1	X	0	0	0	
0	1	1	0	X	X	X	X	
1	1	1	0	X	X	X	X	
0	1	1	1	X	X	X	X	
1	1	1	1	X	X	X	X	

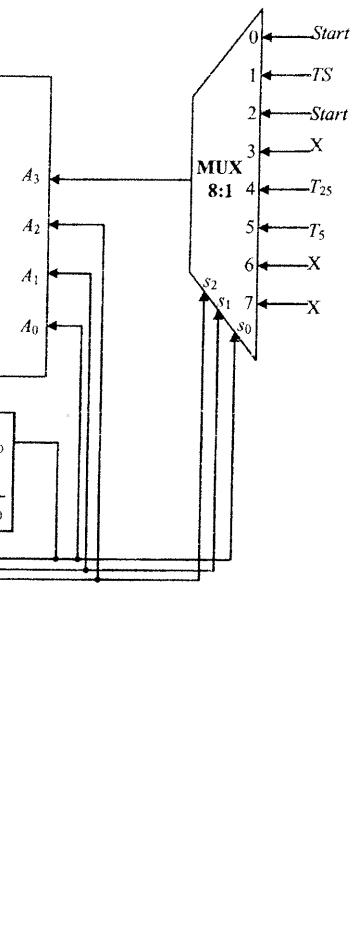
Pentru ieșiri creăm un tabel separat:

For the outputs we create a separate table:

În final, schema sistemului este următoarea:



Finally, the system's scheme is the following one:



4.6. Probleme propuse

- Proiectați cu bistabile de tip T un numărător sincron reversibil a cărui buclă de numărare conține numerele prime din intervalul 0-15. Auto-corecția și auto-inițializarea se vor realiza într-un singur tact.

4.6. Proposed problems

- Design using T Flip-Flops a reversible synchronous counter whose counting loop contains prime numbers in the 0-15 interval. The self-correction and self-initialization will be done in a single clock cycle.

2. Proiectați cu bistabile de tip D un numărător sincron reversibil a cărui buclă de numărare conține numerele $0 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 10 \rightarrow 0$. Rezolvați problema auto-corecției și auto-inițializării.

3. Proiectați un sistem numeric cu următorul comportament:

- pentru intrarea $A = 0$ semnalează numerele pare din intervalul $0 - 5$
 - pentru intrarea $A = 1$ semnalează numerele prime din intervalul $5 - 0$
- Auto-corecția și auto-inițializarea se vor realiza asincron.

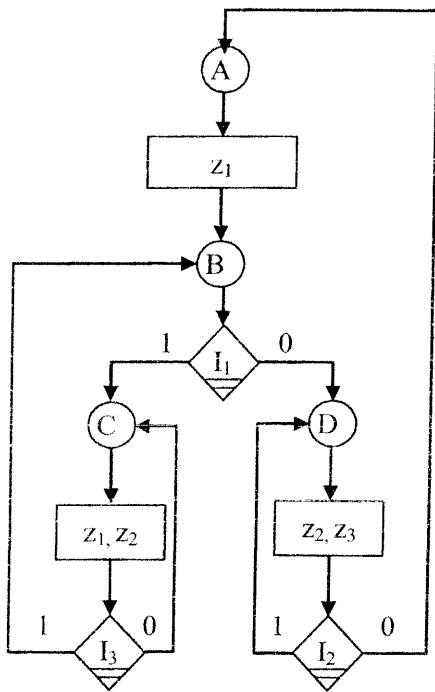
4. Proiectați sistemele logice secvențiale descrise de organigramele următoare:

2. Design using D Flip-Flops a reversible synchronous whose counting loop contains the numbers $0 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 10 \rightarrow 0$. Solve the self-correction and self-initialization problem.

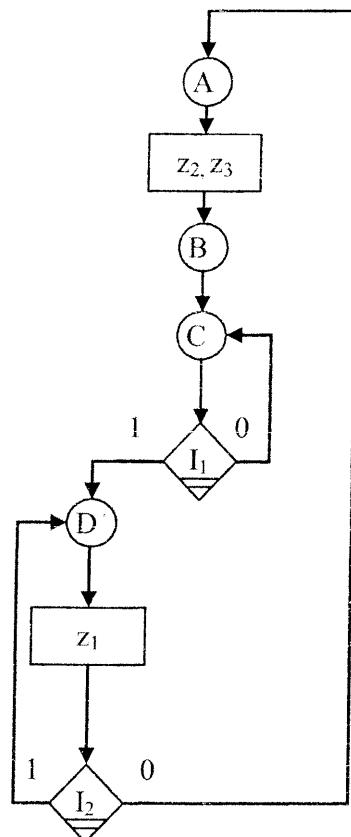
3. Design a digital system with the following behavior:

- for the input $A = 0$, it signals the even numbers in the $0 - 5$ interval
 - for the input $A = 1$, it signals the prime numbers in the $5 - 0$ interval
- The self-correction and self-initialization will be done asynchronously.

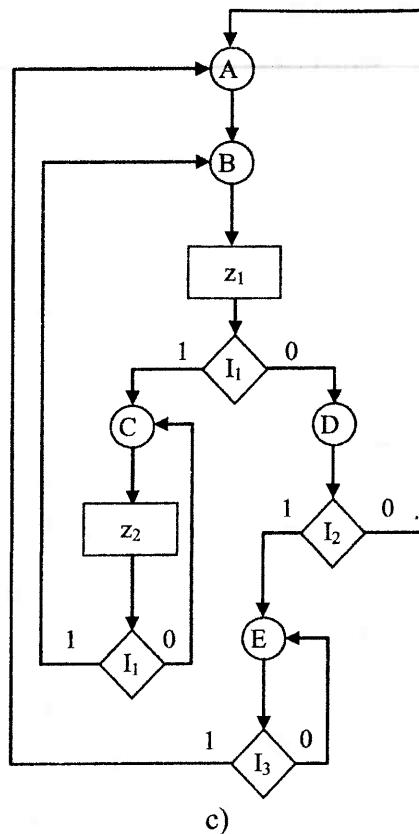
4. Design the sequential logic systems described by the following state diagrams:



a)



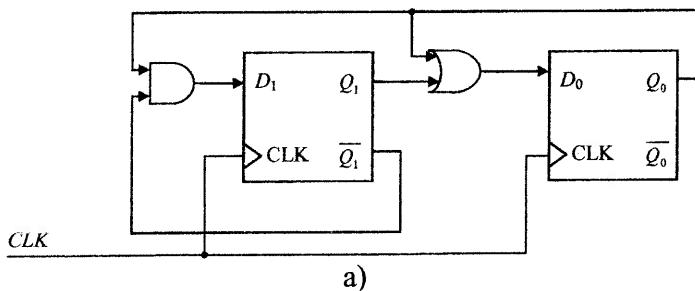
b)



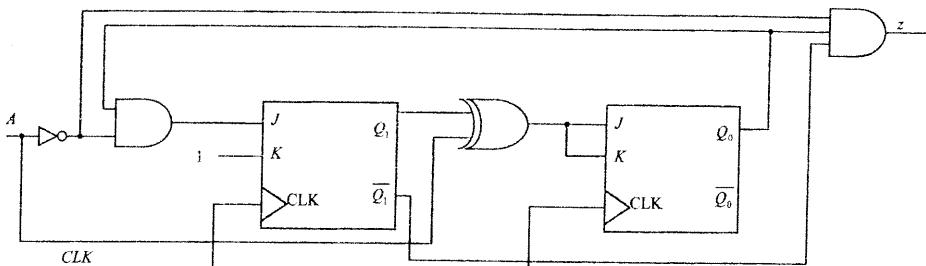
c)

5. Determinați grafurile de tranziții ale circuitelor de mai jos:

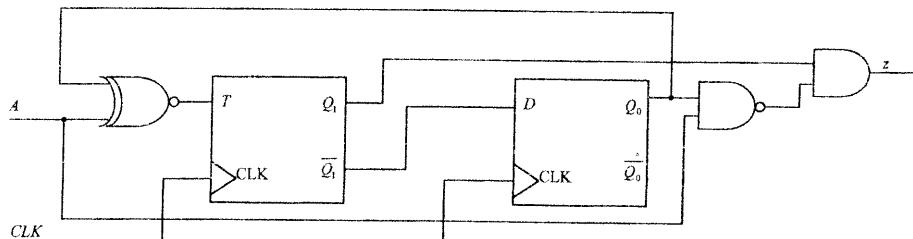
5. Determine the transition graphs of the following circuits:



a)



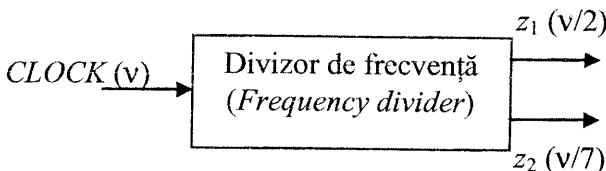
b)



c)

6. Proiectați un dispozitiv divizor de frecvență conform schemei:

6. Design a frequency divider with the following structure:



7. Determinați succesiunea stărilor următorului sistem numeric secvențial: pentru sistemul de la problema rezolvată 6, se conectează o poartă SAU-EXCLUSIV ale cărei intrări le constituie ieșirile Q_3 și Q_0 ale sistemului, ieșirea porții fiind dusă la *Serial In*. Regimul de funcționare este cu intrarea *MOD* = 0.

7. Determine the sequence of states for the following sequential logic circuit: for the system from the solved problem 6, connect an XOR logic gate whose outputs are the system's Q_3 and Q_0 outputs, the gate's output being connected to *Serial In*. The working regime is with *MOD* = 0.

8. Proiectați cu bistabile de tip JK un sistem logic secvențial care citește date de pe o linie serială și detectează apariția secvenței „0011” din sirul de intrare, afișând și numărul de asemenea secvențe detectate.

9. Proiectați un dispozitiv universal pe 4 biți cu următoarele regimuri de funcționare:

- a) încărcare paralelă
- b) deplasare dreapta
- c) numărare directă (*count-up*)
- d) reset

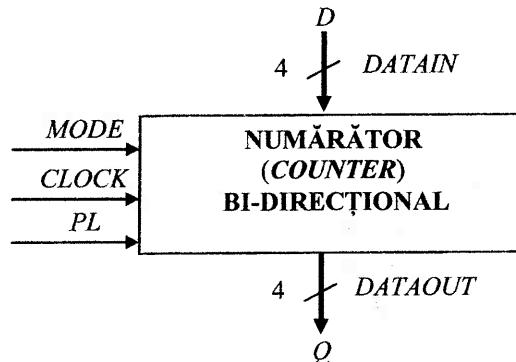
10. Proiectați cu bistabile de tip D un numărător bidirecțional. Dacă *PL* = 1, atunci codul prezent pe liniile de intrare *DATAIN* este încărcat în paralel, asincron, în numărător. Dacă *PL* = 0, atunci numărătorul numără crescător sau descreșcător, în funcție de *Mode*. Dacă *Mode* = 1, atunci numărătorul numără crescător. Dacă *Mode* = 0, atunci numărătorul numără descreșcător.

8. Design using JK Flip-Flops a sequential logic system that reads data from a serial input line and detects the appearance of the “0011” sequence in the input stream, also displaying the number of such detected sequences.

9. Design an universal 4-bit device with the following functioning regimes:

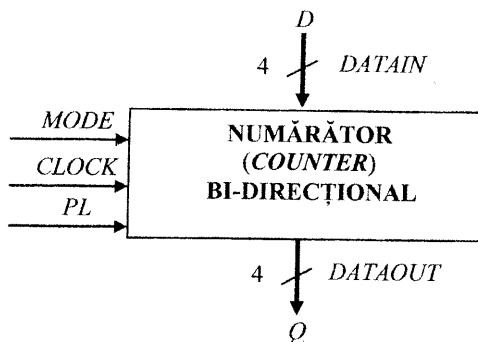
- a) parallel load
- b) shift right
- c) count-up
- d) reset

10. Design with D Flip-Flops a bi-directional counter. If *PL* = 1, then the code present on the *DATAIN* lines is loaded in parallel, asynchronously, in the counter. If *PL* = 0, then the counter counts up or down, according to *Mode*. If *Mode* = 1, then the counter counts up. If *Mode* = 0, then the counter counts down.



11. Proiectați cu bistabile de tip D un numărător bidirecțional. Dacă $PL = 1$, atunci codul prezent pe liniile de intrare $DATAIN$ este încărcat în paralel, sincron, în numărător. Dacă $PL = 0$, atunci numărătorul numără crescător sau descrescător, în funcție de $Mode$. Dacă $Mode = 1$, atunci numărătorul numără crescător. Dacă $Mode = 0$, atunci numărătorul numără descrescător.

11. Design with D Flip-Flops a bi-directional counter. If $PL = 1$, then the code present on the $DATAIN$ lines is loaded in parallel, synchronously, in the counter. If $PL = 0$, then the counter counts up or down, according to $Mode$. If $Mode = 1$, then the counter counts up. If $Mode = 0$, then the counter counts down.



12. Proiectați cu componente MSI și / sau SSI standard un sistem logic care citește date de pe o linie de intrare cu lățimea de 4 biți numită X și returnează la ieșirea pe 6 biți numită Y numărul $Y = X * 3$.

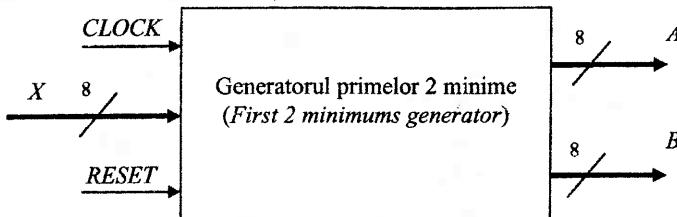
13. Pe o linie de date sosesc cuvinte de 8 biți în mod serial, sincron. Proiectați un sistem numeric care returnează primele două cele mai mici numere din *stream*-ul de intrare [A este numărul cel mai mic, B este numărul imediat următor după A ($A < B <$

12. Design using standard MSI and / or SSI components a logic system that reads data from a 4-bit input line called X and yields at the 6-bit output called Y the number $Y = X * 3$.

13. On a data line, 8-bit words come in a serial manner, synchronously. Design a digital system that yields the two smallest numbers from the input stream [A is the minimal number, B is the number following immediately after A ($A < B <$

oricare alt număr din stream-ul de intrare)].

any_other_number_from_the_input_stream)].



14. Un circuit secvențial are o intrare X și o ieșire Z . Proiectați un sistem numeric care face ca ieșirea Z să devină 1 dacă și numai dacă numărul total de biți de 1 recepționați este divizibil cu 3 sau dacă numărul total de biți de 0 recepționați este un număr par mai mare decât 0. (Indicație: nouă stări sunt suficiente).

15. Proiectați un numărător binar a cărui buclă conține numerele multiplu de 5 din intervalul 0-30 (0, 5, 10 etc.)

16. Un controlor de semafor pentru pietoni funcționează după cum urmează: ledul ROȘU este APRINS și ledul VERDE este STINS. Dacă cineva dorește să traverseze strada, el / ea apasă un buton. Imediat după apăsarea butonului, ledul ROȘU mai rămâne aprins încă 5 (cinci) secunde, apoi se stinge. Simultan, ledul VERDE se APRINDE timp de 22 de secunde. După aceste 22 de

14. A sequential circuit has one input X and one output Z . Design a digital system that makes the output Z become 1 if and only if the total number of bits of 1 that have been received is divisible by 3 or if the total number of bits of 0 that have been received is an even number greater than 0 (Hint: nine states are sufficient).

15. Design a binary counter whose counting loop contains the multiple of 5 numbers from the 0-30 interval (0, 5, 10, etc.)

16. A traffic light controller for pedestrians functions as follows: The RED light is ON and the GREEN light is OFF. If somebody wants to cross the street, he / she presses a button. Immediately after the button has been pressed, the RED stays ON 5 (five) more seconds, then it shuts down. Simultaneously, the GREEN light goes ON for 22 seconds. After those

secunde, ledul VERDE clipește de 5 ori în 5 secunde, după care se stinge și din nou, ledul ROȘU este APRINS. Timp de 32 de secunde, apăsarea butonului de către pietoni nu are nici un efect (ledul ROȘU va fi APRINS și ledul VERDE STINS), după care apăsarea butonului de către pietoni are din nou efect. Implementați un sistem numeric având comportamentul descris mai sus. Există disponibil un semnal de TACT cu frecvență de 1 Hz și un factor de umplere de 50%.

17. Faceți parte dintr-o echipă care are de proiectat un cuptor electric compact și ieftin, conceput special pentru camere de cămin studențesc. Cuptorul trebuie să aibă un comportament deosebit de simplu: el va găti întotdeauna la aceeași temperatură (200°C) și timp de 20 de minute (mereu aceeași perioadă). Cuptorul trebuie să funcționeze după cum urmează. Starea inițială este una de așteptare, iar încălzitorul este stins. Când se apasă butonul de „Start” al cuptorului, încălzitorul trebuie pornit pentru a pre-încălzi cuptorul (deci pentru a-l aduce la temperatura de gătit). În timp ce se realizează aceasta, ledul „Preheating” („PH”) trebuie aprins pe panoul frontal al cuptorului. În interiorul cuptorului există un senzor de temperatură („TS”) care îi

22 seconds, the GREEN light blinks 5 times in 5 seconds, then it goes OFF and, again, the RED light is ON. For 32 seconds, pressing the button by pedestrians has no effect (the RED light will be ON and the GREEN light will be OFF), then pressing the button by pedestrians is effective again. Implement a digital system with the behavior described above. There is a CLOCK signal with a frequency of 1 Hz and a duty cycle of 50% available.

17. You are part of a team designing a compact and inexpensive electronic oven, especially designed for very small flats or student halls of residence. The oven is to have a particularly simple operation: it will always cook to the same temperature (200°C) and for the same time (20 minutes).

The oven is to operate as follows. The oven is initially in an idle state and the heating element is switched off. When the “Start” button of the oven is pressed, the heating element should turn on to pre-heat the oven (i.e. to bring it up to its cooking temperature). While it is doing this, a “Preheating” (“PH”) lamp should light up on the front of the cooker. A temperature sensor (“TS”) inside the oven indicates to the oven controller when the oven is hot

semnalează unității de control a cuptorului momentul când cuptorul este suficient de bine încălzit (200°C) pentru a se putea începe gătitul. În acest moment se va aprinde un led „Insert Food” („IF”), pentru ca utilizatorul să știe că acum cuptorul este pregătit pentru a găti mâncarea.

După ce pune mâncarea în cuptor, utilizatorul apasă din nou butonul „Start”, iar încălzitorul va rămâne pornit pentru a găti mâncarea un timp pre-stabilit, de 20 de minute. Pe durata acestui interval de timp, ledul „Cooking” trebuie să fie aprins. După încheierea procesului de gătit, încălzitorul trebuie oprit și ledul „Cooking” trebuie stins. În plus față de regimul de funcționare normal descris mai sus, cuptorul trebuie să aibă următoarea funcționalitate de siguranță: după terminarea etapei de pre-încălzire, dacă utilizatorul nu introduce mâncarea și nu apasă „Start” în 5 minute, cuptorul se va închide automat și va reveni în starea de așteptare.

Senzorul de temperatură există deja. Timer-ele de 5 și de 20 de minute vor fi construite de dumneavoastră folosind numărătoare MSI. Este disponibil un semnal de tact cu perioada de 1 minut. Senzorul de temperatură funcționează non-stop, iar când se atinge temperatura de

enough (200°C) to begin cooking. At this point, an “Insert Food” (“IF”) lamp turns on to tell the user that the oven is now ready to cook their meal.

After putting the item to be cooked into the oven, the user presses “Start” again, and the heating element will remain on to cook the food for the pre-set time of 20 minutes. During this time, the “Cooking” (“CK”) lamp should light. After cooking has completed, the heating element should turn off and the “Cooking” lamp should go out. In addition to the above normal operation, the oven is to have the following safety feature: once pre-heating is complete, if the user does not insert their item and press “Start” within 5 mins, the oven will switch off automatically and return to its idle state.

The temperature sensor is already designed. The 5 min and 20 min timers will be built by you using MSI counters. A clock signal with the period of 1 minute is available. The temperature sensor is always enabled, and once the temperature reaches the required value of 200°C , it produces the output “TS”. The 20 min timer is started when the oven is in its cooking state, and sets its output, T_{20} , to ‘1’ when the 20

200°C, el generează ieșirea „TS”. Timer-ul de 20 de minute este pornit când cuptorul este în starea în care gătește, și generează ieșirea $T_{20} = 1$ după trecerea a 20 de minute. Timer-ul de 5 de minute este pornit de îndată ce cuptorul este suficient de încălzit pentru a începe să gătească și așteaptă ca utilizatorul să introducă mâncarea. După trecerea a 5 minute, timer-ul generează ieșirea $T_5 = 1$.

- Desenați schema bloc a sistemului. Evidențiați Unitatea de execuție și Unitatea de control. Indicați ce ieșiri sunt vizibile pe panoul frontal (cel pe care-l vede utilizatorul) și ieșirile care sunt interne sistemului.
- Trasați organograma unității de control a cuptorului descrise mai sus. Determinați cu atenție câte stări sunt necesare, inclusiv starea de așteptare. Determinați de asemenea care sunt intrările și ieșirile sistemului, știind că anumite ieșiri aprind ledurile, pe când altele pornesc timer-ele. Ieșirile senzorului de temperatură și ale timerelor vor constitui intrări în UC a sistemului nostru.
- Precizați care intrări sunt sincrone și / sau asincrone.
- Implementați sistemul numeric care controlează cuptorul.

Remarcă: Nu trebuie modelat senzorul de temperatură. Nu trebuie proiectate numărătoarele MSI necesare.

minutes are over. The 5 min timer is started as soon as the oven is hot enough to start cooking and waiting for the user to insert the item to be cooked. When the 5 min are up, the timer sets its output, T_5 , to ‘1’.

- Draw the system's block diagram. Highlight the Execution Unit and the Control Unit. Show the outputs that are visible on the front panel (the one that the user sees) and the outputs that are internal to the system.
- Draw a state diagram for the simple oven controller described above. Think carefully about how many states you will require, including the idle state. Also consider what are the inputs and outputs of the system, bearing in mind that some outputs light up the lamps, while others start the timers. The outputs of the temperature sensor and timers will act as inputs to your system.
- Think carefully and specify which inputs are synchronous and / or asynchronous.
- Implement the digital system that controls the oven.

Remark: You do not need to model the temperature sensor. You do not need to design the necessary MSI counters.

Capitolul 5. Probleme propuse finale

În acest capitol propunem o serie de probleme cu caracter sintetic, sumativ, care înglobează toate aspectele prezentate în cadrul cursului de Proiectare logică (Logic Design). Pentru rezolvarea lor sunt necesare și cunoștințe care nu au fost prezentate în paginile acestei Culegeri, dar care sunt bineînțeles expuse în cadrul unui curs de specialitate.

5.1. Probleme propuse

1. La un concurs de admitere la facultate, fiecare candidat este identificat printr-un cod binar propriu, unic. Proiectați un sistem numeric care repartizează fiecare candidat în sala stabilită, pe măsură ce aceștia se prezintă.
2. Se dă funcția Booleană: $f = B \bullet \bar{C} + \bar{A} \bullet C$. Se cere să se implementeze funcția cu eliminarea hazardului combinational. Explicați.
 $f = \Sigma(0, 1, 5, 6, 8, 14, 15)$.
3. Arătați că MULTIPLEXORUL este o funcție universală și demonstrați pentru implementarea funcției Booleene

Chapter 5. Final proposed problems

In this chapter we propose a series of problems of a synthetic, summative character, that comprise all aspects presented in the Logic Design course. In order to solve them, it is necessary make usage of knowledge that has not been presented in this book, but which are of course exposed during the lectures.

5.1. Proposed problems

1. At an University admission contest, each candidate is identified by an unique, own, binary code. Design a digital system that distributes each candidate in the appointed room, as they present themselves to the exam.
2. Be given the Boolean function: $f = B \bullet \bar{C} + \bar{A} \bullet C$. It is requested to implement the function, eliminating the combinational hazard. Explain.
 $f = \Sigma(0, 1, 5, 6, 8, 14, 15)$.
3. Show that the MULTIPLEXER is an universal function and prove it for the implementation of the Boolean function
 $f = \Sigma(0, 1, 5, 6, 8, 14, 15)$.

4. Arătați care componente sunt cele mai adecvate pentru implementarea următoarelor construcții logice:

- a) IF <expresie_logică_1> THEN 0 ELSE <expresie_logică_2>
- b) IF <expresie_logică_1> THEN 1 ELSE <expresie_logică_2>
- c) IF <expresie_logică_1> THEN <expresie_logică_2> ELSE <expresie_logică_3>

5. Implementați funcțiile Booleene $f_1 = \Sigma(0, 2, 6, 7, 8, 10, 14, 15)$ și $f_2 = \Sigma(0, 2, 8, 10)$ cu un dispozitiv PLA cu 4 intrări și 4 ieșiri. Indicați conexiunile făcute și modul de funcționare.

6. Realizați Maparea Tehnologică a următoarelor circuite folosind următoarele celule logice:

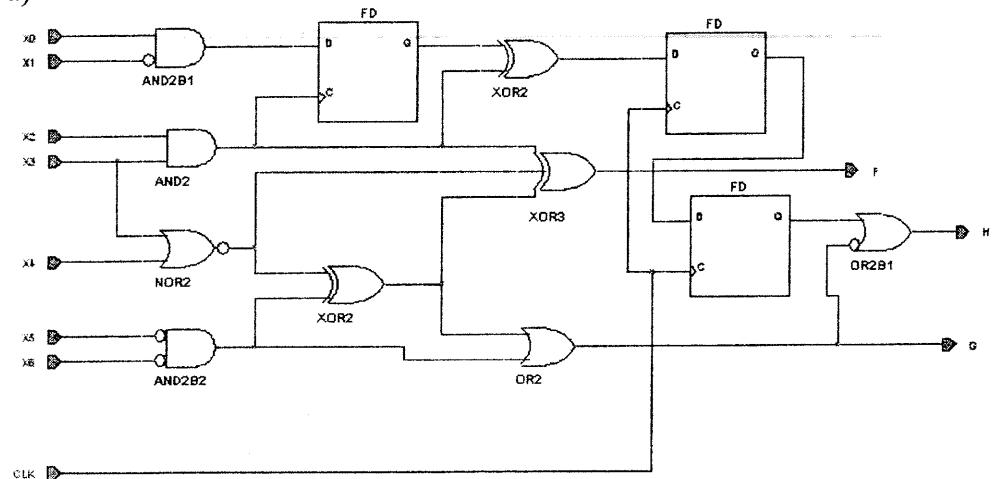
4. Show which are the most appropriate components to implement the following logic constructs:

- a) IF <logic_expression_1> THEN 0 ELSE <logic_expression_2>
- b) IF < logic_expression_1> THEN 1 ELSE < logic_expression_2>
- c) IF < logic_expression_1> THEN < logic_expression_2> ELSE < logic_expression_3>

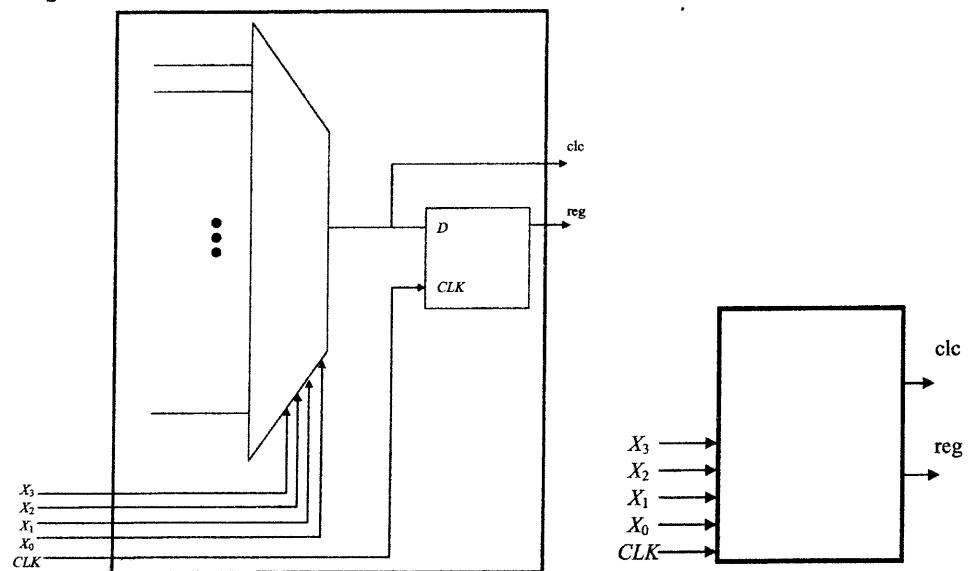
5. Implement the Boolean functions $f_1 = \Sigma(0, 2, 6, 7, 8, 10, 14, 15)$ and $f_2 = \Sigma(0, 2, 8, 10)$ with a 4-input, 4-output PLA device. Indicate the connections made and its functioning mode.

6. Realize the Technology Mapping of the following circuits using the following logic cells:

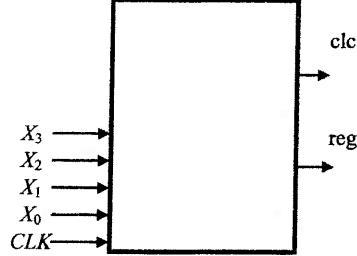
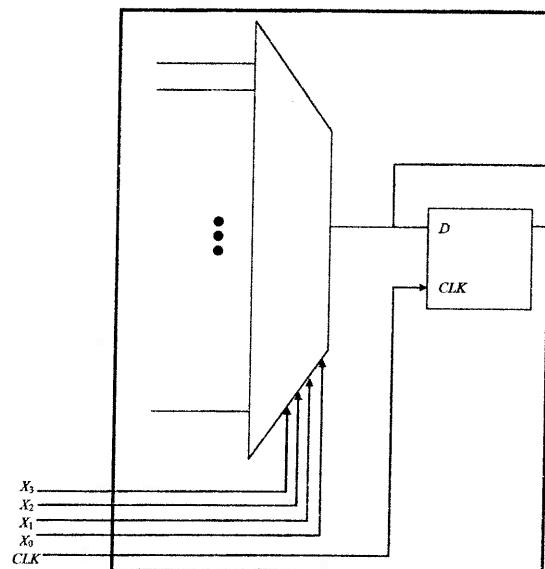
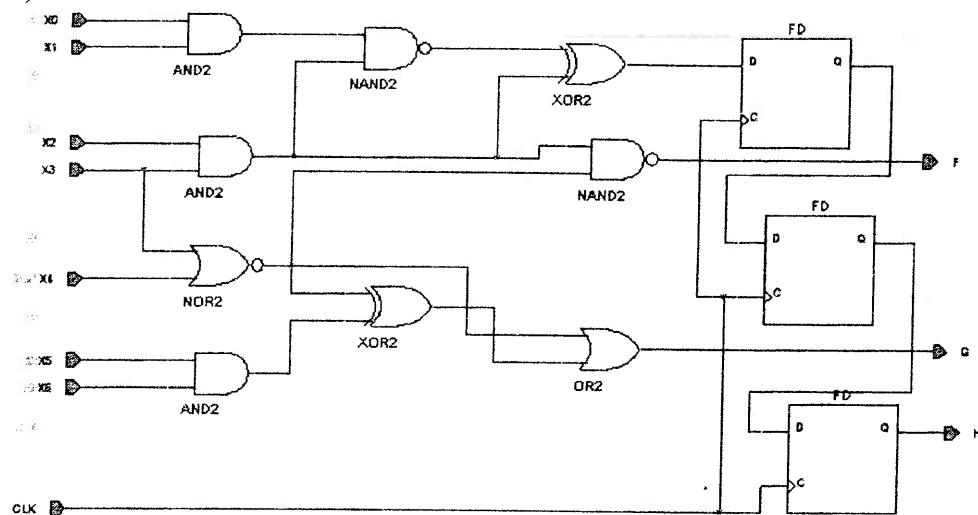
a)



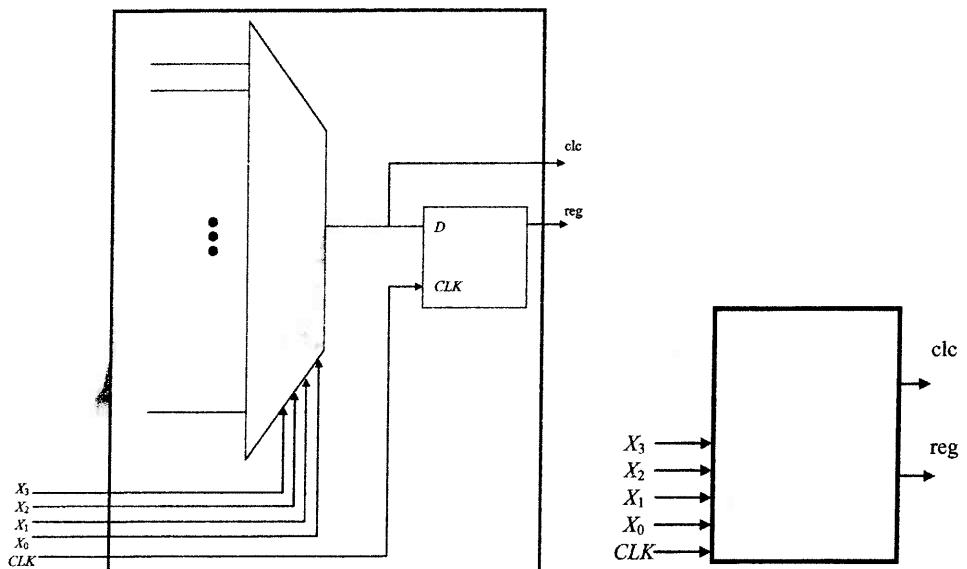
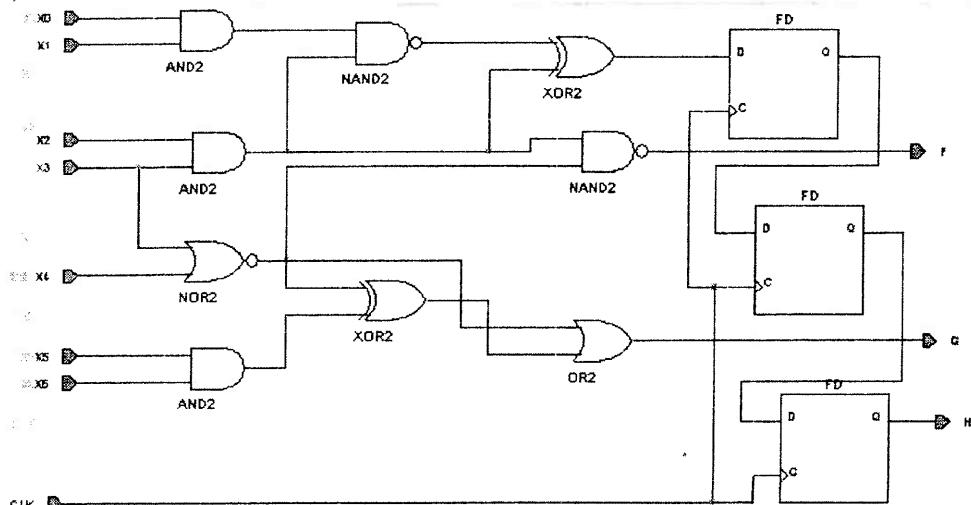
CLK



b)



c)



6. Proiectați un generator de secvențe pseudo-aleatoare pe 4 biți, cu o secvență maximă de lungime

6. Design a 4-bit pseudo-random sequence generator, having a maximal sequence of length 10 (the

10 (starea inițială se repetă după 10 impulsuri de tact).

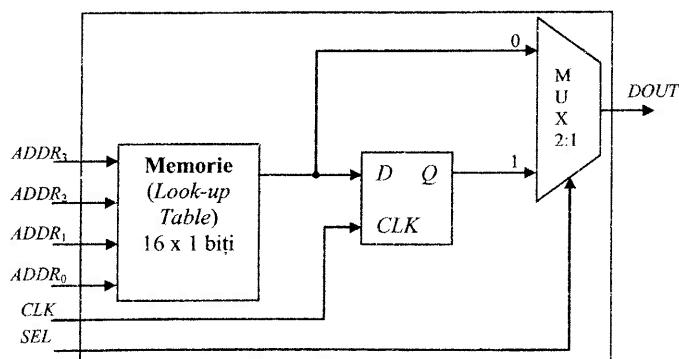
7. a) Construiți CLB-ul unui dispozitiv FPGA care conține: 1) un Generator de Funcții pe 4-bit (cu completitudine funcțională și care este în același timp o funcție universală); 2) 2 (două) bistabile de tip D; 3) o ieșire combinațională; 3) 2 ieșiri prin registru.
 b) Rezolvați problema Mapării Tehnologice pentru arhitectura propusă la problema 6), folosind CLB-uri de acest tip (cel determinat la punctul a)).

8. Realizați Maparea Tehnologică a circuitului de la problema 6, presupunând că aveți la dispoziție CLB-ul reprezentat mai jos. Determinați de asemenei Harta memoriei și valoarea semnalelor SEL.

initial state is repeated after 10 clock cycles).

7. a) Build the CLB of an FPGA device that contains: 1) a 4-bit Function Generator (functionally complete and universal function); 2) 2 (two) D Flip-Flops; 3) a combinational output; 3) 2 buffered (registered) outputs.
 b) Solve the Technology Mapping problem for the architecture you propose for problem 6), using CLBs of this type (the one determined at point a)).

8. Realize the Technology Mapping of the circuit of problem 6, assuming you have at your disposition the CLB presented below. Determine also the Memory maps and the value of the SEL signals.



9. Proiectați schema internă (cu porți logice) a unui Decodificator 3:8 și schema internă a unui Codificator priorităr 8:3. Implementați un modul de memorie 8×2 cu următoarea hartă a memoriei ("C" înseamnă "Conținut"): $C(0) = \text{"00"}, C(1) = \text{"10"}, C(2) = \text{"11"}, C(3) = \text{"00"}, C(4) = \text{"01"}, C(5) = \text{"01"}, C(6) = \text{"10"}, C(7) = \text{"00}"$ alcătuit din Decodificatorul proiectat mai sus și un Codificator priorităr adecvat. Proiectați schema internă (cu porți logice) a acestui Codificator priorităr.

10.

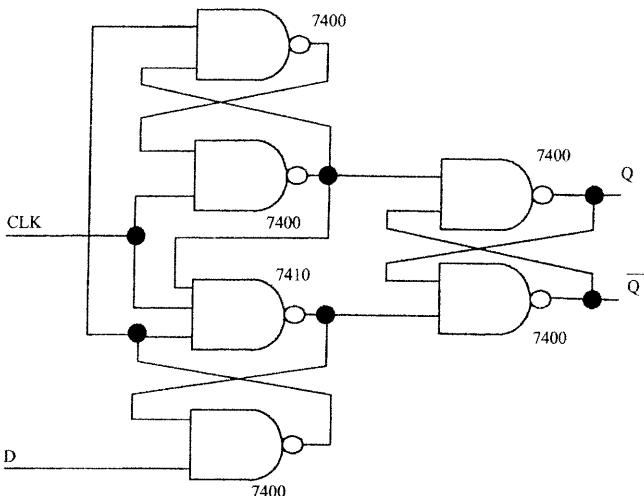
- a) Explicați care este diferența dintre un bistabil de tip ZĂVOR (*LATCH*) și un bistabil de tip *Flip-Flop*.
- b) Schema următoare reprezintă un bistabil de tip D *Flip-Flop* cu acționare pe frontul ascendent. Explicați modul său de funcționare.
(*Indicație:* utilizați semnale intermedii pentru a facilita procesul de explicare).

9. Design the internal scheme (with logic gates) of a 3-to-8 Decoder and the internal scheme of an 8-to-3 Priority Encoder. Implement an 8×2 Memory module with the following memory map ("C" means "Content"): $C(0) = \text{"00"}, C(1) = \text{"10"}, C(2) = \text{"11"}, C(3) = \text{"00"}, C(4) = \text{"01"}, C(5) = \text{"01"}, C(6) = \text{"10"}, C(7) = \text{"00"}$ composed of the previously designed Decoder and an appropriate Encoder. Design the internal schematic (with logic gates) of this Priority Encoder.

10.

- a) Explain what is the difference between a LATCH and a Flip-Flop.
- b) The following scheme represents a positive edge-triggered D Flip-Flop. Explain its functioning.

(*Hint:* use intermediate signals to facilitate the explaining process).



11. Proiectați cu bistabili JK o memorie FIFO de capacitate 4×8 biți și explicați funcționarea sa.

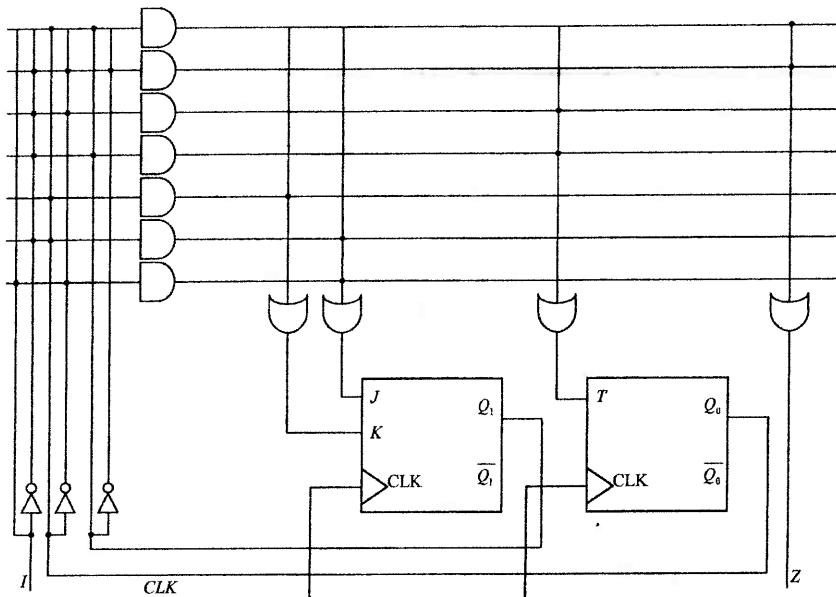
12. Proiectați cu bistabili D o memorie LIFO de capacitate 4×8 biți și explicați funcționarea sa. Este necesar să specificați structura internă la nivel de bistabili și porți.

13. Deduceți graful (sau tabelul) de tranzitii al implementării automatului finit din figura următoare (funcțiile corespunzătoare stării următoare și ieșirii sunt implementate cu ajutorul unei structuri PLA). Automatul are o intrare I și o ieșire Z .

11. Design with JK Flip-Flops a 4×8 bits FIFO memory and explain its functioning.

12. Design with D Flip-Flops a 4×8 bits LIFO memory and explain its functioning. You must specify its structure at the Flip-Flops and gates level.

13. Derive the state transition graph (or table) for the implementation of the finite state machine of the next figure (the next-state and output functions are implemented by a PLA structure). The machine has one input I and one output Z .



14. (Problemă de logică)

Într-un dulap sunt 3 borcane de compot: unul de cireșe, unul de vișine și unul de cireșe + vișine. Cineva a lipit însă greșit toate etichetele de pe borcane. Cum se poate determina conținutul corect al fiecărui borcan scoțând un singur fruct dintr-un singur borcan?

14. (Logic problem)

In a repository there are three pots of compote: one of cherries, one morello cherries, and one which is a mixture of cherries and morello cherries. Somebody put the labels all wrong on these pots. How can we determine the correct content of each pot by extracting a single fruit from a single pot?

Bibliografie

References

1. Văcariu, L., Creț, O. *Analiza și sinteza dispozitivelor numerice. Îndrumător de laborator.* Cluj-Napoca, Editura U.T. Press, 2005.
2. Creț, O. *Sisteme de calcul reconfigurabile.* Cluj-Napoca, Editura U. T. Pres, 2005.
3. Jenkins, J. *Designing with FPGAs and CPLDs.* New York, Prentice Hall, 1994.
4. Katz, R., Borriello, G. *Contemporary Logic Design.* New York, Prentice Hall, 2005.
5. Nedevschi, S., Baruch, Z., Creț, O. *Proiectarea sistemelor numerice folosind tehnologia FPGA.* Cluj-Napoca, Editura Mediamira, 1999.
6. Toacșe, Gh., Nicula, D., *Electronică digitală – Dispozitive. Circuite. Proiectare.* București, Ed. Tehnică, 2005.
7. Ștefan, Gh., Bistrițeanu, V. *Circuite integrate digitale. Probleme, proiectare.* Cluj-Napoca, Editura Albastră, 2000.
8. Wakerly, JF. *Digital Design Principles and Practices.* Prentice Hall, 2000.
9. Wirth, N. *Digital Circuit Design.* Zurich, Springer-Verlag, 1995.
10. *** *ActiveCAD schematic editor - reference manual.* ALDEC, 1996.



EDITURA U.T.PRESS

str. Observatorului nr.34

C.P.42, O.P. 2

400775 Cluj-Napoca

e-mail: utpress@biblio.utcluj.ro

ISBN 978-973-662-412-4



9 789736 624124