PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

IBN-KHALDOUN UNIVERSITY IN TIARET

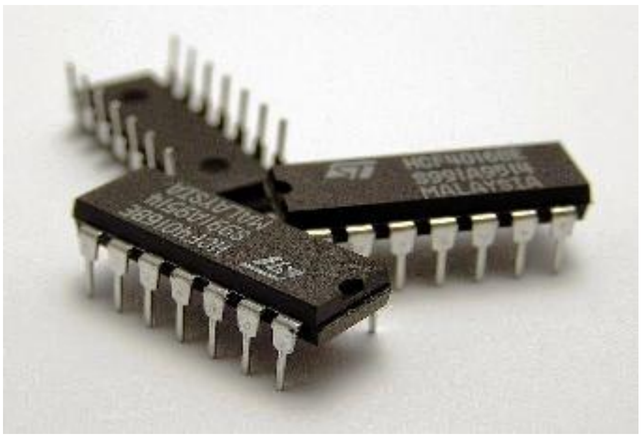
FACULTY OF APPLIED SCIENCES
ELECTRICAL ENGINEERING DEPARTMENT

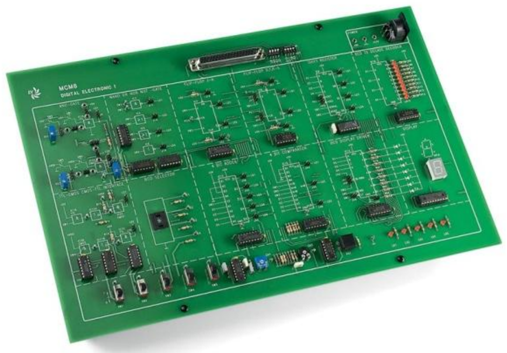
COURSE HANDOUT

COMBINATORIAL AND SEQUENTIAL LOGIC

(Course)

Prepared by: **BERKANI Abderrahmane**





Promotion :	2nd year engineer, electrical engineering specialty
Semester:	03

Reviewed by:
Pr : SAHLI Belguacem
Pr : NASRI Djilali

Academic year: 2024/2025

Table of Contents

Chapter 1: Numbering system and coding of information

1. Objectives:....................1
2. Numbering systems....................1
3. Change of base....................3
4. Operations in the bases....................6
5. Coding of information....................11

Chapter 2: BOOLE Algebra and Logical Functions

1. Objectives....................16
2. Variables and logical functions....................16
3. Basic operations of BOOLE algebra and associated properties....................17
4. Materialization of logical operators....................18

Chapter 3: Representation and simplification of combinatorial logic functions

1. Objectives....................25
2. Representation of a logical function....................25
3. Simplification of logical functions....................32
4. Summary: Synthesis of a logical function....................37

Chapter 4: Combinational Logic Circuits

1. Objectives....................38
2. Arithmetic circuits....................38

Chapter 5: : Sequential Logic

1. Objectives....................59
2. Introduction....................59
3. Asynchronous Switches....................60
4. Synchronous Flip-flops....................65

Chapter 6: : The Registers

1. Objectives....................71
2. General information....................71

3. Storage register..........71

4. Shift register..........72

5. Mixed register..........74

Chapter 7: The Counters

1. Objectives..........76

2. Introduction..........76

3. Asynchronous counters and downcounters..........76

4. Synchronous counters and downcounters..........82

Chapter 7: The Counters

1. Objectives..........85

2. Introduction..........85

3. Examples..........86

Bibliographic References..........95

CHAPTER 1
NUMBERING SYSTEMS AND CODING OF INFORMAT

1. OBJECTIVES

- Cover in detail the different number systems: decimal, binary, octal and hexadecimal systems as well as the methods of conversion between number systems.
- Deal with arithmetic operations on numbers.
- Study several digital codes such as DCB, GRAY and ASCII codes.

2. NUMBER SYSTEMS

For digital information to be processed by a circuit, it must be put into a form suitable for it. To do this, a base B numbering system must be chosen (B is a natural whole number - 2).

There are many numbering systems used in digital technology. The most commonly used are: Decimal (base 10), Binary (base 2), Tetral (base 4), Octal (base 8) and Hexadecimal (base 16).

The table below represents a summary of these systems:

Decimal	Binary	Tetral	Octal	Hexadecimal
0	0	0	0	0
1	1	1	1	1
2	10	2	2	2
3	11	3	3	3
4	100	10	4	4
5	101	11	5	5
6	110	12	6	6
7	111	13	7	7
8	1000	20	10	8
9	1001	21	11	9
10	1010	22	12	HAS
11	1011	23	13	B
12	1100	30	14	C
13	1101	31	15	D
14	1110	32	16	E
15	1111	33	17	F

2.1 Polynomial representation

Any number N can be decomposed into integer powers of the base of its numbering system. This decomposition is called the polynomial form of the numberNand which is given by:

$$\boxed{N=a_nB_n+ a_{n-1}B_{n-1}+ a_{n-2}B_{n-2}+ ...+ a_2B_2+ a_1B_1+ a_0B_0}$$

- B: The basis of the numbering system, it represents the number of different digits that this numbering system uses.
- hasi:a number (or digit) among the numbers in the base of the numbering system.
- i: rank of the numberhasi.

2.2Decimal system (base 10)

The decimal system consists of 10 digits which are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} it is a system that has imposed itself quite naturally on man who has 10 fingers. Let us write some decimal numbers in polynomial form:

Examples:

$(5462)_{10}= 5*10_3+ 4*10_2+ 6*10_1+ 2*10_0$

$(239,537)_{10}= 2*10_2+ 3*10_1+ 9*10_0+ 5*10_{-1}+ 3*10_{-2}+ 7*10_{-3}$

2.3 Binary system (base 2)

In this number system there are only two possible digits {0, 1} which are often called bits "binary digit". As the following examples show, a binary number can be written in polynomial form.

Examples:

$(111011)_2= 1*2_5+ 1*2_4+ 1*2_3+0*2_2+ 1*2_1+ 1*2_0$

$(10011.1101)_2= 1*2_4+ 0*2_3+ 0*2_2+ 1*2_1+ 1*2_0+ 1*2_{-1}+ 1*2_{-2}+ 0*2_{-3}+ 1*2_{-4}$

2.4Tetral system (base 4)

This system, also called base 4, includes four possible digits {0, 1, 2, 3}. A tetral number can be written in polynomial form as shown in the following examples:

Examples:

$(2331)_4= 2*4_3+ 3*4_2+ 3*4_1+ 1*4_0$ $(130.21)_4= 1*4_2+ 3*4_1+1*4_0+ 2*4_{-1}+ 1*4_{-2}$

2.5 Octal System (base 8)

The octal or base 8 system consists of eight digits which are {0, 1, 2, 3, 4, 5, 6, 7}. The digits 8 and 9 do not exist in this base. For example, let's write the numbers 45278and 1274,6328:

Examples:

$(4527)_8 = 4*8^3 + 5*8^2 + 2*8^1 + 7*8^0$

$(1274,632)_8 = 1*8^3 + 2*8^2 + 7*8^1 + 4*8^0 + 6*8^{-1} + 3*8^{-2} + 2*8^{-3}$

2.6 Hexadecimal system (base 16)

The Hexadecimal or base 16 system contains sixteen elements which are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}. The digits A, B, C, D, E, and represent 10, 11, 12, 13, 14 and 15 respectively.

Examples:

$(3256)_{16} = 3*16^3 + 2*16^2 + 5*16^1 + 6*16^0$ $(9C4F)_{16} = 9*16^3 + 12*16^2 + 4*16^1 + 15*16^0$
$(A2B.E1)_{16} = 10*16^2 + 2*16^1 + 11*16^0 + 14*16^{-1} + 1*16^{-2}$

3. CHANGE OF BASIS

This is the conversion of a number written in a base$B_1$to its equivalent in another baseB_2

3.1 Converting a base B number N to a decimal number

The decimal value of a number N, written in a database B, is obtained by its polynomial form described previously.

Examples:

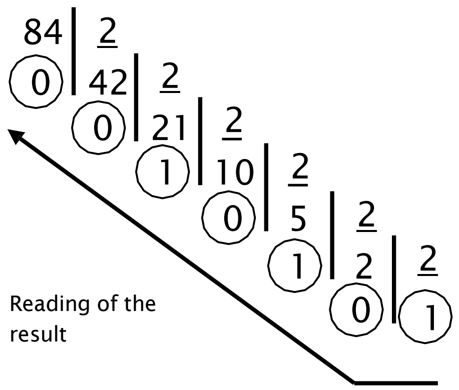
$(1011101)_2 = 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = (93)_{10}$ $(231102)_4 = 2*4^5 + 3*4^4 + 1*4^3 + 1*4^2 + 0*4^1 + 2*4^0 = (2898)_{10}$ $(7452)_8 = 7*8^3 + 4*8^2 + 5*8^1 + 2*8^0 = (3882)_{10}$
$(D7A)_{16} = 13*16^2 + 7*16^1 + 10*16^0 = (3450)_{10}$

3.1.1 Conversion of a decimal whole number

To convert a whole decimal number to a base numberBany, it is necessary to make successive integer divisions by the baseBand keep the remainder of the division each time. We stop when we obtain a result less than* the baseB. The number searchedNin the baseBis written from left to right, starting with the last result and going to the first remainder.

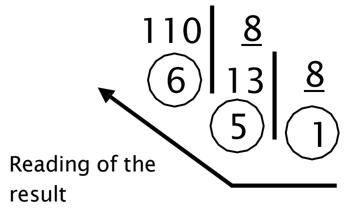
Examples:

- $(84)_{10}=(?)_2$



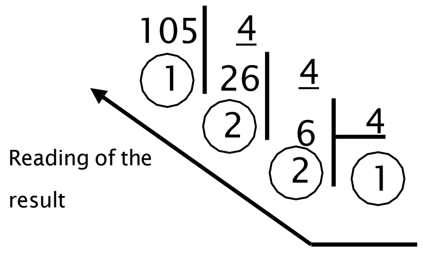
$(84)_{10}=(1010100)_2$

- $(110)_{10}=(?)_8$



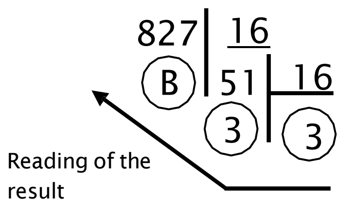
$(110)_{10}=(156)_8$

- $(105)_{10}=(?)_4$



$(105)_{10}=(1221)_4$

- $(827)_{10}=(?)_{16}$



$(827)_{10}=(33B)_8$

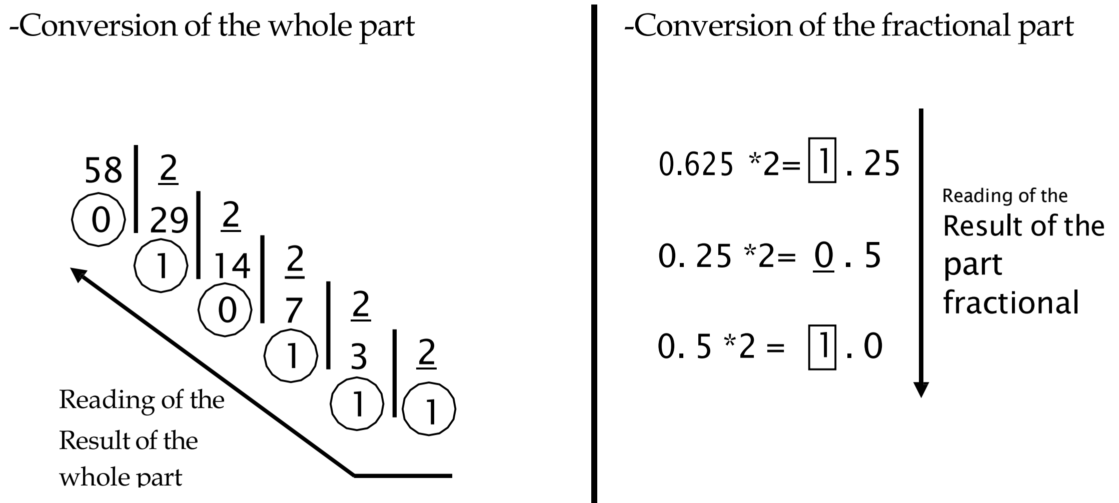
3.1.3 Converting a decimal number to a comma

To convert a decimal number to a comma in a base B any, you must:

- Convert the whole part by performing successive divisions by B(as we saw previously).
- Convert the fractional part by performing successive multiplications by Band each time keeping the number becoming a whole number.

Examples:

Converting the number (58.625) to base 2


-Conversion of the whole part
58
2
0
29
2
1
14
2
0
7
2
1
3
2
1
1
Reading of the Result of the whole part
-Conversion of the fractional part
0.625 *2= 1 . 25
0. 25 *2= 0 . 5
0. 5 *2 = 1 . 0
Reading of the Result of the part fractional

$(58,625)_{10}=(111010.101)_2$

Remarks :

Sometimes by multiplying the fractional part by the base B we cannot convert the entire fractional part. This is mainly due to the fact that the number to be converted does not have an exact equivalent in the base B and its fractional part is cyclic

Example :$(0.15)_{10}=(\ ?\ )_2$

0.15 *2	=**<u>0</u>** .3
0.3 *2	=**<u>0</u>** .6
0.6 *2	=**<u>1</u>** .2
0.2 *2	=**<u>0</u>** .4
0.4*2	=**<u>0</u>** .8
0.8*2	=**<u>1</u>** .6
0.6 *2	=**<u>1</u>** .2
0.2 *2	=**<u>0</u>** .4
0.4*2	=**<u>0</u>** .8
0.8*2	=**<u>1</u>** .6

- $(0.15)_{10}=(0.001001\underline{\mathbf{1001}}\ )_2$

We say that the number $(0.15)_{10}$is cyclic in the period base 2 **1001.**

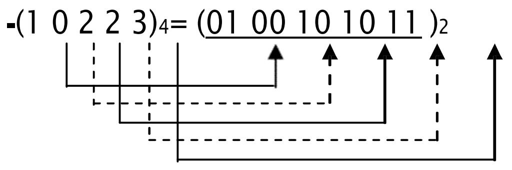
3.1.4 Other conversions

To convert a number from any baseB1to another baseB2you have to go through the base10. But if the baseB1AndB2are written respectively in the form of a power of 2 we can go through the base 2 (binary):

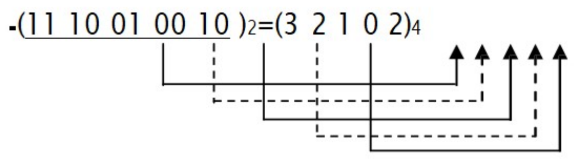
Tetral base (base 4): $4=2^2$ each tetra digit converts itself to 2 bits. Octal base (base 8): $8=2^3$ each octal digit converts itself to 3 bits.

Hexadecimal base (base 16): $16=2^4$ each hexadecimal digit converts itself to 4 bits.

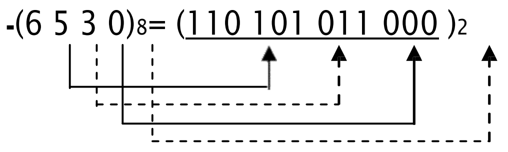
Examples:



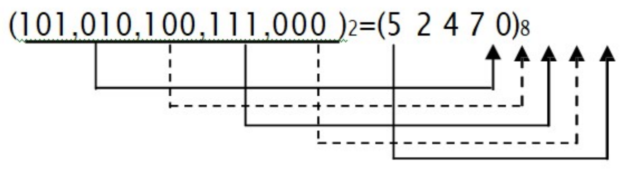
-$(1\ 0\ 2\ 2\ 3)_4 = (01\ 00\ 10\ 10\ 11)_2$



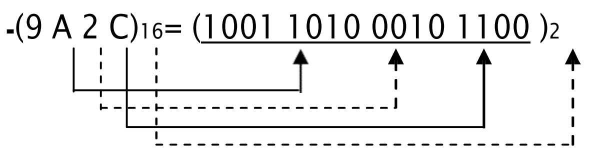
-$(11\ 10\ 01\ 00\ 10)_2 = (3\ 2\ 1\ 0\ 2)_4$



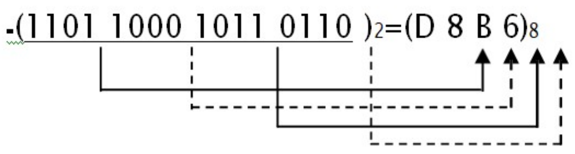
-$(6\ 5\ 3\ 0)_8 = (110\ 101\ 011\ 000)_2$



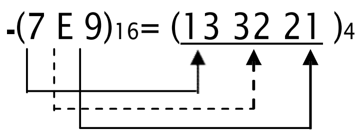
$(101,010,100,111,000)_2 = (5\ 2\ 4\ 7\ 0)_8$



-$(9\ A\ 2\ C)_{16} = (1001\ 1010\ 0010\ 1100)_2$



-$(1101\ 1000\ 1011\ 0110)_2 = (D\ 8\ B\ 6)_8$



-$(7\ E\ 9)_{16} = (13\ 32\ 21)_4$

4. OPERATIONS IN THE BASES

We proceed in the same way as that used in the decimal base. Thus, we must carry out the operation in the base 10, then convert the result by column of the baseB.

4.1 Addition

Binary Base	
11001001	1101110
+ 110101	+ 100010
= $(11111110)_2$	= $(10010000)_2$

Tetral Base	
32210	20031
+ 1330	+ 1302
= $(100200)_4$	= $(21333)_4$

Octal Base	
63375	5304
+ 7465	+ 6647
= $(73062)_8$	= $(14153)_8$

Hexadecimal base	
89A27	5 3 0 4
+ EE54	+ CC3B
= $(9887B)_{16}$	= $(11F3F)_{16}$

4.2 Subtraction

Binary Base	
1110110 - 110101 =$(1000001)_2$	1000001001 - 11110011 = $(100010110)_2$

Tetral Base	
13021 - 2103 = $(10312)_4$	2210 - 1332 = $(21333)_4$

Octal Base	
52130 - 6643 = $(43265)_8$	145126 - 75543 = $(47363)_8$

Hexadecimal Base	
725B2 - FF29 = $(62689)_{16}$	45DD3 - 9BF6 = $(3C1DD)_{16}$

4.3 Multiplication

Binary Base	
$\begin{array}{r} 1110110 \\ * \quad 11011 \\ \hline 1110110 \\ 1110110 \\ 1110110 \\ 1110110 \\ \hline = (110001110010)_2 \end{array}$	$\begin{array}{r} 1010111 \\ * \quad 10011 \\ \hline 1010111 \\ 1010111 \\ 1010111 \\ \hline = (11001110101)_2 \end{array}$

Tetral Base	
$\begin{array}{r} 3021 \\ * \quad 113 \\ \hline 21123 \\ 3021 \\ 3021 \\ \hline = (1020033)_4 \end{array}$	$\begin{array}{r} 13320 \\ * \quad 210 \\ \hline 13320 \\ 33300 \\ \hline = \quad (10123200)_4 \end{array}$

Octal Base	
$\begin{array}{r} 7506 \\ * \quad 243 \\ \hline 26722 \\ 36430 \\ 17214 \\ \hline = (2334622)_8 \end{array}$	$\begin{array}{r} 4327 \\ * \quad 651 \\ \hline 4327 \\ 26063 \\ 32412 \\ \hline = \quad (3526357)_8 \end{array}$

Hexadecimal Base	
$\begin{array}{r} A928 \\ * \quad 7D3 \\ \hline 1FB78 \\ 89708 \\ 4A018 \\ \hline = (52B83F8)_{16} \end{array}$	$\begin{array}{r} 6340 \\ * \quad B51 \\ \hline 6340 \\ 1F040 \\ 443C0 \\ \hline = (4632740)_{16} \end{array}$

2.1 <u>Division</u>

Binary Base	Tetral Base
$\begin{array}{r\|l} 110000000110 & 1110010 \\ \cline{2-2} -\underline{1110010} & 11011 \\ 10011100 & \\ -\underline{1110010} & \\ 10101011 & \\ -\underline{1110010} & \\ 1110010 & \end{array}$	$\begin{array}{r\|l} 300012 & 1302 \\ \cline{2-2} -\underline{1302} & 123 \\ 10321 & \\ -\underline{3210} & \\ 11112 & \end{array}$

Octal Base	Hexadecimal Base
$\begin{array}{r\|l} 50064 & 72 \\ \cline{2-2} -\underline{442} & 542 \\ 366 & \\ -\underline{350} & \\ 164 & \end{array}$	$\begin{array}{r\|l} 24328 & 2B \\ \cline{2-2} -\underline{22F} & D78 \\ 142 & \\ -\underline{12D} & \\ 158 & \end{array}$

5. CODING OF INFORMATION

Coding of information is necessary for its automatic processing. Among the most commonly encountered codes, other than the natural binary code, we cite the code DCB , the codeGRAY, the codepamongn, the ASCII code …

5.1 Digital codes

5.1.1 The Natural Binary Code

It is a numerical representation of numbers in base 2

Decimal	Natural Binary Code			
	has3	has2	has1	has0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

This code has the disadvantage of changing more than a single bit when going from one number to an immediately higher one.

5.1.2 Reflected binary code (GRAY code)

Its interest lies in incrementation applications where a single bit changes state at each increment.

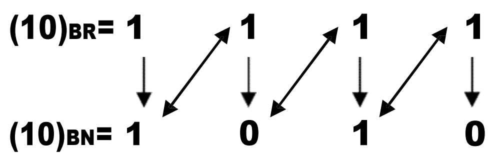
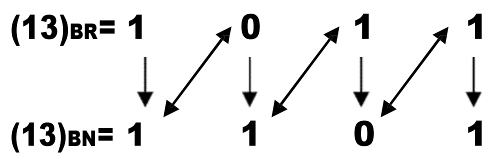
Decimal	Natural Binary Code				Reflected Binary Code			
	has$_3$	has$_2$	has$_1$	has$_0$	has'$_3$	has'$_2$	has'$_1$	has'$_0$
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Remarks :

- Conversion from Natural Binary to Reflected Binary: this involves comparing the bitsbn+1and the bitbnfrom natural binary, the result isbrof the reflected binary which is worth 0 ifbn+1=bnor 1 otherwise. The first bit on the left remains unchanged.

$(6)_{10}=(?)_{BR}$	$(10)_{10}=(?)_{BR}$
$\begin{tikzcd} (6)_{BN}= & 1 \arrow[r, leftrightarrow] \arrow[d] & 1 \arrow[r, leftrightarrow] \arrow[d] & 0 \arrow[d] \\ (6)_{BR}= & 1 & 0 & 1 \end{tikzcd}$	$\begin{tikzcd} (10)_{BN}= 1 \arrow[r, leftrightarrow] \arrow[d] & 0 \arrow[r, leftrightarrow] \arrow[d] & 1 \arrow[r, leftrightarrow] \arrow[d] & 0 \arrow[d] \\ (10)_{BR}= 1 & 1 & 1 & 1 \end{tikzcd}$
$(6)_{10}=(110)_{BN}=(101)_{BR}$	$(10)_{10}=(1010)_{BN}=(1111)_{BR}$

- Conversion from Reflected Binary to Natural Binary: this involves comparing the bitbn+1 natural binary and bit bn from the reflected binary the result is bn of the natural binary which is worth 0 if bn+1=bn or 1 otherwise. The first bit on the left remains unchanged.

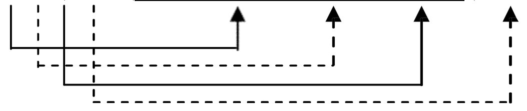
$(10)_{10}=(?)_{BN}$	$(13)_{10}=(?)_{BN}$
 $(10)_{10}=(1111)_{BR}=(1010)_{BN}$	 $(13)_{10}=(1011)_{BR}=(1101)_{BN}$

5.1.3 Binary Coded Decimal Code (BCD Code)

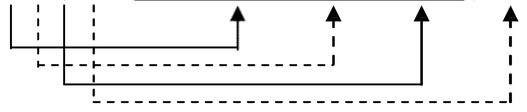
Its property is to associate 4 bits representing each digit in natural binary. The most common application is that of digital display where each digit is associated with a group of 4 bits carrying the DCB code.

Examples:

-$(9\ 4\ 2\ 7)_{10}=(\underline{1001\ 0100\ 0010\ 0111}\ )_{DCB}$



-$(6\ 8\ 0\ 1)_{10}=(\underline{0110\ 1000\ 0000\ 0001}\ )_{DCB}$



5.1.4 The P code among N

The P among N code is an N-bit code in which P bits are at 1 and (NP) bits are at 0. Reading this code can be associated with checking the number of 1s and 0s in the information, which makes it possible to check the information read by detecting the erroneous code.

Example : code 2 of 5

Decimal	Code 2 of 5				
	has7	has4	has2	has1	has0
0	1	1	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	1	0	0	0	1
8	1	0	0	1	0
9	1	0	1	0	0

5.1.5 ASCII code

ASII (American Standard Code for Information Interchange) is an alphanumeric code that has become an international standard. It is used for transmission between computers or between a computer and peripherals. In its standard form, it uses 7 bits. This allows for the generation of 27=128 characters. This code represents uppercase and lowercase alphanumeric letters, decimal digits, punctuation marks, and control characters.

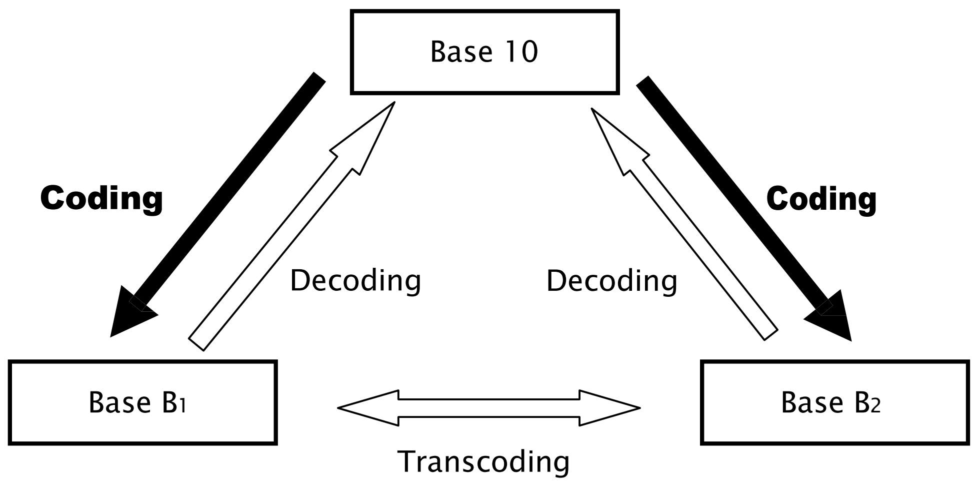
Each code is defined by 3 higher order bitsb6b5b4and 4 lower order bits b3b2b1b0. Thus the character "A" has the hexadecimal code 41H

Example :

HAS -$(65)_{ASCII}$	-$(01000001)_2$	-$(41)_H$
B -$(66)_{ASCII}$	-$(01000010)_2$	-$(42)_H$
Z -$(90)_{ASCII}$	-$(01011010)_2$	-$(5A)_H$
has -$(97)_{ASCII}$	-$(01100001)_2$	-$(61)_H$
b -$(98)_{ASCII}$	-$(01100010)_2$	-$(62)_H$
z -$(122)_{ASCII}$	-$(01111010)_2$	-$(7A)_H$
[-$(91)_{ASCII}$	-$(01011011)_2$	-$(5B)_H$
{ -$(123)_{ASCII}$	-$(01111011)_2$	-$(7B)_H$

5.2 Transcoding

One of the applications related to information coding is the transition from one code to another. This operation is called transcoding:



- The coding of information is done by means of a combinational circuit called **Coder**.
- The decoding of information is done by means of a combinational circuit called **Decoder.**
- A trans coder is a Decoder associated with a Coder.

CHAPTER 2

BOOLIAN ALGEBRA AND LOGICAL FUNCTIONS

1. OBJECTIVES

- Study the rules and theorems of Boolean algebra.
- Understand how logic gates work.

2. VARIABLES AND LOGICAL FUNCTIONS

2.1 Logical variables

A logical variable is a quantity that can only take two logical states. We symbolize them by 0 or 1.

Examples:

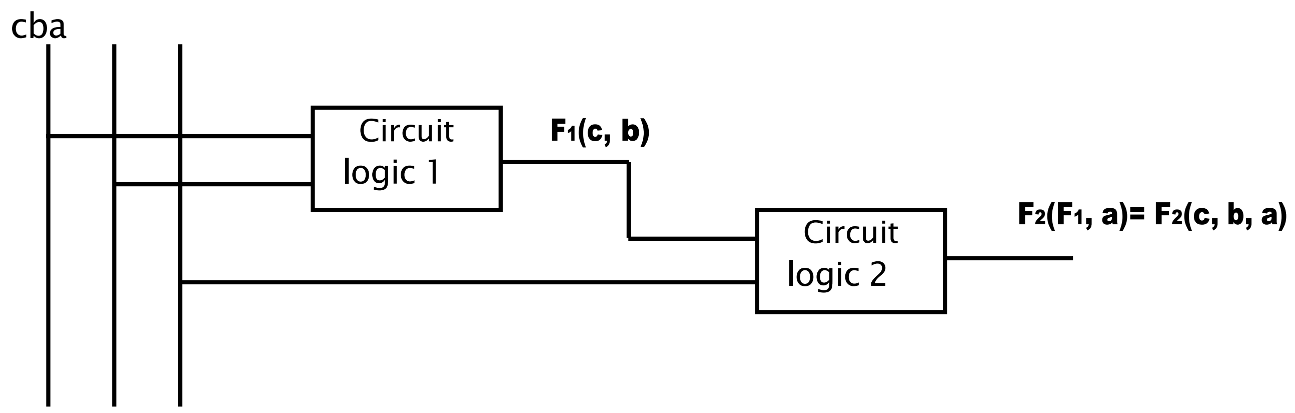
- A switch can be either closed (logic 1) or open (logic 0). It therefore has 2 possible operating states.
- A lamp also has 2 possible operating states which are off (logic 0) or on (logic 1).

2.2 Logical functions

A logical function is a logical variable whose value depends on other variables,

- The operation of a logical system is described by one or more simple logical propositions which have the binary character "TRUE" or "FALSE".
- A logical function that takes the values 0 or 1 can be thought of as a binary variable for another logical function.
- To describe the operation of a system by looking for the state of the output for all possible combinations of inputs, we will use "The truth table".

EXAMPLE :



3. BASIC OPERATIONS OF BOOLIAN ALGEBRA AND ASSOCIATED PROPERTIES

Boolean algebra is a set of two-state variables {0 and 1} also called Boolean, equipped with 3 elementary operators presented in the following table:

Logical operation	Addition	Multiplication	Inversion
	OR	**AND**	**NO**
Algebraic Notation	A OR B = A + B	A AND B = AB	No A=$\overline{A}$
Truth table	HA B A+B 0 0 0 0 1 1 1 0 1 1 1 1	HA B AB 0 0 0 0 1 0 1 0 0 1 1 1	HAS NO TO 0 1 1 0

3.1 Properties of basic operations

Some remarkable properties are worth knowing:

Functions	OR	AND	Comments
1 variable	A+A=A	AA=A	Idempotence
	A+1=1	A.0=0	Absorbent element
	A+0=A	A.1=A	Neutral Element
	$A+\overline{A}=1$	$A\overline{A}=0$	Complement
	$\overline{\overline{A}}=A$		Involution

Functions	**OR**	**AND**	**Comments**
2 variables	A+B=B+A	AB=BA	Commutativity
3 variables	A+(B+C)=(A+B)+C =A+B+C	A.(BC)=(AB).C =ABC	Associativity
	A+BC=(A+B).(A+C)	A.(B+C)=A.B+AC	Distributivity

3.2 Theorems of Boolean algebra

To perform any Boolean calculation, we use, in addition to the properties, a set of theorems:

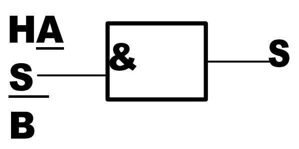
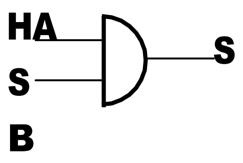
Theorems	OR	AND
Of DEMORGAN	$\overline{A+B}=\overline{A}\,.\,\overline{B}$	$\overline{AB}=\overline{A}+\overline{B}$
	This theorem can be generalized to several variables	
	$\overline{A+B+\ldots+Z}=\overline{A}\,.\,\overline{B}.\ldots.\overline{Z}$	$\overline{AB\ldots.Z}=\overline{A}+\overline{B}+\ldots+\overline{Z}$
Absorption	$A+AB=A$	$A.(A+B)=A$
Of lightening	$A+\overline{A}B=A+B$	$A.(\overline{A}+B)=AB$
	$A.B+\overline{A}C+BC=AB+\overline{A}C$	

4. MATERIALIZATION OF LOGICAL OPERATORS

4.1 Basic logic gates

Logic gates are electronic circuits whose transfer functions (relationships between inputs and outputs) materialize the basic operations applied to electrical variables.

4.1.1 The AND gate

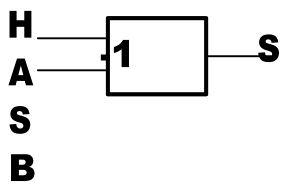
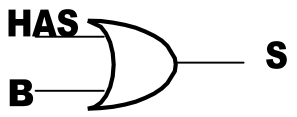
Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)		
		S=AB	TTL: 7408 CMOS: 4081

If V0 represents the LOW voltage level (state 0) AndV1represents the HIGH level (state1), we note at the output of the circuit the voltages given in the operating table and we deduce the truth table.

Operating table		
V_{HAS}	V_B	V_S
V_0	V_0	V_0
V_0	V_1	V_0
V_1	V_0	V_0
V_1	V_1	V_1

Truth table		
HAS	B	S
0	0	0
0	1	0
1	0	0
1	1	1

4.1.2 The OR (OR) gate

Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)		
		$S=A+B$	TTL: 7432 CMOS: 4071

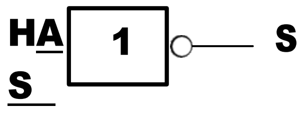
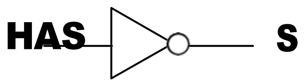
Operating table		
V_{HAS}	V_B	V_S
V_0	V_0	V_0
V_0	V_1	V_1
V_1	V_0	V_1
V_1	V_1	V_1

Truth table		
HAS	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Noticed : There are 2, 3, 4, 8, and 13 input OR and AND logic gates available in integrated circuit form.

4.1.3 The NO gate

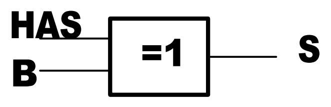
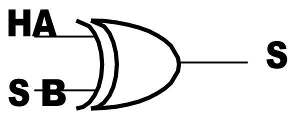
It is a single-entry gate, it materializes the reversing operator.

Logical symbol		**Equation**	**Integrated circuit**
International Symbol (IEC)	European symbol (MIL)		
		$S=\overline{A}$	TTL: 7404 CMOS: 4069

Operating table	
VHAS	**Vs**
V_0	V_1
V_1	V_0

Truth table	
HAS	**S**
0	1
1	0

4.1.4 The exclusive-OR (XOR) gate

Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)	S=AB _ =AB*AB	TTL: 7486 CMOS: 4070
			

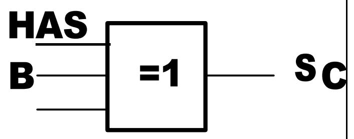
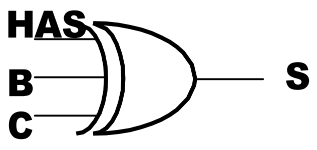
Operating table		
VHAS	**VB**	**Vs**
V_0	V_0	V_0
V_0	V_1	V_1
V_1	V_0	V_1
V_1	V_1	V_0

Truth table		
HAS	**B**	**S**
0	0	0
0	1	1
1	0	1
1	1	0

The exclusive-OR function is1if only one of the inputs is in the state1and the other is the state 0.

Generalizations of the EXCLUSIVE-OR function: The output of the EXCLUSIVE-OR function takes the logical state1if an odd number of input variables are in the logical state 1.

Example: Three-way exclusive-OR

Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)	S=ABC	TTL: 74386
			

Operating table			
V_{HAS}	V_B	V_C	V_S
V_0	V_0	V_0	V_0
V_0	V_0	V_1	V_1
V_0	V_1	V_0	V_1
V_0	V_1	V_1	V_0
V_1	V_0	V_0	V_1
V_1	V_0	V_1	V_0
V_1	V_1	V_0	V_0
V_1	V_1	V_1	V_1

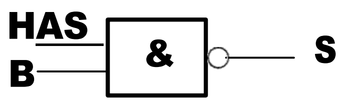
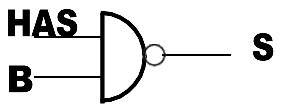
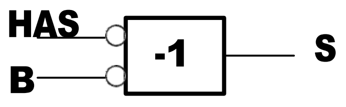
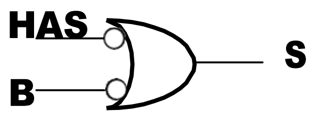
Truth table			
HAS	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

4.2 Universal gates

Other than basic (or elementary) logic gates, there are gates called universal (complete) logic gates such as NAND and NOR gates.

4.2.1 The NAND gate

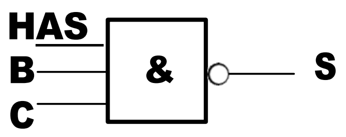
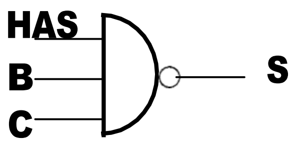
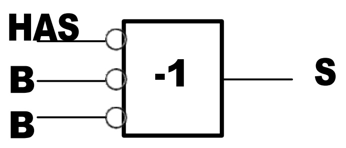
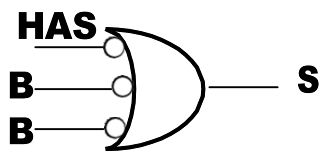
It is equivalent to a gate followed by an inverter.

Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)	$S=A\vert B$ $S=\overline{AB}$ $S=\overline{A}+\overline{B}$	TTL: 7400 CMOS: 4011-4093
			
			

Operating table		
V_{HAS}	V_B	V_S
V_0	V_0	V_1
V_0	V_1	V_1
V_1	V_0	V_1
V_1	V_1	V_0

Truth table		
HAS	B	S
0	0	1
0	1	1
1	0	1
1	1	0

For the three-input NAND gate we find:

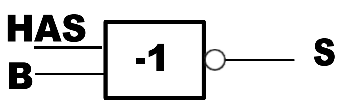
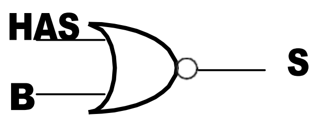
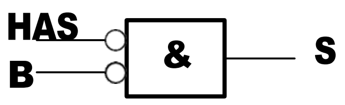
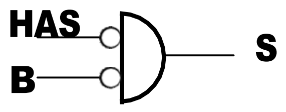
Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)	$S=A\|B\|C$ $S=\overline{ABC}$ $S=\overline{A}+\overline{B}+\overline{C}$	TTL: 7410 CMOS: 4023
			
			

Operating table			
VHAS	VB	VC	VS
V_0	V_0	V_0	V_1
V_0	V_0	V_1	V_1
V_0	V_1	V_0	V_1
V_0	V_1	V_1	V_1
V_1	V_0	V_0	V_1
V_1	V_0	V_1	V_1
V_1	V_1	V_0	V_1
V_1	V_1	V_1	V_0

Truth table			
HAS	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

4.2.2 The NOR gate:

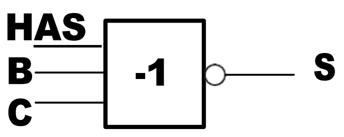
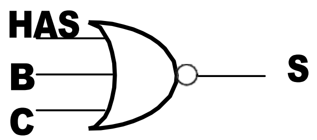
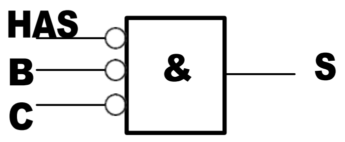
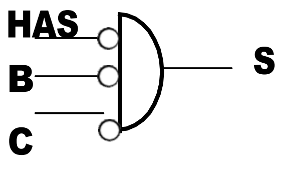
It is equivalent to a gate followed by an inverter.

Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)	$S=AB$ $S=\overline{A+B}$ $S=\overline{A}\overline{B}$	TTL: 7402 CMOS: 4001
			
			

Operating table		
V_{HAS}	V_B	V_S
V_0	V_0	V_1
V_0	V_1	V_0
V_1	V_0	V_0
V_1	V_1	V_0

Truth table		
HAS	B	S
0	0	1
0	1	0
1	0	0
1	1	0

For the three-input NOR gate we find:

Logical symbol		Equation	Integrated circuit
International Symbol (IEC)	European symbol (MIL)		
		S=ABC $S=\overline{A+B+C}$ $S=\bar{A}\bar{B}\bar{C}$	TTL: 7427 CMOS: 4025
			

Operating table			
V_{HAS}	V_B	V_C	V_S
V_0	V_0	V_0	V_1
V_0	V_0	V_1	V_0
V_0	V_1	V_0	V_0
V_0	V_1	V_1	V_0
V_1	V_0	V_0	V_0
V_1	V_0	V_1	V_0
V_1	V_1	V_0	V_0
V_1	V_1	V_1	V_0

Truth table			
HAS	B	C	S
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

4.2.3 Exercise

1) Demonstrate whether universal functions are associative: (A | ? ?

B) | C=A | (B | C)= A | B | C

(AB)-C=A-$^{?}$(BC)= ABC$^{?}$

2) Implement the three-input NAND function using the two-input NAND operators.

Answer :

1)

- $(A|B)|C=\overline{(AB)}|C=(\overline{A}+\overline{B})|C=\overline{(\overline{A}+\overline{B}).C}=\overline{\overline{A}+\overline{B}}+\overline{C}=(AB)+\overline{C}$

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

  (A|B)|CA|(B|C) thenthe functionNAND is not associative

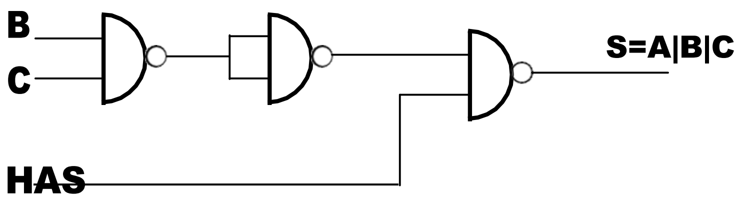
- $(AB)-C=\overline{(A+B)}-C=(\overline{A}\overline{B})-C=\overline{(\overline{A}\overline{B})+C}=\overline{(\overline{A}\overline{B})}.\overline{C}=(A+B).\overline{C}$

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

  (AB)-CA-(BC) thenthe functionNOR is not associative

2)

$-A|B|C=\overline{ABC}=\overline{A}+\overline{BC}= \overline{\overline{\overline{A}+\overline{BC}}} = \overline{A\overline{\overline{BC}}}=A|[(B|C)|(B|C)]$



CHAPTER 3
REPRESENTATION AND SIMPLIFICATION OF LOGICAL FUNCTIONS COMBINATORIES

1. OBJECTIVES

- Study the algebraic representation of a logical function,
- Understand the algebraic simplification of a logical function,
- Synthesize combinatorial applications.

2. REPRESENTATION OF A LOGICAL FUNCTION

A logical function is a combination of binary variables connected by the operators AND, OR and NOT. It can be represented by an algebraic notation or a truth table or a KARNAUGH table or a flowchart.

2.1 Algebraic representation

A logical function can be represented in two forms:

- SD P: -(-) sum of products,
- PDS: - (-) product of sums,

2.1.1 Sum-of-products form (Disjunctive form)

It corresponds to a sum of logical products: F=-(-(ei)), or eirepresents a logical variable or its complement.

Example : F1(A, B, C)=AB+BC. If each of the products contains all the input variables in direct or complemented form, then the form is called:"first canonical form »or form "disjunctive canonical ». Each of the products is called midterm.

Example :F_1(A, B, C)=ABC+ABC+ABC+ABC.

2.1.2 Product of sums)

It corresponds to a product of logical sums: F=-(-(e_i)), or $e_i$represents a logical variable or its complement.

Example :$F_2(A, B, C)=(A+B).(A+B+\overline{C}).$

If each of the sums contains all the input variables in direct or complemented form, then the form is called:"second canonical form »or form "conjunctive canonical". Each of the products is calledmaxterm.

Example :$F_2(A, B, C)=(A+B+C).(A+B+C).(\overline{A}+B+\overline{C})$

2.2 Truth table

A logical function can be represented by a truth table which gives the values that the function can take for each combination of input variables.

2.2.1 Fully defined function

It is a logical function whose value is known for all possible combinations of variables.

Example : The "Majority of 3 variables" function: MAJ(A, B, C) The MAJ function is equal to 1 if the majority (2 or 3) of the variables are at state 1.

Truth table				
Combination	**HAS**	**B**	**C**	**S=SHIFT(A, B, C)**
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

2.2.2 Incompletely defined function

This is a function whose value is<u>unspecified</u> for certain combinations of variables. This is indicated by the symbol X or -; that is, the function is <u>indifferent</u> for certain combinations of input variables corresponding to situations which are:

- Can never follow in the system, Do not change
- The behavior of the system.

Example: Consider a keyboard that has 3 push buttons P1, P2and P3which control a machine and which have a mechanical lock such that 2 adjacent buttons cannot be pressed simultaneously:

P1-	P2-	P3-
Manual Walking	Stop	Increase speed

It is assumed thatPisupported is worth1and released is worth0. Hence the truth table of the function "keyboard»which detects at least one triggered push button:

Truth table				
Combination	**HAS**	**B**	**C**	**Keyboard**
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	-
4	1	0	0	1
5	1	0	1	1
6	1	1	0	-
7	1	1	1	-

2.2.3 Equivalence between the truth table and canonical forms

- To establish the disjunctive canonical expression (the canonical sum) of the function: it is sufficient to carry out the logical sum (or union) of the minterms associated with the states for which the function is equal to “1”.

- To establish the conjunctive canonical expression (the canonical product) of the function: it is sufficient to carry out the logical product (or intersection) of the maxterms associated with the states for which the function is equal to “0”.

Example : The "Majority of 3 variables" function: MAJ(A, B, C)

Truth table						
Combination	**HAS**	**B**	**C**	**S=SHIFT(A, B, C)**	**Minterme**	**Maxterme**
0	0	0	0	0	$\overline{A}\overline{B}\overline{C}$	$A+B+C$
1	0	0	1	0	$\overline{A}\overline{B}C$	$A+B+\overline{C}$
2	0	1	0	0	$\overline{A}B\overline{C}$	$A+\overline{B}+C$
3	0	1	1	1	$\overline{A}BC$	$A+\overline{B}+\overline{C}$
4	1	0	0	0	$A\overline{B}\overline{C}$	$\overline{A}+B+C$
5	1	0	1	1	$A\overline{B}C$	$\overline{A}+B+\overline{C}$
6	1	1	0	1	$AB\overline{C}$	$\overline{A}+\overline{B}+C$
7	1	1	1	1	ABC	$\overline{A}+\overline{B}+\overline{C}$

- We notice thatSHIFT(A,B,C)=1for the combinations 3, 5, 6, 7. We write the function thus specified in a so-called numerical form:MAJ= R(3,5,6,7), Union of states 3, 5, 6, 7. The first canonical form of the function NAJcan be directly deduced from this:

$$\text{UPDATE}_{(A,\,B,\,C)}=\overline{A}BC+A\overline{B}C+AB\overline{C}+ABC.$$

- We notice thatSHIFT(A,B,C)=0for the combinations 0, 1, 2, 4. We write the function thus specified in a so-called numerical form:MAJ= I(0,1,2,4), Intersection of states 0, 1, 2, 4. The second canonical form of the function NAJ can be directly deduced from this:

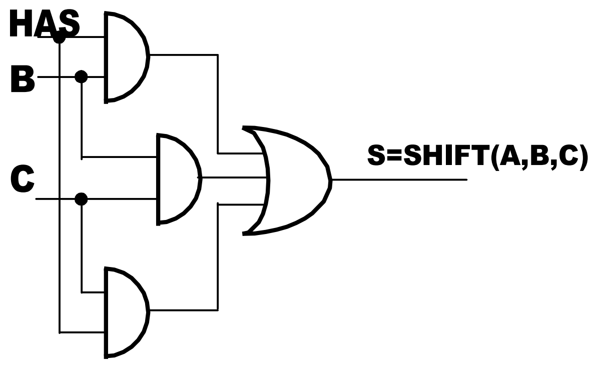
$$\text{PDATE}_{(A,\,B,\,C)}=(A+B+\overline{C}).(A+\overline{B}+C).(\overline{A}+B+C).(A+B+C)$$

- NB:We are generally interested in the representation of a function in the form of a sum or canonical sum (disjunctive form).

2.3 Flowchart

It is a graphical method based on symbols or gates.

Example : The "Majority of 3 variables" function: MAJ(A,B,C) UPDATE$_{(ABC)}$=AB+BC+AC.



2.4 The painting by KARNAUGH (TK)

The Karnaugh table method allows you to visualize a function and intuitively derive a simplified function. The basic element of this method is the Karnaugh table, which is represented as a table formed by rows and columns.

2.4.1Adjacency of boxes

Two binary words are said to be adjacent if they differ only by the complement of one and only one variable. If two adjacent words are summed, they cannt be merged and the variable that differs from them will be eliminated. The words ABC and ABC are adjacent since they differ only by the complementarity of the variable C. The adjacency theorem therefore states that ABC and ABC = AB.

2.4.2 Construction of the table

KARNAUGH's painting was constructed in such a way as to bring out the logical visual adjacency.

- Each box represents a combination of variables (minterm),
- The truth table is transported into the array by putting the value of the corresponding function in each box.

The function represented by a KARNAUGH table is written as the sum of the products associated with the different boxes containing the value 1.

2.4.3 Rules to follow for a problem with n variables: (n>2)

The KARNAUGH table has 2ncases or combinations, The order of the variables is not important but it only respects the following rule:

- The monomials identifying the rows and columns are assigned in such a way that 2 consecutive monomials only differ in the state of a variable, it results that 2 consecutive boxes in row or column identify adjacent combinations, we therefore use the GRAY code.

Example

- n=2

HAS \ B	$\overline{B}$(0)	B(1)
$\overline{A}$(0)	00	01
A(1)	10	11

- n=3

HAS \ BC	$\overline{B}\overline{C}$(00)	$\overline{B}$C(01)	BC(11)	B$\overline{C}$(10)
$\overline{A}$(0)	000	001	011	010
A(1)	100	101	111	110

- n=4

B \ CD	$\overline{C}\overline{D}$(00)	$\overline{C}$D(01)	CD(11)	C$\overline{D}$(10)
$\overline{A}\overline{B}$(00)	0000	0001	0011	0010
$\overline{A}$B(01)	0100	0101	0111	0110
AB(11)	1100	1101	1111	1110
A$\overline{B}$(10)	1000	1001	1011	1010

NB: The Karnaugh Table has a structure wrapped around the rows and columns.

It has a spherical shape.

2.4.4 Example of filling the KARNAUGH table from the truth table:

Truth table

Combination	HA	B	C	D	F(A,B,C,D)
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

Painting by KARNAUGH

AB \ CD	CD(00)	C$\overline{D}$(01)	CD(11)	CD(10)
AB(00)	0	1	0	0
$\overline{A}$B(01)	1	1	1	0
AB(11)	0	1	0	0
A$\overline{B}$(10)	0	0	1	0

3. SIMPLIFICATION OF LOGIC FUNCTIONS

The goal of simplifying logic functions is to minimize the number of terms in order to obtain a simpler hardware implementation, therefore easier to build and troubleshoot and less expensive.

Two simplification methods are used:

- Algebraic simplification.
- Graphical simplification by KARNAUGH table**.**

3.1 Algebraic simplification of logical expressions

To obtain a simpler expression of the function by this method, one must use:

- Theorems and properties of Boolean algebra (see chapter 2).
- Multiplication by 1 (X+X).
- The addition of a zero term (XX).

Example : Simplification of the "Majority" function: MAJ(A,B,C)

$$AJ_{(ABC)}=A\bar{B}C+AB\bar{C}+ABC+\bar{A}BC.$$

$$AJ_{(ABC)}=A\bar{B}C+ABC+\bar{A}BC+A\bar{B}C+ABC+ABC.$$

$$AJ_{(ABC)}=BC(A+\bar{A})+AB(C+\bar{C})+AC(B+B).$$

$$AJ_{(ABC)}=BC+AB+AC$$

NB: The rules and properties of Boolean algebra allow functions to be simplified, but it remains a relatively cumbersome method. It never allows us to know whether or not we arrive at a minimal expression of the function.

We can then use the KARNAUGH table method

3.2 Graphical simplification of logical expressions (by KARNAUGH table)

The KARNAUGH table allows you to visualize a function and intuitively derive a simplified function from it

3.2.1Grouping adjacent boxes

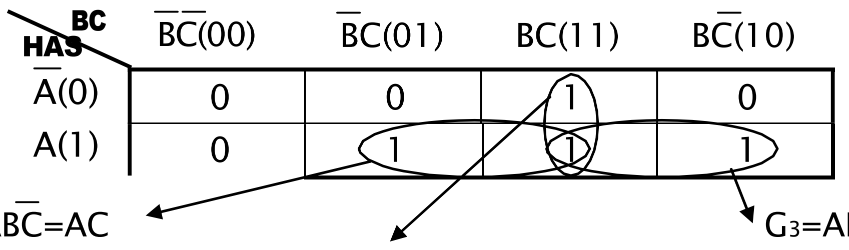
The method consists of making groups of adjacent squares. These groupings of squares must be of maximum size (maximum number of cases) and equal to 2k(i.e. 2, 4, 8, 16, …). We stop grouping when all the ones belong to at least one of them.

NB:Before deriving the equations from the KARNAUGH table, the following rules must be observed:

- Group all together.
- Group as many of them as possible into a single
- grouping. A grouping has a rectangular shape.
- The number of ones in a group is a power of 2 is equal to 2k
- A 1 can appear in more than one grouping.
- A grouping must respect the axes of symmetry of the TK

Grouping of the 2 adjacent boxes

Simplification of the Majority function of 3 variables (MAJ(A,BC))



$G_1=ABC+AB\bar{C}=AC$

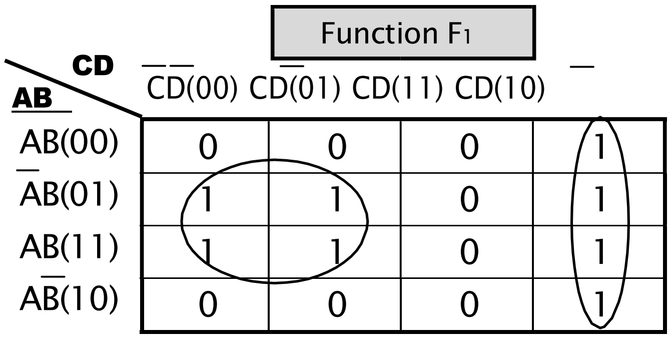
$G_3=ABC+AB\bar{C}=AB$

$G_2=\bar{A}BC+ABC=BC$

$$\boxed{\underline{SHIFT(A,B,C)=G_1+G_2+G_3=AB+BC+AC}}$$

Ruler: Combining two adjacent squares containing 1 each eliminates one only variable that changes state when moving from one box to another.

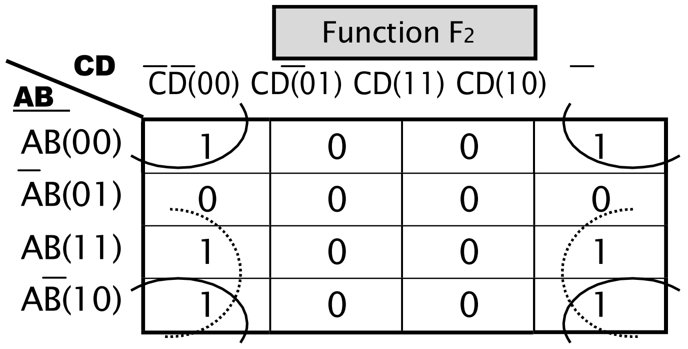
Grouping of the 4 adjacent boxes



Function F1

AB \ CD	$\bar{C}\bar{D}$(00)	$\bar{C}D$(01)	CD(11)	$C\bar{D}$(10)
AB(00)	0	0	0	1
$\bar{A}B$(01)	1	1	0	1
AB(11)	1	1	0	1
$A\bar{B}$(10)	0	0	0	1

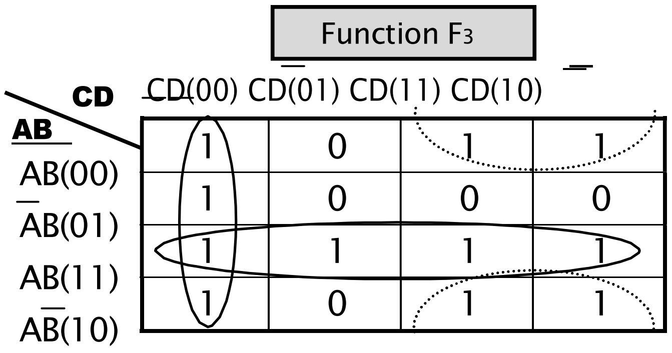
$F_{1(A,B,C,D)}=B\bar{C}+C\bar{D}$



Function F2

AB \ CD	$\bar{C}\bar{D}$(00)	$\bar{C}D$(01)	CD(11)	$C\bar{D}$(10)
AB(00)	1	0	0	1
$\bar{A}B$(01)	0	0	0	0
AB(11)	1	0	0	1
$A\bar{B}$(10)	1	0	0	1

$F_{2(A,B,C,D)}=A\bar{D}+\bar{B}\bar{D}$



Function F3

AB \ CD	$\bar{C}\bar{D}$(00)	$\bar{C}D$(01)	CD(11)	$C\bar{D}$(10)
AB(00)	1	0	1	1
$\bar{A}B$(01)	1	0	0	0
AB(11)	1	1	1	1
$A\bar{B}$(10)	1	0	1	1

$F_{3(A,B,C,D)}=\bar{C}\bar{D}+AB+\bar{B}C$

Ruler:2 variables disappear when we group 4 adjacent boxes, we can then replace the sum of the 4 boxes (4 minterms with 4 variables each) by a single term which only has 2 variables.

Grouping of 8 adjacent boxes

Function F4

AB \ CD	$\overline{C}\overline{D}$(00)	$\overline{C}D$(01)	CD(11)	$C\overline{D}$(10)
AB(00)	1	0	0	1
$\overline{A}B$(01)	1	0	0	1
AB(11)	1	0	0	1
$A\overline{B}$(10)	1	0	0	1

$$F_{4(A,B,C,D)}=\overline{D}$$

Ruler:2 variables disappear when we group 8 adjacent boxes, we can then replace the sum of the 8 boxes (8 minterms with 4 variables each) by a single term which contains only 1 variable.

Noticed: We will limit ourselves to tables of 4 variables, to solve by example of problems with 5 variables; we break them down each into two problems with 4 variables.

3.22 Handling 5-variable problems

To solve this problem we will break it down into 2 problems with 4 variables by applying the expansion theorem (SHANNON).

$$\text{we have: } F_{(A,B,C,D,E)}=\overline{E}F_{(A,B,C,D,0)}+ EF_{(A,B,C,D,1)}$$

NB:SHANNON's expansion theorem remains applicable whatever the number of variables we have:

$$F_{(A,B,C,\,\ldots\,,Z)}=\overline{Z}F_{(A,B,C,\,\ldots\,,0)}+ ZF_{(A,B,C,\,\ldots\,,1)}$$

Example : Simplify the function F(A,B,C,D,E)=-(4, 5, 6, 7, 24, 25, 26, 27)

F(A,B,C,D,0)

AB \ CD	$\bar{C}\bar{D}$(00)	$\bar{C}D$(01)	CD(11)	$C\bar{D}$(10)
$\bar{A}\bar{B}$(00)	0	0	0	1
$\bar{A}B$(01)	0	0	0	1
AB(11)	0	0	0	1
$A\bar{B}$(10)	0	0	0	1

F(A,B,C,D,1)

AB \ CD	$\bar{C}\bar{D}$(00)	$\bar{C}D$(01)	CD(11)	$C\bar{D}$(10)
$\bar{A}\bar{B}$(00)	0	1	0	0
$\bar{A}B$(01)	0	1	0	0
AB(11)	0	1	0	0
$A\bar{B}$(10)	0	1	0	0

$F_{(A,B,C,D,0)}=C\bar{D}$ $\longrightarrow$ $F_{(A,B,C,D,1)}=\bar{C}D$

What results from this: $F_{(A,B,C,D,E)}=\bar{E}C\bar{D}+E\bar{C}D$

3.23 Indifferent or undefined values

The symbol - (or X) can take the value 0 or 1 indifferently: we therefore replace by 1 only those which allow us to increase the number of boxes in a grouping and those which reduce the number of groupings.

Example

Truth table				
Combination	**HAS**	**B**	**C**	**F(A,B,C)**
0	0	0	0	-
1	0	0	1	0
2	0	1	0	1
3	0	1	1	-
4	1	0	0	0
5	1	0	1	0
6	1	1	0	-
7	1	1	1	1

F(ABC)

HAS \ BC	$\overline{B}\overline{C}$(00)	BC(01)	BC(11)	B$\overline{C}$(10)
A(0)	-	0	-	1
A(1)	0	0	1	-

F(ABC)=B

4. SUMMARY: SYNTHESIS OF A LOGICAL FUNCTION

- Step 1: Reading and analysis of the statement of the function.
- Step 2: writing the function in the canonical form of a truth table.
- Step 3: Simplification of the function expression by the method algebraic or by the TK method
- Step 3: Creation of the flowchart:
  - ✓ With only one type of operators using universal logical functions.
  - ✓ With a minimum of operators using basic logic functions

CHAPTER 4
COMBINATORIAL LOGIC CIRCUITS

1. OBJECTIVES

- Study the main combinational logic circuits used in digital systems (such as: arithmetic circuits, encoders, transcoders, etc.),
- Implement logic functions using combinational circuits.

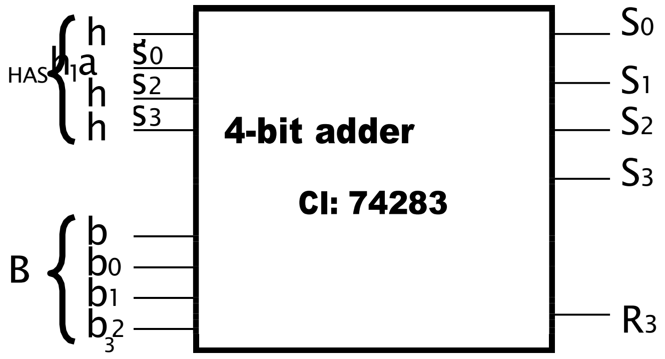
2. ARITHMETIC CIRCUITS

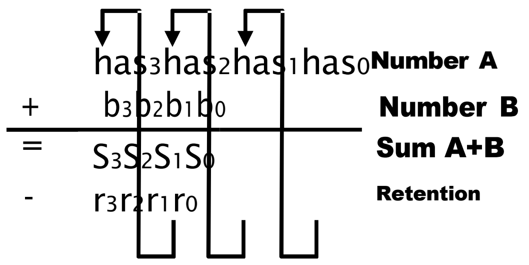
2.1 Adders

An adder is a circuit capable of adding two binary numbers HASAndB. An addition implements two outputs:

- The sum, generally noted S,
- The restraint, generally noted R(Or C: carry).

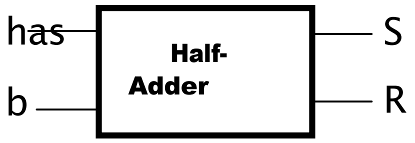
As in decimal, we must take into account the possible carryover, the result of a previous calculation. The following figure shows the decomposition of the addition of two 4-bit binary numbers.

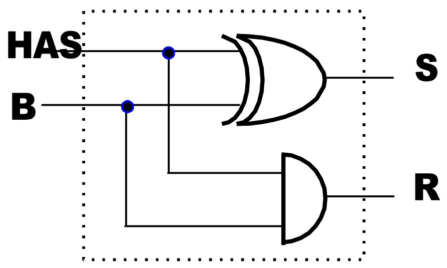




2.1.1 The Half Adder (2 bits)

It is a 2-bit adder without taking into account the previous carry.

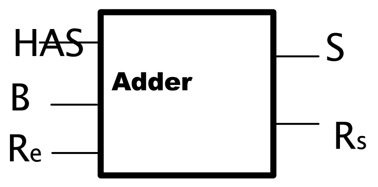


Truth table	Output equation	Flowchart
see below	$S=\overline{A}B+A\overline{B}=AB$ $R=AB$	

HA	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

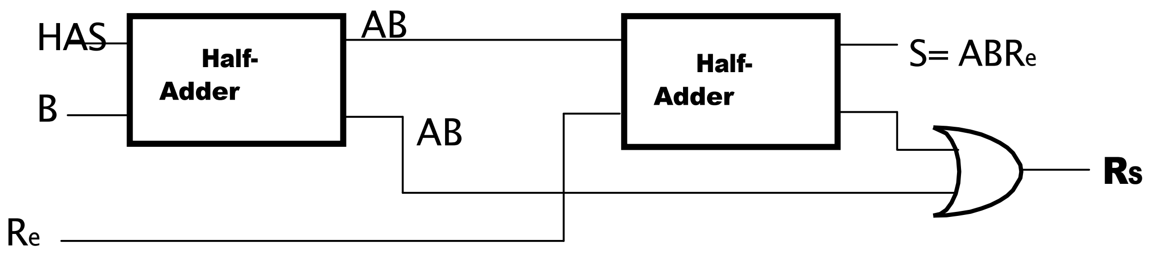
2.1.2 The Full Adder (2 bits)

It has three inputs A, B and Reand two exits S and RS: Rerepresents the carry of rank n-1 and Rsthat of rank n.

Truth table	**Output equation**	**Flowchart**
see below	$S=\overline{A}\overline{B}R_e+\overline{A}B\overline{R_e}+A\overline{B}\overline{R_e}+ABR_e$ $=ABR_e$ $R_S= R_eA\text{-}B+AB$	 **Integrated circuit:** **74LS183**

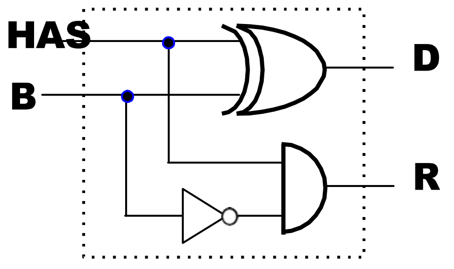
A	B	R_e	S		R_s
0	0	0	0		0
0	0	1	1		0
0	1	0	1		0
0	1	1	0		1
1	0	0	1		0
1	0	1	0		1
1	1	0	0		1
1	1	1	1		1

Flowchart:



2.2 The subtractors

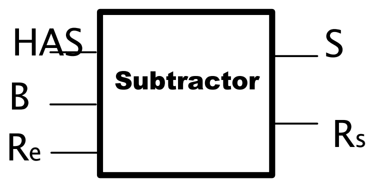
A half-subtract or ignores any carry from lower-order bits. D represents the result of the difference (AB) and R restraint.

Truth table	**Output equation**	**Flowchart**
	$D=\overline{A}B+A\overline{B}=AB$ $R=\overline{A}B$	

HA	B	D	R
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

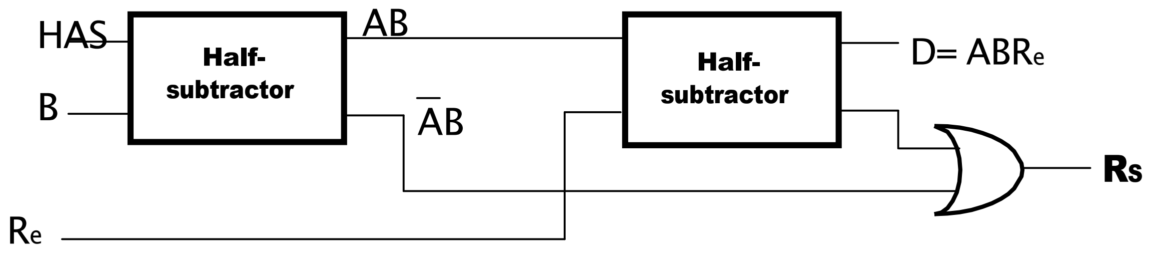
2.2.1 The complete subtractor (2 bits)

It has three inputs A, B and Reand two exits D and RS: Rerepresents the carry of rank n-1 and Rsthat of rank n.

Truth table	**Output equation**	**Flowchart**
	$D=\overline{A}\overline{B}R_e+\overline{A}B\overline{R_e}+A\overline{B}\overline{R_e}+ABR_e$ $=ABR_e$ $R_S= R_eA-B+A\overline{B}$	

AB	ReD	s		
0	0 0	0		0
0	0 1	1		0
0	1 0	1		0
0	1 1	1		1
1	0 0	1		0
1	0 1	0		1
1	1 0	0		1
1	1 1	1		1

Flowchart:



2.3 Adder-subtractors

- A number coded on n bits can take a value between 0 and 2n-1.
- The complement of an n-bit word is obtained by taking the complement of each of its n bits. Thus, we have:

$$\mathbf{A+\overline{A}=2_n-1 \quad --A= \; A+\overline{1}-2_n}$$

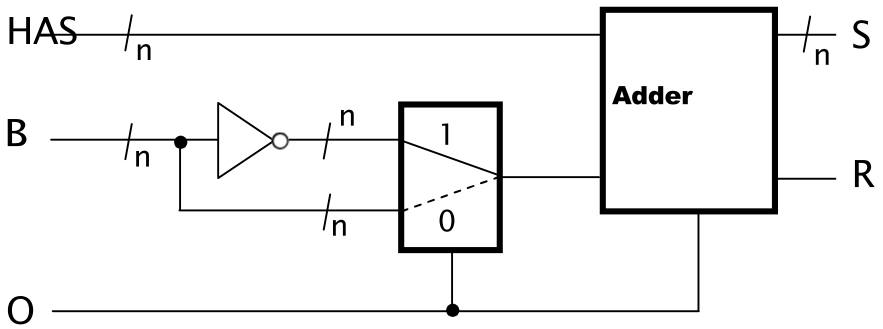
- For a variable coded on n bits: 2n=0. That is to say, it is possible to write a negative integer as the "2's complement" of its value absolute.

$$-A=\overline{A}+1$$

- We can use this property to write the subtraction of two n-bit words in the following form:

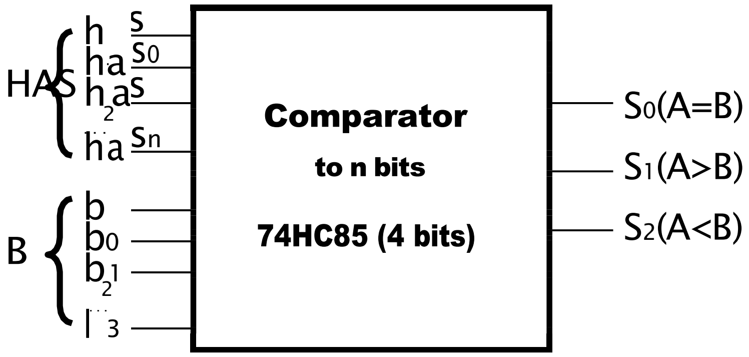
$$AB=A+\overline{B}+1$$

- A single device shown in the figure below can be used for addition and subtraction according to the operation codeO:
    - ✓ O=0: addition
    - ✓ O=1: subtraction



2.4 Comparator

It is a circuit that allows you to compare two binary numbers. It indicates whether the first number is less than (S2), equal (S0) or higher (S1) to the second number.



Basic principle

The principle is to first compare the most significant bits (MSB). If they are different, there is no point in continuing the comparison. On the other hand, if they are equal, the next lowest-order bits must be compared, and so on.

2.4.1 The 1-bit comparator

Truth table

B	HA	S_0	S_1	S_2
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

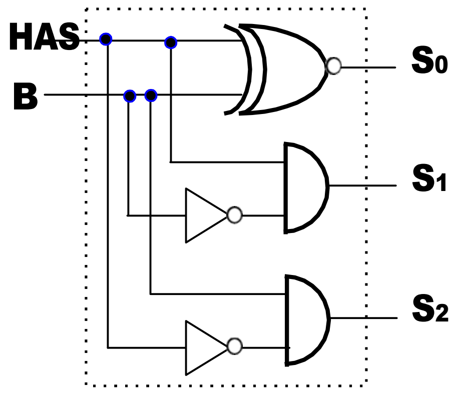
Equation of exits

$S_0=\bar{A}\bar{B}+AB=\overline{AB}$

$S_1=A\bar{B}$

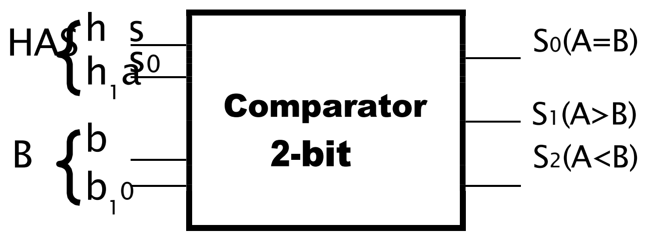
$S_2=\bar{A}B$

Flowchart

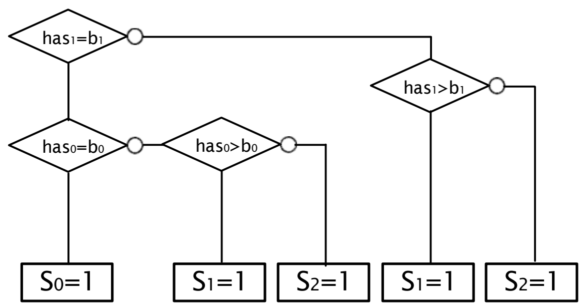


2.4.2 The 2-bit comparator

Operating diagram



Organizational chart



Truth table

b_1	b_0	has_1	has_0	S_0	S_1	S_2
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

Equations

We have S_0 is worth 1 if $a_1=b_1$ and if $a_0=b_0$

$S_0=(a_1-b_1).(has_0-b_0).$

And S1 is worth 1 if a1>b1 or if (a1=b1 and has 0>b0)

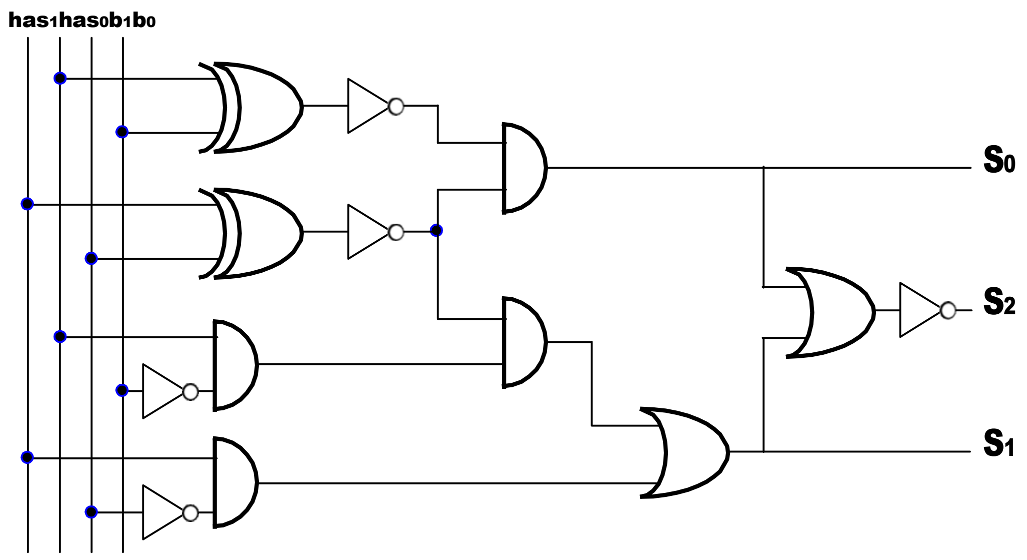
$S_1=a_1\overline{b}_1+(a_1-b_1)has_0\overline{b}_0$

And S2 is worth 1 if a1<b1 or if (a1=b1 and has 0<b0)

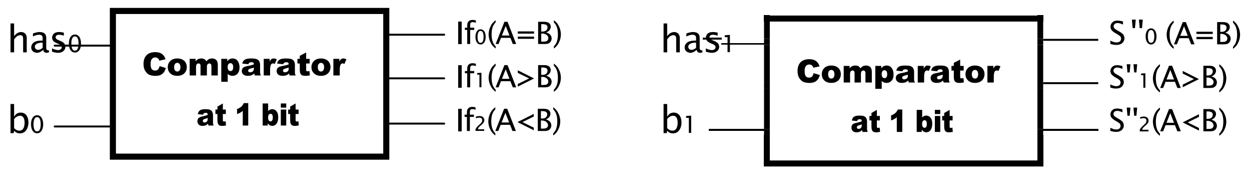
$S_2=\overline{a}_1b_1+(a_1-b_1)has_0\overline{b}_0$

$S_2=\overline{S_0+S_1}$

Logic diagram using basic logic gates



Flowchart using the 2 1-bit comparators.



$S_0=(a_1\text{-}b_1).(has_0\text{-}b_0)\ =S''_0If_0.$

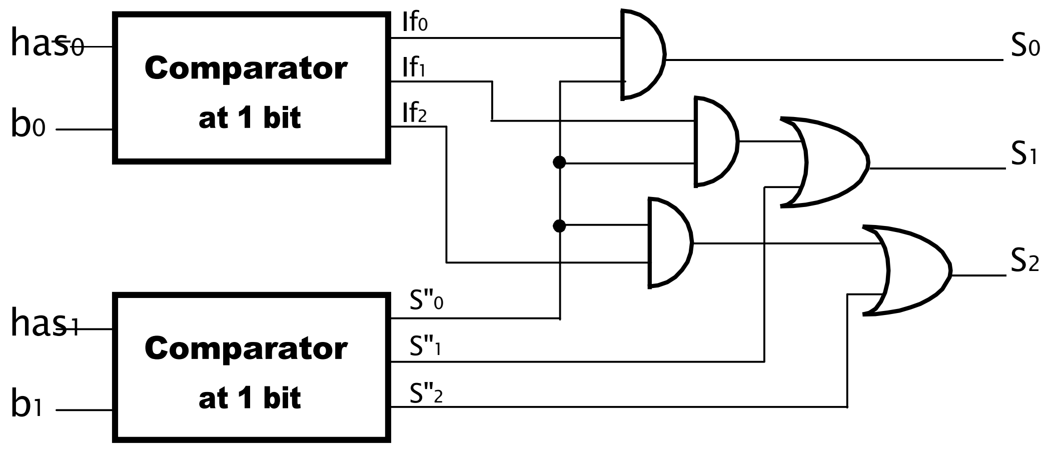
And $S_1$is worth 1 if $a_1>b_1$or if ($a_1=b_1$and $has_0>b_0$)

$S_1=a_1\overline{b_1}+(a_1\text{-}b_1)has_0\overline{b_0}=S''_1+S''_0If_1$

And $S_2$is worth 1 if $a_1<b_1$or if ($a_1=b_1$and $has_0<b_0$)

$S_2=\overline{a_1}b_1+(a_1\text{-}b_1)\overline{has_0}b_0=S''_2+S''_0If_2$

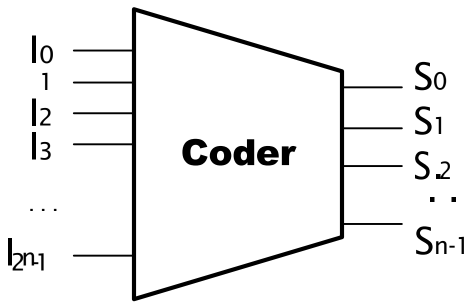
$S_2=\overline{S_0+S_1}$



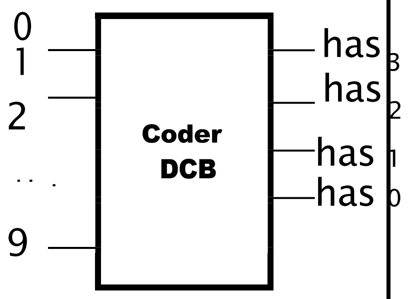
2.5 Coders and decoders

2.5.1 The coders

It is a circuit that translates the values of an input into a chosen code. An encoder (or encoder) is a logic circuit that has2ninput channels of which only one is activated andNexit routes.

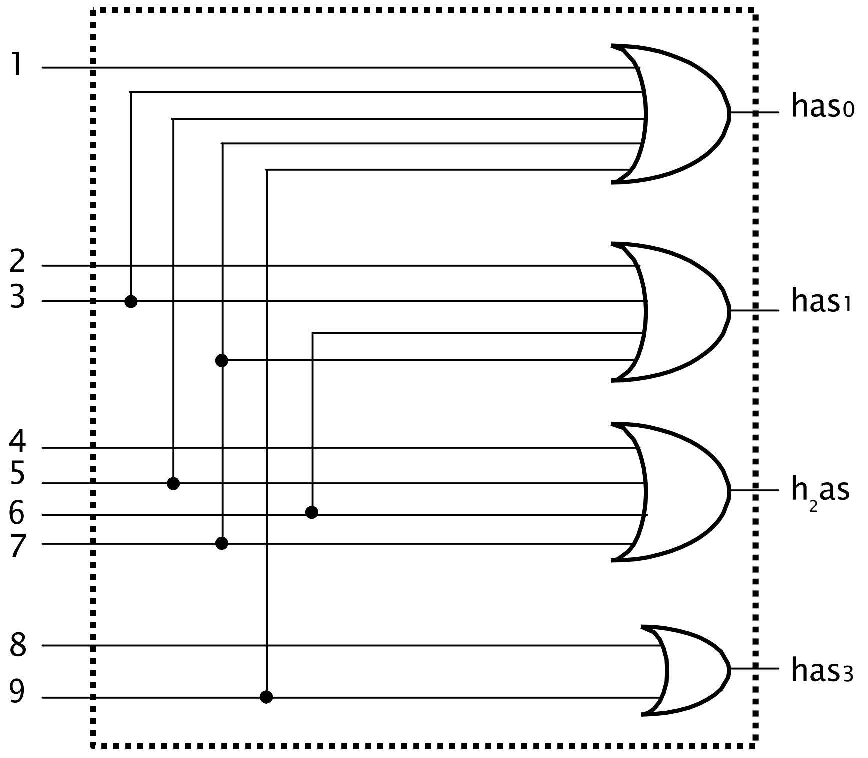


Example: DCB Encoder

Truth table	Output equation	Flowchart
	a_0= 1 + 3+5+7+9 has_1=2+3+6+7 has_2=4+5+6+7 has_3=8+9	 **Integrated circuit:** **74LS147**

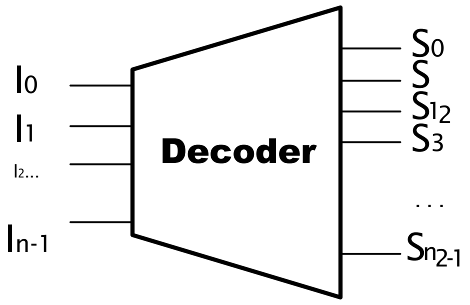
Entrances	Exits			
	has_3	has_2	has_1	has_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Flowchart:



2.5.2 The decoders

A decoder is a circuit with N inputs and 2noutputs of which only one is active at a time. It detects the presence of a specific combination of bits (code) at these inputs and indicates it by a specific output level.



Example: DCB Decoder

Operating table

Entrances				Exits
has3	has2	has1	has0	
0	0	0	0	S0
0	0	0	1	S1
0	0	1	0	S2
0	0	1	1	S3
0	1	0	0	S4
0	1	0	1	S5
0	1	1	0	S6
0	1	1	1	S7
1	0	0	0	S8
1	0	0	1	S9

Output equation

$S_0=\overline{a_3}\,\overline{has_2}\,\overline{has_1}\,\overline{has_0}$

$S_1=\overline{a_3}\,\overline{has_2}\,\overline{has_1}\,has_0$

$S_2=\overline{a_3}\,\overline{has_2}\,has_1\,\overline{has_0}$

$S_3=\overline{a_3}\,\overline{has_2}\,has_1\,has_0$

$S_4=\overline{a_3}\,has_2\,\overline{has_1}\,\overline{has_0}$

$S_5=\overline{a_3}\,has_2\,\overline{has_1}\,has_0$

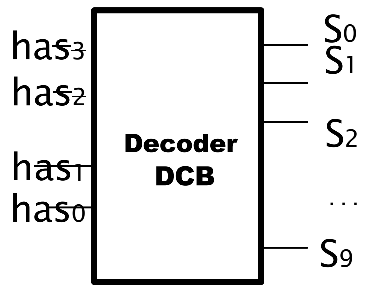
$S_6=\overline{a_3}\,has_2\,has_1\,\overline{has_0}$

$S_7=\overline{a_3}\,has_2\,has_1\,has_0$

$S_8=a_3\,\overline{has_2}\,\overline{has_1}\,\overline{has_0}$

$S_9=a_3\,\overline{has_2}\,\overline{has_1}\,has_0$

Flowchart

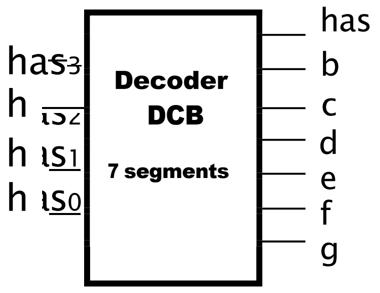


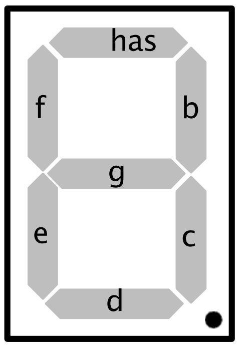
Integrated circuit:

74145

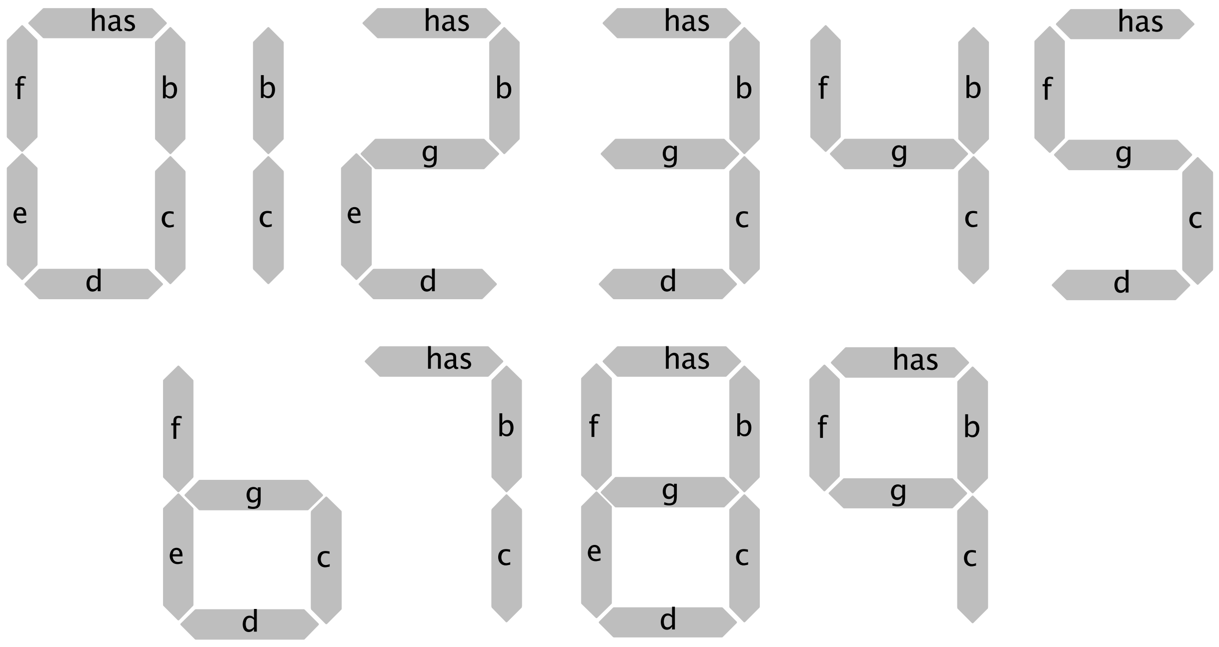
2.5.3 The 7-segment DCB decoder

The 7-segment decoder accepts 4 BCD bits as input (a_0, has_1, has_2, has_3) and activates the outputs which will allow a current to pass through the segments of a digital display to form the decimal digits (from 0 to 9).





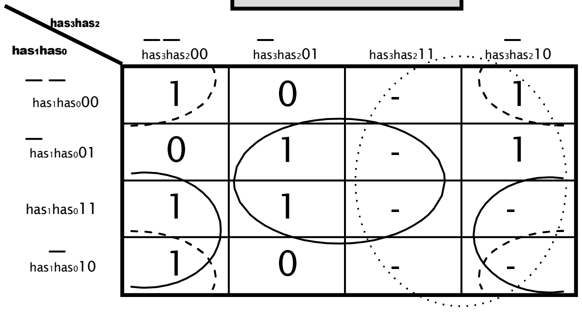
- RNote:There are 6 titled combinations10, 11, 12, 13, 14, 15which will be noted -. The other figures are displayed as follows:



Truth table											Display
Entrances				Exits							
has	3has	2has	1has	0ha	sb	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9

Example: DCB Decoder

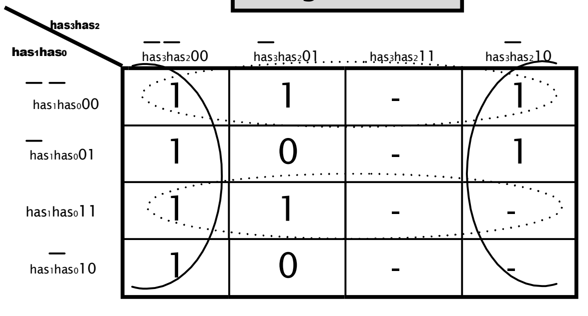
Segment a



has1has0 \ has3has2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	has3has2 11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	1	0	-	1
$\overline{has_1}has_0$01	0	1	-	1
has1has0 11	1	1	-	-
$has_1\overline{has_0}$10	1	0	-	-

$a=a_2\overline{has_1}+a_2has_0+\overline{a_2}\,\overline{has_0}+has_3$

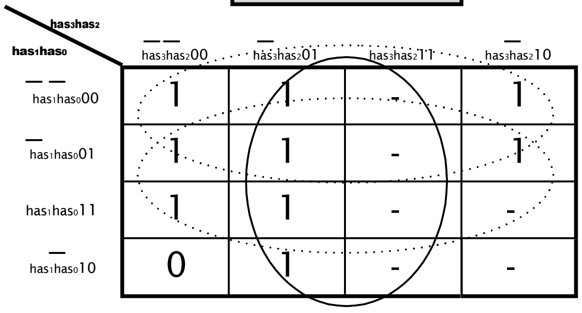
Segment b



has1has0 \ has3has2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	has3has2 11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	1	1	-	1
$\overline{has_1}has_0$01	1	0	-	1
has1has0 11	1	1	-	-
$has_1\overline{has_0}$10	1	0	-	-

$b=\overline{a_2}+\overline{a_1}\,\overline{has_0}+a_1has_0=\overline{a_2}+\overline{a_1}\text{-}has_0$

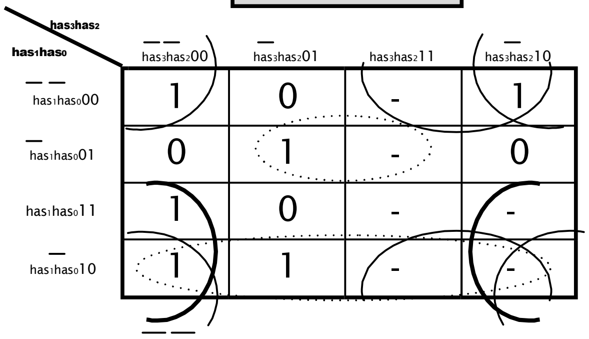
Segment c



has1has0 \ has3has2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	has3has2 11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	1	1	-	1
$\overline{has_1}has_0$01	1	1	-	1
has1has0 11	1	1	-	-
$has_1\overline{has_0}$10	0	1	-	-

$c=a_2+\overline{a_1}+a_0$

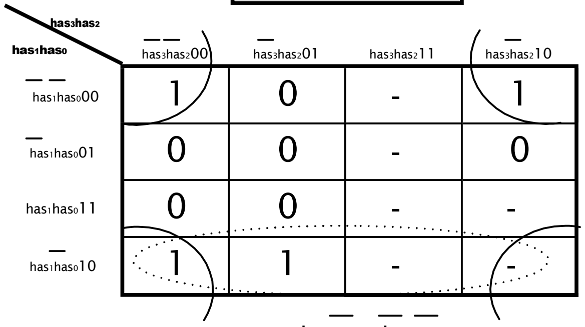
Segment d



has1has0 \ has3has2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	has3has2 11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	1	0	-	1
$\overline{has_1}has_0$01	0	1	-	0
has1has0 11	1	0	-	-
$has_1\overline{has_0}$10	1	1	-	-

$d=\overline{a_2}\,\overline{has_0}+a_3has_{\overline{0}}+a_2\overline{has}_1+a_1has_{\overline{0}}+a_2has_{\overline{1}}\overline{has}_0$

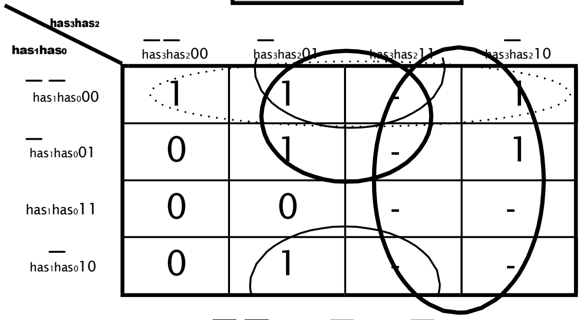
Segment e



has1has0 \ has3has2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	has3has2 11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	1	0	-	1
$\overline{has_1}has_0$01	0	0	-	0
has1has0 11	0	0	-	-
$has_1\overline{has_0}$10	1	1	-	-

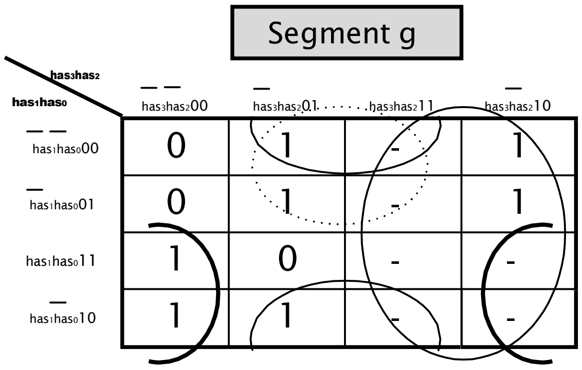
$e=a_1\overline{has_0}+\overline{a_2}\,\overline{has_0}$

Segment f



has1has0 \ has3has2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	has3has2 11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	1	1	-	1
$\overline{has_1}has_0$01	0	1	-	1
has1has0 11	0	0	-	-
$has_1\overline{has_0}$10	0	1	-	-

$f=\overline{a_1}\,\overline{has_0}+a_2\overline{has_1}+a_2\overline{has_0}+a_3$



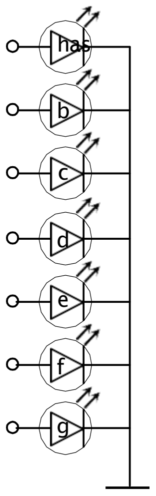
has_1has_0 \ has_3has_2	$\overline{has_3}\,\overline{has_2}$00	$\overline{has_3}has_2$01	$has_3has_2$11	$has_3\overline{has_2}$10
$\overline{has_1}\,\overline{has_0}$00	0	1	-	1
$\overline{has_1}has_0$01	0	1	-	1
$has_1has_0$11	1	0	-	-
$has_1\overline{has_0}$10	1	1	-	-

$g=\overline{a_2}has_1+a_2\overline{has_0}+a_2\overline{has_1}+a_3$

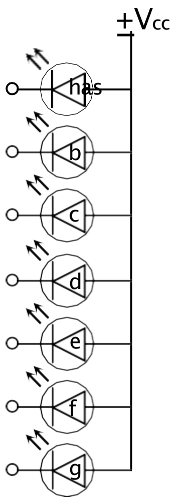
Noticed : The display is made up of 7 LEDs (segments), a, d, c, d, e, f, g which require a specific polarization depending on the type of display (common anode or common cathode):

- For a common anode display: The anodes are connected together at the high level and the decoder outputs are active at the low level (CI: 74LS47) and are connected to the cathodes of the display.

- For a common cathode display: The cathodes are connected together to ground and the decoder outputs are active at high level (CI: 74LS48) and are connected to the anodes of the display.



Cathode display
municipalities

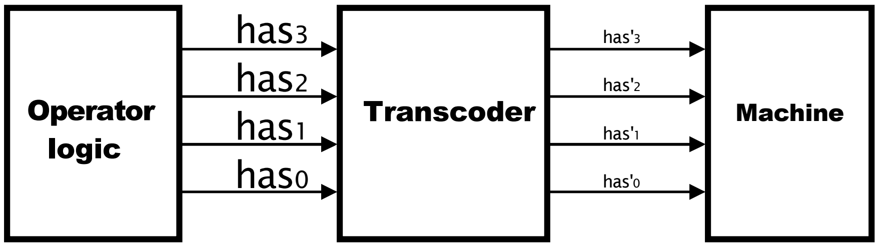


Anode display
municipalities

2.6Transcoders

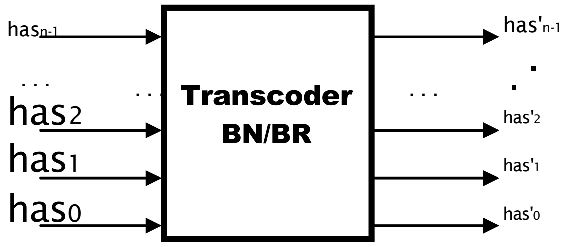
A transcoder is a circuit that allows information written in a C code to be passed1 to a C code2.

It is usually formed by a decoder cascaded with an encoder.



2.6.1 Natural Binary-Reflected Binary Transcoder

Example: BN/BR Transcoder (4 bits)



Truth table								Decimal
BN entries				BR releases				
has	3has	2has	1has	0has'3	has'2	has'1	has'0	
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	2
0	0	1	1	0	0	1	0	3
0	1	0	0	0	1	1	0	4
0	1	0	1	0	1	1	1	5
0	1	1	0	0	1	0	1	6
0	1	1	1	0	1	0	0	7
1	0	0	0	1	1	0	0	8
1	0	0	1	1	1	0	1	9
1	0	1	0	1	1	1	1	10
1	0	1	1	1	1	1	0	11
1	1	0	0	1	0	1	0	12
1	1	0	1	1	0	1	1	13
1	1	1	0	1	0	0	1	14
1	1	1	1	1	0	0	0	15

Operating table

Bit a'$_3$

has_1has_0 \ has_3has_2	$\overline{has_3}\,\overline{has_2}$ 00	$\overline{has_3}has_2$ 01	has_3has_2 11	$has_3\overline{has_2}$ 10
$\overline{has_1}\,\overline{has_0}$ 00	0	0	1	1
$\overline{has_1}has_0$ 01	0	0	1	1
has_1has_0 11	0	0	1	1
$has_1\overline{has_0}$ 10	0	0	1	1

Bit a'$_2$

has_1has_0 \ has_3has_2	$\overline{has_3}\,\overline{has_2}$ 00	$\overline{has_3}has_2$ 01	has_3has_2 11	$has_3\overline{has_2}$ 10
$\overline{has_1}\,\overline{has_0}$ 00	0	1	0	1
$\overline{has_1}has_0$ 01	0	1	0	1
has_1has_0 11	0	1	0	1
$has_1\overline{has_0}$ 10	0	1	0	1

Bit a'$_1$

has_1has_0 \ has_3has_2	$\overline{has_3}\,\overline{has_2}$ 00	$\overline{has_3}has_2$ 01	has_3has_2 11	$has_3\overline{has_2}$ 10
$\overline{has_1}\,\overline{has_0}$ 00	0	1	1	0
$\overline{has_1}has_0$ 01	0	1	1	0
has_1has_0 11	1	0	0	1
$has_1\overline{has_0}$ 10	1	0	0	1

Bit a'$_0$

has_1has_0 \ has_3has_2	$\overline{has_3}\,\overline{has_2}$ 00	$\overline{has_3}has_2$ 01	has_3has_2 11	$has_3\overline{has_2}$ 10
$\overline{has_1}\,\overline{has_0}$ 00	0	0	0	0
$\overline{has_1}has_0$ 01	1	1	1	1
has_1has_0 11	0	0	0	0
$has_1\overline{has_0}$ 10	1	1	1	1

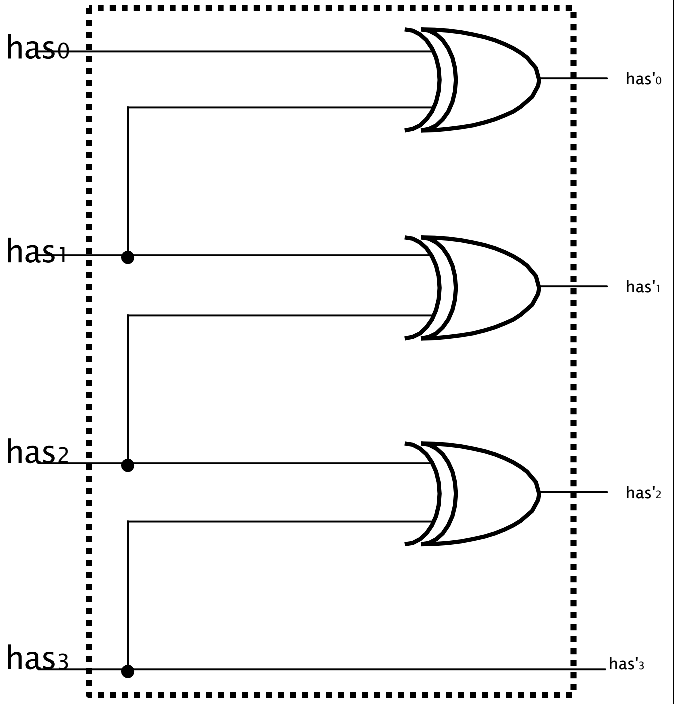
Output equation and flow chart

$has'_3=a_3$

$has'_2=a_3\text{-}has_2$

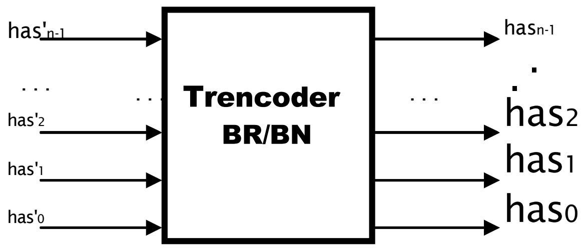
$has'_1=a_2\text{-}has_1$

$has'_0=a_1\text{-}has_0$



2.6.2 Reflected Binary Transcoder - Natural Binary

Example: BR/BN Transcoder (4 bits)



Truth table								
BR entries				BN releases				
has'$_3$	has'$_2$	has'$_1$	has'$_0$	has$_3$	has$_2$	has$_1$	has$_0$	
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	2
0	0	1	1	0	0	1	0	3
0	1	0	0	0	1	1	1	4
0	1	0	1	0	1	1	0	5
0	1	1	0	0	1	0	0	6
0	1	1	1	0	1	0	1	7
1	0	0	0	1	1	1	1	8
1	0	0	1	1	1	1	0	9
1	0	1	0	1	1	0	0	10
1	0	1	1	1	1	0	1	11
1	1	0	0	1	0	0	0	12
1	1	0	1	1	0	0	1	13
1	1	1	0	1	0	1	1	14
1	1	1	1	1	0	1	0	15

Operating table	Output equation and flowchart
see Bit a3 – Bit a0 maps below	$has_3=a'_3$ $has_2 = a'_3 - has'_2 = a_3 - has'_2$ $has_1=a_2-has'_1$ $has_0=a_1-has'_0$

Bit a3

$has'_1has'_0$ \ $has'_3has'_2$	$\overline{has'_3}\,\overline{has'_2}$ 00	$\overline{has'_3}has'_2$ 01	$has'_3has'_2$ 11	$has'_3\overline{has'_2}$ 10
$\overline{has'_1}\,\overline{has'_0}$ 00	0	0	1	1
$\overline{has'_1}has'_0$ 01	0	0	1	1
$has'_1has'_0$ 11	0	0	1	1
$has'_1\overline{has'_0}$ 10	0	0	1	1

Bit a2

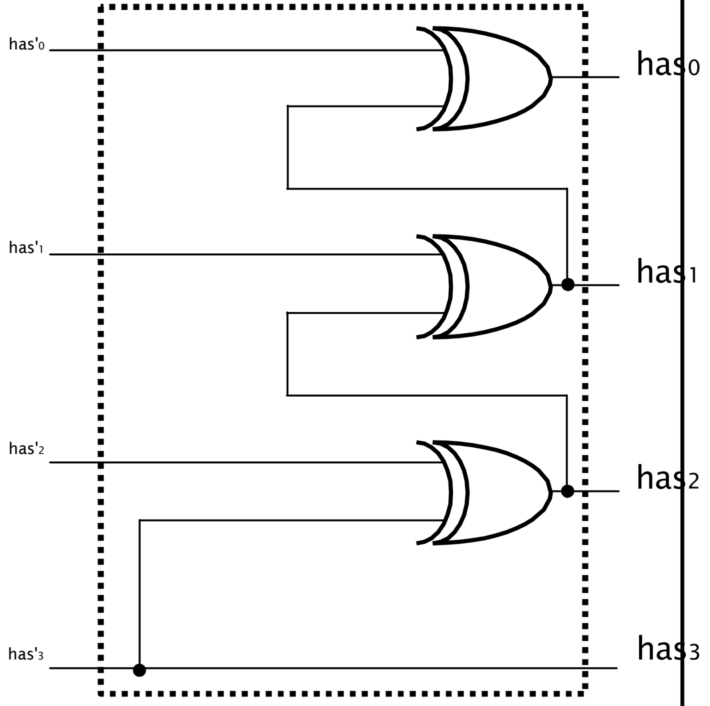
$has'_1has'_0$ \ $has'_3has'_2$	$\overline{has'_3}\,\overline{has'_2}$ 00	$\overline{has'_3}has'_2$ 01	$has'_3has'_2$ 11	$has'_3\overline{has'_2}$ 10
$\overline{has'_1}\,\overline{has'_0}$ 00	0	1	0	1
$\overline{has'_1}has'_0$ 01	0	1	0	1
$has'_1has'_0$ 11	0	1	0	1
$has'_1\overline{has'_0}$ 10	0	1	0	1

Bit a1

$has'_1has'_0$ \ $has'_3has'_2$	$\overline{has'_3}\,\overline{has'_2}$ 00	$\overline{has'_3}has'_2$ 01	$has'_3has'_2$ 11	$has'_3\overline{has'_2}$ 10
$\overline{has'_1}\,\overline{has'_0}$ 00	0	1	0	1
$\overline{has'_1}has'_0$ 01	0	1	0	1
$has'_1has'_0$ 11	1	0	1	0
$has'_1\overline{has'_0}$ 10	1	0	1	0

Bit a0

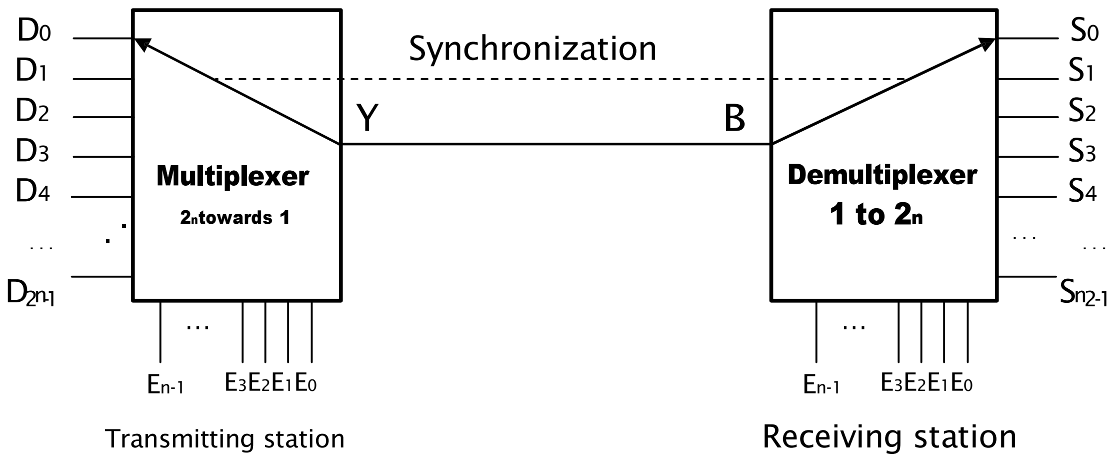
$has'_1has'_0$ \ $has'_3has'_2$	$\overline{has'_3}\,\overline{has'_2}$ 00	$\overline{has'_3}has'_2$ 01	$has'_3has'_2$ 11	$has'_3\overline{has'_2}$ 10
$\overline{has'_1}\,\overline{has'_0}$ 00	0	1	0	1
$\overline{has'_1}has'_0$ 01	1	0	1	0
$has'_1has'_0$ 11	0	1	0	1
$has'_1\overline{has'_0}$ 10	1	0	1	0



2.7 Multiplexers and demultiplexers

Transmitting information from one station to another requires several lines in parallel, which is difficult to achieve and very expensive when the stations are geometrically distant from each other.

The solution is then to transmit serially on a single line, using a parallel/serial converter (Multiplexer) at the transmitting station and a serial/parallel converter (Demultiplexer) at the receiving station.



2.7.1 Multiplexers

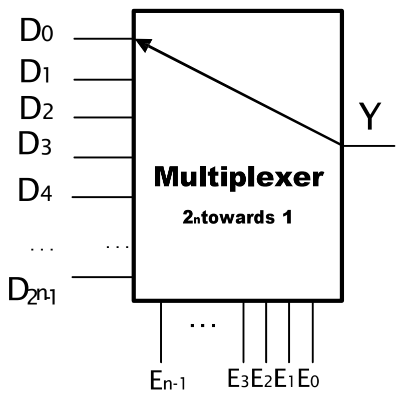
A multiplexer (MUX) is a logic circuit that has2nentries (D0, D1, D2, … Dn2-1),n selection entries (E0, E1, E2, … En-1) and only one exitY. It is said:MUX 2ntowards 1Or MUX 2nx 1.

Its function is to switch one of the inputs to the output based on the address code applied to the selection inputs.

Truth table

Truth table	**Flowchart**

Decimal	Entrances					Exits
	E_{n-1}	…	E_2	E_1	E_0	Y
0	0	…	0	0	0	D_0
1	0	…	0	0	1	D_1
2	0	…	0	1	0	D_2
3	0	…	0	1	1	D_3
4	0	…	1	0	0	D_4
5	0	…	1	0	1	D_5
….	…	…	…	…	…	…
2_n-1	1	…	1	1	1	D_{2^n-1}



Integrated circuit:

74LS157 MUX 1 of 2

74LS153 MUX 1 of 4

74LS151 MUX 1 of 8

74LS150 MUX 1 of 16

2.7.2 Demultiplexers

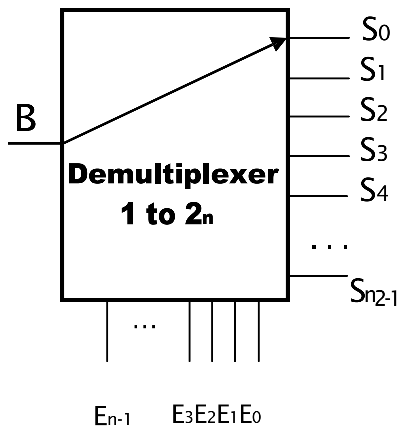
A demultiplexer (DEMUX) is a logic circuit that has a single inputB,nentries selection (E0, E1, E2, … En-1) And2nexits (S0, S1, S2, … Sn 2-1). It is said:DEMUX 1 to 2nOrDEMUX 1 x 2n.

It performs the inverse function of a multiplexer, it transmits the input data to one of the outputs according to the word written to the selection inputs, it works like a switch.

- **Truth table**

Decimal	Entrances					Exits				
	E_{n-1}	…	E_2	E_1	E_0	S_0	S_1	S_2	…	S_{n2-1}
0	0	…	0	0	0	B	0	0	…	0
1	0	…	0	0	1	0	B	0	…	0
2	0	…	0	1	0	0	0	B	…	0
3	0	…	0	1	1	0	0	0	…	0
4	0	…	1	0	0	0	0	0	…	0
5	0	…	1	0	1	0	0	0	…	0
….	…	…	…	…	…	…	…	…	…	…
2_n-1	1	…	1	1	1	0	0	0	…	B

- **Flowchart**



Integrated circuit:

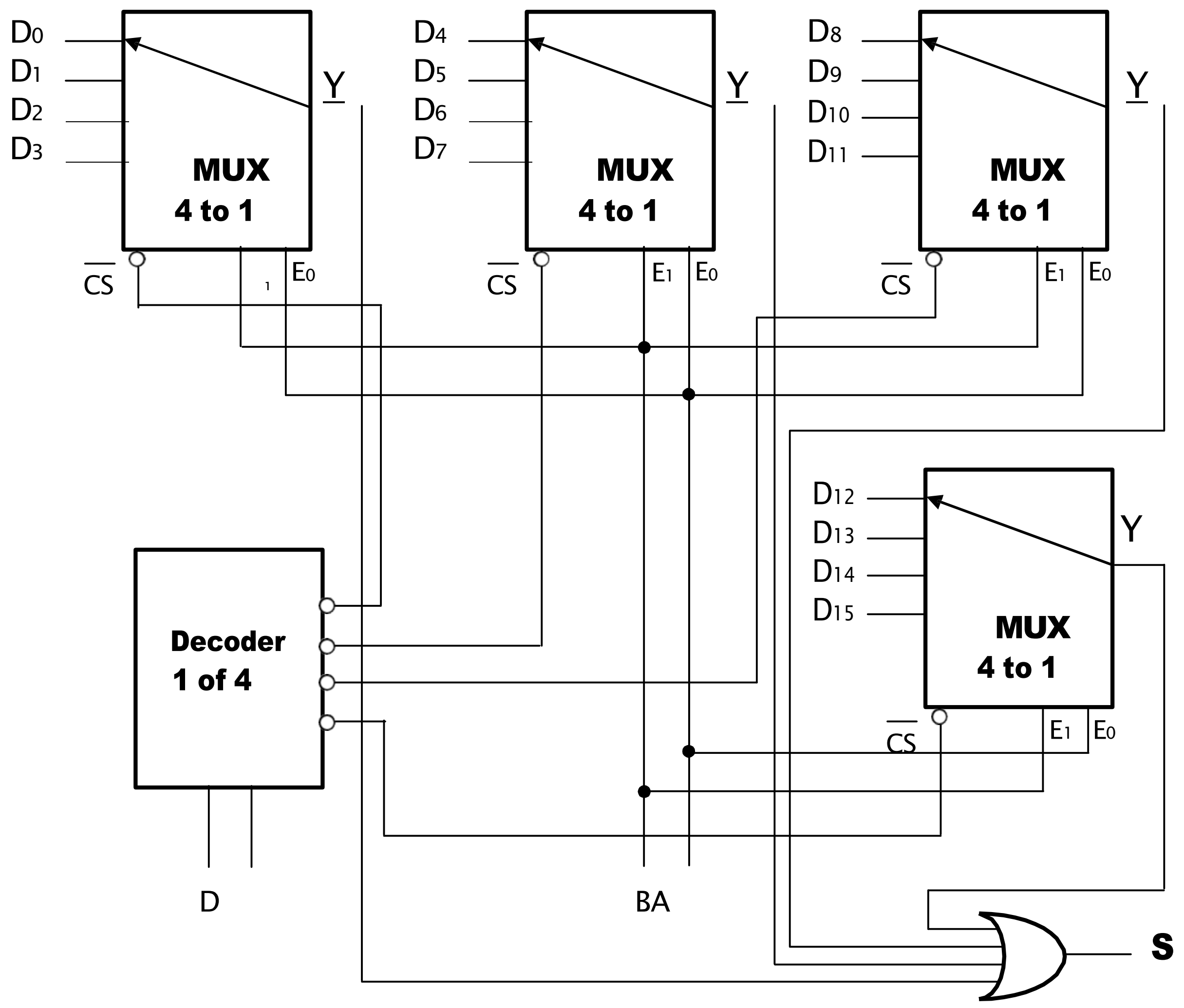
4067 DEMUX 1 to 16

74LS154 DEMUX 1 to 16

74LS138 DEMUX 1 to 8

74LS156 DEMUX 1 to 4

2.7.3 Realization of a 1 of 16 multiplexer using 4 1 of 4 multiplexers and a 1 of 4 decoder



2.7.4 Implementation of logic functions using multiplexers

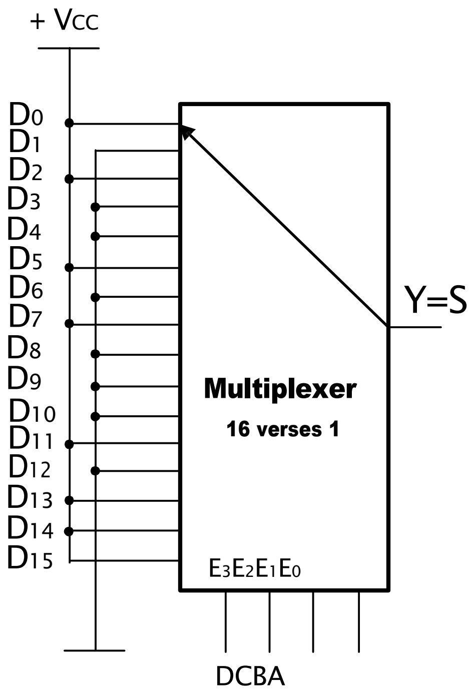
Issue

Let the function F be$_{(A, B, C, D)}$=-(0, 2, 5, 7, 11, 13, 14, 15). Perform this function using a multiplexer.

Solution

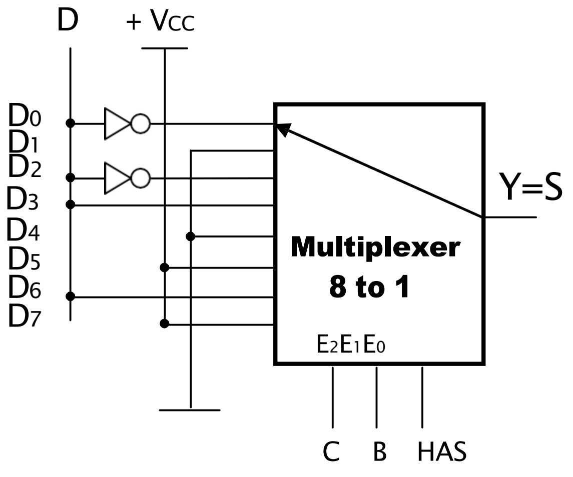
Using a 16 to 1 multiplexer (number of variables equal to the number of selection inputs).

Decimal	Entrances				Exits	
	E_3=D	E_2=C	E_1=B	E_0=A	Y	S
0	0	0	0	0	D_0	1
1	0	0	0	1	D_1	0
2	0	0	1	0	D_2	1
3	0	0	1	1	D_3	0
4	0	1	0	0	D_4	0
5	0	1	0	1	D_5	1
6	0	1	1	0	D_6	0
7	0	1	1	1	D_7	1
8	1	0	0	0	D_8	0
9	1	0	0	1	D_9	0
10	1	0	1	0	D_{10}	0
11	1	0	1	1	D_{11}	1
12	1	1	0	0	D_{12}	0
13	1	1	0	1	D_{13}	1
14	1	1	1	0	D_{14}	1
15	1	1	1	1	D_{15}	1



Using an 8 to 1 multiplexer (number of variables less than the number of selection inputs).

D \ **CBA**	~~CBA(000)~~	CBA(001)	CBA(010)	CBA(011)	CBA(100)	CBA(101)	CBA(110)	CBA(111)
$\overline{D}$(0)	$\textcircled{0}$	1	$\textcircled{2}$	3	4	$\textcircled{5}$	6	$\textcircled{7}$
D(1)	8	9	10	$\textcircled{11}$	12	$\textcircled{13}$	$\textcircled{14}$	$\textcircled{15}$
	$D_0=\overline{D}$	$D_1=0$	$D_2=\overline{D}$	$D_3=D$	$D_4=0$	$D_5=1$	$D_6=D$	$D_7=1$



CHAPTER 5
SEQUENTIAL LOGIC

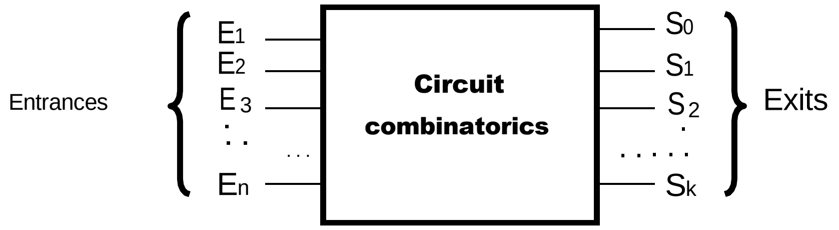
1. OBJECTIVES

- Treat sequential systems in detail.
- Understanding flip-flops.

2.INTRODUCTION

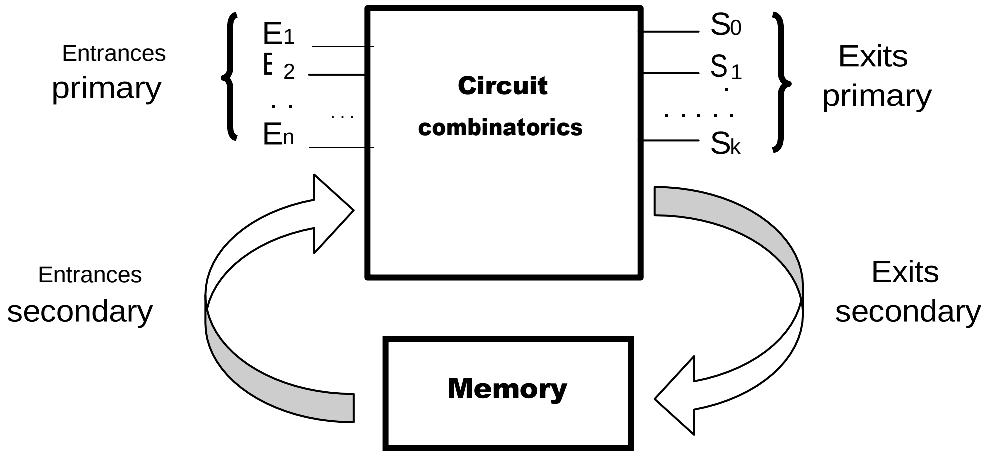
2.1 Reminder on combinational circuits

In a combinational system, the outputs depend only on the state of the inputs at a given time.



2.2 Sequential circuits

The output function of sequential systems depends in addition to the states of the inputs (called primary inputs) on the previous states of the outputs (called secondary inputs). The sequential circuit is said to have a memory function.



Sequential systems are classified into 2 categories:

- **Asynchronous sequential circuits**

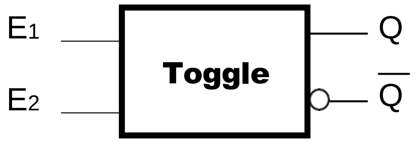
In asynchronous sequential circuits, the outputs change states as soon as the input states change.

- **Synchronous sequential circuits**

In this type of circuit the outputs change state after having been authorized by a synchronization signal often called "Clock" signal noted H or CLK.

3. ASYNCHRONOUS FLIP-FLOATS

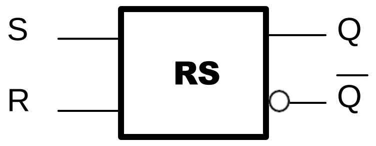
The flip-flop is the most common memory circuit. It also serves to create a frequency divider by two. It is a sequential system consisting of one or two inputs and two complementary outputs.



The flip-flop is the most common storage circuit. It also serves to create a frequency divider by two. It is a sequential system consisting of one or two inputs and two complementary outputs.

It is called a "bistable flip-flop" because it has two stable states. There are 4 types of flip-flops:RS,D,JK, AndT.

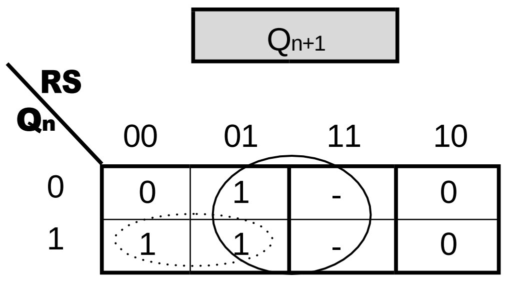
3.1 RS rocker

Symbol	Explanation
	-S -R A pulse onS (set)-Up to 1 ofQ((walk) A pulse on R (Reset)-Reset to 0deQ(Stop)

Truth table

Entrances			Exits		Mode of functioning
R	S	Q_n	Q_{n+1}	$\overline{Q}_{n+1}$	
0	0	0	0	1	Previous state
0	0	1	1	0	Previous state
0	1	0	1	0	Engagement
0	1	1	1	0	Maintain at 1
1	0	0	0	1	Maintain at 0
1	0	1	0	1	Triggering
1	1	0	-	-	Forbidden
1	1	1	-	-	Forbidden

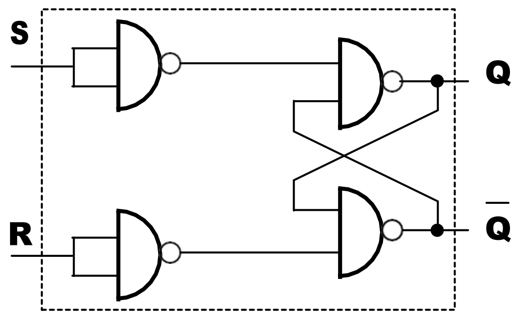
Output equation



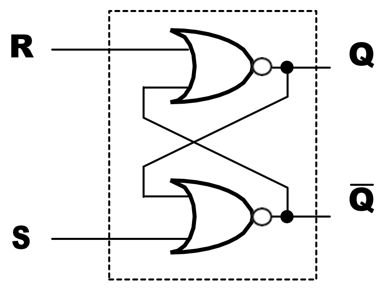
$$Q_{n+1}=\overline{R}Q_n+S$$

Flowchart

Using NAND gates

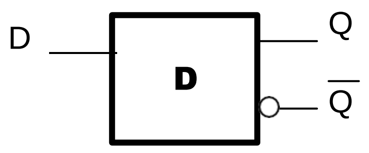


Using NOR gates



NB: The state R=S=1e is a forbidden state since it gives us the two complementary outputs Q and Q in the same state which is not logical.

3.2 D-Flip

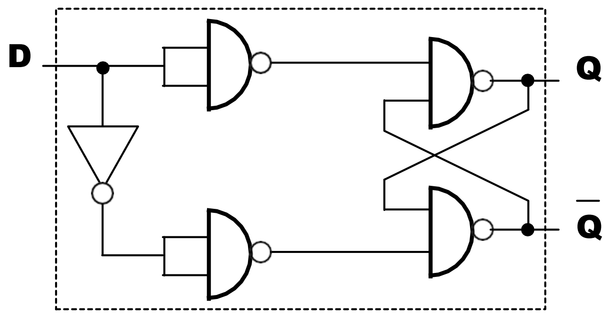
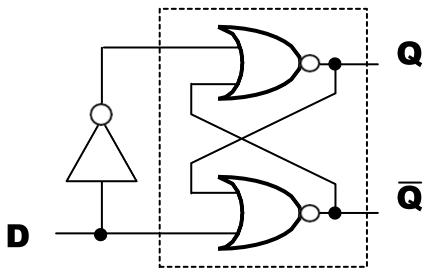
Symbol	Explanation
	A press on D-Up to 1 of Q A release of D-Reset to 0Q

Truth table					Output equation
Entrances		Exits		Mode of functioning	
D	Qn	Qn+1	Qn+1		
0	0	0	1	Maintain at 0: -0	
0	1	0	1	Trigger: -	
1	0	1	0	Engagement: -	
1	1	1	0	Maintain at 1: -1	

Qn+1

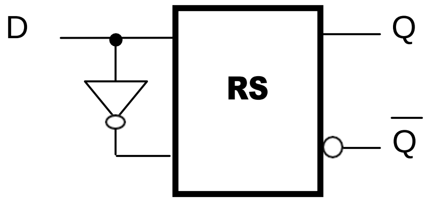
Qn \ D	0	1
0	0	1
1	1	1

Qn+1=D

Flowchart	
Using NAND gates	Using NOR gates
	

Noticed : By putting S=D and R=Din the seesaw equation RSwe will haveQn+1 =DQn+D=D(1+Qn)=D.

So we get a flip-flopDby adding an inverter between S And R.



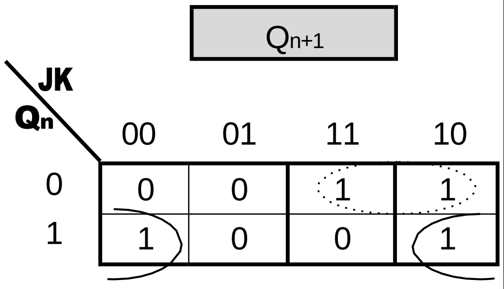
3.3 JK rocker

Unlike the seesaw RS, the condition J=K=1, does not give rise to an indeterminate condition, but on the other hand the flip-flop goes to the opposite state.

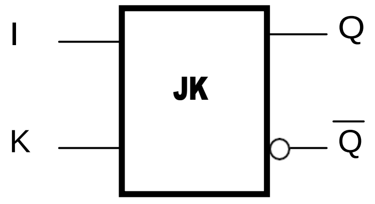
Truth table

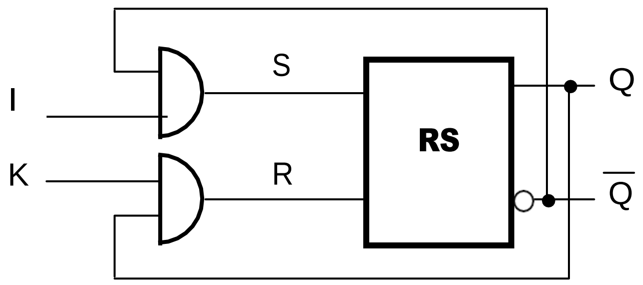
Entrances			Exits		Mode of functioning
I	K	Qn	Qn+1	Qn+1	
0	0	0	0	1	Previous state
0	0	1	1	0	Previous state
0	1	0	0	1	Maintain at 0: -0
0	1	1	0	1	Trigger: -
1	0	0	1	0	Engagement: -
1	0	1	1	0	Maintain at 0: -1
1	1	0	1	0	Engagement: -
1	1	1	0	1	Trigger: -

Output equation



$$Q_{n+1}=J\overline{Q}_n+\overline{K}Q_n$$





3.4 T-toggle

The seesaw T is obtained by connecting the inputs I And K of a seesaw JK.

Truth table

Entrances		Exits		Mode of functioning
T	Q_n	Q_{n+1}	$\overline{Q_{n+1}}$	
0	0	0	1	Maintain at 0: -0
0	1	1	0	Maintain at 1: -1
1	0	1	0	Engagement: -
1	1	0	1	Trigger: -

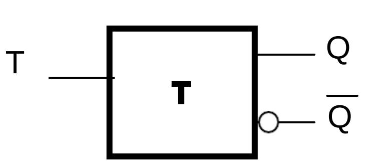
Output equation

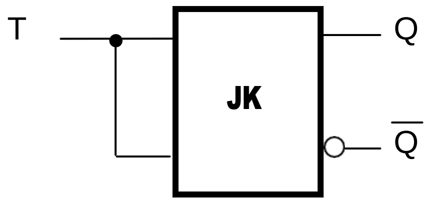
Q_{n+1}

Q_n \ T	0	1
0	0	1
1	1	0

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

Noticed :En refill there cient I And K by T in the seesaw equation JK we will have Q n+1=TQn+TQn=TQn.

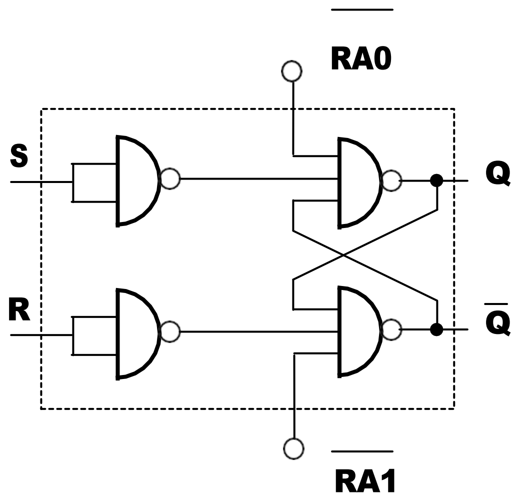


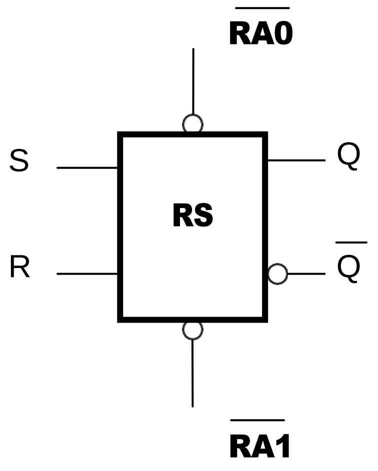


3.5 Forcing the flip-flops

Some scales are equipped with special inputs:

- Reset input: PRESET (RA1),
- Reset input: RESET (RA0),





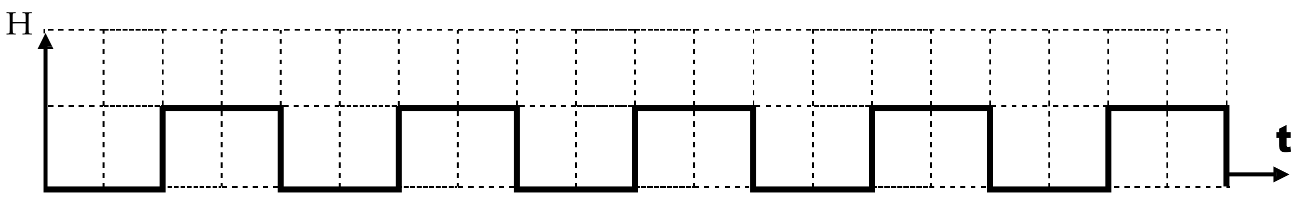
The same reasoning is applied to the D, T and JK flip-flops.

3.5.1 Truth table

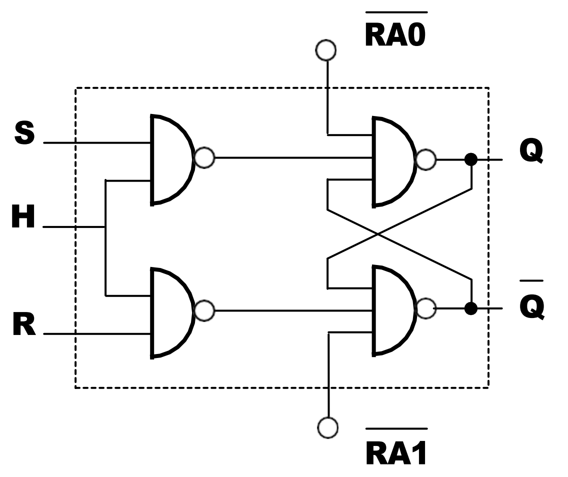
Entrances		Exits		Mode of functioning
PRESET	CLEAR	Q_{n+1}	$\overline{Q_{n+1}}$	
0	0	Q_n	$\overline{Q_n}$	Memorization
0	1	0	1	Force to 1
1	0	1	0	Force to 0
1	1	-	-	Forbidden

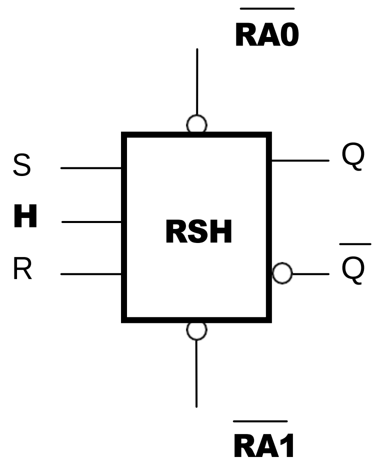
4. SYNCHRONOUS FLIP-FLOATS

A flip-flop is synchronous when its outputs only change state if an additional signal is applied to an input, called input clock (noted HOrCLK).



4.1 Synchronization on high level



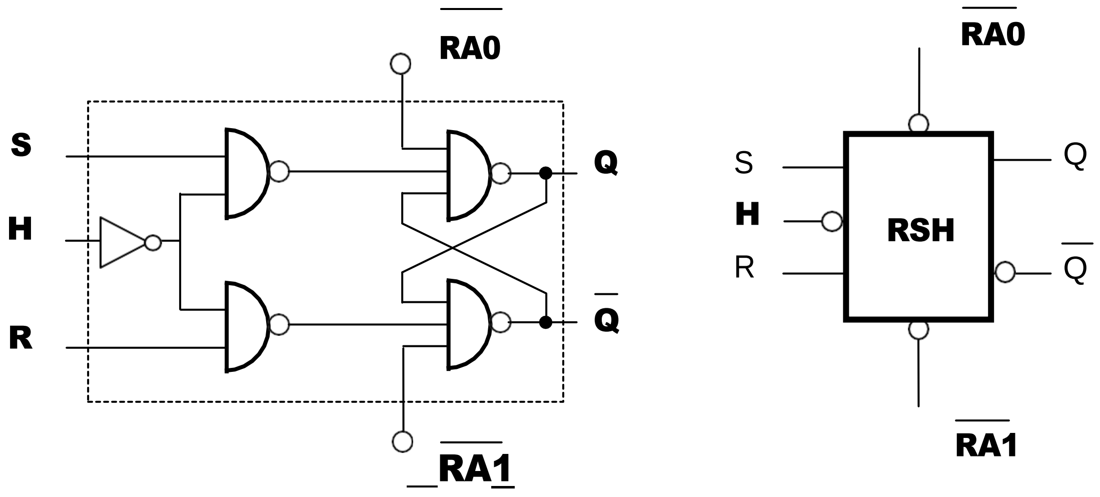


- IfH=0:the exits$\overline{S}$And$\overline{R}$are stuck at1whatever they areRAndS, (inputs are hidden from outputs) the output keeps the previous state.
- IfH=1:the seesawRSworks normally the outputs obey the entries.
- So the switchRSonly works normally ifH=1(High Level).
- Same thing for the other switches.

4.2 Synchronization on high level

At the lower level, the opposite is true:

- IfH=1:Qkeeps the previous state.
- IfH=0:Normal operation of the scale.



- IfH=1:the exitsSAndRare stuck at1whatever they areRAndS, (inputs are hidden from outputs) the output keeps the previous state.

- IfH=0:the flip-flop works normally the outputs obey the inputs.
- So the switchRSonly works normally ifH=0(Low level). The synchronous flip-flop is identical to the asynchronous one.
- Same thing for the other switches.

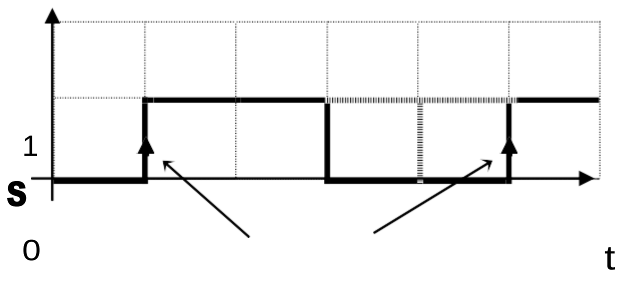
Noticed :

This type of synchronization (on level) has many disadvantages: the flip-flop is sensitive to inputs for the entire duration of the clock state for high level (or 0 for low level). If, while H = 1 (or H = 0), parasites appear on the S and R inputs, they can cause unexpected state changes on the Q output.
In order to minimize the duration of this sensitive state as much as possible, we arrange for the flip-flop to remain in its memory state except for a brief instant, just when the input changes from 0 to 1 (or from 1 to 0).

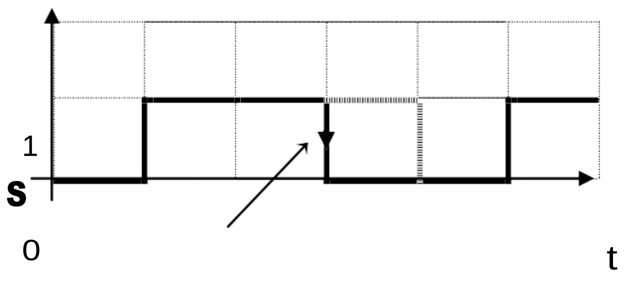
The flip-flop is said to be edge-synchronized.

4.3 Edge synchronization

A logical variableScan have two levels: high level (True) or low logic level (False). When it goes from low level to high level, it sets the<u>rising front</u> . Otherwise, it defines the<u>falling edge</u> .

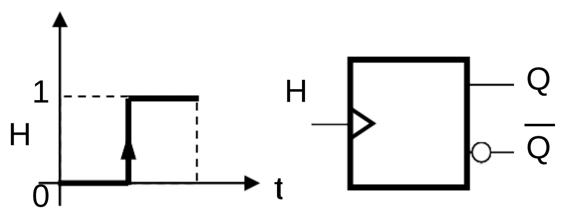
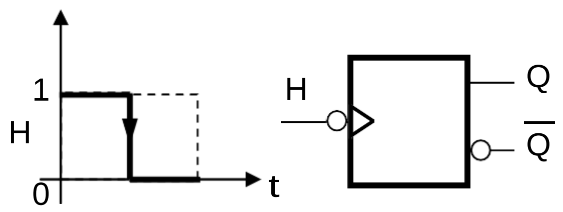


Rising front



Falling edge

Symbol :

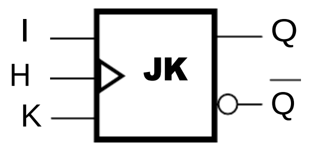
Rising front	Falling edge
	

4.4 Operating principle of a JK flip-flop synchronized on rising edge

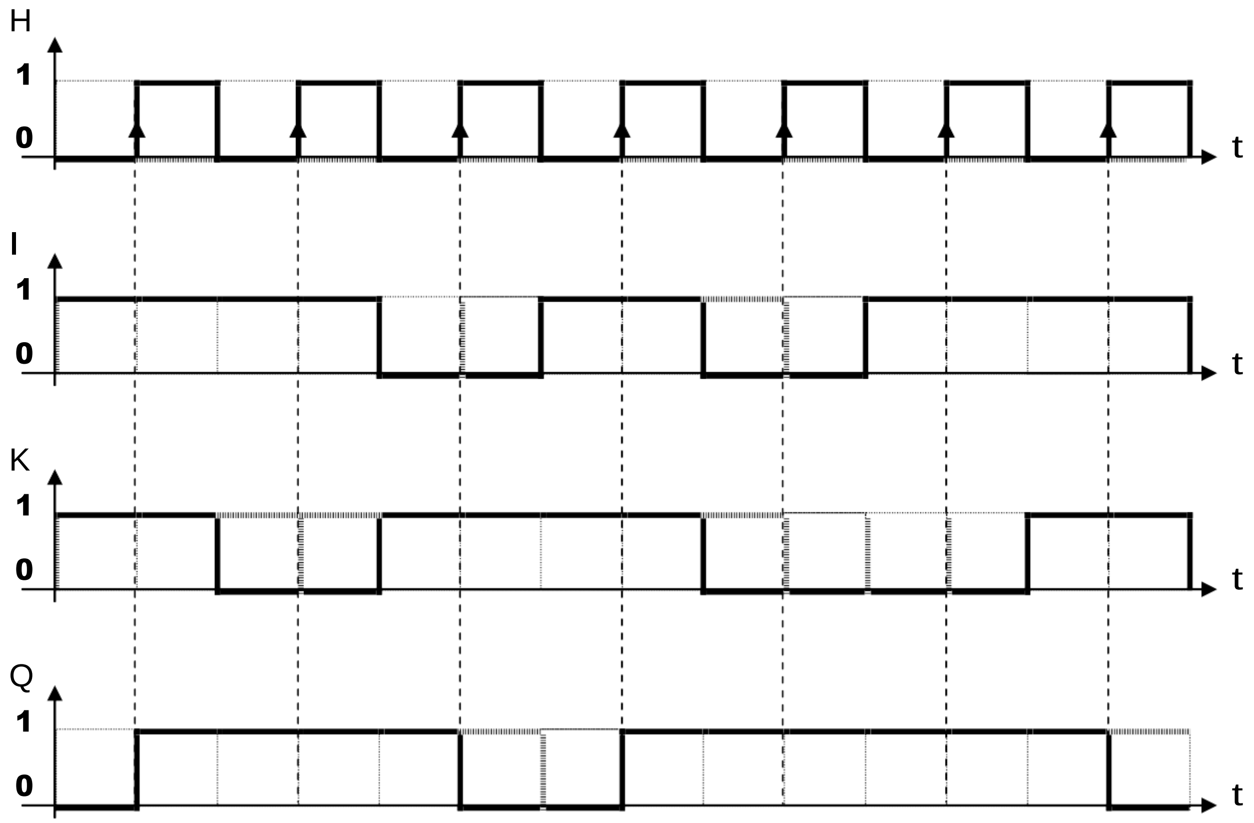
Operating table

Entrances			Exits		Mode of operation
H	I	K	Qn+1	Qn+1	
0	x	x	Q_n	$\underline{Q_n}$	Previous state
1	x	x	Q_n	$\underline{Q_n}$	Previous state
-	x	x	Q_n	$\underline{Q_n}$	Previous state
-	0	0	Q_n	Q_n	Previous state
-	0	1	0	1	Trigger: -
-	1	0	$\underline{1}$	0	Engagement: -
-	1	1	Q_n	Q_n	change of state

Symbol

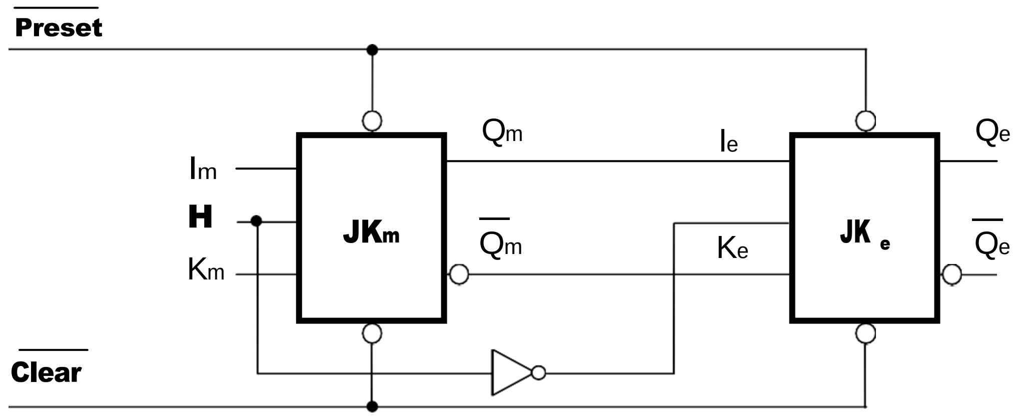


Timeline:



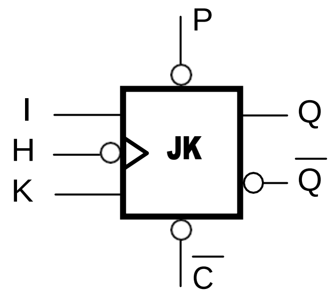
4.5 JK master slave switch

4.5.1 Synchronization on rising edge

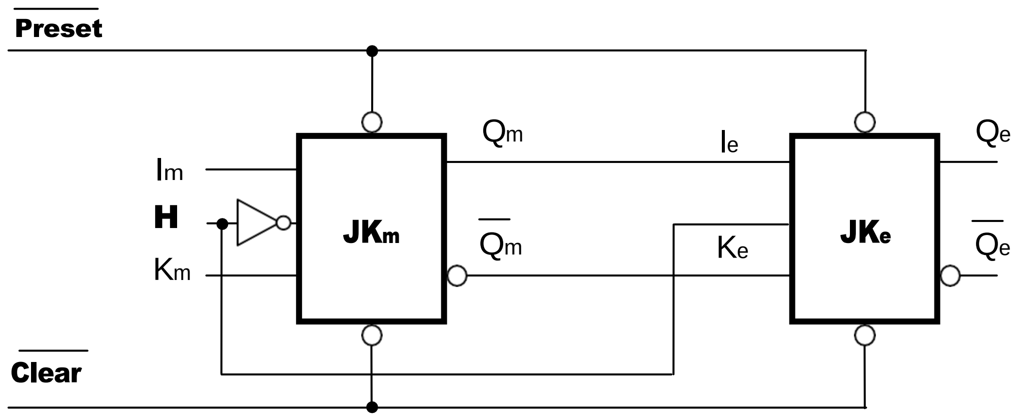


Both flip-flops operate normally if $\overline{PRESET}=\overline{CLEAR}=1$ and if H=1 the first flip-flop operates normally while the second is blocked and when H=0 the first flip-flop is blocked while the second operates
normally and the two flip-flops only work together at the time of
passage of H from 1 to 0, that is to say at the moment of the falling edge (-).

So any master-slave flip-flop where the master is working on the high level and the slave is working on the low level is a falling-edge synchronized flip-flop.

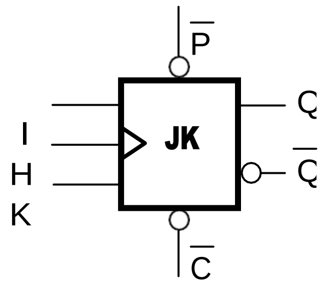


4.5.2 Synchronization on rising edge



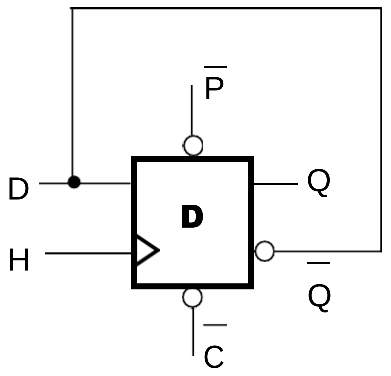
The two flip-flops operate normally if $\overline{PRESET}=\overline{CLEAR}=1$ and if H=0 the first flip- flop operates normally while the second is blocked and when H=1 the first flip- flop is blocked while the second operates normally and the two flip-flops only operate together at the moment of passage of H from 0 to 1, that is to say at the moment of the rising edge (-).

So any master-slave flip-flop where the master is working on the low level and the slave is working on the high level is a rising-edge synchronized flip-flop.

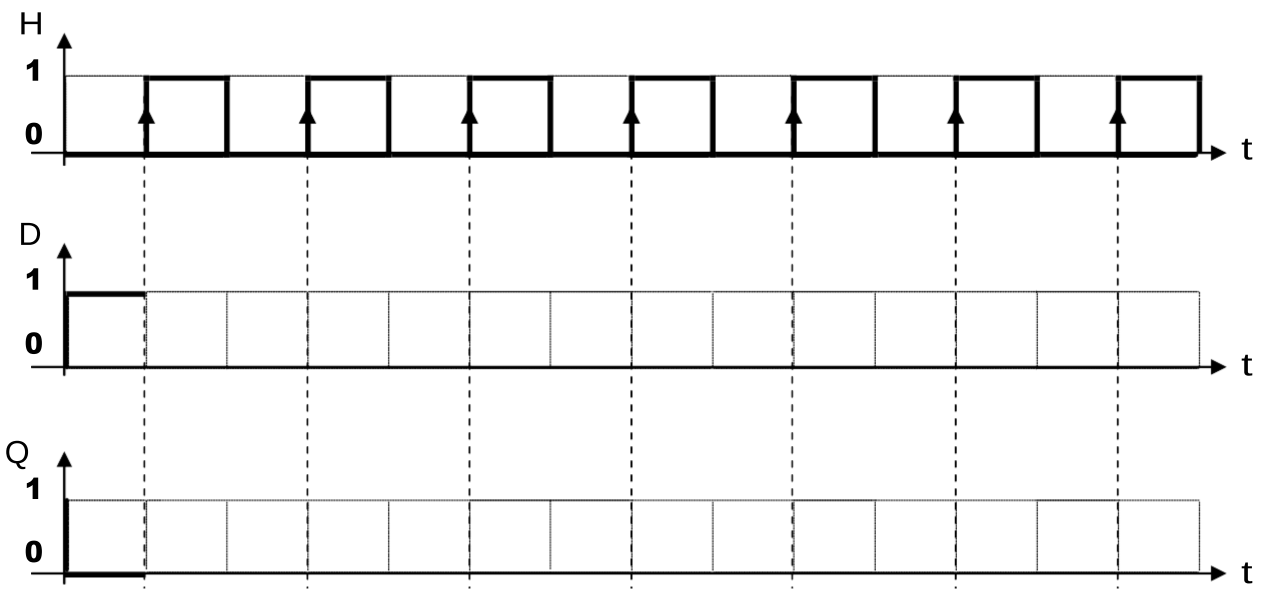


Exercise

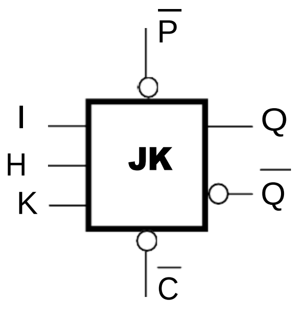
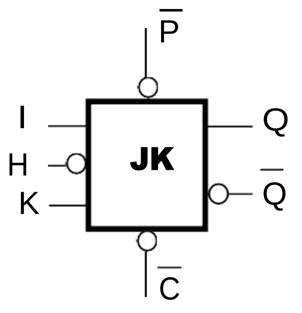
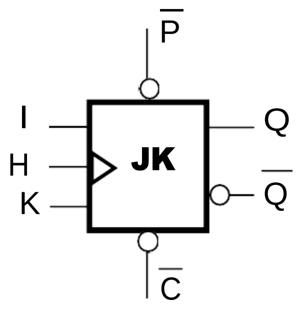
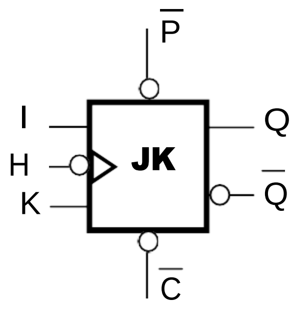
Let's take the following setup:



Complete the chronogram of D and Q. Deduce the function thus produced.



1.1 ***Summary***

Synchronization on high level	Synchronization on low level	Synchronization on rising front	Synchronization on falling edge
			

CHAPTER 6
THE REGISTERS

1.OBJECTIVES

- Study the different types of register
- Know the operating principle of each type.

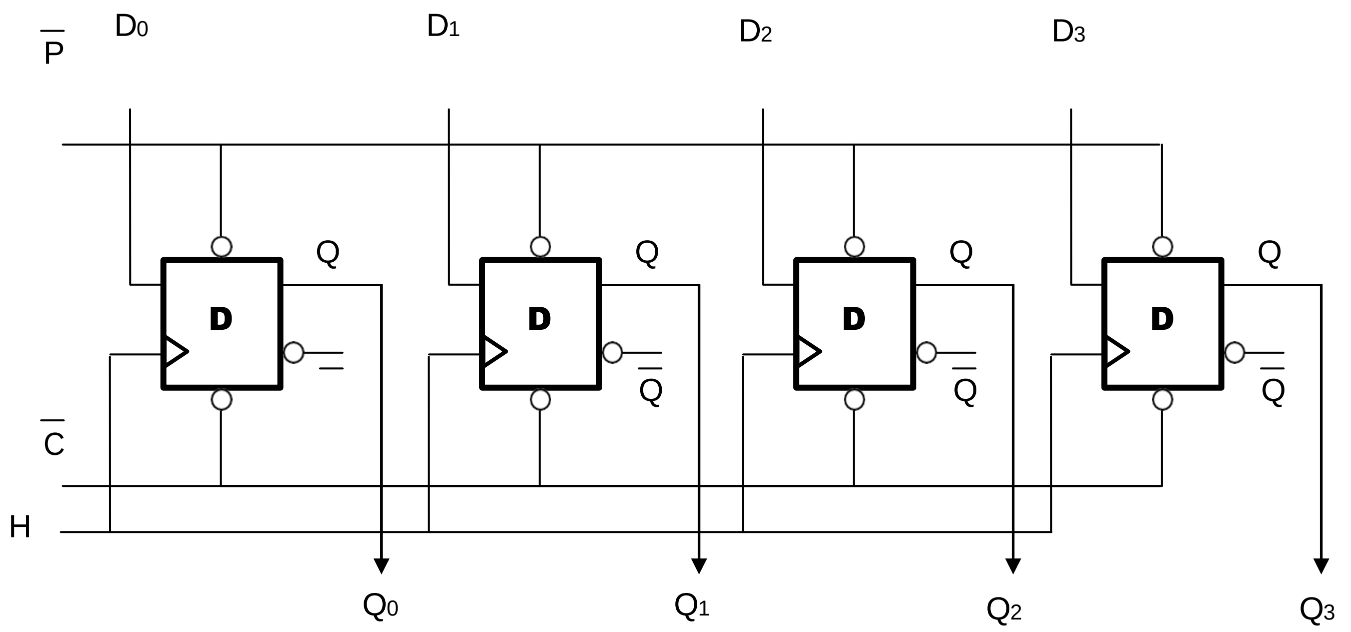
2. GENERAL INFORMATION

- A register is a collection of basic memory cells.
- Data can be written/read at the same time (parallel) or one after the other (serial).
- The number of bits in the register corresponds to the number of memory cells (number of D or JK flip-flops) in the register.
- Note that all clock inputs (H) of the cells are connected (write line).
- The registers are classified by:
  - ✓ The number of bits.
  - ✓ The operating mode (single or multiple).
- The classification of operating modes is as follows:
- Parallel input and parallel output registers:PIPO*(Parallel IN-Parallel OUT).*
- Parallel input and serial output registers:PISO*(Parallel IN-Serial OUT).*
- Registers with serial inputs and parallel outputs:SIPO*(Serial IN - Parallel OUT).*
- Serial input and serial output registers:SISO*(Serial IN- Serial OUT).*

3. STORAGE REGISTER (Parallel Register)

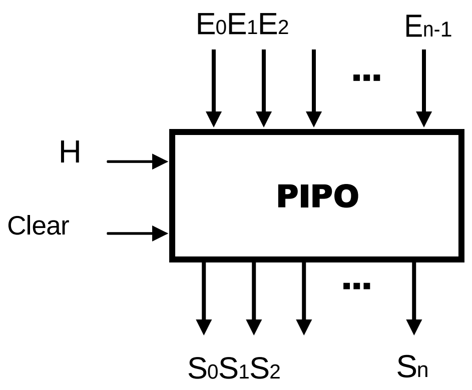
A storage register (or data register) is a register in which the different stages are independent of each other, however certain signals act on all the stages; such as reset to 0 and reset to 1.

3.1 bit storage register



In the example below, the 4 flip-flops are loaded in parallel and read in parallel synchronously with the write signal H. This type of register is also called a registerPIPO.

3.2 Functional diagram of a PIPO register.

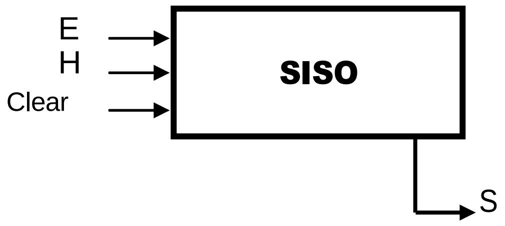


4. SHIFT REGISTER (Serial Register)

This type of register is mainly used as dynamic information memory; the shift function consists of sliding the information from each elementary cell into another adjacent elementary cell.

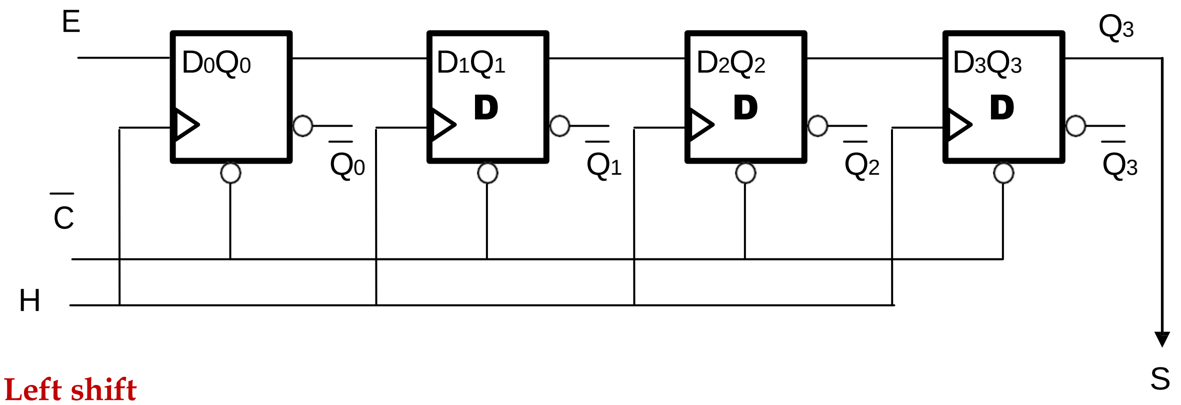
This type of register is also called a registerSISO.

4.1 Functional diagram



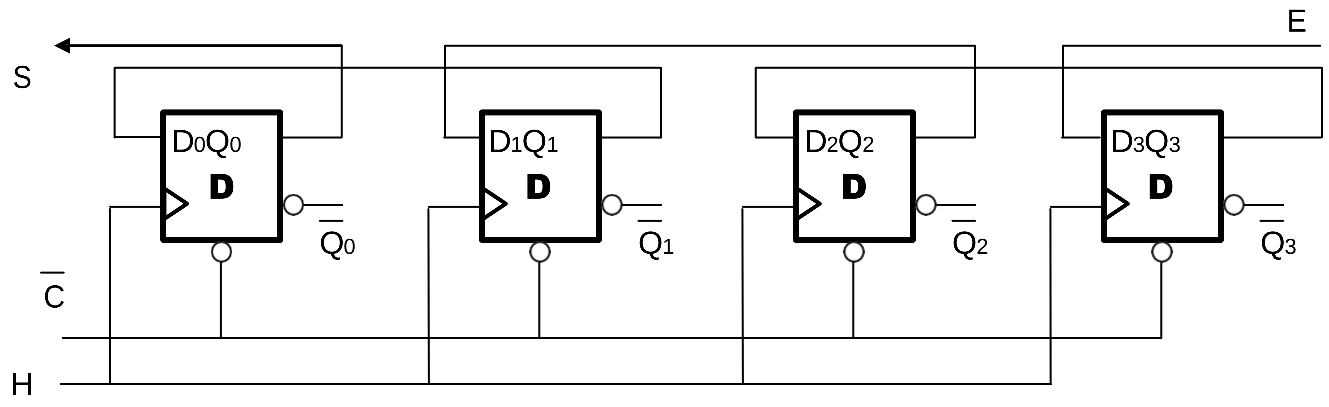
4.2Right shift

The flip-flop of rank i must copy the output of the flip-flop of rank (i-1) so its input must be connected to the output (i-1).



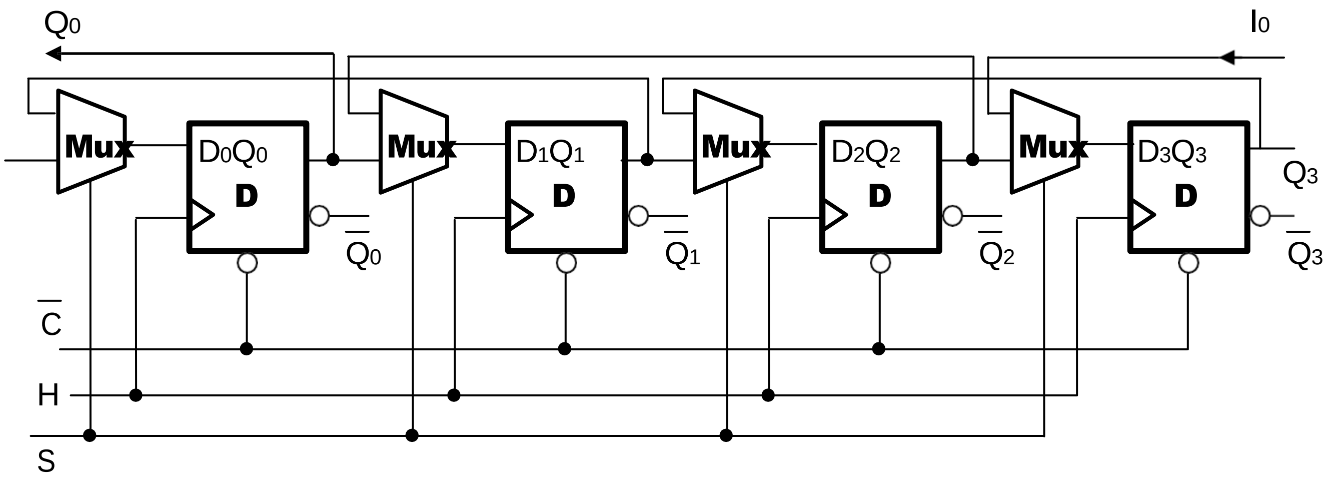
4.3 Left shift

The input of the flip-flop of rank i must copy the output of the flip-flop of rank (i+1).



4.4 Reversible shift

There are reversible shift registers, that is to say registers where the shift is carried out to the right and to the left depending on the logic level applied to the S input: "shift direction".



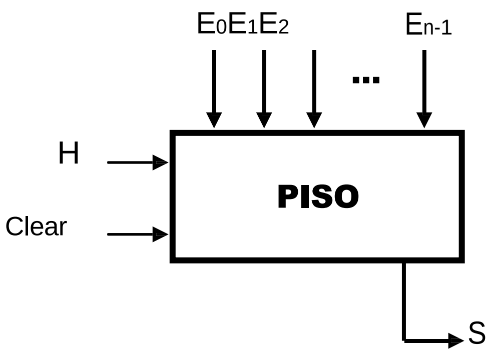
Depending on the value of the input S, we have the following operation:

S	Operation
0	Left shift
1	Right shift

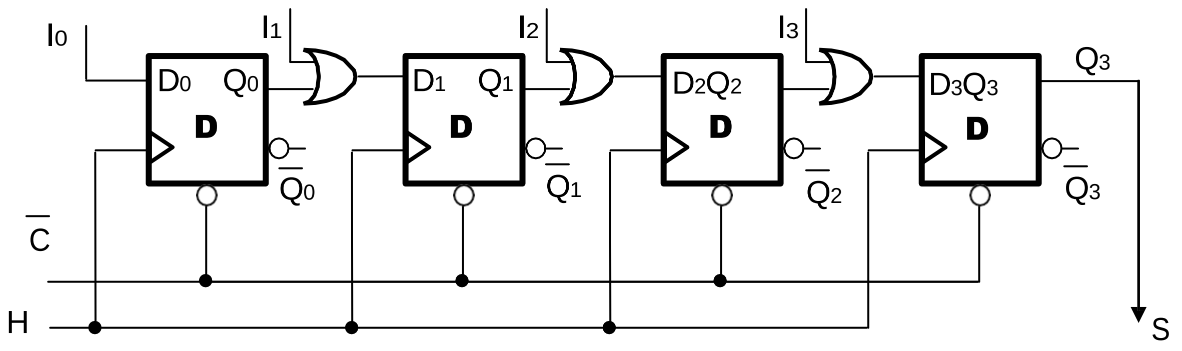
5. MIXED REGISTER

We can find mixed registers, so we can write in parallel and read in serial (PISO), or vice versa (PISO), or which offer both possibilities "to choose from".

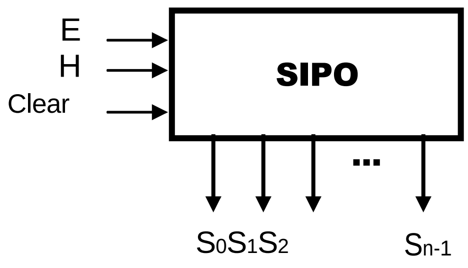
5.1PISO Registry



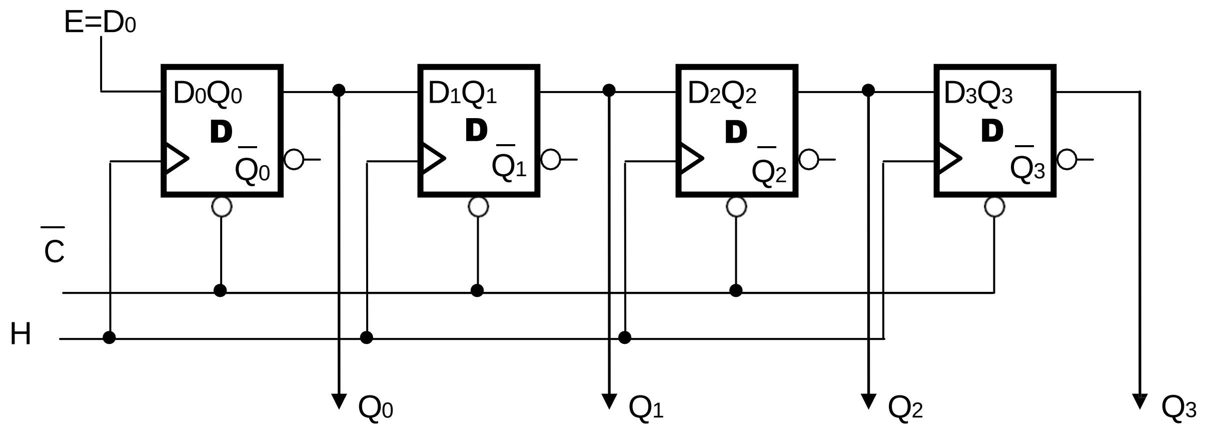
5.1.1 Flowchart using D flip-flops



5.2 SIPO Register



Flowchart using D flip-flops



5.3 Application example

Two types of registers (PISOAndSIPO) are used in serial connections; they form the basis of modems. For example, if we want to transmit information between two computers a few dozen meters apart. Transmitting information in parallel requires a lot of wires and is very expensive. The solution is then to use a register PISO to send the bits on a single line. At the end of which, a register SIPO receives these bits and reconstructs bytes which are transmitted to the destination computer.

CHAPTER 7
THE COUNTERS

1. OBJECTIVES

- Study the different types of meters.
- Understand the operating principle of each type.
- Master the steps of synthesizing a counter.

2. INTRODUCTION

In many applications we are led to do counting: counting pulses in a given time for frequency measurement for example. In one case it is necessary to count in other it is necessary to count down from zero or another given number. A counter, in the broad sense of the term, will be likely to function as a counter itself (up counter) or even in down counter (down counter) and in which we can introduce any starting number, that is to say that we can initialize or load.
Counters can be classified according to their principle as follows:

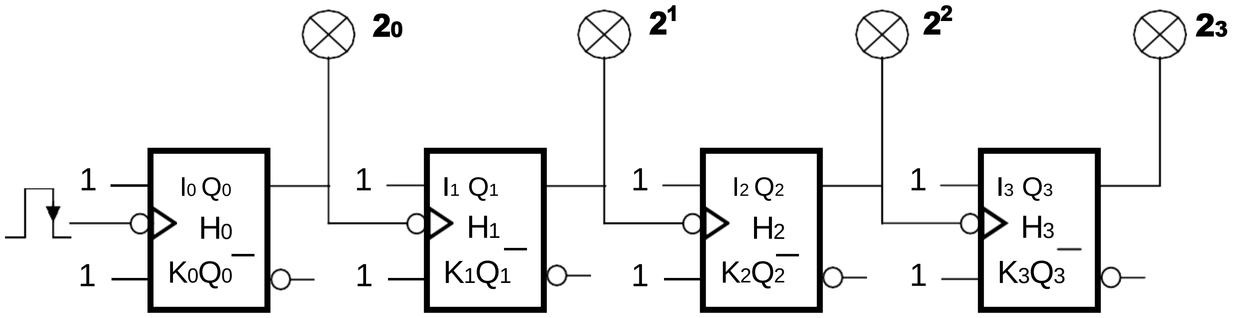
- Asynchronous up-down counters.
- Synchronous up-down counters.
- The basic element of counters is a clock-input flip-flop (synchronous flip-flop), either D, T, or JK type.

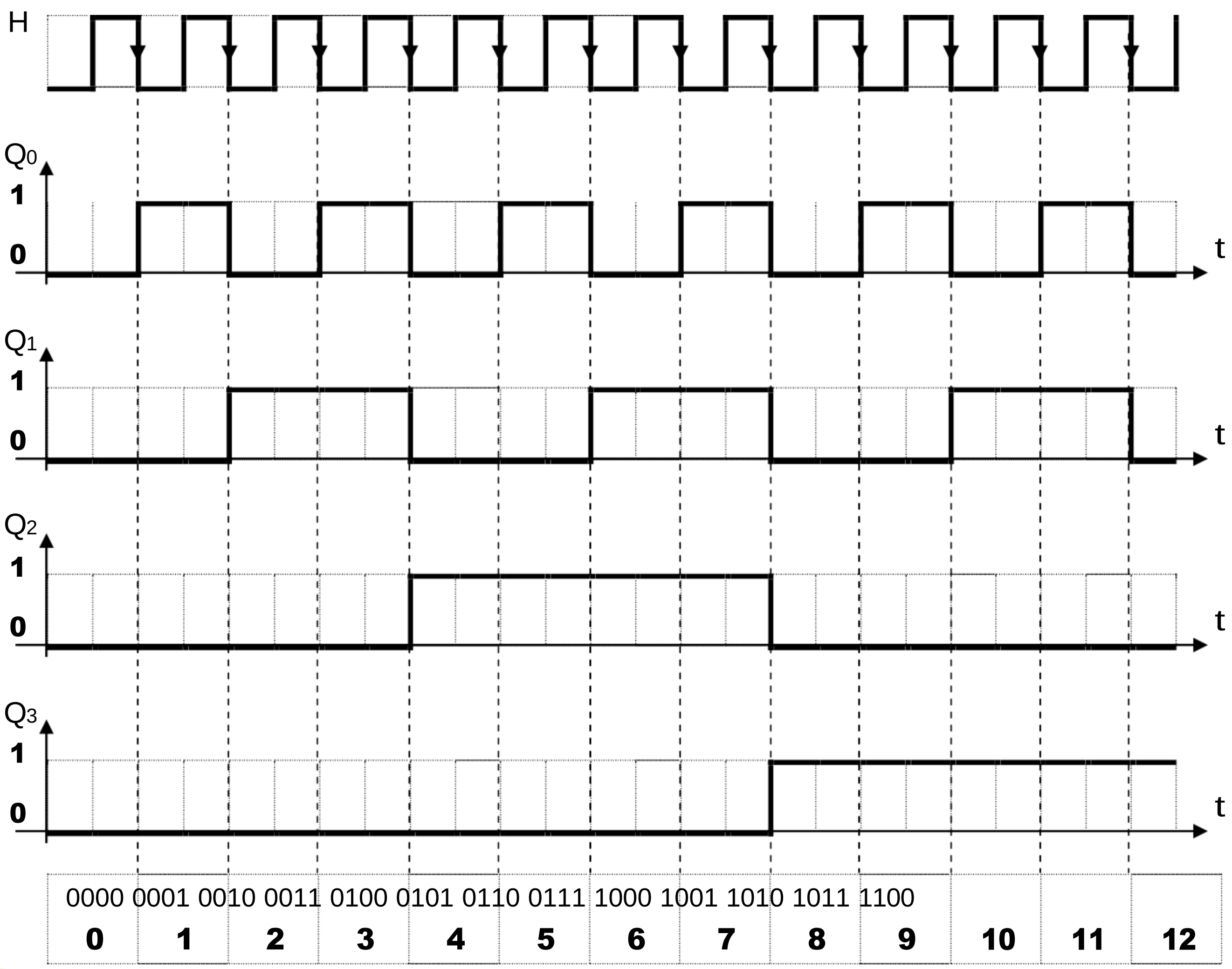
3. ASYNCHRONOUS COUNTERS AND DOWNCOUNTERS:

The term asynchronous means that the events have no temporal relationship to each other. The flip-flops forming an asynchronous counter do not change state at the same time, because they are not connected to the same clock signal, the periodic triggering only on the first flip-flop of the counter. The triggeringt of the following flip-flops is done step by step so that the output Qnor Qnwill be applied to the H clockn+1depending on whether we are working on a rising or falling edge and whether we want to obtain an up or down counter.

This type of meter is generally simple to make and has the disadvantage of generating operating hazards (propagation delay).

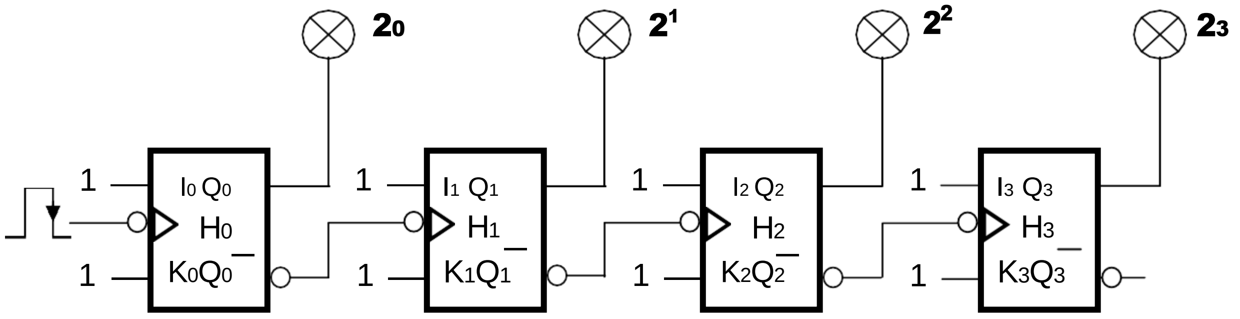
3.1 Asynchronous counters

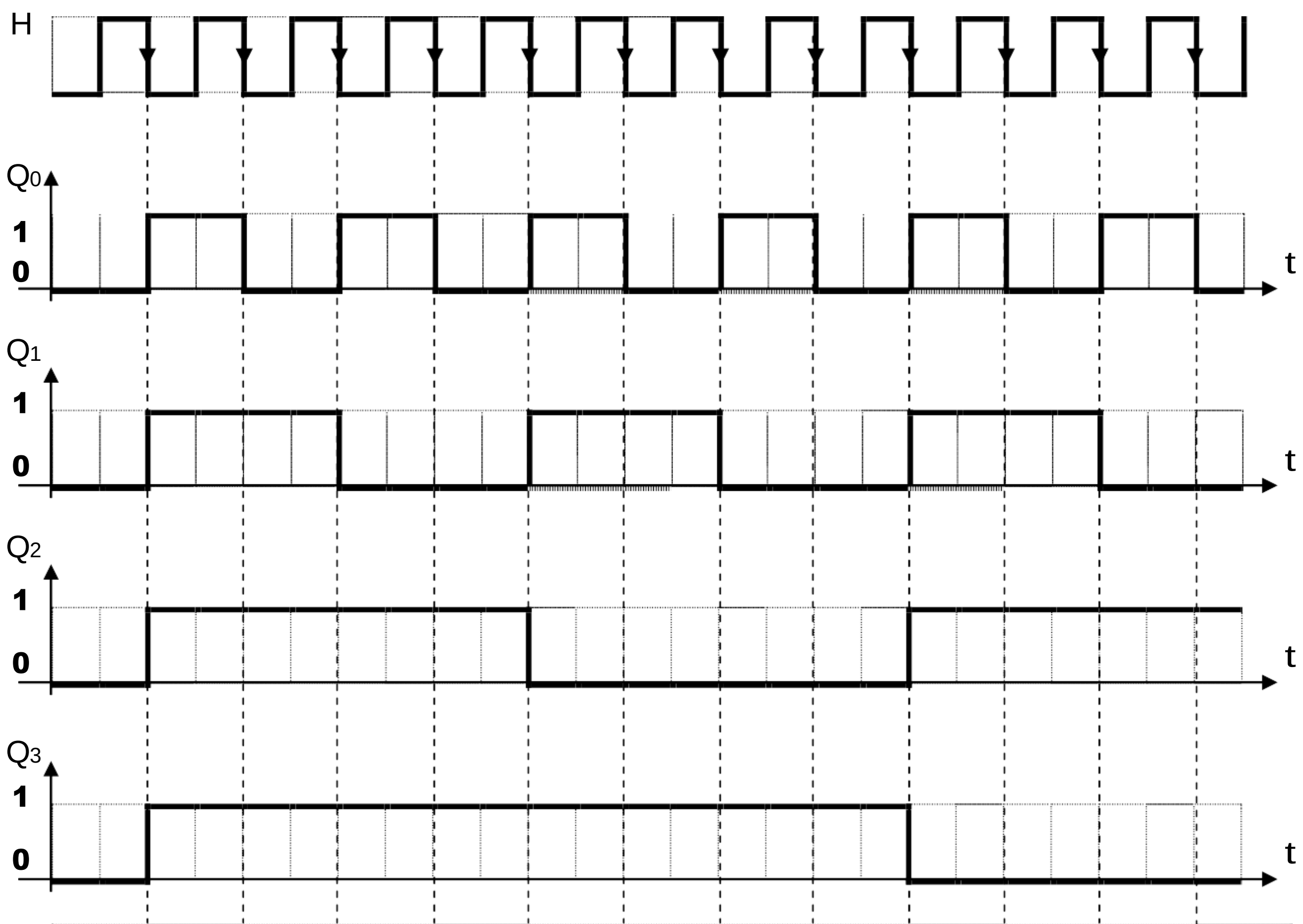




- We therefore obtain a Counter asynchronous modulo 16 .
- The same counter can be made using stockingsc ules synchronized on rising edge whose clock Hiwill be connected to output Qi-1.

3.2 Asynchronous downcounters





0000	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100
0	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**

- We therefore obtain a <u>Down counter</u> asynchronous <u>modulo 16</u> .

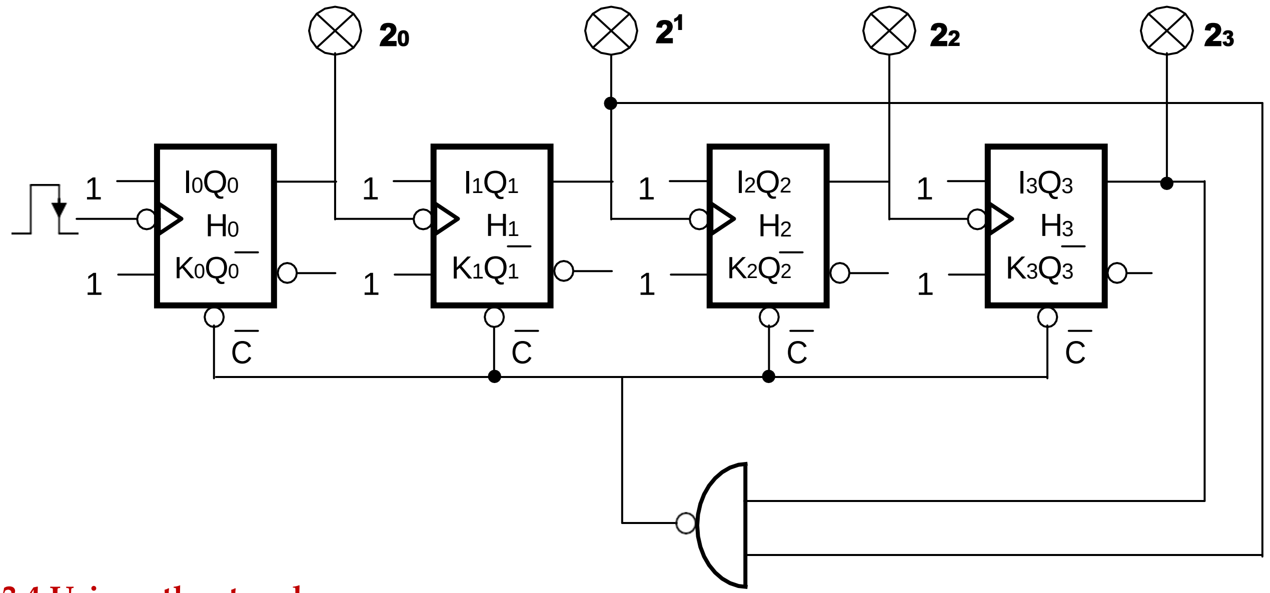
- The same counter can be made using stockingsc ules synchronized on rising edge whose clock Hiwill be connected to output Qi-1.

3.3 Truncated sequence:

The modulo is the number of distinct states occupied by a counter before it is recycled to the initial state. The maximum number of possible states, or maximum modulo, of a counter is equal to2n, where n represents the number of flip-flops in the counter.

We can construct counters to obtain a sequence whose number of states is less than2n. The sequence is then called a truncated sequence.
To obtain a truncated sequence, it is necessary to force the recycling of the counter before the latter has occupied all the states. It is necessary to have flip- flops equipped with reset predisposition inputs.0 RA0(also known RESET).

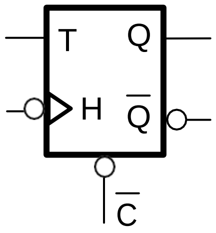
Example of a modulo 10 counter (decade counter)



3.4 Using other toggles

Other types of flip-flops can be used to make up/down counters. asynchronous:

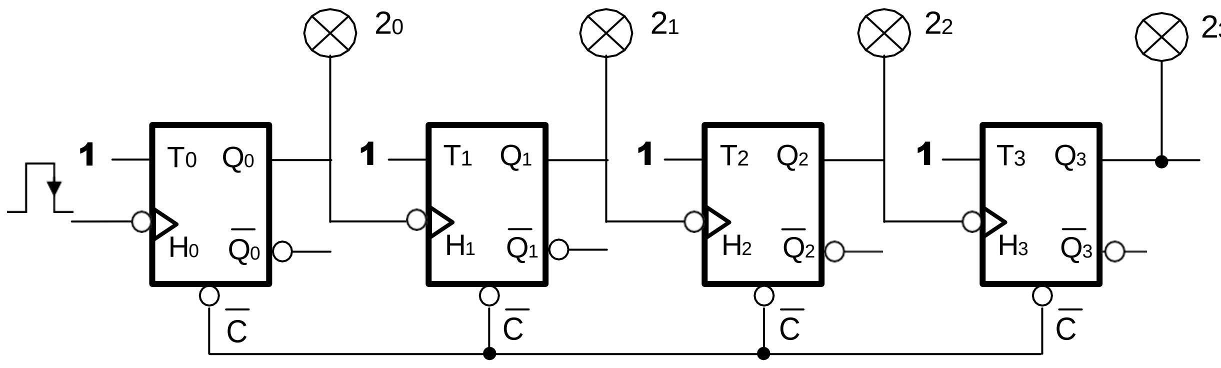
3.4.1T-toggle:



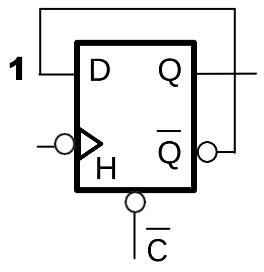
This type of flip-flops change states at each clock pulse, if the input T=1, so we can build asynchronous up/down counters based on T flip-flops using the assembly below.

Q_n \ T	0	1
0	0	1
1	1	0

Q_{n+1}



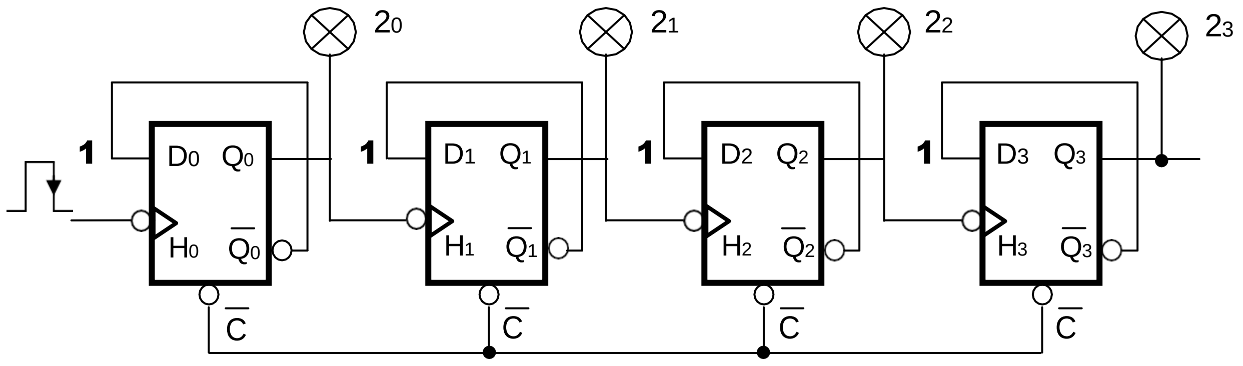
3.4.2 Flip-flop D:



This type of flip-flops change state at each clock pulse. The trigger is performed if D=1 and the trigger is performed if the D input=0, so if we connect D to Q, we obtain a change of state at each clock pulse. We can build asynchronous up/down counters based on D flip-flops using the circuit below:

Q_n \ D	0	1
0	0	1
1	0	1

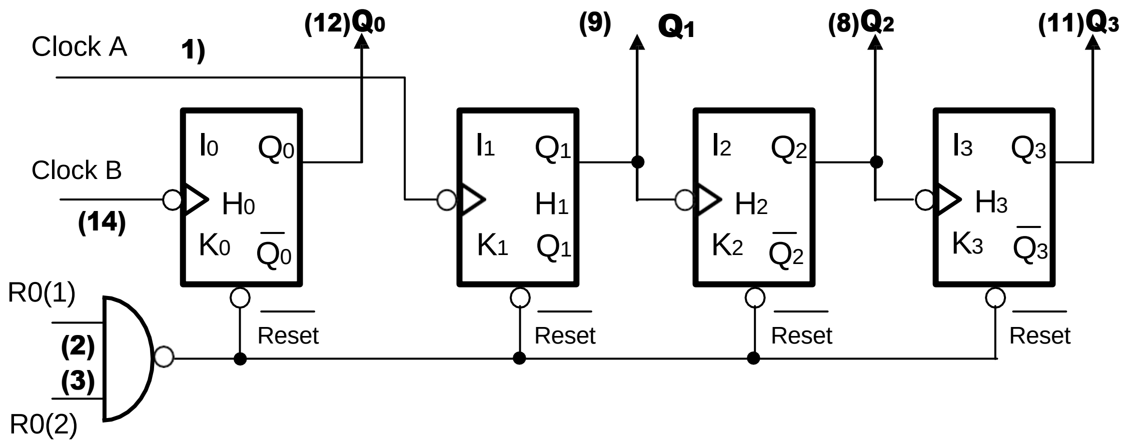
Q_{n+1}



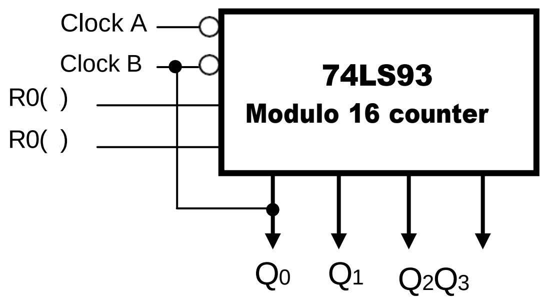
3.5 Integrated counter 7493:

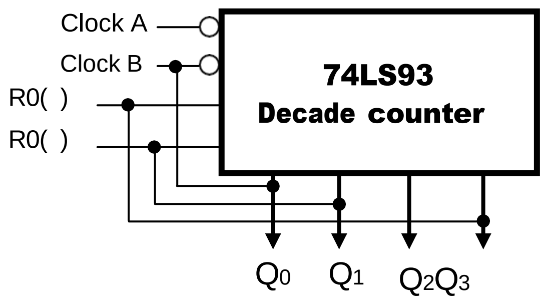
The 74LS93 integrated circuit is an example of an asynchronous counter. It consists of a flip-flop and a 3-bit asynchronous counter. It has reset inputs connected to a NAND gate, designated R0(1) and R0(2). When these two inputs are HIGH, the counter is initialized to 0000.

3.5.1 Logic diagram:



3.5.2 Examples of use of the 74LS93 counter:





3.6 Propagation delay:

Asynchronous counters are often called propagation counters because the effect of the clock pulse is initially felt only by the first flip-flop. This effect cannot reach the next flip-flop immediately because of the propagation delay of the first flip-flop. This effect is cumulative so that a clock pulse propagates through the counter for some time before reaching the last flip-flop, due to propagation delay.

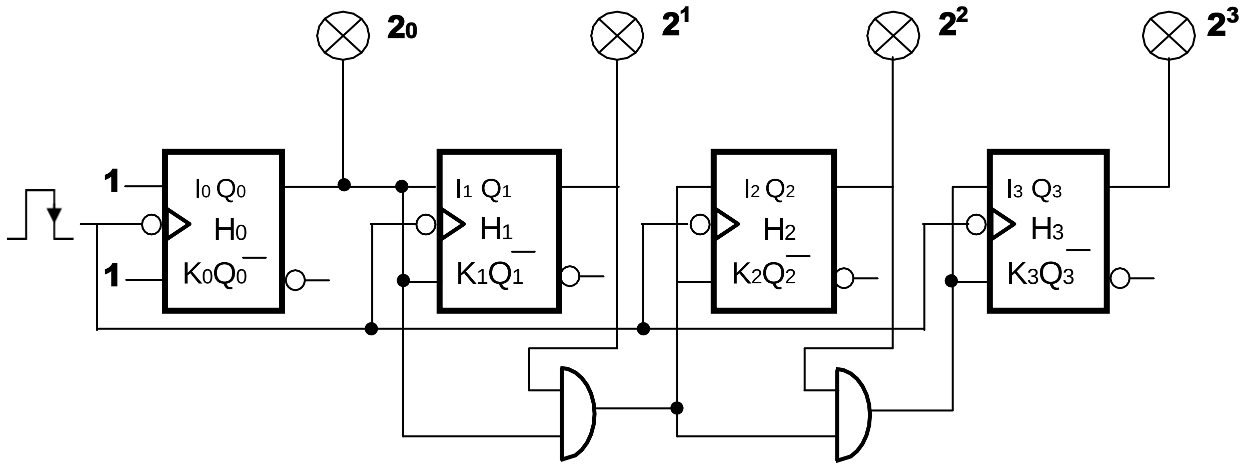
The propagation delay associated with asynchronous counters is one of the major disadvantages for this type of counters because it limits the frequency of use. The propagation delay for a flip-flop is of the order of 5 ns, which is why frequencies lower than 200 MHz must be used.

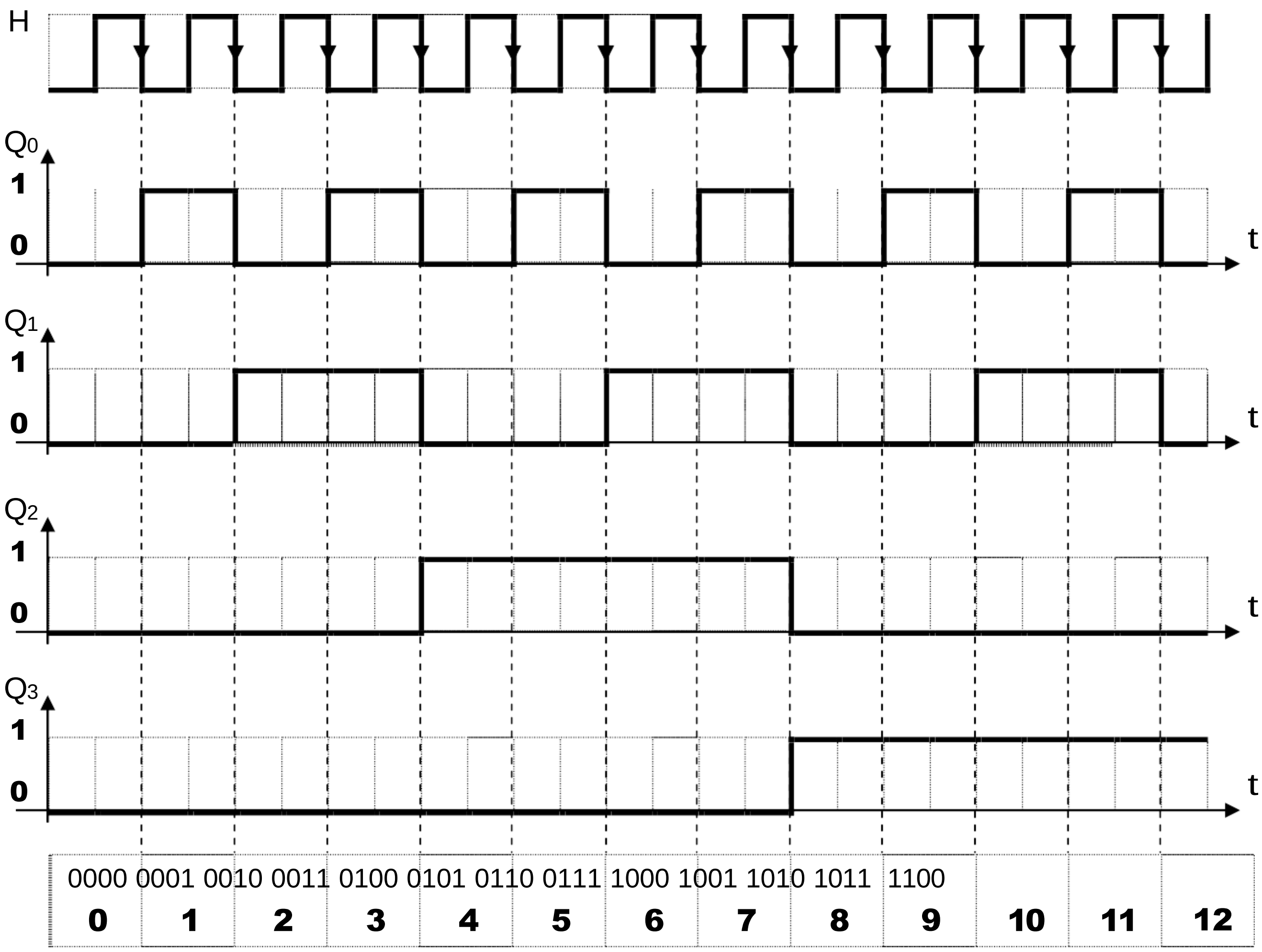
4. SYNCHRONOUS COUNTERS AND DOWN COUNTERS:

The term synchronous refers to events that have a fixed temporal relationship to each other. In terms of counter operation, the word synchronous means that all flip-flops in the counter are synchronized to the same clock signal. This solves the propagation delay problem.

The flip-flops are associated with each other, in such a way that for the flip-flop of rank i we apply all the outputs of the flip-flops which precede it to the inputs J and K.

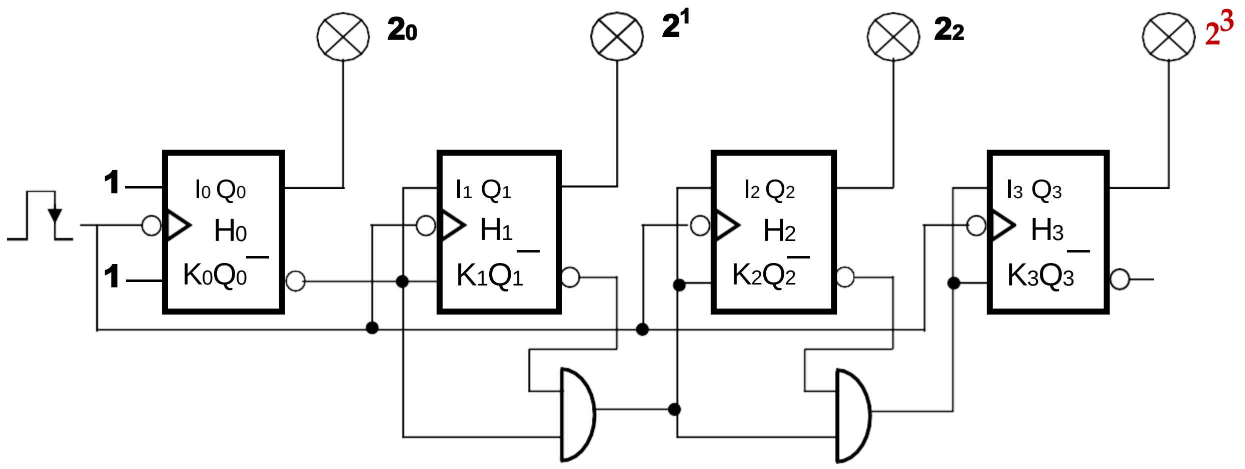
4.1 Synchronous counters

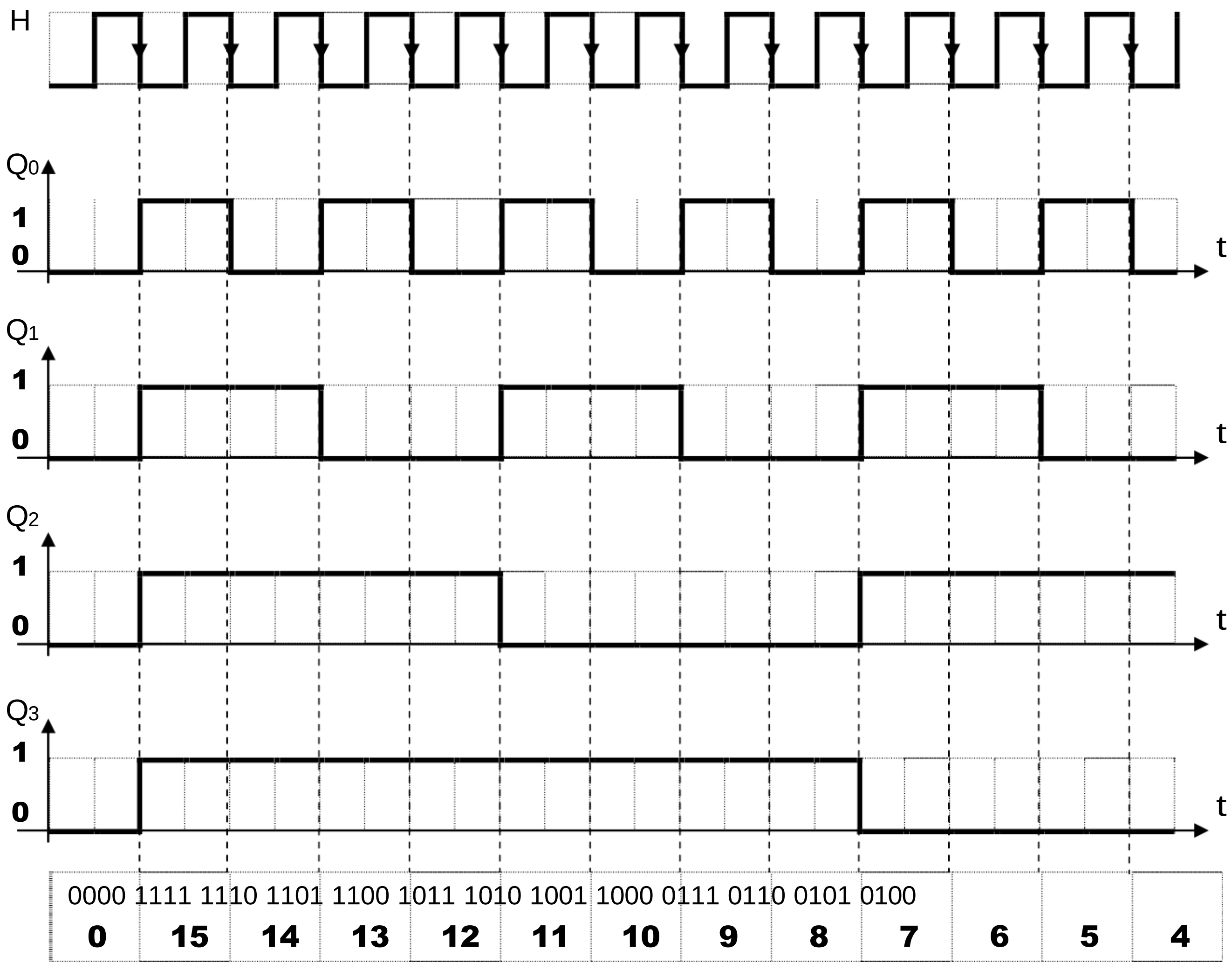




- We therefore obtain a **Counter** synchronous **modulo 16** .
- We can achieve the same compt eur using rising edge synchronized flip-flops and Q outputs$_i$ instead of Q_i.

4.2 Synchronous downcounters





- We therefore obtain a Down counter synchronous modulo 16 .
- The same down counter can be achieved using rising edge synchronized flip- flops and Q out puts$_i$instead of Q$_i$.

CHAPTER 8
SYNTHESIS OF SYNCHRONOUS COUNTERS

1. OBJECTIVES.

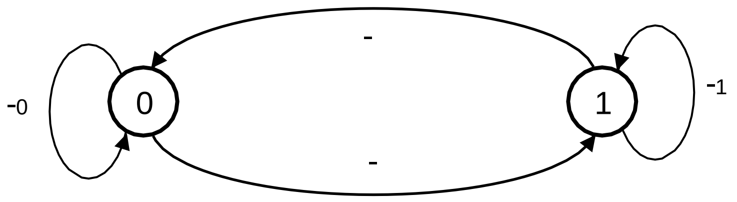
- Understanding the synthesis of synchronous counters.
- Understanding the synthesis of synchronous downcounters.

2.INTRODUCTION

At each clock pulse, the clock undergoes a transition. There are four possible transitions that can be respected by a transition table or by a state graph.

Transition	Exits		Description	Rating
	Qn	Qn+1		
0	0	0	Maintain at 0	-0
1	0	1	Engagement	-
2	1	0	Triggering	-
3	1	1	Maintain at 1	-1

Transition table



State graph

The table below gives a summary of the transitions for the different switches:

Transition	Rating	JK rocker		RS rocker		D-Flip	T-toggle
		I	K	S	R	D	T
0	-0	0	-	0	-	0	0
1	-	1	-	1	0	1	1
2	-	-	1	0	1	0	1
3	-1	-	0	-	0	1	0

From the table above we can conclude that if we want to use:

- JK flip-flops

  ✓ We regroup necessarily the interlocks (-) And optionally the triggers (-) and keeps them at 1 (-1) for the equations of theI.

  ✓ e regroup necessarily the triggers (-) And optionally the interlocks (-) and keeps them at 0 (-0) for the equations of theK.

- RS flip-flops

  ✓ We regroup necessarily the interlocks (-) And optionally keeps them at 1 (-1) for the equations of the S.

  ✓ We regroup necessarily the triggers (-) And optionally keeps them at 0 (-0) for the equations of the R.

- D flip-flops

  ✓ We regroup necessarily the interlocks (-) and keeps them at 1 (-1) for the equations of the D.

- T-swings

  ✓ We regroup necessarily the interlocks (-) and the triggers (-) for the equations of the T.

EXAMPLES

Example 1: modulo 12 counter

We want to make a modulo 12 counter using JK, RS and T flip-flops

Solution

To design this counter, you need to determine the number of flip-flops and then the equations for each input.

With 3 rockers we can achieve23=8combinations and with 4 switches we can achieve24=16combinations and a modulo 12 counter therefore requires 4 rockers since the number 2nwhich is first greater than or equal to 12 is 16.

Truth table

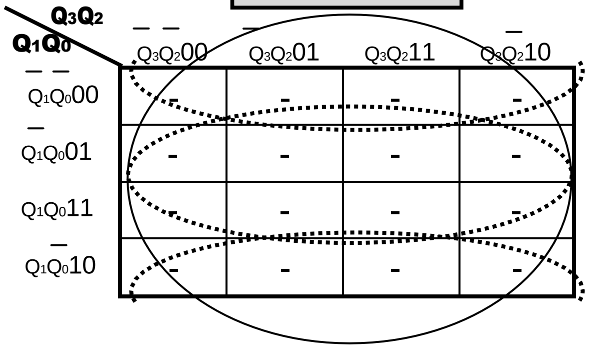
Transition	Previous state				Next state			
	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	0	0	0	0
12	0	0	0	0	0	0	0	1

It can also be presented by the KARNAUGH table below:

Sequences

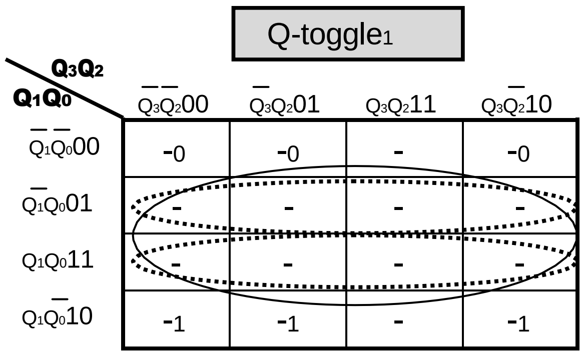
Q_1Q_0 \ Q_3Q_2	$\overline{Q_3Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	0001	0101	-	1001
$\overline{Q_1}Q_0$01	0010	0110	-	1010
$Q_1Q_0$11	0100	1000	-	0000
$Q_1\overline{Q_0}$10	0011	0111	-	1011

Q-toggle$_0$



Q_1Q_0 \ Q_3Q_2	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-	-	-	-
$\overline{Q_1}Q_0$01	-	-	-	-
$Q_1Q_0$11	-	-	-	-
$Q_1\overline{Q_0}$10	-	-	-	-

JK rocker	$I_0=K_0=1$
RS rocker	$R_0=Q_0;\ S_0=\overline{Q_0}$
T-toggle	$T_0=1$

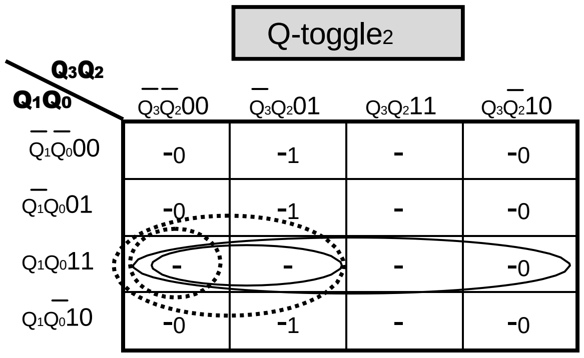


Q_1Q_0 \ Q_3Q_2	$\bar{Q}_3\bar{Q}_2$00	$\bar{Q}_3Q_2$01	$Q_3Q_2$11	$Q_3\bar{Q}_2$10
$\bar{Q}_1\bar{Q}_0$00	-0	-0	-	-0
$\bar{Q}_1Q_0$01	-	-	-	-
$Q_1Q_0$11	-	-	-	-
$Q_1\bar{Q}_0$10	-1	-1	-	-1

JK Toggle:$I_1=K_1=Q_0$

RS rocker:$R_1=Q_1\bar{Q}_0$; $S_1=Q_1Q_0$

T-toggle:$T_1=Q_0$



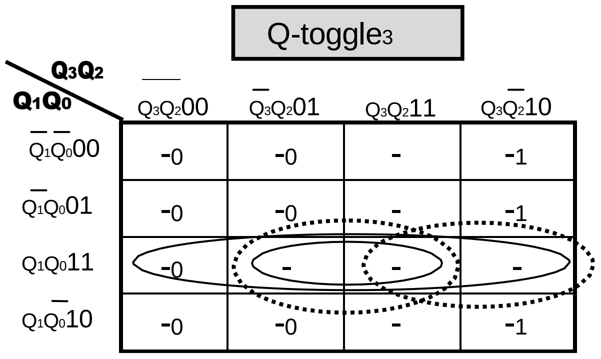
Q_1Q_0 \ Q_3Q_2	$\bar{Q}_3\bar{Q}_2$00	$\bar{Q}_3Q_2$01	$Q_3Q_2$11	$Q_3\bar{Q}_2$10
$\bar{Q}_1\bar{Q}_0$00	-0	-1	-	-0
$\bar{Q}_1Q_0$01	-0	-1	-	-0
$Q_1Q_0$11	-	-	-	-0
$Q_1\bar{Q}_0$10	-0	-1	-	-0

JK Toggle:$I_2= Q_3\bar{Q}_1Q_0$;$K_2=Q_1Q_0$

RS rocker:$R_2=Q_3Q_2\bar{Q}_1\bar{Q}_0$;

$S_2=Q_1Q_0$

T-toggle:$T_2= Q_3\bar{Q}_1Q_0$



Q_1Q_0 \ Q_3Q_2	$\overline{Q_3Q_2}$00	$\bar{Q}_3Q_2$01	$Q_3Q_2$11	$Q_3\bar{Q}_2$10
$\bar{Q}_1\bar{Q}_0$00	-0	-0	-	-1
$\bar{Q}_1Q_0$01	-0	-0	-	-1
$Q_1Q_0$11	-0	-	-	-
$Q_1\bar{Q}_0$10	-0	-0	-	-1

JK Toggle:$I_3= Q_2Q_1Q_0$

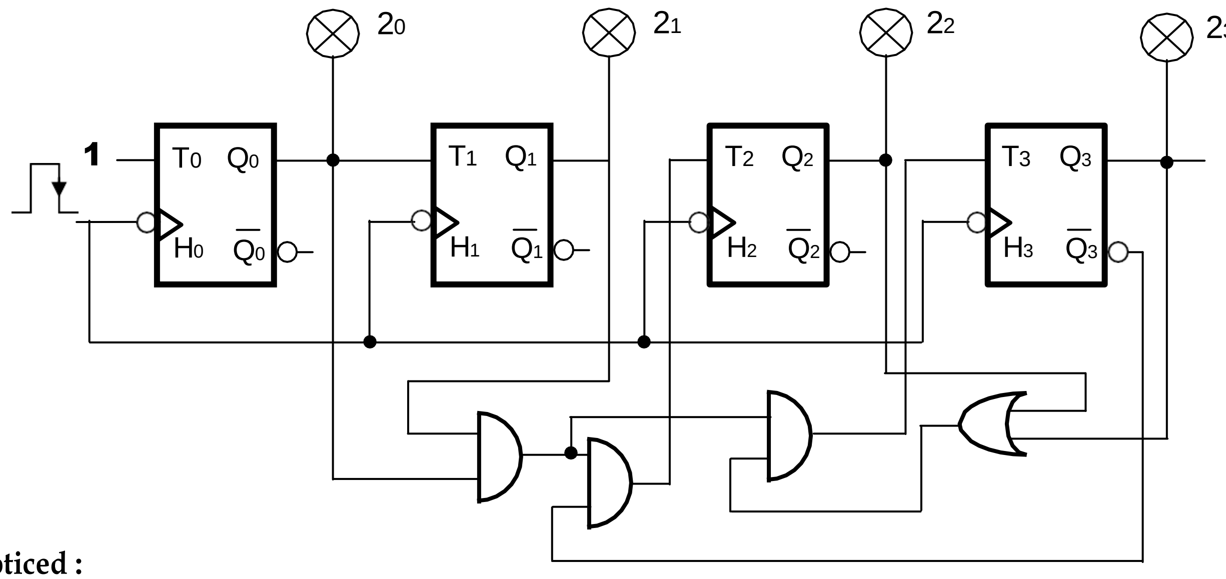
$K_3=Q_1Q_0$

RS rocker:$R_3=Q_2Q_1Q_0$

$S_3=Q_3Q_1Q_0$

T-toggle:$T_3= Q_1Q_0(Q_3+Q_2)$

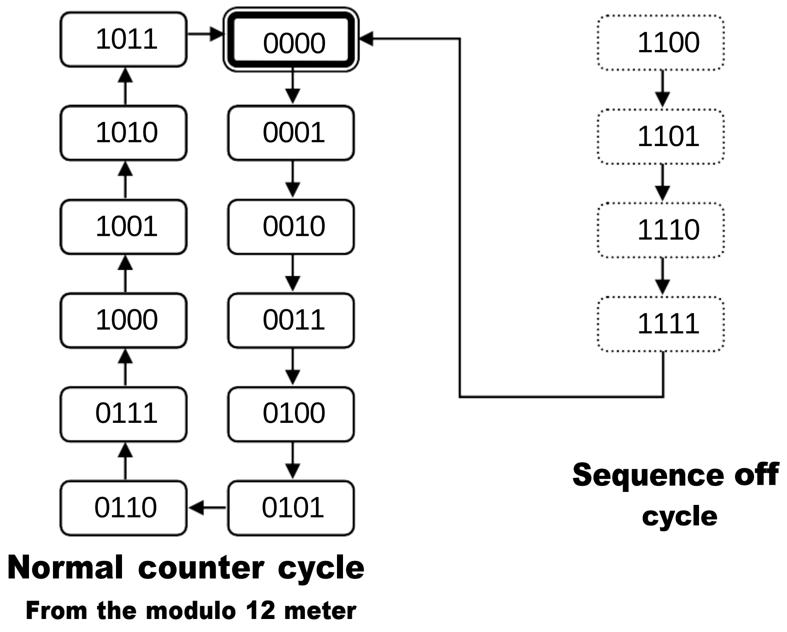
Implementation using T flip-flops



Noticed :

After the synthesis of the synchronous counter, it is necessary to check whether this counter is self-correcting or not, that is to say that if by any accident we find ourselves in a combination of outputs which is out of cycle, it is necessary to check that this counter can return to the cycle after a few pulses.

For example for the previous counter:



Example 2: modulo 16 down-counter

We want to create a modulo 16 up/down counter using JK flip-flops. The operating mode is changed using a control input A (if A=0: up/down mode; if A=1: down/down mode)

Solution

To design this counter, 4 flip-flops are needed, which can be made24=16 combinations. We will use Channon's expansion theorem to use only 4 variables

Truth table of counting (A=0)

Transition	Previous state				Next state			
	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	1	1	0	0
12	1	1	0	0	1	1	0	1
13	1	1	0	1	1	1	1	0
14	1	1	1	0	1	1	1	1
15	1	1	1	1	0	0	0	0

It can also be presented by the KARNAUGH table below

Sequences

Q_3Q_2 / Q_1Q_0	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	0001	0101	1101	1001
$\overline{Q_1}Q_0$01	0010	0110	1110	1010
$Q_1Q_0$11	0100	1000	0000	1100
$Q_1\overline{Q_0}$10	0011	0111	1111	1011

Q-toggle0

Q_3Q_2 / Q_1Q_0	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-	-	-	-
$\overline{Q_1}Q_0$01	-	-	-	-
$Q_1Q_0$11	-	-	-	-
$Q_1\overline{Q_0}$10	-	-	-	-

Flip 0:$I_0=K_0=1$

Q-toggle1

Q_3Q_2 / Q_1Q_0	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-0	-0	-0	-0
$\overline{Q_1}Q_0$01	-	-	-	-
$Q_1Q_0$11	-	-	-	-
$Q_1\overline{Q_0}$10	-1	-1	-1	-1

Flip 1:$I_1=K_1=Q_0$

Q-toggle2

Q_3Q_2 / Q_1Q_0	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-0	-1	-1	-0
$\overline{Q_1}Q_0$01	-0	-1	-1	-0
$Q_1Q_0$11	-	-	-	-
$Q_1\overline{Q_0}$10	-0	-1	-1	-0

Flip 2:$I_2=Q_1Q_0$;

$K_2=Q_1Q_0$

Q-toggle$_3$

Q_3Q_2 \ Q_1Q_0	$\bar{Q}_3\bar{Q}_2$ 00	$\bar{Q}_3Q_2$ 01	Q_3Q_2 11	$Q_3\bar{Q}_2$ 10
$\bar{Q}_1\bar{Q}_0$ 00	-0	-0	-1	-1
$\bar{Q}_1Q_0$ 01	-0	-0	-1	-1
Q_1Q_0 11	-0	-	-	-1
$Q_1\bar{Q}_0$ 10	-0	-0	-1	-1

Flip 3:I3= Q2Q1Q0

$K_3 = Q_2Q_1Q_0$

Truth table of the countdown (A=1)

Transition	Previous state				Next state			
	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	1	1	1	1
1	0	0	0	1	1	1	1	0
2	0	0	1	0	1	1	0	1
3	0	0	1	1	1	1	0	0
4	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	0	1	1	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	0	1
11	1	0	1	1	0	1	0	0
12	1	1	0	0	0	0	1	1
13	1	1	0	1	0	0	1	0
14	1	1	1	0	0	0	0	1
15	1	1	1	1	0	0	0	0

It can also be presented by the KARNAUGH table below

Sequences

Q_1Q_0 \ Q_3Q_2	$\overline{Q_3Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	1111	0011	1011	0111
$\overline{Q_1}Q_0$01	0000	0100	1100	1000
$Q_1Q_0$11	0010	0110	1110	1010
$Q_1\overline{Q_0}$10	0001	0101	1101	1001

Q-toggle$_0$

Q_1Q_0 \ Q_3Q_2	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-	-	-	-
$\overline{Q_1}Q_0$01	-	-	-	-
$Q_1Q_0$11	-	-	-	-
$Q_1\overline{Q_0}$10	-	-	-	-

Flip 0: $I_0=K_0=1$

Q-toggle$_1$

Q_1Q_0 \ Q_3Q_2	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-	-	-	-
$\overline{Q_1}Q_0$01	-0	-0	-0	-0
$Q_1Q_0$11	-1	-1	-1	-1
$Q_1\overline{Q_0}$10	-	-	-	-

Flip 1: $I_1=K_1=Q_0$

Q-toggle$_2$

Q_1Q_0 \ Q_3Q_2	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-	-	-	-
$\overline{Q_1}Q_0$01	-0	-1	-1	-0
$Q_1Q_0$11	-0	-1	-1	-0
$Q_1\overline{Q_0}$10	-0	-1	-1	-0

Flip 2: $I_2=\overline{Q_1}\,\overline{Q_0}$;

$K_2=\overline{Q_1}\,\overline{Q_0}$

Q-toggle$_3$

$Q_1Q_0 \backslash Q_3Q_2$	$\overline{Q_3}\,\overline{Q_2}$00	$\overline{Q_3}Q_2$01	$Q_3Q_2$11	$Q_3\overline{Q_2}$10
$\overline{Q_1}\,\overline{Q_0}$00	-	-0	-1	-
$\overline{Q_1}Q_0$01	-0	-0	-1	-1
$Q_1Q_0$11	-0	-0	-1	-1
$Q_1\overline{Q_0}$10	-0	-0	-1	-1

Flip 3: $I_3=\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}$

$K_3=\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}$

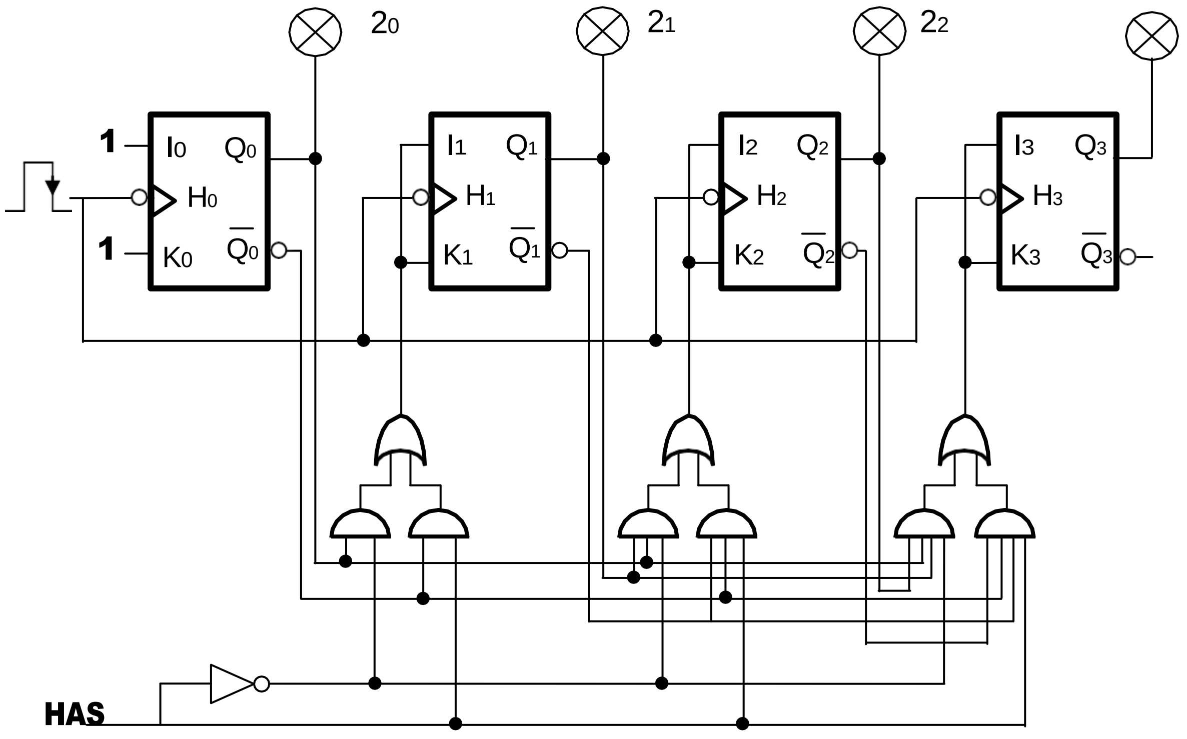
Final equations

Flip 0: $I_0=K_0=A.1+A.\overline{1}=1$

Flip 1: $I_1=K_1=AQ_0+\overline{A}\,\overline{Q_0}$

Flip 2: $I_2=K_2=AQ_1Q_0+\overline{A}\,\overline{Q_1}\,\overline{Q_0}$

Flip 3: $I_2=K_2=AQ_2Q_1Q_0+\overline{A}\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}$



Bibliographic References

[1] Digital Circuits Theory and Applications, Ronald J. Tocci , Reynald Goulet Inc, Reynald Goulet Inc, ISBN; 2-89377-108-4

[2] Combinatorial Logic and Technology, Marcel Gindre, Denis Roux, BELIN , 1984, ISBN: 2-7011-0857-8

[3] Digital Systems, Jaccob Millman, Arvin Grabel, 1989, ISBN: 2-7042-1182-5

[4] Digital Electronics, Rached Tourki, University Publishing Center, 2005, ISBN: 9973-37-019-8

[5] Logical Systems course support, Mohamed Habib BOUJMIL,2005, ISBN:

[6] Educational Support for Logical Systems, Fedia DOUIRI , 2012

[7] Digital Electronics Courses and Problems, Jean-Claude Laffont, Jean-Paul Vabre , Marketing Edition , ISBN :1986, 2-7298-8650-8