الجمهورية الجزائرية الديمقراطية الشعبية
People's **D**emocratic **R**epublic of **A**lgeria
وزارة التعليم العالي و البحث العلمي
Ministry of **H**igher **E**ducation and **S**cientific **R**esearch
جامعة الدكتور يحي فارس بالمدية
Dr. **Y**ahia **F**arès **U**niversity of **M**édéa
كلية العلوم
Faculty of **S**ciences



COURSE HANDOUT

Machine Structure 01

Dr. HASSANI DJAMEL

MACHINE STRUCTURE 01

SUMMARY

CHAPTER 01:
NUMBER SYSTEMS AND INFORMATION REPRESENTATION

	Introduction	**2**
1.2.	Definitions	**2**
1.3.	Decimal system	**2**
1.4.	Binary system	**3**
1.5.	Octal system	**4**
1.6.	Hexadecimal system	**5**
1.7.	Conversion from one number system to another	**5**
1.7.1.	Base b to base 10	**5**
1.7.2.	Base 10 to base b	**6**
1.7.3.	Base 2 to base 2n	**7**
1.7.4.	Base 2n to base 2	**7**
1.7.4.	Base i to base j	**7**
1.8.	BCD number (binary coded decimal)	**8**
1.9.	Binary arithmetics	**9**
1.9.1.	The four operations	**9**
1.9.1.1.	Addition	**9**
1.9.1.2.	Subtraction	**10**
1.9.1.3.	Multiplication	**10**
1.9.1.4.	Division	**10**
1.9.2.	Negative numbers	**11**
1.9.2.1.	Representation with absolute value and sign	**11**
1.9.2.2.	Representation by ones' complement	**11**
1.9.2.3.	Representation by two's complement	**11**
1.9.3.	Coding in two's complement	**12**
1.9.4.	Number length issues	**12**
1.9.4.1.	Addition of two positive numbers	**12**
1.9.4.2.	Addition of two negative numbers	**13**
1.9.4.3.	Addition of a positive number and a negative number	**13**
1.10.	Floating-point numbers	**14**
1.10.1.	The ieee 754 standard	**14**
1.11.	Character encoding	**16**
1.11.1.	ASCII encoding	**16**
1.11.2.	Extended ascii encoding	**17**
1.11.3.	Unicode	**18**

CHAPTER 02
BOOLEAN ALGEBRA (BINARY ALGEBRA)

	Introduction	**20**
2.1.	Binary variables and functions	**20**
2.2.	Logic states	**21**
2.3.	Basic logic functions	**21**
2.3.1.	The logical or gate	**21**
2.3.2.	The logical and gate	**22**
2.3.3.	Logical inversion	**23**
2.3.4.	The nor logic gate	**24**
2.3.5.	The nand logic gate	**24**
2.3.6.	The exor (or exclusive) logic gate	**25**
2.3.7.	The nexor logic gate	**25**
2.4.	Simplification of logical equations	**26**
2.4.1.	Fundamental relationships :	**27**
2.4.2.	De Morgan's theorem:	**28**
2.5.	Notions of min-terms and max-terms	**29**
2.6.	Products of sums and sums of products	**29**
2.7.	Representation by contact chain	**30**
2.8.	Logic expression using nand and nor gates	**31**
2.9.	Graphic simplification "karnaugh table"	**32**
2.9.1.	Adjacent terms and the karnaugh diagram:	**32**
2.9.2.	Incomplete function	**33**
2.9.3.	Simplification by karnaugh table:	**34**
2.9.3.1.	Simplification rules.	**34**
2.9.3.2.	Representation of groups	**34**
2.10.	Quine-McCluskey method	**36**

CHAPTER 01

NUMBER SYSTEMS AND INFORMATION REPRESENTATION

CHAPTER 01

NUMBER SYSTEMS AND INFORMATION REPRESENTATION

1.1. INTRODUCTION

Over the centuries, the way of representing numbers has adapted to the use we want to make of them and to the needs generated by new discoveries. Thus, the shepherds could content themselves with keeping a cord to which they added a knot at the birth of a sheep; this counting practice suited them, since the only need was to "count" the sheep in order to ensure that none were lost. This practice was also at the origin of the Chinese abacus and the rosary.

Over the course of the evolution of needs, the modes of representation and manipulation of numbers, that is to say the numbering systems, have evolved. The decimal system using ten symbols is now in use and suits the needs of modern science. However, the discovery of electricity and its development created new needs, including that of the binary number system and two other systems related to it: the octal system and the hexadecimal system.

1.2. DEFINITIONS

1. 1. The number system is a system for representing numbers using symbols called digits.
2. 2. The basis of a number system is the number of symbols (or digits) used to represent the numbers in the number system.
3. 3. In the representation of a base number B, each digit has a place value expressed by the successive powers of B. These powers being negative to the right of the decimal point and positive to the left of the decimal point. A base number B will be written in the form:

$$(a_n\ a_{n-1}\ \ldots\ a_2\ a_1\ a_0\ a_{-1}\ a_{-2}\ \ldots\ a_{-n})_B$$

where the a_i are the digits used in base B. The value of the index i represents the position (or the rank) of the digit in the number.

1.3. DECIMAL SYSTEM

Our number system is a base 10 positional system. This means that we use 10 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and the value of each of these symbols depends on the position he occupies. Thus in the number 534, each symbol or digit has a value which depends on its position, that is to say that:

$$534 = 500 + 30 + 4$$

or:

$$\mathbf{534 = 5.10^2 + 3.10^1 + 4.10^0}$$

This expression is called the polynomial representation of the number 534.

In general, any number comes down to the sum of the products of each digit by the weight of its rank in the number:

$$\mathbf{N_{10} = a_{n-1}.10^{n-1} + a_{n-2}.10^{n-2} + \ldots + a_0.10^0}$$

$$\mathbf{N_{10} = a_{n-1}\,a_{n-2} \ldots a_0}$$

with:

$$\mathbf{0 \leq a_i \leq 9}$$

If we have the comma:

$$\mathbf{N_{10} = a_{n-1}.10^{n-1} + a_{n-2}.10^{n-2} + \ldots + a_0.10^0 + a_{-1}.10^{-1} + \ldots + a_{-m}.10^{-m}}$$

we call :

n - 1 : the heaviest weight;
- m : the lowest weight.

1.4. BINARY SYSTEM

The decimal system is difficult to adapt to digital mechanisms, it is very difficult to design electronic equipment that works with ten different voltage levels, on the other hand it is very easy to imagine electronic systems that work with two voltage levels. This is the reason why most number systems use the binary system (base 2). In the binary system there are only two possible symbols 0 and 1 which can represent any quantity expressed in the decimal system or in other systems.

$$\mathbf{N_2 = a_{n-1}.2^{n-1} + a_{n-2}.2^{n-2} + \ldots + a_0.2^0}$$

with:

$$\mathbf{0 \leq a_i \leq 1}$$

Physically, in electronic circuits, a "0" bit is represented by a low voltage and a "1" bit, by a high voltage.

How, with only two values, to encode all numbers? The reasoning was as follows. Decimal zero is coded binary zero; there is no difference. Similarly, a decimal is encoded as a binary. But since we have exhausted the binary digits, we must invent a technique for coding the two decimals: it simply consists of adding one more digit. Thus the 2 decimal will be written 10 in binary. Once we have exhausted all the two-digit combinations,

we go to a three-digit number, then to four, etc. The following table summarizes the first combinations, with a maximum of 4 digits:

DECIMAL	*BINARY*
0	**0**
1	**1**
2	**1 0**
3	**1 1**
4	**1 0 0**
5	**1 0 1**
6	**1 1 0**
7	**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**

Table 1.1

1.5. OCTAL SYSTEM

A mode of numeration which had its glory days is the octal, based on the base 8. In octal we use 8 digits which go from 0 to 7. So, we will code in octal on 0, 1, 2, 3, 4, 5, 6 and 7 the decimal values also ranging from 0 to 7. But we are not allowed to go beyond 7; consequently, the 8 decimal will be coded 10 in octal, the 9 decimal will be written 11 in octal, and so on according to the following table:

DECIMAL	*OCTAL*
0	**0**
1	**1**
2	**2**
3	**3**
4	**4**
5	**5**
6	**6**
7	**7**
8	**10**
9	**11**
10	**12**
11	**13**
12	**14**
13	**15**
14	**16**
15	**17**
16	**20**
17	**21**
etc.	**etc.**

Table 1.2

The octal system is also based on the notion of place value:

$$N_8 = a_{n-1} . 8^{n-1} + a_{n-2} . 8^{n-2} + \ldots + a_0 . 8^0$$

with:

$$0 \leq a_i \leq 7$$

1.6. HEXADECIMAL SYSTEM

With hexadecimal, the counting base is 16 and we use 16 fundamental values ranging from 0 to 15. This is even the origin of a writing problem: if we want to encode these 16 base values using a single digit, we will have to invent additional writing symbols for the numbers from 10 to 15. We code these last 6 values using the first 6 letters of the alphabet:

DECIMAL	*HEXADECIMAL*
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
etc.	etc.

Table 1.3

The hexadecimal system is also based on the notion of place value:

$$\mathbf{N_{16} = a_{n-1} . 16^{n-1} + a_{n-2} . 16^{n-2} + \ldots + a_0 . 16^0}$$

with:

$$\mathbf{0} \leq a_i \leq \mathbf{F}$$

1.7. CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER

1.7.1. Base B to Base 10

Two examples will suffice to explain the conversion method:

For B = 2

$$\begin{aligned}(100101)_2 &= 1.2^5 + 0.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0 \\ &= 32 + 4 + 1 \\ &= (37)_{10}\end{aligned}$$

For B = 16

$$\begin{aligned}(A3D)_{16} &= A.16^2 + 3.16^1 + D.16^0 \\ &= A.256 + 48 + D \\ &= 2560 + 48 + 13 \\ &= (2621)_{10}\end{aligned}$$

1.7.2. Base 10 to Base B

The method consists of dividing by the base B as many times as necessary to obtain a zero quotient, we write the remainders in the reverse order in which they were obtained. For example:

For B = 2

Successive divisions	stay
43 : 2 = 21	+ 1
21 : 2 = 10	+ 1
10 : 2 = 5	+ 0
05 : 2 = 2	+ 1
02 : 2 = 1	+ 0
01 : 2 = 0	+ 1

Resulting binary word **1 0 1 0 1 1**

For B = 9

Successive divisions	stay
363 : 9 = 40	+ 3
040 : 9 = 4	+ 4
004 : 9 = 0	+ 4

Resulting base 9 word **4 4 3**

The previous method is suitable for the integer part of a decimal number. When there will be a fractional part we convert it by successive multiplications by the base B as many times as necessary to obtain the desired precision or for the conversion to fall just (zero result). For example:

$(0,6875)_{10} = (\ ?\ )_2$

0,6875 x 2 = 1 + 0,3750
0,3750 x 2 = 0 + 0,7500
0,7500 x 2 = 1 + 0,5000
0,5000 x 2 = 1 + 0,0000

So the conversion result is:

$(0,6875)_{10} = (0,1011)_2$

Second example:

$(0,3)_{10} = (\ ?\ )_2$

0,3 x 2 = 0 + 0,6
0,6 x 2 = 1 + 0,2
0,2 x 2 = 0 + 0,4
0,4 x 2 = 0 + 0,8
0,8 x 2 = 1 + 0,6
0,6 x 2 = 1 + 0,2
. . . . . .

So the conversion result is:

$(0,3)_{10} = (\ 0,010011\ \ldots)_2$

$(0,3)_{10} \cong (\ 0,01001)_2$

1.7.3. Base 2 to Base 2^n

The conversion from binary to a 2^n base can be done directly without having to divide or multiply. We can associate with each of the 2^n symbols of the base 2^n a word - code, or symbol of n digits, which gives the value in binary of the number of the base 2^n. Such a decoding method has the advantage of writing the numbers in a more condensed form than in binary. This conversion mode makes it possible to pass directly from binary to base 2^n and vice versa.

To do this, we take the digits in increments of n starting from the decimal point, going to the left for the integer part and to the right for the fractional part. For example:

$\underbrace{0011}_{3}\ \underbrace{1010}_{A}\ \underbrace{1001}_{9}$ so $(1110101001)_2 = (3A9)_{16}$

$\underbrace{010}_{2}\ \underbrace{110}_{6}\ \underbrace{100}_{4}$ so $(10110100)_2 = (264)_8$

1.7.4. Base 2^n to Base 2

Conversely, we replace each symbol of the base 2n by n binary digits. For example:

$(\ 7F\ )_{16} = (\ ?\ )_2$

$\underbrace{7}_{0111}\ \underbrace{F}_{1111}$ so $(\ 7F\ )_{16} = (\ 1111111\ )_2$

1.7.4. Base i to Base j

- If i and j are both powers of 2, we use base 2 as a relay base : $\boldsymbol{i \rightarrow 2 \rightarrow j}$

Example:

$(7F)_{16} = (?)_8$
$(7F)_{16} = (1111111)_2$
$(1111111)_2 = (177)_8$

from where:

$(7F)_{16} = (177)_8$

- If i and j are not both powers of 2, we use base 10 as a relay base : $i \rightarrow 10 \rightarrow j$

Example:

$(16B)_{16} = (?)_9$

$(16B)_{16} = (363)_{10}$
$(363)_{10} = (443)_9$

from where:

$(16B)_{16} = (443)_9$

1.8. BCD NUMBER (Binary Coded Decimal)

Conversion to natural binary is a long and tedious operation. To overcome this drawback, we use the BCD code which has the advantages of decimal and binary. There are several BCD codes: 8421, 6311, excess-3, 2 out-of 5 and code gray.

In the BCD code, we associate a four-bit code word with each of the symbols of the decimal system.

For example, the decimal number 937.25 will be written in BCD (8421):

$$\overbrace{1001}^{9}\ \overbrace{0011}^{3}\ \overbrace{0111}^{7},\overbrace{0010}^{2}\ \overbrace{0101}^{5}$$

Décimal	*BCD8421*	*BCD6311*	*Excess-3*	*2 out-of 5*	*Code Gray*
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	0111
6	0110	1000	1001	10001	0101
7	0111	1001	1010	10010	0100
8	1000	1011	1011	10100	1100
9	1001	1100	1100	11000	1101

Table 1.4

- The 8421 (6311): indicates the weight of the positions for the 4 bits of the BCD.
- Excess-3 code: is obtained by adding 3 (0011) to code 8421.
- The 2 out-of 5 code: has the property that exactly 2 out of 5 bits have a value equal to "1". This code is very useful for verification since if there will be an error in the logic circuit, the number of "1"s will no longer be 2 for each combination.
- Gray code (reflected binary): has the property that when passing from one combination to another, only one bit changes.

1.9. BINARY ARITHMETICS

When a computer performs an arithmetic operation, two constraints are involved:

- circuits work on numbers that always have the same length (format); for example, in an 8-bit machine, the number 10111 must be completed with zeros 00010111;
- the circuits handle only two numbers at a time, to calculate S=X+Y+Z, they perform X + Y = T then T + Z = S.

1.9.1. The four operations

The same calculation rules apply in all number systems; binary arithmetic resembles that of decimal numbers.

1.9.1.1. Addition

Truth table

a	*b*	*S*	*R*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S : the sum
R : the restraint

Example:

	1 1 1 . . . . . Restraint
$11)_{10}$	$1\ 0\ 1\ 1)_2$
$7)_{10}$	$0\ 1\ 1\ 1)_2$
.....	.....
$18)_{10}$	$1\ 0\ 0\ 1\ 0)_2$

$10010)_2 = 1 . 2^4 + 0 . 2^3 + 0 . 2^2 + 1 . 2^1 + 0 . 2^0 = 18)_{10}$

1.9.1.2. Subtraction

Truth table

a	*b*	*D*	*E*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

D : Difference
E : Footprint

Example:

1. . . . . Footprint
1 1 1 0 1)$_2$
1 0 0 1 1)$_2$
.....
0 1 0 1 0)$_2$

1.9.1.3. Multiplication

Truth table

a	*b*	*c*
0	0	0
0	1	0
1	0	0
1	1	1

c = a . b

Example:

1 0 1 1)$_2$
0 1 0 1)$_2$
.....
1 0 1 1
1 0 1 1 . .
.....
1 1 0 1 1 1)$_2$

1.9.1.4. Division

Example:

110111 | 101
101 | 1011
.....
00111
101

or:

21 - 7 = 14
14 - 7 = 7 three operations the rest "0"
7 - 7 = 0 so the result is: 3.

1.9.2. Negative numbers

In order to use an adder as a subtractor, a proper representation of negative numbers is needed.

1.9.2.1. Representation with absolute value and sign

We adopt the convention: "0" for the "+" sign and "1" for the "-" sign. For example:

$+ 35)_{10} = 0100011)_2$
$- 35)_{10} = 1100011)_2$

For an n-bit word, the first bit is the monkey and the remaining (n - 1) bits represent the word value.

1.9.2.2. Representation by ones' complement

A negative number (- N) is represented by its ones' complement as follows:

$$\overline{N} = (2^n - 1) - N$$

Example:

If n = 4 then (-N) is represented by $\overline{N} = 15 - N$.

Note that 1111 represents "0" in ones' complement, and (-8) has no representation in the 4-bit system.

How to find ones' complement ?

Example:

N = 010101 $\qquad\qquad \overline{N} = ?$

$2^n - 1 = 1\ 1\ 1\ 1\ 1\ 1$
$N = 0\ 1\ 0\ 1\ 0\ 1$
.....
$\overline{N} = (2^n - 1) - N = 1\ 0\ 1\ 0\ 1\ 0$

We obtain the ones' complement of a number N by replacing its "0" by "1" and its "1" by "0".

1.9.2.3. Representation by two's complement

A negative number (- N) is represented by its 2's complement as follows:

$$N^* = 2^n - N$$

Example:

If n = 4 then (− N) is represented by $N^* = 16 - N$.
- 3 is represented by 16 - 3 = 13 = 1101

All negative numbers in two's complement have a "1" in the first bit to the left.

How to find two's complement?

Example:

N =	0	1	0	1	1	0	0		or :				
$\overline{N}$ =	1	0	1	0	0	1	1			N	=	0	1	0	1	1	0	0
+							1			N* =	1	0	1	0	1	0	0	
N* =	1	0	1	0	1	0	0											

Noticed:

1. N is also the two's complement of N* :

$$N = 2^n - N^*$$

2. N is also the ones' complement of N:

$$N = \left(2^n - 1\right) - \overline{N}$$

1.9.3. Coding in two's complement

In order to fully understand the principle of coding in two's complement, useful for distinguishing positive and negative numbers, we take the following example, even if it is absurd. We suppose that we take the road with a car whose counter indicates 00000 km.

- If you go forward, after 1 km, the counter indicates 00001 km then 00002 km etc. . . .
- Suppose that from 00000, we start in reverse and that the counter counts down: it will display the greatest number possible, ie 99999 after 1Km, then 99998 after 2Km etc. . .

We can then consider that 99999 means - 1 and that 99998 means - 2, etc . . . But it will be simpler once transposed into binary, our odometer is this time graduated in binary and starts from 00000000:

- If you drive forward, it will indicate 00000001 after 1Km, then 00000010 after 2Km etc. . . .
- In reverse, it marks 11111111 after 1Km, then 11111110 after 2Km, etc. . . .

1.9.4. Number length issues

1.9.4.1. Addition of two positive numbers

By adding two positive numbers (binary element of sign "0"), one can obtain a result whose element of sign is "1" (although the result is positive) if the result is out of the authorized interval.

Example:

+ 49	0 0110001
+ 33	0 0100001
.....	.....
+ 82	0 1010010
	+ 82

```
 + 49          0  0110001
 + 88          0  1011000
.....          ...........
+137           1  0001001
                    ↑
                Over flow
```

1.9.4.2. Addition of two negative numbers

By adding two negative numbers represented by their two's complement (the binary element of sign "1"), one can obtain a result whose binary element of sign is "0" (although the result is negative). In addition there is always a carry because the binary element of weight (n+1) is always at "1".

Example:

```
 - 32          1  1100000   2's complement
 - 31          1  1100001   2's complement
.....          ...........
 - 63        1  1  1000001   2's complement
                      -  63
          Restraint
```

N* = 11000001 ⇒ N = 00111111 = - 63

```
 -  32         1  1100000
 - 127         1  0000001
.....          ...........
 - 159       1  0  1100001
                   +   97  ⟶  Over flow
          Restraint
```

1.9.4.3. Addition of a positive number and a negative number

A - B = A + B*

With:

B* : 2's complement of B

Subtraction is reduced to an addition in which we neglect the carry which appears at the weight (n + 1): the presence of this term in a comparison concludes with A ≥ B if there is no carry then A < B.

Example:

with : n = 5 bit

```
+ 9          0  1001
- 5          1  1011        2's complement
.....      .....
+ 4        1  0  0100
                 + 4
        to rejected
```

```
+ 7          0  0111
- 10         1  0110        2's complement
.....      .....
-  3        1  1101
```

N* = 11101 ⇒ N = 00011 = - 3

1.10. FLOATING-POINT NUMBERS

Floating-point numbers, more commonly referred to as floating-point numbers, are generally non-integer numbers whose digits are written only after the decimal point, and to which an exponent is added to specify how many positions the decimal point must be moved. For example, in decimal notation, we write the number 31,41592 as :

$$\mathbf{0{,}3141592 \cdot 10^2}$$

So there are several ways of representing the exponent and mantissa.

Example: base-2 coding, floating-point format of (3,25)

$$\begin{aligned} 3{,}25_{(10)} &= 11{,}01_{(2)} \quad \text{(Fixed point)} \\ &= 1{,}101 \cdot 2^{1}_{(2)} \\ &= 110{,}1 \cdot 2^{-1}_{(2)} \end{aligned}$$

In computing, a standard has been established for the representation of floating-point numbers. This is the **IEEE 754 standard**.

1.10.1. The IEEE 754 standard

In the IEEE 754 standard, a floating-point number is always represented by a triplet : (s, e, m). This triplet is assembled in the order sign, exponent, mantissa to form a number :

$$N = s \cdot m \cdot B^e$$

or else

$$N = (-1)^s \cdot m \cdot B^e$$

So, for base B = 10, the two preceding numbers are represented by the triplets :

(0, 2, 3141592) **= 0,3141592 x 10^2**

and

(1, - 1, 1732) **= - 0,1732 x 10^{-1}**

The IEEE 754 standard then defines the coding, with base B = 2, of a number in single-precision on 32 bits and in double-precision on 64 bits. A standardized format:

- **Single-precision format: 32 bits**

  Sign bit (1 bit)

  Exponent (8 bits)

  Mantissa (23 bits)

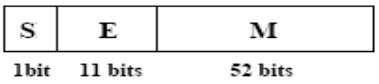


- **Double precision format: 64 bits**

  Sign bit (1 bit)

  Exponent (11 bits)

  Mantissa (52 bits)



- ***s***: is the sign coded on 1 bit (MSB is the most significant bit): 0 for a positive number, 1 for a negative number;
- ***e***: designates the exponent, which is an integer coded on n_e = 8 bits for single precision and on n_e = 11 bits for double precision. The exponent can be positive or negative. n_e represents the number of bits of the exponent:

$$e_b = e + 2^{(ne-1)} - 1$$

- ***m***: designates the mantissa, which is a decimal point number coded on 23 bits in single precision, 52 in double precision. In the binary representation of m, the bit to the left of the decimal point is necessarily a 1 (m = 1.....). It is therefore unnecessary to represent it. Only the digits after the decimal point in the mantissa are coded in binary.

Example:

1. Represent 525,5 in IEEE 754 single precision format.

Write 525.5 in binary: $(525,5)_{10} = (1000001101,1)_2$

Find the exponent: $(525,5)_{10} = (1,0000011011)2 \,.\, 2^9$

Identify the triplet (s, e, m):

- $s = 0$
- $e_b = e + \underbrace{2^{n_e-1} - 1}_{biais} = 9 + 2^{8-1} - 1 = (136)_{(10)} = (10001000)_{(2)}$
- $m = (00000110110000000000000)_{(2)}$

So:

$$(525,5)_{(10)} = (0\ 10001000\ 00000110110000000000000)_{(2)}$$

2. Decimal conversion - IEEE 754 single precision :

$35{,}5_{(10)} = ?_{(IEEE\ 754\ SP)}$

$S = 0$

$35{,}5_{(10)} = 100011{,}1_{(2)}$

$= 1{,}000111 \cdot 2^{5}_{(2)}$

$E = E_b - 127 = 5$

$E_b = 132$

$1{,}M = 1{,}000111 \qquad M = 00011100...$

S M

$0\underbrace{10000100}_{E_b}\overbrace{00011100000000000000000}_{(IEEE\ 754\ SP)}$

3. IEEE 754 single precision - Decimal conversion :

S M

$0\underbrace{10000001}_{E_b}\overbrace{11100000000000000000000}_{(IEEE\ 754\ SP)}$

$S = 0$, so positive number

$E_b = 129$, so exponent $= E_b - 127 = 2$

$1{,}M = 1{,}111$

$+ 1{,}111 \cdot 2^{2}_{(2)} = 111{,}1_{(2)} = 7{,}5_{(10)}$

1.11. CHARACTER ENCODING

1.11.1. ASCII encoding

The computer's memory stores all data in digital form. There is no method of storing characters directly. Each character therefore has its equivalent in binary code: this is the ASCII code (American Standard Code for Information Interchange). (standardized in 1963) This coding consists of associating a binary numerical value (which can be interpreted as hexadecimal, decimal, etc.) with each character used in computer data exchange: alphabetic and numeric (alphanumeric) characters, punctuation, various control codes.

The basic ASCII code uses one byte to encode 128 characters, representing the characters on 7 bits (128 possible characters) and leaving the 8th bit at 0:

- Codes 0 to 31 are not characters. They are called control characters because they allow actions such as :
  - line feed (CR)
  - Beep (BEL)

- Codes 65 to 90 represent uppercase letters
- Codes 97 to 122 represent lower-case letters

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Table 1.5 : *The basic ASCII code*

1.11.2. Extended ASCII encoding

The ASCII code was developed for the English language, so it does not contain accented or language-specific characters. To encode such characters, another code must be used. The ASCII code has therefore been extended to a single byte, using the 8th bit to define characters numbered from 128 to 255 (this is known as the extended ASCII code).

The ISO/IEC 8859 standard provides extensions for various languages. For example, ISO 8859-1, also known as Latin-1, extends ASCII with accented characters useful for Western European languages such as French and German.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	^{n}
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	^{2}
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Table 1.6 : *Extended ASCII*

1.11.3. Unicode

Unicode encodes characters as sequences of 1 to 4 code elements of one byte each. Among other things, the Unicode standard defines a character set. Each character is identified within this set by an integer index, also known as a code point. For example, the character € (euro) is the 8365th character in the Unicode repertoire, so its index, or code point, is 8364.

The Unicode directory can contain over a million characters, which is far too many to be encoded by a single byte. The Unicode standard therefore defines standardized methods for encoding and storing this index as a sequence of bytes: UTF-8 is one of these, along with UTF-16, UTF-32 and their various variants.

The main feature of UTF-8 is that it is backward-compatible with the ASCII standard, i.e. every ASCII character is encoded in UTF-8 as a single byte, identical to the ASCII code. For example, A (capital A) has ASCII code 65 and is encoded in UTF-8 by byte 65. Each character with a code point greater than 127 (non-ASCII character) is coded on 2 to 4 bytes. The € (euro) character, for example, is encoded on 3 bytes: 226, 130 and 172 (**1110001010000010 10101100**).

Type	Caractère	Point de code (Hex)	Valeur scalaire		Codage UTF-8	
			Base 10	binaire	binaire	Base 16
Contrôles	[NUL]	U+0000	0	*0000000*	**0***0000000*	00
	[US]	U+001F	0	*0*011111	**0***0*011111	1F
Texte	[SP]	U+0020	32	*0*100000	**0***0*100000	20
	A	U+0041	65	1000001	**0**1000001	41
	~	U+007E	126	1111110	**0**1111110	7E
Contrôles	[DEL]	U+007F	127	1111111	**0**1111111	7F
	[PAD]	U+0080	128	*00010* 000000	**110***00010* **10**000000	C2 80
	[APC]	U+009F	159	*00010* 011111	**110***00010* **10**011111	C2 9F
Texte	[NBSP]	U+00A0	160	*00010* 100000	**110***00010* **10**100000	C2 A0
	é	U+00E9	233	*00010* 101001	**110***00010* **10**101001	C3 A9
	??	U+07FF	2047	11111 111111	**110**11111 **10**111111	DF BF
	?!	U+0800	2048	*0000* 100000 000000	**1110***0000* **10**100000 **10**000000	E0 A0 80
	€	U+20AC	8364	*0010* 000010 101100	**1110***0010* **10**000010 **10**101100	E2 82 AC
	▒	U+D7FF	55295	1101 011111 111111	**1110**1101 **10**111111 **10**111111	ED 9F BF

Table 1.7 : *Unicode*

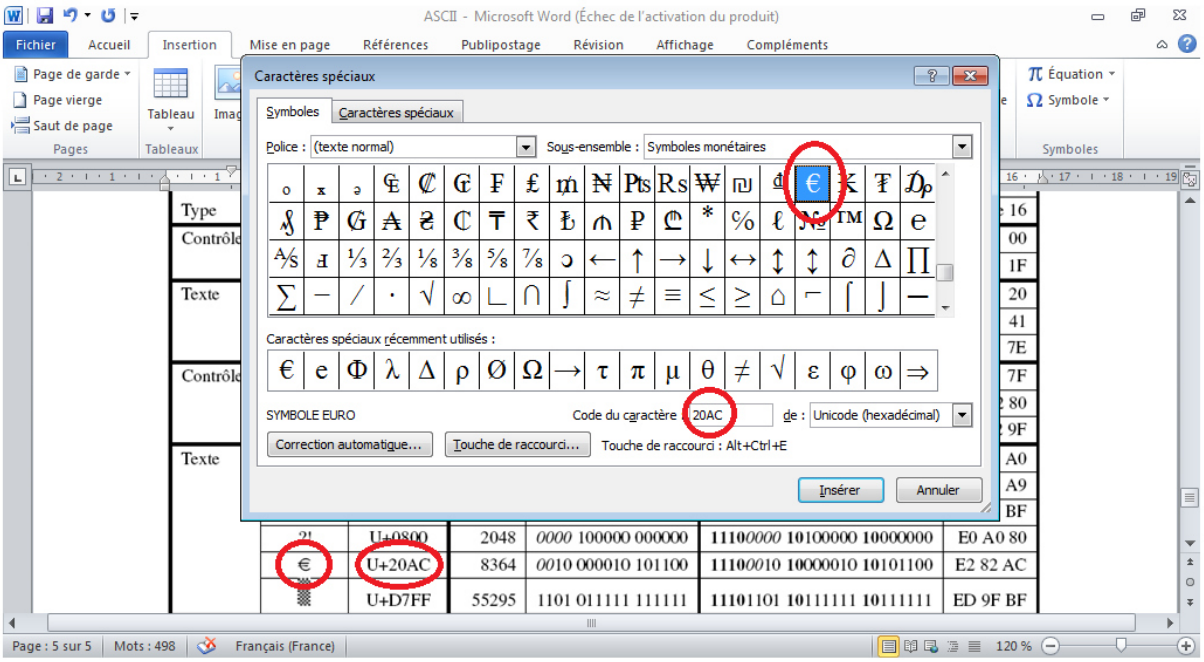


Figure 1.1: *Example of the € (euro) character*

CHAPTER 02

BOOLEAN ALGEBRA
(BINARY ALGEBRA)

CHAPTER 02

BOOLEAN ALGEBRA
(BINARY ALGEBRA)

INTRODUCTION

Boolean algebra is a mathematical tool for equating binary variables to represent the effects of logic inputs on the outputs of a digital circuit. Boolean algebra makes it possible to express the behavior of any digital function, whatever its complexity, according to a logical expression composed from the 3 fundamental operators of the algebra. The circuit is a combinatorial system in which each input vector can be associated with a unique output vector. A combinatorial function is therefore an application defining the value of a binary output as a function of input variables. An implementation of the logic representation then enables its materialization at the electronic level and the possibility of applying input voltage levels to the various components.

2.1. BINARY VARIABLES AND FUNCTIONS

- A binary variable is represented by a physical quantity that can only assume two states ("0" or "1"), between which it makes transitions.
- A binary function depends on a number of independent variables.

Boolean algebra studies the states of binary variables and functions.

Example :

X	Y	F_1	F_2	F_3	F_4	F_5
0	0	0	0	1	0	1
0	1	0	0	0	0	1
1	0	0	1	1	0	1
1	1	0	1	0	1	0

Table 2.1

Hence :

$\mathbf{F_1} = 0$
$\mathbf{F_2} = X$
$\mathbf{F_3} = \overline{Y}$
$\mathbf{F_4} = X \cdot Y$
$\mathbf{F_5} = \overline{X.Y}$

2.2. LOGIC STATES

State	*Binary*
True	1
false	0

Table 2.2: *The logic states*

From an electrical point of view, true and false states can be represented by voltage levels. Regardless of the absolute value of the voltage, we call it:

- High level, the highest voltage. This is often marked with the letter H.
- Low level, the lowest voltage. It is often marked with L

State	*Binary*	*Level*	*Symbol*	*Voltage (example)*
True	1	High	H	+ 5V
False	0	Low	L	0V

Table 2.3: *The logic states*

Note:

1. Positive logic is that for which the highest voltage level is the "1" state.
2. Conversely, negative logic is that for which the lowest voltage level is the "1" state.

2.3. BASIC LOGIC FUNCTIONS

Many digital electronic circuits provide one output from several inputs. We will only look at circuits with two inputs and one output. Since the two inputs A and B can be 0 or 1, all possible input configurations are:

A = 0 B = 0
A = 0 B = 1
A = 1 B = 0
A = 1 B = 1

Each different law of association between A and B leads to the creation of a different circuit. All these circuits are called logic gates. The logic gate is characterized by the data association law used to create the output value.

2.3.1. The logical OR gate

We often illustrate logical OR with two switches connected in parallel: either switch needs to be closed for lamp L to light up.

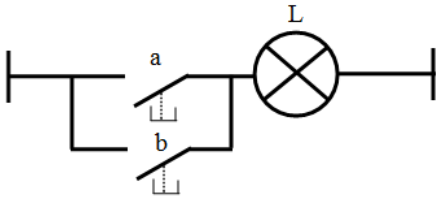


Figure 2.1: *Equivalent diagram of an OR logic gate*

So, if a or b (or both) take the value 1 (switch closed), then L is 1 (lamp L on). This is the law of association of the "OR" gate. The corresponding truth table is:

a	***b***	***L***
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.4: *OR logic gate truth table.*

The logic function is :

$$L = a + b \text{ stands for } a \text{ or } b$$

The logical symbol is :

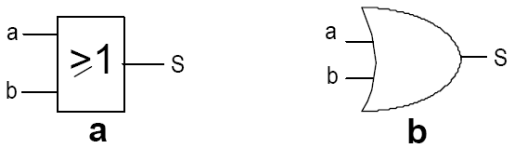


Figure 2.2 *: OR logic gate*
a- European symbol
b- American symbol.

We can multiply the number of input variables, the general rule being "with logical OR, you only need one variable to be 1 for the output to go to 1".

2.3.2. The logical AND gate

With two switches in series, we create a logical AND: both must be closed for the LED to light up.

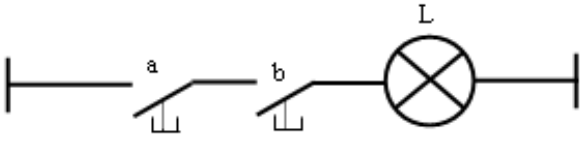


Figure 2.3 *: Equivalent diagram of an AND logic gate.*

Therefore, the result S of the combination A and B can only take the value 1 if A and B are both equal to 1. The resulting truth table is as follows:

A	*B*	*S*
0	0	0
0	1	0
1	0	0
1	1	1

***Table 2.5** : AND logic gate truth table.*

The logic function is:

$$S = A \,.\, B \quad \text{stands for} \quad A \text{ and } B$$

The logical symbol is:

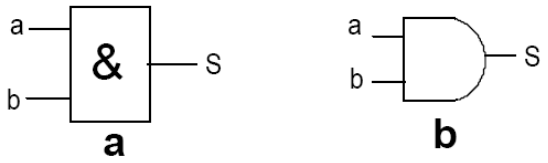


***Figure 2.4** : OR logic gate*
a- European symbol
b- American symbol.

2.3.3. Logical inversion

Logical inversion is called NOT, or complement function: in fact, it aims to make a 1 out of a 0 and vice-versa. Its state table is as follows:

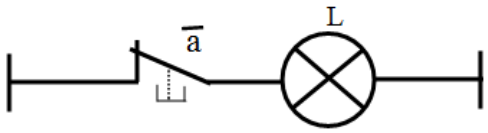


***Figure 2.5 :** Equivalent diagram of an NOT logic gate.*

Input	*Output*
0	1
1	0

***Table 2.6 :** NOT logic gate truth table.*

The logic function is:

$$\boldsymbol{Output} = \overline{\boldsymbol{Input}}$$

The inversion is marked by over-lining.

The logical symbol is a simple little circle at the end of a triangle:

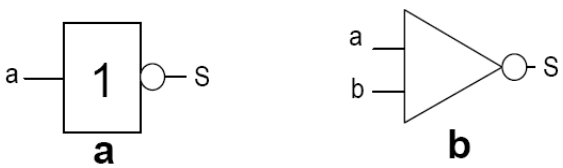


***Figure 2.6:** Logic inverter.*

2.3.4. The NOR logic gate

The result of an OR between A and B may well be inverted, giving the following truth table:

A	*B*	*S*
0	0	1
0	1	0
1	0	0
1	1	0

Table 2.7 : *NOR logic gate truth table.*

The logic function is:

$$S = \overline{A + B}$$ stands for ***A no or B***

The logical symbol is:

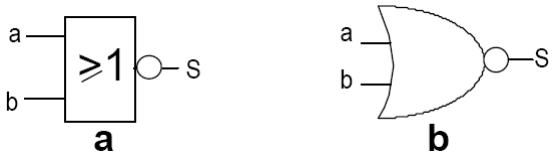


Figure 2.7 *: NOR logic gate*
a- European symbol
b- American symbol.

2.3.5. The NAND logic gate

By inverting the result of the AND operation, we obtain the following truth table:

A	*B*	*S*
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.8 : *NAND logic gate truth table.*

The logic function is:

$$S = \overline{A.B}$$ stands for ***A no and B***

The logical symbol is:

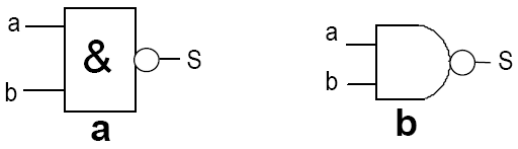


Figure 2.8 *: NAND logic gate*
a- European symbol
b- American symbol.

2.3.6. The EXOR (or exclusive) logic gate

Combined, the basic functions we've just studied can be used to set up practical circuits. The logical OR studied above is called or inclusive: "the or inclusive implies that one or both of the inputs must change to 1 in order for the output to change to 1".

Conversely, the exclusive or excludes the possibility of a simultaneous passage of the inputs to 1: "the exclusive or implies that one or other of the inputs passes to 1, but not both simultaneously, for the output to pass to 1". We can therefore establish its state table for two inputs:

A	***B***	***S***
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.9 : *EXOR logic gate truth table.*

The logic function is:

$$S = A \oplus B \text{ stands for } A \text{ EXOR } B$$

In fact, the logical equation is equivalent to the following formulas:

$$A \oplus B = A\bar{B} + \bar{A}B$$
$$A \oplus B = (A + B)\overline{AB}$$
$$A \oplus B = \overline{\bar{A}\bar{B} + AB}$$

The logical symbol is:

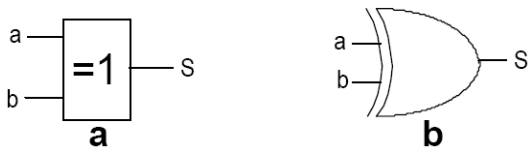


Figure 2.9 *: EXOR logic gate*

To create it, several doors need to be grouped together, and numerous combinations are possible:

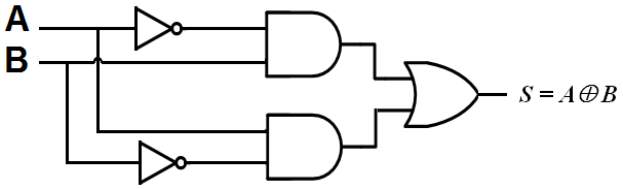


Figure 2.10 : *Exclusive OR (EXOR) with logic gates*

2.3.7. The NEXOR logic gate

By inverting the output of an EXOR gate, we obtain the relationship characterized by the following truth table:

A	*B*	*S*
0	0	1
0	1	0
1	0	0
1	1	1

Table 2.10 : *NEXOR logic gate truth table*

The logic function is:

$$S = \overline{A \oplus B}$$
$$S = \overline{A\bar{B} + \bar{A}B}$$
$$S = AB + \bar{A}\bar{B}$$

The logical symbol is:

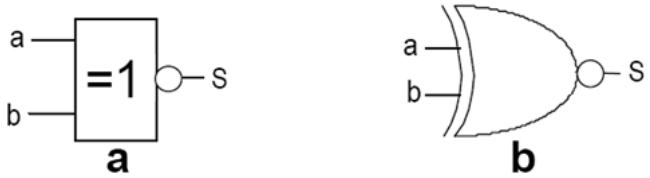


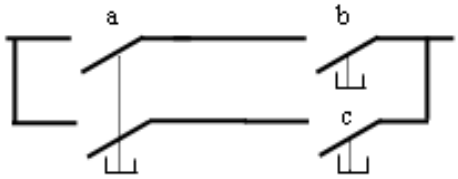
Figure 2.11 *: EXOR logic gate.*

2.4. SIMPLIFICATION OF LOGICAL EQUATIONS

Once the logical equations of a system have been established, an important task arises: simplifying them. There are a number of methods for simplifying Boolean expressions, including algebraic and graphical simplification.

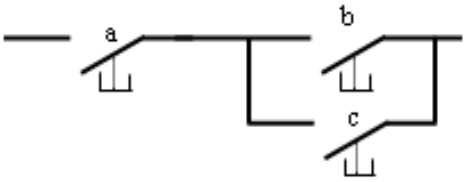
Example :

a- Before simplification:



S = (A . B) + (A . C)

b- After simplification:



S = A.(B + C)

Figure 2.12 : *Simplification of logic equations.*

2.4.1. Fundamental relationships :

**** Negation :***

$$\overline{\mathbf{0}} = \mathbf{1}$$
$$\overline{\mathbf{1}} = \mathbf{0}$$
$$\overline{\overline{A}} = A$$
$$A + \overline{A} = \mathbf{1}$$
$$A.\overline{A} = \mathbf{0}$$
$$AB + \overline{AB} = \mathbf{1}$$
$$AB.\overline{AB} = \mathbf{0}$$

**** Addition :***

$$A + A + A + \cdots + A = A$$
$$A + \mathbf{1} = \mathbf{1}$$
$$A + B + \mathbf{1} = \mathbf{1}$$
$$A + \mathbf{0} = A$$

**** Multiplication :***

$$A.A.A \ldots A = A$$
$$A.\mathbf{0} = \mathbf{0}$$
$$A.\mathbf{1} = A$$

**** Symmetry:***

Si $A = B$ alors $B = A$

**** Commutativity:***

$$AB = BA$$
$$A + B = B + A$$
$$A \oplus B = B \oplus A$$

**** Transitivity:***

Si $A = B$ et si $B = C$ alors $A = C$

**** Associativity:***

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

**** Distributivity:***

$$(AB) + (AC) = A(B + C)$$
$$(A + B)(A + C) = A + BC$$

Demonstration :

$$(A + B)(A + C) = AA + AC + BA + BC$$
$$= A + AC + BA + BC$$
$$= A(1 + C + B) + BC$$
$$= A + BC$$

**** Absorption :***

$$A(1 + B) = A$$
$$A(A + B) = A$$
$$A + (AB) = A$$

**** Theorem 1:***

$$\boldsymbol{A + (\bar{A}B) = A + B}$$
$$\boldsymbol{B + (\bar{B}A) = A + B}$$

Demonstration :

$$\begin{aligned}(A+B)(A+\bar{A}) &= AA + A\bar{A} + BA + B\bar{A}\\ &= A + BA + B\bar{A}\\ &= A(1+B) + B\bar{A}\\ &= A + B\bar{A}\end{aligned}$$

**** Theorem 2 :***

$$\boldsymbol{A(\bar{A} + B) = AB}$$
$$\boldsymbol{B(\bar{B} + A) = AB}$$

Demonstration :

$$\begin{aligned}A(\bar{A}+B) &= A\bar{A} + AB\\ &= 0 + AB\\ &= AB\end{aligned}$$

**** Theorem 3 :***

$$\boldsymbol{AB + BC + \bar{A}C = AB + \bar{A}C}$$

Demonstration :

$$\begin{aligned}AB + BC + \bar{A}C &= (AB + \bar{A}C) + BC(A + \bar{A})\\ &= (AB + \bar{A}C) + (BCA + BC\bar{A})\\ &= (AB + ABC) + (\bar{A}C + \bar{A}BC)\\ &= AB(1 + C) + \bar{A}C(1 + B)\\ &= AB + \bar{A}C\end{aligned}$$

2.4.2. DE MORGAN's Theorem:

$$\boldsymbol{\overline{A + B} = \bar{A}.\bar{B}}$$
$$\boldsymbol{\overline{AB} = \bar{A} + \bar{B}}$$

Demonstration :

$$\begin{aligned}(A+B) + \bar{A}\bar{B} &= [(A+B) + \bar{A}][(A+B) + \bar{B}]\\ &= [(A + \bar{A}) + B][(B + \bar{B}) + A]\\ &= (1+B)(1+A)\\ &= 1.1\\ &= 1\end{aligned}$$

$$\begin{aligned}(A+B)\bar{A}\bar{B} &= A\bar{A}\bar{B} + \bar{A}\bar{B}B\\ &= (A\bar{A})\bar{B} + \bar{A}(\bar{B}B)\\ &= 0.\bar{B} + 0.\bar{A}\\ &= 0\end{aligned}$$

$$\left.\begin{matrix}(A+B) + \bar{A}\bar{B} = 1\\ (A+B)\bar{A}\bar{B} = 0\end{matrix}\right\} \Rightarrow \boldsymbol{\overline{A + B} = \bar{A}.\bar{B}}$$

For the second Theorem we do the same demonstration: $\boldsymbol{\overline{AB} = \bar{A} + \bar{B}}$

In other words, this means that AB is the complement of $\overline{A} + \overline{B}$, which we can verify by assigning 0 and 1 to A and B :

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

These two columns are identical so : $\overline{AB} = \overline{A} + \overline{B}$

Table 2.11

2.5. NOTIONS OF MIN-TERMS AND MAX-TERMS

Suppose we have n Boolean variables (capable of taking the state 0 or 1):

- The Min-term is the Boolean product of these variables.
- The Max-term is the Boolean sums of these variables.

Each variable must be represented by its true or complimented form. The following table illustrates the case of three ABC variables.

Variables			*Min - term*		*Max - term*	
A	***B***	***C***	***Writing***	***Notation***	***Writing***	***Notation***
0	0	0	$\overline{A}\overline{B}\overline{C}$	m_0	$A+B+C$	M_0
0	0	1	$\overline{A}\overline{B}C$	m_1	$A+B+\overline{C}$	M_1
0	1	0	$\overline{A}B\overline{C}$	m_2	$A+\overline{B}+C$	M_2
0	1	1	$\overline{A}BC$	m_3	$A+\overline{B}+\overline{C}$	M_3
1	0	0	$A\overline{B}\overline{C}$	m_4	$\overline{A}+B+C$	M_4
1	0	1	$A\overline{B}C$	m_5	$\overline{A}+B+\overline{C}$	M_5
1	1	0	$AB\overline{C}$	m_6	$\overline{A}+\overline{B}+C$	M_6
1	1	1	ABC	m_7	$\overline{A}+\overline{B}+\overline{C}$	M_7

Table 2.12 : *Min-term and Max-term.*

2.6. PRODUCTS OF SUMS AND SUMS OF PRODUCTS

Any Boolean function can be expressed as :

- A product of Max-terms, i.e. a product of sums.
- Or a sum of Min-terms, i.e. a sum of products.

Example:

$$F = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$$

Note:

An expression is said to be in canonical Min-term (or Max-term) form when all variables (true or complimented) appear in each term of the logic function.

Example:

We take as our function the carry of the addition of (X + Y + Z) in the binary system:

X	*Y*	*Z*	*R*	$\bar{R}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Table 2.13

- Canonical form of Min-terms:

We take the terms for which the function R = 1 and connect them by OR (+):

$$R = \bar{X}YZ + X\bar{Y}Z + XY\bar{Z} + XYZ$$

$$R = m_3 + m_5 + m_6 + m_7$$

$$R = \Sigma\, m(3,5,6,7)$$

- Canonical form of Max-terme :

We take the terms for which the function R = 0 ($\bar{R} = 1$) and connect them by AND (.):

$$R = (X + Y + Z).(X + Y + \bar{Z}).(X + \bar{Y} + Z).(\bar{X} + Y + Z)$$

$$R = M_0 M_1 M_2 M_4$$

$$R = \prod M(0,1,2,4)$$

2.7. REPRESENTATION BY CONTACT CHAIN

We're going to use contactors to represent binary functions. By convention, a closed contactor represents the binary value 1, while an open contactor represents the binary value 0. In practice, there are two cases:

- **First case:** we have an equation and need to represent it by a contact chain.
- **Second case:** we have a contact chain and need to deduce the Boolean function.

Example 1 :

Determine the equation of a contact chain from a diagram.

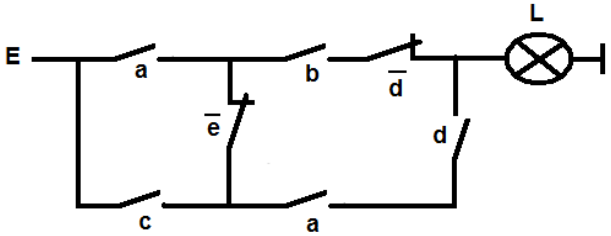


Figure 2.13 : *contact chain*

The equation will be found by traversing the contact chain from input to output in such a way that no loop is ever completed:

$$f_1 = a.b.\bar{d}$$
$$f_2 = c.a.d$$
$$f_3 = c.\bar{e}.b.\bar{d}$$
$$f_4 = a.\bar{e}.a.d = a.\bar{e}.d$$

$$L = \Sigma f_i$$
$$L = ab\bar{d} + cad + c\bar{e}b\bar{d} + a\bar{e}d$$

Example 2 :

Consider a lamp L which, via contactors ABC and source E, must light under the following conditions: the lamp lights when A is energized, or if at the same time B is energized and C is not.

A	*B*	*C*	*L*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 2.14

So :

$$L = A + B\bar{C}$$

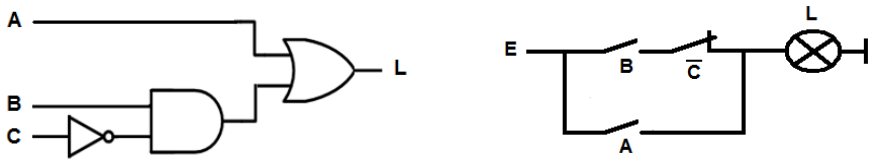


Figure 2.14

2.8. LOGIC EXPRESSION USING NAND AND NOR GATES

In practice, it is often useful to build logic circuits using only one type of gate, i.e. NAND gates or NOR gates.

- The AND operation can be obtained using NAND gates:

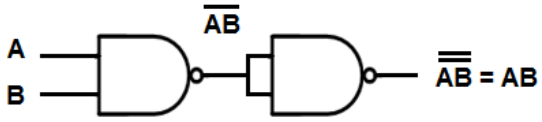


Figure 2.15

- The OR operation can be obtained using NAND gates:

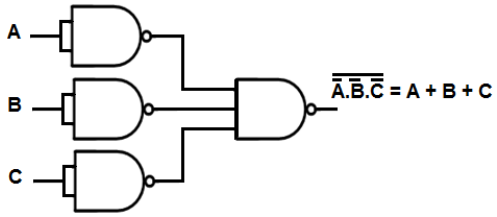


Figure 2.16

- The OR operation can be obtained using NOR gates :

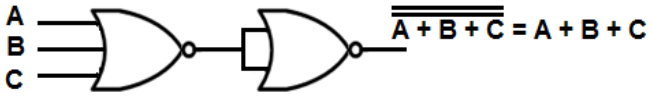


Figure 2.17

- The AND operation can be obtained using NOR gates:

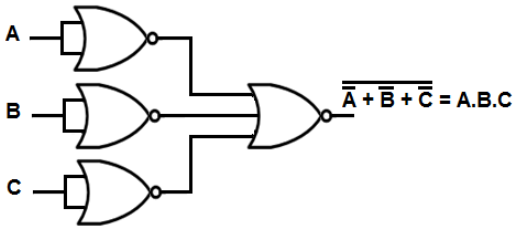


Figure 2.18

Example:

Let $L = A + B\bar{C}$, implement L using only NAND:

$$L = A + B\bar{C}$$
$$\bar{L} = \overline{A + \bar{B}C} = \bar{A}.\overline{\bar{B}C}$$
$$\bar{\bar{L}} = L = \overline{\bar{A}.\overline{\bar{B}C}}$$

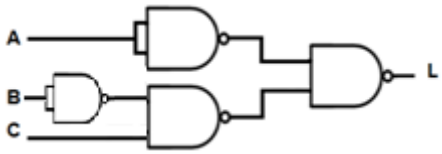


Figure 2.19

2.9. GRAPHIC SIMPLIFICATION "KARNAUGH TABLE"

2.9.1. Adjacent terms and the Karnaugh diagram:

- Two terms are adjacent when they differ from each other by a single variable: for example, ABC and AB$\bar{C}$.
- A Boolean function can be represented graphically by a table, called a "Karnaugh table", which shows the interdependence of the variables.
- An imperative rule: when moving from one row or column to another, only one variable needs to change state (00, 01, 11, 10).

Example: *(Table 2.14 to 2.17)*

B \ A	0	1
0	$\bar{A}\bar{B}$	$A\bar{B}$
1	$\bar{A}B$	AB

$$F = \bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB$$

B \ A	0	1
0	1	
1		1

$$F = \bar{A}\bar{B} + AB$$

C \ BA	00	01	11	10
0	1			
1	1	1	1	1

$$F = \bar{C}\bar{B}\bar{A} + C\bar{B}\bar{A} + C\bar{B}A + CBA + CB\bar{A}$$

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

$$F = \bar{D}\bar{C}\bar{B}\bar{A} + \bar{D}\bar{C}B\bar{A} + \bar{D}CBA + DC\bar{B}A + DCB\bar{A} + D\bar{C}BA$$

2.9.2. Incomplete function

It often happens that a certain combination of variables in a logic function gives the latter an indeterminate value.

Example 1 :

Consider an elevator and two variables X and Y which indicate its state. X to go up, Y to go down.

- If the elevator goes up: X = 1 Y = 0 F = 1
- If the elevator descends: Y = 1 X = 0 F = 0

Note that for X = 1 and Y = 1 the function is impossible. The indeterminate function is represented in the Karnaugh table by Φ.

Example 2 :

Let the logical function: F = Σ m(1,3,5,6,8) + Σi (7,13) avec i : undetermined

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

Table 2.18

Φ : Can take state 1 or 0 does not affect function.

2.9.3. Simplification by Karnaugh table:

Observations on the usual simplifications lead to the following process, to be used in the Karnaugh diagram:

1. put the function under consideration into the first canonical form;
2. represent this function in a Karnaugh diagram;
3. determine adjacent states;
4. group adjacent states into the largest possible blocks.

2.9.3.1. Simplification rules.

Principle: for each cell with the logic state "1" corresponds to a term of the non-simplified equation. Grouping boxes with "1" and simplifies the expression of the grouping.

1. only a number of boxes corresponding to a power of 2 can be grouped together.
2. make the largest groupings possible.
3. we make groups until all the "1" are part of a group.
4. Look for groupings starting with the largest.
5. the equation of a grouping is given by the logical AND of the variables which have the same state for all the cells of the grouping.
6. the equation sought is obtained by the logical OR of the grouping equations

2.9.3.2. Representation of groups

a. Groupings of 2: *(Table 2.19 to 2.23)*

b\a	0	1
0		
1		

b\a	0	1
0		
1		

c\ab	00	01	11	10
0				
1				

c\ab	00	01	11	10
0				
1				

c\ab	00	01	11	10
0				
1				

b. Groupings of 4: (*Table 2.24 to 2.30)*

c\ab	00	01	11	10
0				
1				

c\ab	00	01	11	10
0				
1				

c\ab	00	01	11	10
0				
1				

cd\ab	00	01	11	10
00				
01				
11				
10				

cd\ab	00	01	11	10
00				
01				
11				
10				

cd\ab	00	01	11	10
00				
01				
11				
10				

cd\ab	00	01	11	10
00				
01				
11				
10				

Example 1 :

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

Table 2.31

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

Example 2 :

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

Table 2.32

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

Example 3 :

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

Table 2.33

$$F = \overline{C} + \overline{A}$$

2.10. QUINE-MCCLUSKEY METHOD

The Quine-McCluskey method is an algorithm for simplifying logic functions. The Quine-McCluskey method has several steps:

1. Express the function in canonical disjunctive form.
2. Group min-terms according to their weights
3. Compare terms two by two, until no longer possible
4. Repeat step (3) as many times as necessary.
5. Tick next to the terms that have been combined.
6. Identify the prime implicants, i.e. the terms that are not ticked.
7. Determine the essential prime implicants: we draw a table in which the prime implicants represent the rows and the min-terms represent the columns. The min-terms covered by the implicants are then indicated by 'x'.
8. Check that the function is fully expressed by its essential implicants
9. If you didn't finish in step (8), choose the appropriate prime implicants.

We'll illustrate the technique with a four-variable example (which could just as easily be solved using a Karnaugh table). Consider the following function:

$$F = \Sigma\, m(2,3,4,5,10,12,11,13,15)$$

Step 1: Express the function in canonical disjunctive form.

$$\boldsymbol{F = \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + A\bar{B}CD + AB\bar{C}D + ABCD}$$

Step 2: Group terms according to weight

Group	Canonical form
1	$\bar{A}\bar{B}C\bar{D}$ $\bar{A}B\bar{C}\bar{D}$
2	$\bar{A}\bar{B}CD$ $\bar{A}B\bar{C}D$ $A\bar{B}C\bar{D}$ $AB\bar{C}\bar{D}$
3	$A\bar{B}CD$ $AB\bar{C}D$
4	$ABCD$

Table 2.34

Step 3: Compare each term in a group with each term in the next group. If two terms differ by a single bit, a new term is generated without that bit. Tick the two terms that have generated a new term.

Step 4: Repeat step 3 with the new terms until no more associations are possible.

Group	Canonical form		1st simplification		2nd simplification
1	$\bar{A}\bar{B}C\bar{D}$ $\bar{A}B\bar{C}\bar{D}$	✔ ✔	$\bar{A}\bar{B}C$ $\bar{B}C\bar{D}$	✔ ✔	$\bar{B}C$ $B\bar{C}$
2	$\bar{A}\bar{B}CD$ $\bar{A}B\bar{C}D$ $A\bar{B}C\bar{D}$ $AB\bar{C}\bar{D}$	✔ ✔ ✔ ✔	$\bar{A}B\bar{C}$ $B\bar{C}\bar{D}$ $\bar{B}CD$ $B\bar{C}D$	✔ ✔ ✔ ✔	
3	$A\bar{B}CD$ $AB\bar{C}D$	✔ ✔	$A\bar{B}C$ $AB\bar{C}$	✔ ✔	
4	$ABCD$	✔	ACD ABD		

Table 2.35: *Quine McCluskey method*

Step 5: Terms that have not been associated with others, i.e. those that have not been ticked, represent the prime implicants.

$$\boldsymbol{F = B\bar{C} + \bar{B}C + ABD + ACD}$$

Step 6: In the implicant table, associate each prime implicant with the function terms it can represent. If the canonical term appears only once in a prime intersection, the latter is an essential term, and so it must appear in the simplified form.

	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}\bar{B}CD$	$\bar{A}B\bar{C}D$	$A\bar{B}C\bar{D}$	$AB\bar{C}\bar{D}$	$A\bar{B}CD$	$AB\bar{C}D$	$ABCD$
ACD							x		x
ABD								x	x
$\bar{B}C$	x		x		x		x		
$B\bar{C}$		x		x		x		x	

Table 2.36: *Prime implicant - canonical term.*

Step 7: Check whether the essential prime implicants are sufficient to represent all the terms of the function. If so, simplification is complete. If not, select the appropriate prime implicants.

This gives the two minimal simplified forms of F :

$$F = B\bar{C} + \bar{B}C + ABD$$

Or :

$$F = B\bar{C} + \bar{B}C + ACD$$

Bibliography

1. John R. Gregg, Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets 1st Edition , Wiley & sons Inc. publishing, 1998, ISBN: 978-0-7803-3426-7.
2. Bradford Henry Arnold , Logic and Boolean Algebra, Dover publication, Inc., Mineola, New York, 2011, ISBN-13: 978-0-486-48385-6
3. Alain Cazes,Joëlle Delacroix, architecture des machines et des systèmes informatiques : Cours et exercices corrigés, 3° édition, Dunod 2008.
4. Floyd Thomas, Villeneuve Martin - Systèmes Numériques, 9e édition. Reynald Goulet.
5. Ronald J. Tocci , Circuits numériques (théorie et applications) 2ème édition. Dunod
6. Mesnard Emmanuel, Du binaire au processeur, 2004, Ellipse
7. Philippe Larcher - Introduction à la synthèse logique ; Eyrolles
8. Noël Richard - Electronique numérique et séquentielle (pratique des langages de description de haut niveau) ; Dunod
9. John F.Wakerly, Digital design (principles et practice) third edition updated, Prentice Hall
10. Maurice Gaumain, Cours de systèmes logiques ; Fonctions standards combinatoires, Aspects techniques
11. Jean-Pierre Ginisti, La logique combinatoire, Paris, PUF (coll. Que sais-je? n°3205), 1997.
12. H. Curry, J. R. Hindley et J. P. Seldin, Combinatory Logic II, North-Holland, 1972
13. Philippe Darche, Architecture des ordinateurs, Vuibert
14. Alexandre Nketsa, Circuits logiques programmables Mémoires, CPLD et FPGA. Ellipse
15. Kuntzmann J., Algèbre de Boole, 1968, Paris Dunod.