STRM1

Information Encoding and Representation

Info4 - SEC4 - USTHB

By Dr L.ABADA abada.lyes@gmail.com

1st semester

Ch 1 : Encoding & representation of numbers

Ch 2 : Boolean algebra

Ch 3 : Combinational Circuits

2nd semester

Ch 4 : Sequential circuits

Ch 5 : Memories

Ch 6 : Structure and Functioning of a Basic Computer

- **Chapter 1 : Encoding & representation of numbers,**
- **Chapter 2 : Boolean algebra**
- **Chapter 3 : Combinational Circuits**

Introduction

- Our written language is encoded using a system of 26 letters (in both uppercase and lowercase), 10 digits, punctuation marks, and mathematical symbols

- Thanks to this code and its associated rules, we can transmit information, give instructions, and perform calculations.

- Although computers are called "artificial intelligence," they cannot understand the outside world.

Introduction

- Their intelligence lies in their speed of execution and their ability to work with combinations of two states (0 and 1), equivalent to (off and on).

- The term “**bit**” means "**binary digit**". It is the smallest unit of information that can be processed by a digital machine.

- This binary information can be physically represented by an **electrical or magnetic signal**.

Introduction

- **Encoding** : involves creating a correspondence between :

  - External representation of the information we use

  and

  - Its internal representation within the machine, which consists of a series of bits (0 and 1)

Numeral Systems

- A **Numeral System** is defined by a set of **symbols (numbers or letters)** and **writing rules** for the **positioning** of these symbols.

- The most commonly used numeral system is the **decimal system**.
- Which consists of ten digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

- Numbers are represented as sequences of thses digits.

- with each digit holding a specific **weight** based on its position within the number.

Numeral Systems

$2_3\,3_2\,4_1\,1_0 = 2*10^3+3*10^2+4*10^1+1*10^0$

$abcd = a*10^3+b*10^2+c*10^1+d*10^0$

Base 10
0
1
2
3
4
5
6
7
8
9
10

Base	Base16	Base12	Base 10	Base 9	Base8	Base 5	Base 3	Base 2
	0	0	00	00	0	0	0	00
	1	1	01	01	1	1	1	01
	2	2	02	02	2	2	2	10
	3	3	03	03	3	3	10	11
	4	4	04	04	4	4	11	100
	5	5	05	05	5	10	12	101
	6	6	06	06	6	11	20	110
	7	7	07	07	7	12	21	111
	8	8	08	08	10	13	22	1000
	9	9	09	10	11	14	100	1001
	A	A	10	11	12	20	101	1010
	B	B	11	12	13	21	102	1011
	C	10	12	13	14	22	110	1100
	D	11	13	14	15	23	111	1101
	E	12	14	15	16	24	112	1110
	F	13	15	16	17	30	120	1111
	10	14	16	17	20		121	10000
	11	15	17	18	21		122	10001
	12	16	18	20	22		200	10010
	13	17	19	21	23		201	
		18	20	22	24	44		
		19	***21***	***23***	25	100		
	19	1A	22	24	26			
	1A	1B	23					
	1B	20	24					
	1C			88	77			
			99	100				
	21		100					
	1F							

Base 10	Base 8
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
18	22
19	23
20	***24***
21	25
22	26
23	27
24	30
25	31
26	32
27	33

$$(24)_8 = 2*8^1+4*8^0 = 16+4 = (20)_{10}$$

Base 10	Base 8
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
18	22
19	23
20	***24***
21	25
22	26
23	27
24	30
25	31
26	32
27	33

$$(24)_8 = 2*8^1+4*8^0 = 16+4 = (20)_{10}$$

$$(a_3a_2a_1a_0)_{10} = a_3*10^3+a_2*10^2+a_1*10^1+a_0*10^0$$

$$(a_3\,a_2\,a_1\,a_0)_8 = a_3*8^3+a_2*8^2+a_1*8^1+a_0*8^0$$

$$(a_3\,a_2\,a_1\,a_0)_{10} = a_3*10^3+a_2*10^2+a_1*10^1+a_0*10^0$$

$$(a_3\,a_2\,a_1\,a_0)_{8} = a_3*8^3+a_2*8^2+a_1*8^1+a_0*8^0$$

Bases

A **numeral System** with **base B** is defined by **B symbols** (numbers or letters)

Let **N** be a number of **n** digits represented in base **B**

$$N = a_{n-1}a_{n-2}\ldots a_i \ldots a_0 \quad i \quad a_i < B$$

(All symbols are strictly less than B)

Whatever the base, the polynomial form of N is :

$$N = a_{n-1} * B^{n-1} + a_{n-2} * B^{n-2} \ldots + a_i * B^i \ldots + a_0 * B^0$$

Binary System (Base 2)

→{0,1}

Octal System (Base 8)

→{0,1,2,3,4,5,6,7}

Hexadecimal System (Base 16)

→{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Base conversion : Base B→Base 10:

Just represent the number in its polynomial form and calculate the sum of all the terms

Bases

A **Numeral System** with **base B** is defined by **B symbols** (numbers or letters)

Let **N** be a number of **n** digits represented in base **B**

$$N = a_{n-1}a_{n-2}\ldots a_i \ldots a_0 \quad i \quad a_i < B$$

(All symbols are strictly less than B)

Whatever the base, the polynomial form of N is :

$$N = a_{n-1} * B^{n-1} + a_{n-2} * B^{n-2} \ldots + a_i * B^i \ldots + a_0 * B^0$$

Base conversion : Base B→Base 10:

Example :

B = 2

N= $(1111011)_2 = 1*2^6 + 1*2^5 + 1*2^4 +1*2^3 + 0*2^2 + 1*2^1 +1*2^0 = (123)_{10}$

B = 16

N = $(7B)_{16} = (7*16^1 + 11*16^0)_{10} = 123$

Base conversion : Base B→Base 10:

For decimal numbers, we will use negative exponents.

Example :

B = 16

$N = (7_1B_0,8_14_2)_{16} = 7*16^1 + 11*16^0 + 8*16^{-1} + 4*16^{-2} = (123,515625)_{10}$

Base conversion : Base10 → Base B:

To convert a number from **base-10** to **base-B** we can use : **the successive division**,
We divide the number by **B**

1. We **save the remainder** then divide the **quotient** by **B**

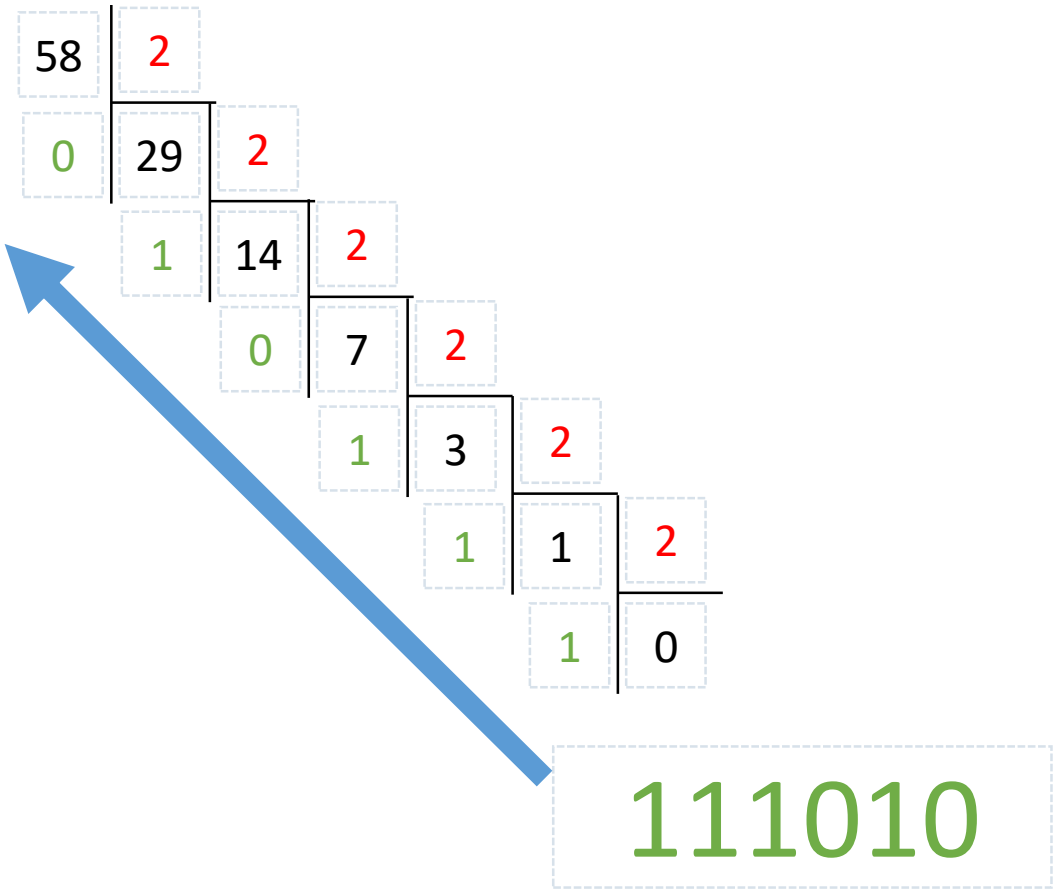
2. And so on, until obtaining a **zero quotient**

3. The sequence of remainders corresponds to the number of the target base

4. **The first remainder** corresponds to the **low weight (Less Significant Bit (LSB))** and the **last** to the **high weight (Most Significant Bit(MSB))**

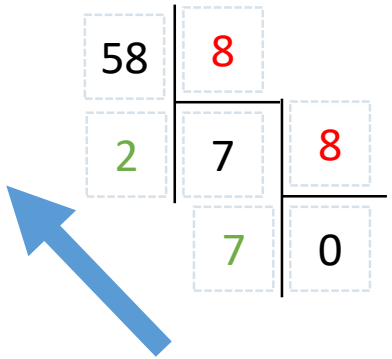
Base conversion : Base 10 → Base B:

Example : binary (base2) → $N=(58)_{10}$ → $(???)_2$



Base conversion : Base 10 → Base B:

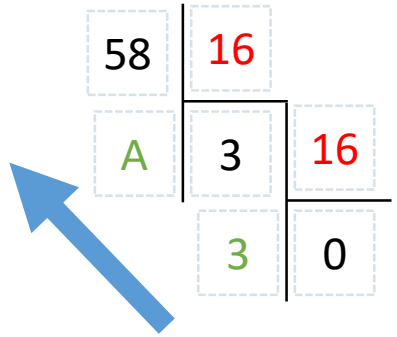
Example : octal (base8) → $N=(58)_{10}$ → $(???)_8$



72

Base conversion : Base 10 → Base B:

Example : hexadecimal (base16) → $N=(58)_{10}$ → $(???)_{16}$



3A

Base conversion : Base 10 → Base B (a decimal number):

Example : binary (base2) →

$N=(58,875)_{10}$ → $(???)_2$

$60,40625_{10} = ????????_2$

$58,875_{10} = 111010,111_2$

$58_{10} = 111010_2$

$0,875_{10} = ???_2$

0,875	0,75	0,5
2	2	2
---	---	---
1,750	1,50	1,0

$0,875_{10} = 111_2$

Base conversion : Base 10 → Base B:

Example : binary (base2) →

$N=(58,875)_{10}$ → $(???)_2$

$58,15625_{10} = 111010,00101_2$

Base conversion : Base B1 → Base B2:

To move from a base B1 to a base B2 we need 2 operations :

1. We must first convert from base B1 to base 10
2. Then from base 10 to base B2

Example : Convert $(3141)_5$ to base 16

$\mathbf{(3141)_5} = (421)_{10}$

$(421)_{10} = \mathbf{(1A5)_{16}}$

→ $\mathbf{(3141)_5 = (1A5)_{16}}$

Base conversion :

Applications to the bases 2, 8 et 16

2➔ 8

To convert a number from **base 2** to **base 8**, we need to split this number into 3-bit groups and replace each group with its octal value (starting from the right).

Example : Convert $(1\ 011\ 010)_2$ to the base 8

$(001\ \ 011\ \ 010)_2 = (1\ 3\ 2)_8$

Base 2	Base 8
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Base conversion :

Applications to the bases 2, 8 et 16

2 ← 8

To convert a number from base 8 to base 2, we just transcribe each digit of this number into 3-bit binary (starting from low weight).

Example : Convert $(645)_8$ to the base 2

$(420)_8 = (\ 100\ \ 010\ \ 000)_2$

Base 2	Base 8
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Base conversion : Applications to the bases 2, 8 et 16

2 ➔ 16

To convert a number from base 2 to base 16, you must split the number into groups of 4 bits and replace each group with its hexadecimal value (starting from the right).

Example :

Convert $(101\ \ 1010)_2$ to the base 16

$(0101\ \ 1010)_2 = (5\ A)_{16}$

Base 2	Base 16
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Base conversion : Applications to the basics 2, 8 et 16

2 ← 16

To convert a number from base 8 to base 2, we just transcribe each digit of this number into 4 -bit binary (starting from low weight).

Example : Convert $(A15)_{16}$ to base 2

$(A15)_{16} = (1010\ \ 0001\ 0101)_2$

Base 2	Base 16
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Base conversion : Applications to the bases 2, 8 et 16

8 → 16

To convert a number from **base 8** to **base 16** or **vice versa**, we can use **base 10 but** we can also use **base 2**

Base8 → **base2** → **base16**

Base16 → **base2** → **base8**

Example : Convert $(232)_8$ to the base 16

$(232)_8 = (010\ \ 011\ \ 010)_2 = (\ 0000\ \ 1001\ \ 1010)_2 = (0\ 9\ A)_{16}$

Base 2	Base 16
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Base conversion : Applications to the bases 2, 8 et 16

Decimal numbers

To convert decimal numbers, we separate the integer part from the decimal part.

The integer part is processed as previously indicated.
The conversion of the decimal part is done from left to right.

$(11101,010\ 11)_2 = (\ 011\ \ 101,\ 010\ 110\ )_2 = (35,26)_8$

$(11101\ ,0101\ 1)_2 = (\ 0001\ \ 1101,\ 0101\ 1000\ )_2 = (1D,58)_{16}$

Base 2	Base 16

Binary Arithmetic

1

(Examples of operations performed in binary)

Addition

	1	1	1	1					
	1	1	1	0	1			29	
+			1	1	1		+	7	
	1	0	0	1	0	0		36	

0+0 = 0

0+1 = 1

1+0 = 1

1+1 = 10 = 2_{10}

1+1+1 = 11 = 3_{10}

Binary Arithmetic

(Examples of operations performed in binary)

1

1

Subtraction

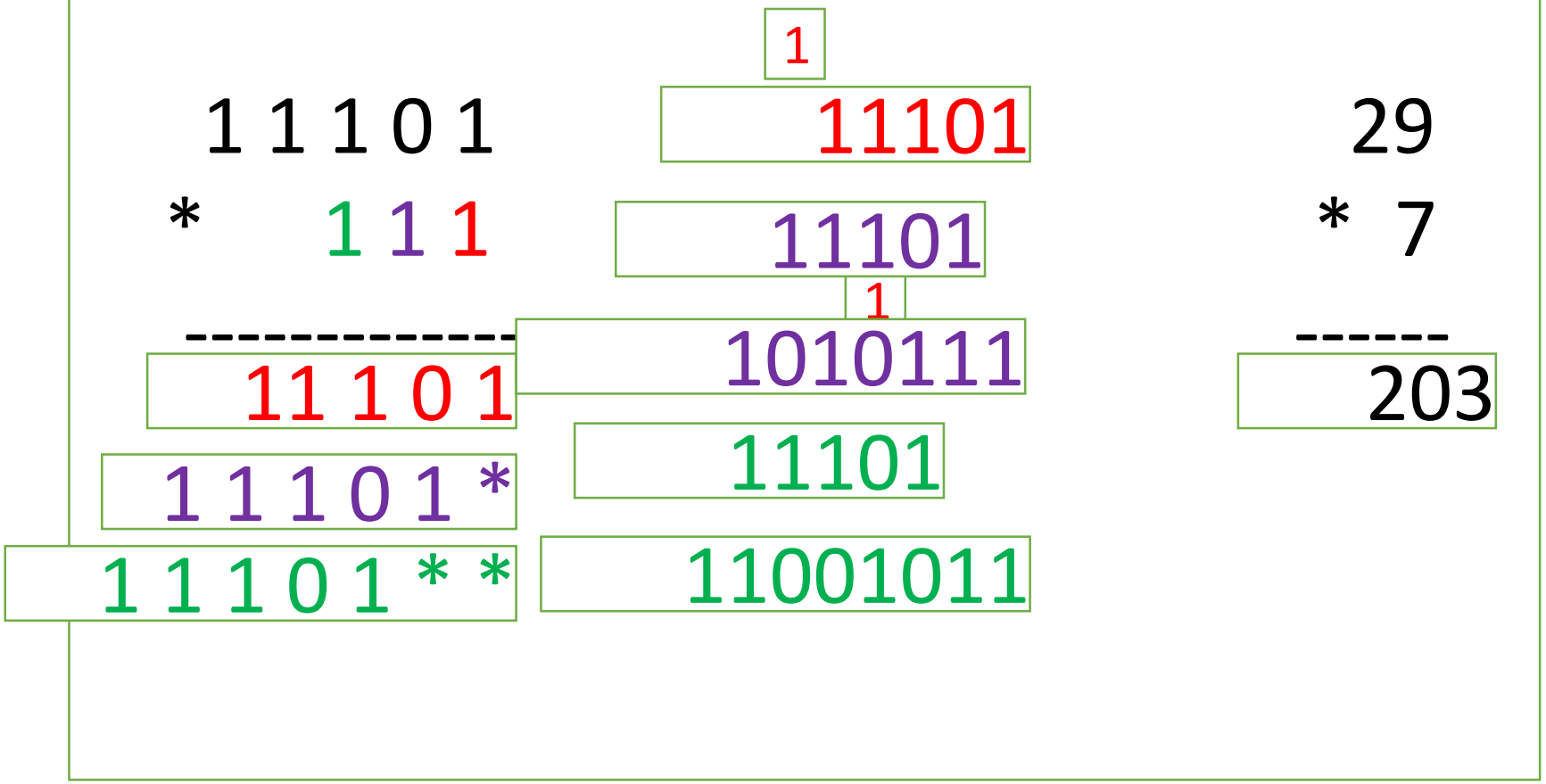
```
  1   1   1   1   0 1          27
  1   1   1   1   0 1          27
-         1   1 1          -  9
-         1   1 1          -  9
      1       1
      1       1
 ----------------          -----
 1  0    1   1 0             22
```

0-0 = 0
0-1 =
1-0 = 1
1-1 = 0

Binary Arithmetic

(Examples of operations performed in binary)

Multiplication



0*0 =
0*1 =
1*0 =
1*1 =

Binary Arithmetic

(Examples of operations performed in binary)

Division

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

2 9	7
1	4

0*0 =
0*1 =
1*0 =
1*1 =

Binary Arithmetic

(Examples of operations performed in binary)

Division

```
11000001 | 101
---------|-----
001000   | 100110
  00110  |
    0011 |
```

```
193 | 5
----|---
 43 | 38
  3 |
```

0*0 =
0*1 =
1*0 =
1*1 =

Representation of relative integers

Relative integers are represented in binary in a **fixed format**

we cannot compare two numbers with two **different numbers of bits** : for example a 5-bit number with an 8-bit number

The most significant bit **(MSB)** represents **the sign** :

- it is equal to 0 if the number is positive,
- or 1 if the number is negative,

Representation of relative integers

- **Sign–magnitude representation, also called sign-and-magnitude or signed magnitude**
- **One's complement representation (C1)**
- **Two's complement representation (C2)**
- **Representing a floating point number in IEEE 754 format**

Representation in Sign and magnitude

The **most significant bit (MSB)** represents the sign of the number (0 for + and 1 for –)
The other bits represent the **absolute value** of the number.
X is represented in an n-bit format : $-(2^{n-1}-1) < X < +(2^{n-1}-1)$

$000 = 7 = 2^n-1$

Examples: Representation a number of an **<u>8 bits</u>**

A = +25

$0\,0011001_{(MS)}$ $\quad -(2^7-1) < X < +(2^7-1) \quad -127 < X < 127$

A = -25

$1\,0011001_{(MS)}$

Representation in Sign and magnitude

The **most significant bit (MSB)** represents the sign of the number (0 for + and 1 for –)
The other bits represent the **absolute value** of the number.
X is represented in an n-bit format : $-(2^{n-1}-1) < X < +(2^{n-1}-1)$

Examples: Representation a number of an n=**8 bits**
$B = +525 = 10\ 0000\ 1101_{(2)}$
$525 > 2^{11}-1 = 127$

525 :

N'est pas possible,

1's complement representation (C1)

The one's complement of a number is obtained by (toggling)reversing all the bits.
The most significant bit represents the sign of the number (0 for + and 1 for –).
X represented in an n-bit format : $-(2^{n-1}-1) < X < +(2^{n-1}-1)$

Examples: Representation a number of an n=**8 bits**

A = 1 0 0 1 1 0 0 1
C1(A) = 0 1 1 0 0 1 1 0

A + C1(A) = 11111111+1 =0000000

2's complement representation (C2)

The 2's complement of a number is equal to the 1's complement (C1) plus 1 :

2's= 1's+1,

X Let X be represented in two's complement of n bits : $\mathbf{-(2^{n-1}) < X < +(2^{n-1}-1)}$

(-X)= 2's (X)

Examples: Representation a number of an **8 bits**

A = 25 = $00011001_{(2)}$

C1(25)= $11100110_{(c1)}$

C2(25) = 11100110 +1 = $11100111_{(c2)}$

C2(25) = -25 (sur 8bit)

2's complement representation (C2)

Example of n=4:

$E = [-2^{4-1}, 2^{4-1} - 1]$

$E = [-8,7]$

0= 0000 1111+1 = 0000
1= 0001 1110+1=1111
2= 0010 1101+1 = 1110
3= 0011 1100+1 = 1101
4= 0100
5= 0101
6= 0110
7= 0111 1000+1 = 1001
1111

1110➔ 0001+1
= 0010

-0= C2(0) = 0000
-1= C2(1) = 1111
-2= C2(2) = 1110
-3= C2(3) = 1101
-4= C2(4) = 1100
-5= C2(5) = 1011
-6= C2(6) = 1010
-7= C2(7)= 1001
-8= C2(8)= 1000

2's complement representation (C2)

Let A and B be 2 numbers represented in C2 of 8 bits, find their decimal values.

Represent in C2 :

A = 53

$A = 00110101_{(2)} = 00110101_{(C2)}$

A = -53

A = -53 = C2(53) = C2(00110101) = C1(00110101)+1

$= 11001010+1 = 11001011_{(C2)}$

Compute the C2 :

A = 13

C2(A) = C2(13) = C1(00001101) +1 = 11110010+1 = 11110011 = -13

Arithmetic operations in 2's complement

In this representation the subtraction must be treated as an addition. For arithmetic operations we must apply the following rule:

$$\forall (A,B) \in E \quad si \quad X = A + B \quad alors \quad X \in E$$

Examples of arithmetic operations with n=4 (E = [- 8,+ 7]) :

Addition

```
    5
+   1
-------
```

```
    0101
+   0001
------------
    0110
```

overflow?

no overflow

Arithmetic operations in 2's complement

In this representation the subtraction must be treated as an addition. For arithmetic operations we must apply the following rule:

$$\forall (A,B) \in E \quad si \quad X = A + B \quad alors \quad X \in E$$

Examples of arithmetic operations with n=4 (E = [- 8,+ 7]) :

Addition

```
   -7          1001
+  -4     +    1100
-------    ----------
   -4        ±0101
```

overflow?

there is an overflow,
(+) + (+) = (-)

Arithmetic operations in 2's complement

In this representation the subtraction must be treated as an addition. For arithmetic operations we must apply the following rule:

$$\forall (A,B) \in E \quad si \quad X = A + B \quad alors \quad X \in E$$

Examples of arithmetic operations with n=4 (E = [- 8,+ 7]) :

Substraction

```
   5
-  3
------
  +2
```

```
    0101
+   1101
------------
(+2) = 10010
```

overflow?

Treatment of overflow for an addition:

- If the two operands have the same sign and the result has the same sign as the operands, there is no overflow.
  - 0001 + 0101 = 0110 ⇔ 1+5= 6
  - 1111 + 1011 = ~~1~~1010 ⇔ -1-5=-6

- If the two operands have the same sign and the result has the opposite sign then there is an overflow.
  - 0101+0100=1001 ⇔ ~~5+4 = -7~~

- If the two operands have opposite signs, there is never an overflow
  - 1110+0111=~~1~~0101 ⇔ -2 + 7 = 5

Treatment of overflow for an addition:

Note :

- Do not confuse these two expressions. :

1. « Represent X in C2 format » : is equivalent to writing X in C2
2. «Give the C2 of X » is equivalent to writing the opposite of X

Ex : X = 35

Represent 35 in C2 of 8bits

$35 = 00100011_{(C2)}$

Represent -35 in C2 of 8bits

-35 = C2(35) = C1(00100011)+1 = 11011101

Compute/give the C2 of 35

C2(35) = -35 = C1(00100011)+1 = 11011101

BCD code

The BCD code, Binary Coded Decimal, is a code that only applies to base 10 digits. Each decimal digit is represented directly by its binary value in a 4-bit format.

BCD code table

DEC	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example :

$(987)_{10} = (1001 \ \ 1000 \ 0111)_{BCD}$

$(13)_{10} = (0001 \ \ 0011 \ \ )_{BCD}$

Addition in BCD

Binaire	BCD
0 → 0000	0000 ← 0
1 → 0001	0001 ← 1
2 → 0010	0010 ← 2
3 → 0011	0011 ← 3
4 → 0100	0100 ← 4
5 → 0101	0101 ← 5
6 → 0110	0110 ← 6
7 → 0111	0111 ← 7
8 → 1000	1000 ← 8
9 → 1001	1001 ← 9
10 → 1010	0001 0000 ← 10
11 → 1011	0001 0001 ← 11
12 → 1100	0001 0010 ← 12
13 → 1101	0001 0011 ← 13
14 → 1110	0001 0100 ← 14
15 → 1111	0001 0101 ← 15
16 → 10000	0001 0110 ← 16

+

---

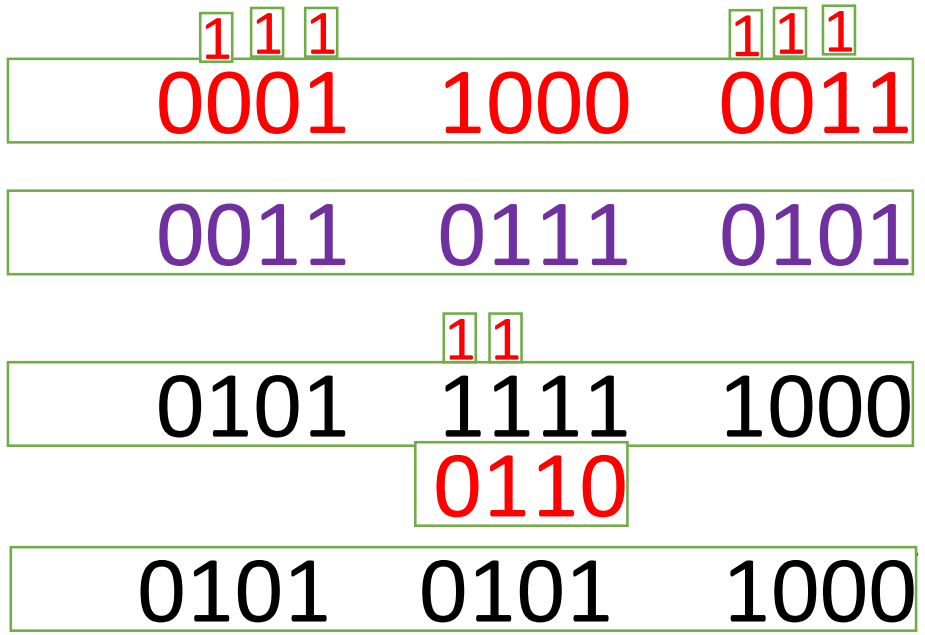
1 0011

Addition in BCD

If the sum of 2 digits coded in BCD is less than or equal to 9 (1001) then we do nothing otherwise we add 6 (0110) to correct it and we retain 1.

Example: Compute the addition A + B in BCD:

$A = 183_{10} = (\ 0001\ 1000\ 0011\ )_{BCD}$ $B = 375_{10} = (\ 0011\ 0111\ 0101\ )_{BCD}$



Addition in BCD

1 0110

If the sum of 2 digits coded in BCD is less than or equal to 9 (1001) then we do nothing otherwise we add 6 (0110) to correct it and we retain 1.

Example 2

4	0000 0100
+ 3	0000 0011

Example 3

4	0100
+ 9	1001
=	----------

GRAY code

Gray code is a binary code that allows you to go from an integer number N to the next number (N+1) by changing a single digit (bit). (It is also called Reflective Code). This code will be used mainly in Karnaugh Tables to simplify Boolean functions.

0000	0		11		22
	1		12		23
	2		13		24
	3		14		25
	4		15		26
	5		16		27
	6		17		28
	7		18		29
	8		19		30
	9		20		31
	10		21		32

GRAY code

Gray code is a binary code that allows you to go from an integer number N to the next number (N+1) by changing a single digit (bit). (It is also called Reflective Code). This code will be used mainly in Karnaugh Tables to simplify Boolean functions.

0000	**0**
0001	**1**
0011	**2**
0010	**3**
0110	**4**
0111	**5**
0101	**6**
0100	**7**
1100	**8**
1101	**9**
1111	**10**

1110	**11**
1010	**12**
1011	**13**
1001	**14**
1000	**15**
11000	**16**
11001	**17**
11011	**18**
11010	**19**
11110	**20**
11111	**21**

11101	**22**
11100	**23**
10100	**24**
10101	**25**
10111	**26**
10110	**27**
10010	**28**
10011	**29**
10001	**30**
10000	**31**
110000	**32**

Compute the next number (successor):

To compute the next number of a gray number (successor):

if the number of 1s is **even**, the least significant bit must be inverted,

if the number of 1s is **odd**, you must reverse the digit located to the left of the rightmost 1.

Example 1 : 32 = 110000

.....

Example 2 : X = 1011101010

.....

Conversion from binary code to Gray code

Let $\mathbf{X = B_n\ B_{n-1}..... B_0}$ a number represented in binary
To convert X into Gray code you must follow the following rules :

$G_n = B_n$
$G_i = 0$ si $B_i = B_{i+1}$
$G_i = 1$ si $B_i <> B_{i+1}$

Example : for n = 4 X = 10001 in binary X = (?)$_{gray}$

1 0 0 0 1
B4 B3 B2 B1 B0

1 1 0 0 1
G4 G3 G2 G1 G0

Conversion from Gray code to binary code

Let $X = G_n\ G_{n-1}.....G_0$ represented in Gray code
To convert X to binary you must follow the following rules:

$Bn = Gn$
$B_i = 0$ si $B_{i+1} = G_i$
$B_i = 1$ si $B_{i+1} <> G_i$

Example : for n = 4 X = 10101 in Gray code $X = (\ 11001\ )_2$

1 0 1 0 1
G4 G3 G2 G1 G0
1 1 0 0 1
B4 B3 B2 B1 B0

- Numeral systems (B2,B8, B16,B10…)
- Sign–magnitude representation
- One's complement representation (C1)
- Two's complement representation (C2)
- BCD Code
- Gray Code
- Representing a floating point number in IEEE 754 format

Representation of a floating point number in IEEE 754 format

Format (32 bits)
Sign bit (1 bit)
Exponent (8 bits)
Mantissa (23 bits)

Let X a real number written in base 2,
In order to present X in **floating point format**, the point must be shifted so that there is only one **1** in the integer part and we increase the exponent at each shift.
Then we get a number in the following format:

$$X_2 = \pm 1,M * 2^e$$

M is the mantissa, **e** is the exponent , **S** is the sign **(0 if X>0 1 if X<0)**

Representation of a floating point number in IEEE 754 format

$$X_2 = \pm 1,M * 2^e$$

M is the mantissa, **e** is the exponent , **S** is the sign **(0 if X>0 1 if X<0)**

The mantissa is the decimal part of **X (23 bits)**

The 1 before the decimal point is not coded in the computer and it is called a **hidden bit**

The exponent e is represented by its real value (e) plus 127_{10},

The obtained value is the characteristic (c) **(8 bits)**

$c = e + 127_{10}$

Representation of a floating point number in IEEE 754 format

Example: Representation of X_{10} = 7,625
Convert to base 2
X2 = 111,101

Shift the decimal point to obtain only one “1” in the entire part
X2 = 1,11101 * 2^2
M = 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
e = 2
c = 2 +127 = 129_{10} = 1 0 0 0 0 0 0 1
Sign = 0
The representation of X in floating point is :

0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
X = 4 0 F 4 0 0 0 0 **(**Condensed form in hexadecimal)

Representation of a floating point number in IEEE 754 format

Let X a number represented in floating point on 32 bits, to find the corresponding value, It must be divided as follows :

- One bit for sign,
- 8 Bits for characteristic
- 23 bits for the mantissa.

We directly deduce the sign and the decimal part then we compute the exponent (e).
If the sign bit = 0 then X>0 else X <0
Decimal part = Mantissa
Integer part = 1
e = c – 127

Converting a number X represented in floating point to decimal :

Example : Compute the decimal value of X

X = C 1 E 9 0 0 0 0

X = 1 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Mantissa = 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

c = 1 0 0 0 0 0 1 1 = 131_{10}

e = 131 – 127

e = 4

X_2 = - 1, 1101001* 2^4 = - 11101,001

X_{10} = -29,125

Converting a number X represented in floating point to decimal :

Remarque :

There is also a double precision format that allows a much larger set of numbers to be represented with greater precision.

In this case we have the following representation:

Format (64 bits)
sign bit (1 bit)
Exponent (11 bits)
Mantissa (52 bits)

Adding two IEEE floating point numbers (32 bits)

To do the addition of two floating point numbers, they must have the same exponent.
Let **A = 4 0 D 9 0 0 0 0** et **B = 3 E 9 A 0 0 0 0** in floating point IEEE
Perform the operation: A + B

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

c = 1 0 0 0 0 0 0 1 = 129_{10} => e = 129 – 127 => e = 2

M = 1 0 1 10 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

So **A = 1 , 1 0 1 1 0 0 1 * 2^2**

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

c = 01111101 = 125_{10} => e = 125 – 127 => e = -2

M = 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

So **B = 1 , 0 0 1 1 0 1 * 2^{-2}**

Adding two IEEE floating point numbers (32 bits)

We put B at the same exponent as A by moving the decimal point 4 places to the left.

B = 0 , 0 0 0 1 0 0 1 1 0 1 * 2^2

We do : A + B

A = 1 , 1 0 1 1 0 0 1 0 0 0 * 2^2

B = 0 , 0 0 0 1 0 0 1 1 0 1 * 2^2

C = 1 , 1 1 0 0 0 1 0 1 0 1 * 2^2

We replace the result in the form IEEE

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

A + B = 4 0 E 2 A 0 0 0

Adding two IEEE floating point numbers (32 bits)

Remarque

If the integer part of the result is greater than 1, we shift the decimal point one place to the left and increase the exponent by 1 :

For example

if A + B = 10,11001101 * 2^2 => A + B = 1,011001101 * 2^3

Codification and representation Alphanumeric

Codification and representation Alphanumeric : ASCII code

Le code ASCII

Definition of ASCII code : American Standard Code For Information Interchange,

Is a computer coding standard developed in the 1960s.

This code defines 128 characters represented on 7bits (8 bits in computer).

This table is presented in a **condensed form**, based on **base 16**.

Each character is found at the intersection of a row and a column.

The **line number followed** by the **column number** represents the character code in **hexadecimal**.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Example :

The letter M is found at the intersection of line **4** and column **D**

its code is **4D** in hexadecimal.

Its representation is therefore : **0100 1101** en code ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Codification and representation Alphanumeric : ASCII code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US

Control characters :

We can consider that ASCII has around thirty control characters.

The usual characters are NUL, LF, CR, DEL et ESC

- **NUL :** indicates the end of a String, particularly in the **C** language.
- **LF** et **CR** : indicate the end of a line.
- We use **LF, CR** or both depending on the operating system :
  - Linux use **LF**,
  - Mac OS use **CR**
  - Windows use **CR** followed by **LF**.
- **ESC** indicates the output of a text.
- **DEL** indicates character delete.

Codification and representation Alphanumeric : ASCII code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Example : In a text editor or in an input field, type without releasing the alt key + the character code in decimal :

$(30)_{16} = (48)_{10}$ ➔ '0' $(41)_{16} = (65)_{10}$ ➔ 'A' $(61)_{16} = (97)_{10}$ ➔ 'a' $(20)_{16} = (32)_{10}$ ➔ ' '

$(4D)_{16} = (77)_{10}$ ➔ 'M' $(3F)_{16} = (63)_{10}$ ➔ '?' $(45)_{16} = (69)_{10}$ ➔ 'E'

Codification and representation Alphanumeric : ASCII code

Exercise N°7:

1/ En code ASCII $(41)_{16}$ correspond to 'A' and $(30)_{16}$ correspond to '0', without using the ASCII code table deduce the coding of the following message : STRM1

2/ Decode the following message:

57454C434F4D4520494E20434F4D505554420552534349454E4345

Codification and representation Alphanumeric : ASCII code

<u>Exercise N°9:</u> (à faire comme un exemple en cours)

1/ En code ASCII

$(41)_{16}$ ➔ 'A'

$(61)_{16}$ ➔ 'a'

$(30)_{16}$ ➔ '0',

$(2D)_{16}$ ➔ '-'

Sans l'utilisation de la table du code ASCII déduire le codage du message suivant :

STRM

Codification and representation Alphanumeric : ASCII code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Exercise N°9:

2/ Décoder le message suivant :

Codification and representation Alphanumeric : UNICODE code

UNICODE code

Unicode is a coding standard developed in the 1990s;
It defines more than **60000** characters from several languages, coded on **16 bits**.
ASCII code is included in Unicode.

The ASCII code is only based on Anglo-Saxon letters; we do not find the accented letters of the French language such as à or é for example.
These letters can be found in the following UNICODE table :

Codification and representation Alphanumeric : UNICODE code

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
008	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
009	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
00A	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY -	®	¯
00B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Codifica

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
060	ـــ	ـــ	ـــ	صـ	□	□	□	□	□	□	□	فـ	،	ء	ـــ	ع
061	ـَ	ـُ	ـّ	ـْ	ـَ	ـُ	ـِ	ـِ	ـِ	ـِ	ـِ	؛	ALM		ـــ	؟
062	□	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د
063	ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	□	□	□	□	□
064	ـ	ف	ق	ك	ل	م	ن	ه	و	ى	ي	ـً	ـّ	ـٍ	ـَ	ـُ
065	ـِ	ـَ	ـْ	ـّ	ـُ	ـِ	ـِ	ـُ	ـْ	ـَ	ـّ	ـْ	ـِ	ـْ	ـْ	□س
066	٠	١	٢	٣	٤	٥	٦	٧	٨	٩	٪	٫	،	*	ـــ	ق
067	ـْ	آ	أ	إ	ٔ	ٴ	ؤ	ۇ	ئ	ٹ	ٹ	پ	ٹ	ٹ	پ	ٹ
068	ڀ	ځ	څ	ڃ	ڃ	ڄ	ڃ	ڃ	ڈ	ډ	ډ	ډ	ډ	ډ	ډ	ډ
069	ډ	ڑ	ړ	ړ	ړ	ړ	ړ	ړ	ڑ	ړ	ښ	ږ	ږ	ږ	ږ	ط
06A	ڠ	ف	ڤ	ڤ	ڤ	ڤ	ڤ	ف	ڤ	ک	ک	ک	ک	ک	ڪ	گ
06B	ګ	گ	ڳ	ڳ	ګ	ڵ	ڵ	ڵ	ڵ	ڼ	ں	ڻ	ڼ	ڻ	ھ	چ
06C	ه	ه	ه	ۀ	ۆ	ۆ	ۆ	ۇ	ۇ	ۋ	ۋ	ۋ	ی	ی	ی	ۆ
06D	ي	ي	ے	ۓ	۔	ه	ۺ	ۺ	ۺ	ۺ	ۺ	ۺ	ۺ	◯	۞	ۺ

Chapter 2 :

Boolean algebra

Boolean algebra

Introduction

The computer is composed of logic circuits.

The fundamental component of these circuits is the transistor, we have two states 0 and 1

0 = Blocked 1 = Conductor

The input variables are those that can be directly manipulated.
They are independent logical variables.

The output variable contains the state of the function after the evaluation of logical operators on the input variables.

To implement these circuits and determine the input variables and output variables, we will use :

Boolean algebra

Boolean algebra

Terminology

- Sum **(OR)** s = a **+** b / s = a **or** b
- Product **(AND)** s = a ***** b / s = a **and** b
- In addition to this, there is a unary application:
- Complementation **(NOT)** $\bar{s}$ / **not**(s)

Boolean algebra

Terminology

- Sum **(OR)** s = a **+** b / s = a **or** b
- Product **(AND)** s = a ***** b / s = a **and** b
- In addition to this, there is a unary application:
- Complementation **(NOT)** $\bar{s}$ / **not**(s)

$s = a \cdot b$

a	b	a . b
0	0	0
0	1	0
1	0	0
1	1	1

$s = a + b$

A	B	a + b
0	0	0
0	1	1
1	0	1
1	1	1

$s = \bar{a}$

A	$\bar{a}$
0	1
1	0

Boolean algebra

Theorems and postulates of Boolean algebra

A Boolean Algebra consists of a set E = {0,1} and Internal binary operation (AND), (OR), (NOT) :

Properties of Boolean Algebra

1- Commutativity

$a + b = b + a$

$a \cdot b = b \cdot a$

2- Associativity

$a + (b + c) = (a + b) + c$

$a \cdot (b \cdot c) = (a \cdot b) \cdot c$

Boolean algebra

Properties of Boolean Algebra

3- Distributivity

$a \cdot (b + c) = a.\ b + a \cdot c$

$a + (b \cdot c) = (a + b).(a + c)$

4- Neutral element

$a + 0 = a$

$a \cdot 1 = a$

5- Inverse element

$a + \bar{a} = 1$

$a \cdot \bar{a} = 0$

Boolean algebra

Deduced Properties

1- Idempotence

$a + a = a$

$a \cdot a = a$

2- Absorbent element

$a + 1 = 1$

$a \cdot 0 = 0$

3- Simplified common expressions

$a + a \cdot b = a$

$a \cdot (a + b) = a$

$a + \bar{a}\, b = a + b$

Boolean algebra

NAND and NOR operators : truth tables

$s = \overline{a \cdot b}$

a	b	$\overline{a \cdot b}$
0	0	1
0	1	1
1	0	1
1	1	0

$s = \overline{a + b}$

a	b	$\overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0

Boolean algebra

The exclusive OR (XOR) and its complement

$s = a \oplus b$

$s = 1$ si $a \neq b$

$s = 0$ si $a = b$

$$a \oplus b = \bar{a} \cdot b + a \cdot \bar{b}$$

$$\overline{a \oplus b} = \bar{a} \cdot \bar{b} + a \cdot b$$

Truth table

a	b	$a \oplus b$	$\overline{a \oplus b}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Boolean algebra

Boolean expressions

Logical functions and normal forms

We call "**min term**" of n variables, one of the products of these **n** variables or their complements.

Example : $a\,\bar{b}\,c$, $\bar{a}\,b\,\bar{c}$ et $a\,b\,c$

are "**min terms**" of a function of 3 variables a, b, and c

Boolean algebra

Boolean expressions

Logical functions and normal forms

We call **"max term"** of n variables, one of the sums of these **n** variables or their complements.

Example : $(a + \bar{b} + c)$, $(\bar{a} + b + \bar{c})$ et $(a + b + c)$

are "**max terms**" of a function of 3 variables a, b, and c

Boolean algebra

First normal form or disjunctive form

A function is in **disjunctive form(1st normal form)** if it is represented by a sum of min terms (sum of products-SOP)

Example :

$$F(a, b, c) = a\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,b\,c$$

Boolean algebra

Second normal form or conjunctive form

A function is in **conjunctive form(2nd normal form)** if it is represented by a product of max terms (product of sums- POS)

Example :

$$F(a, b, c) = (a + \overline{b} + c)\ (\overline{a} + b + \overline{c})\ (a + b + c)$$

Note :
We can go from one form to another using the distributivity

Boolean algebra

DE MORGAN's Law (To check using a truth table)

$$\overline{a + b} = \bar{a} \cdot \bar{b}$$

$$\overline{a \cdot b} = \bar{a} + \bar{b}$$

a	b	$\bar{a}$	$\bar{b}$	a+b	a.b	$\overline{a+b}$	$\overline{a.b}$	$\bar{a},\bar{b}$	$\bar{a}+\bar{b}$
0	0	1	1	0	0	1	1	1	1
0	1	1	0	1	0	0	1	0	1
1	0	0	1	1	0	0	1	0	1
1	1	0	0	1	1	0	0	0	0

Boolean algebra

Complement of a function

∀F a Boolean Function ∃ G such as G = $\overline{F}$

To compute $\overline{F}$ we have to use De Morgan's Laws

Example :

$$F(a, b, c) = a\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,b\,c$$

$$\overline{F}(a, b, c) = \overline{a\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,b\,c}$$

$$\overline{F}(a, b, c) = \overline{a\,\overline{b}\,c} * \overline{\overline{a}\,b\,\overline{c}} * \overline{a\,b\,c}$$

$$\overline{F}(a, b, c) = (\overline{a} + b + \overline{c})\ (a + \overline{b} + c)\ (\overline{a} + \overline{b} + \overline{c})$$

We note that in order to find $\overline{F}$ just invert each variable and each operator

Boolean algebra

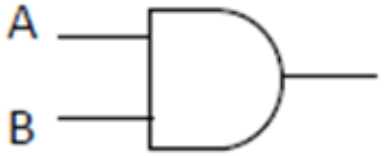
Logic circuits

A logic circuit is a set of interconnected **logic gates** corresponding to an algebraic expression

logic gates (corresponding to a logical operator) :

$Y = A + B$

OR gate



$Y = A \cdot B$

AND Gate

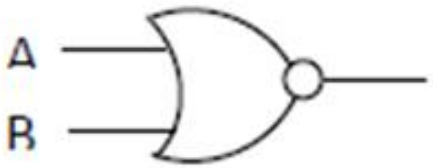
$Y = \overline{A}$

Not Gate

Boolean algebra

Logic circuits

Derived gates :



$Y = \overline{A + B}$

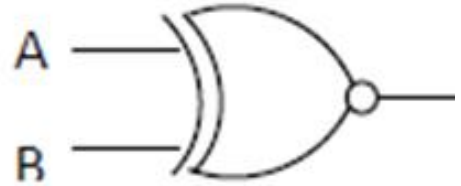
NOR Gate

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

NAND Gate

$Y = A \oplus B$

XOR Gate



$Y = \overline{A \oplus B}$

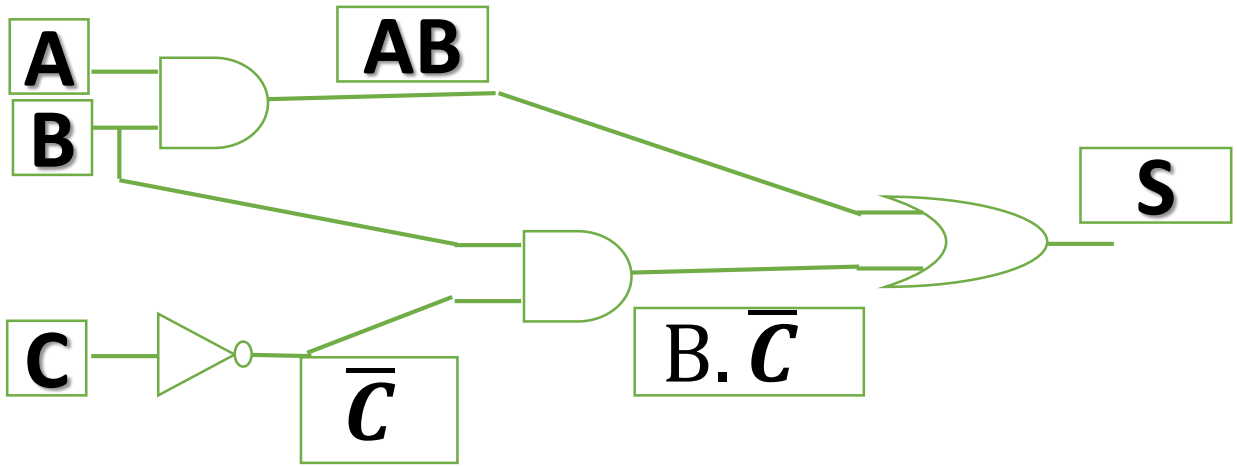
NXOR Gate

Boolean algebra

Logic circuits

Logic circuit corresponding to the algebraic expression :

$S = A.B + B.\overline{C}$



Boolean algebra

Truth table of a function

The Truth Table of a function consists of finding its values for each combination of variables.

Boolean algebra

Truth table of a function

Let the function :

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

F(A, B, C) = 1 si un de ses termes est égal à 1

$\overline{A}\, B\, C = 1$ si A = 0 B = 1 et C = 1 F(0 1 1) = 1

$A\, \overline{B}\, \overline{C} = 1$ si A = 1 B = 0 et C = 0 F(1 0 0) = 1

A B = 1 si A = 1 et B = 1 F(1 1 0) = 1 et F(1 1 1) = 1

Boolean algebra

La Table de Vérité de F est :

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

Pour tirer l'expression d'une fonction à partir d'une Table de Vérité on fait la somme des min termes où F = 1

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

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

$\overline{F}(A, B, C)$ est obtenu en faisant la somme des min termes où F = 0

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

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

Boolean algebra

La Table de Vérité de F est :

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

Pour tirer l'expression d'une fonction à partir d'une Table de Vérité on fait la somme des min termes où F = 1

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

On retrouve la forme conjonctive de cette fonction à partir de $\overline{F}$. $F(A, B, C) = \overline{\overline{F}}(A, B, C)$

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

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

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

Boolean algebra

Simplification of Boolean functions: :

1- Algebraic simplification

To algebraically simplify a Boolean function, we use the properties of Boolean algebra :
Commutativity, Associativity, Distributivity …

Boolean algebra

Example :

$$F(a,b,c) = \overline{a}\, b\, c + a\, \overline{b}\, \overline{c} + a\, b\, c + \overline{a}\, b\, \overline{c} + a\, \overline{b}\, c + a\, b\, \overline{c}$$

$$F(a,b,c) = b\, c\, (\overline{a} + a) + b\, \overline{c}\, (\overline{a} + a) + a\, \overline{b}\, (\overline{c} + c)$$

$$F(a,b,c) = b\, c + b\, \overline{c} + a\, \overline{b}$$

$$F(a,b,c) = b\, (c + \overline{c}) + a\, \overline{b}$$

$$F(a,b,c) = b + a\, \overline{b} \qquad (b + a)(b + \overline{b})$$

$$F(a,b,c) = b + a$$

2-Simplification by the Karnaugh-Map (K-Map)

A Karnaugh-Map is a 2-dimensional truth table.

The numbering of rows and columns is done according to **The gray code,**

We move from a row to the next **by changing one bit**
and from a column to the next **by also changing one bit**.

Boolean algebra

Simplification by the Karnaugh-Map (K-Map)

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

Each cell represents a combination.

If we take a cell in a Karnaugh-Map, all the cells adjacent to it have only one bit that changes

so only one variable changes

A	B	C	D	F
0	0	0	0	
0	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	
1	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	

Boolean algebra

Simplification by the Karnaugh-Map (K-Map)

F (A

Each c

If we t have only o

so onl

AB / CD	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	0	0	0	0

$$F(a,b,c,d) = \bar{a}\bar{c}d + \bar{a}c\bar{d} + \bar{b}\bar{c}d$$

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

Boolean algebra

ab / cd	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	0	0	0	0

$\overline{F}$(a,b,c,d) = $\bar{c}\bar{d} + ab + cd + ac$

$F = \overline{\overline{F}}(a,b,c,d) = \overline{\bar{c}\bar{d} + ab + cd + ac}$

$\overline{\overline{F}}(a,b,c,d) = F$ = (c+d)$(\bar{a} + \bar{b})(\bar{c} + \bar{d})(\bar{a} + \bar{c})$

Incomplete function (Incomplete Karnaugh-Map)

We say that a function is incompletely defined if it is not defined at all its points.

In this case the points where it is not defined will take the value X.

When simplifying the function using the Karnaugh-Map, if a cell containing X is adjacent to a cell containing 1, we can give the value 1 to this X in order to simplify the function.

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

Incomplete function (Incomplete Karnaugh-Map)

ab / cd	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	0	0	X	X
10	1	1	x	x

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

Incomplete function (Inc

ab / cd	00	01	11	10
00	0	0	X	0
01	1	1	X=1	1
11	0	0	X	X
10	1	1	X=1	X=1

$F(a,b,c,d) = \cd + c\d$

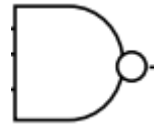
Incomplete function (Inc

ab / cd	00	01	11	10
00	0	1	X	0
01	1	X	0	1
11	1	0	X	1
10	0	0	1	X

$$F(a,b,c,d) = ac + \bar{b}\bar{d} + b\bar{c}\bar{d}$$

Boolean algebra

Using NAND Gates



To create the circuit of a function using NAND gates, the function must be in disjunctive form (1st Normal form or SOP), then just apply the complement twice and use Morgan's laws.

$$F(ABC) = A\,B\,C + A\,\overline{B} + B\,\overline{C}$$

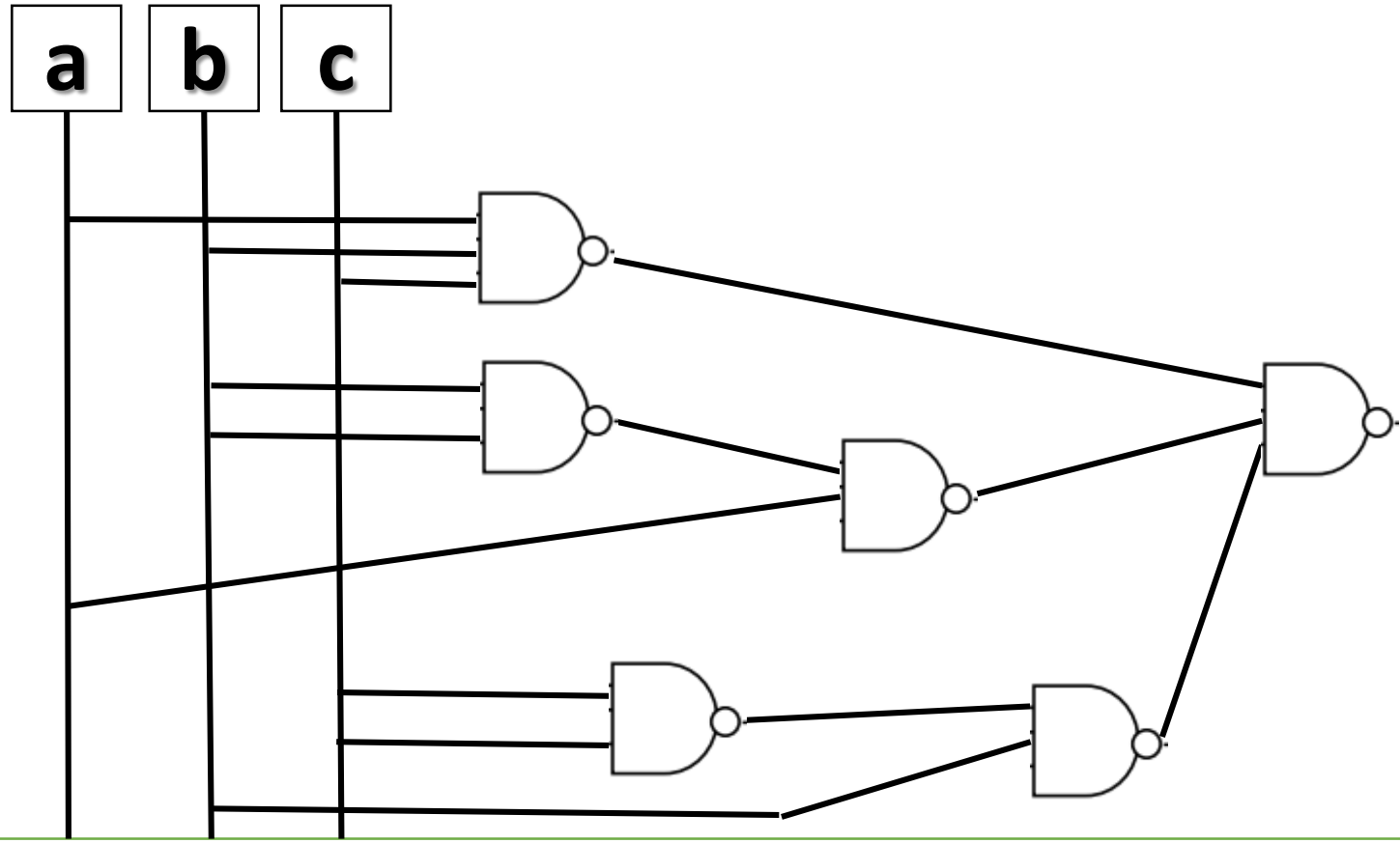
Boolean algebra

Using NAND Gates

/B = /(BB) = /(B+B) **B*B = B**

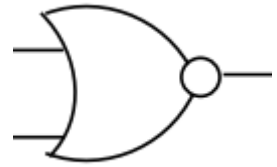
$$F(ABC) = A\,B\,C + A\,\overline{B} + B\,\overline{C}$$

$$F(ABC) = A\,B\,C + A\,\overline{B} + B\,\overline{C} = \overline{\overline{A\,B\,C}\,.\,\overline{A\,\overline{B}}\,.\,\overline{B\,\overline{C}}}$$



Boolean algebra

Using NOR Gates



To create the circuit of a function using NOR gates, the function must be in conjunctive form (2nd Normal form or POS), then just apply the complement twice and use Morgan's laws.

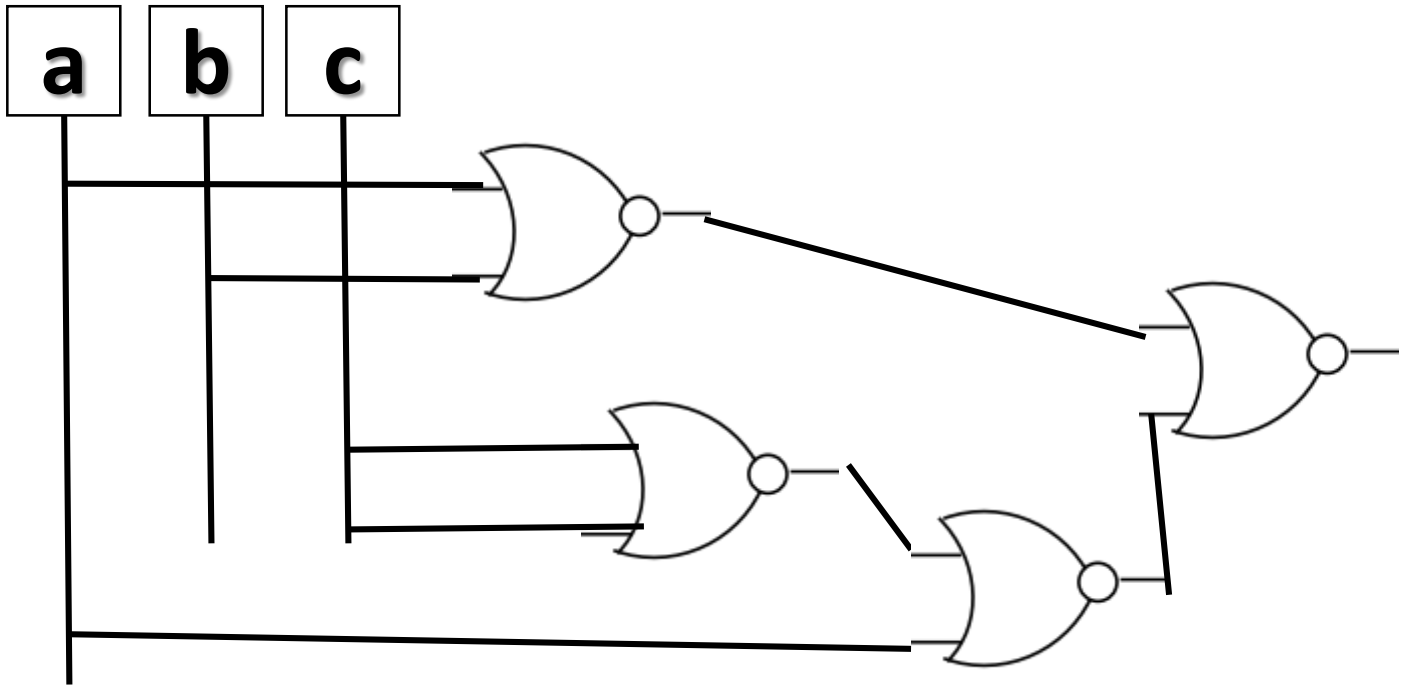
$$G(ABC) = (A + B) (A + \overline{C})$$

Boolean algebra

Using NOR Gates

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

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



Chapter 3 :

Combinational circuits

The adder

Half Adder Circuit

The one-bit **Half Adder** is a circuit that allows to add 2 binary digits without input carry

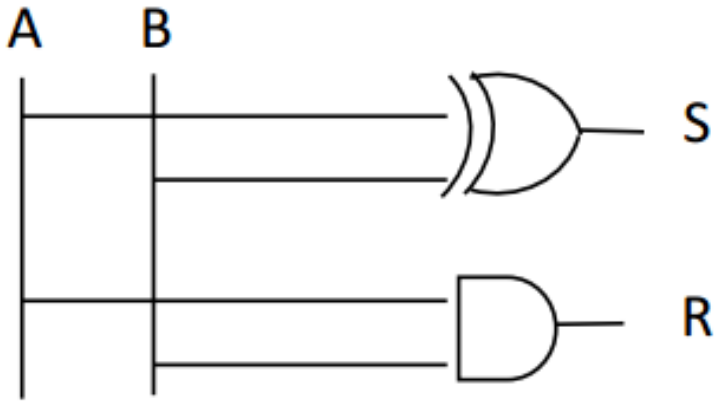
When we add 2 binary digits A and B we obtain the sum S and a carry R.

A + B = S(sum) and R(carry)

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

$$S = A \oplus B$$

$$R = A\,B$$



A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The adder

Full Adder Circuit

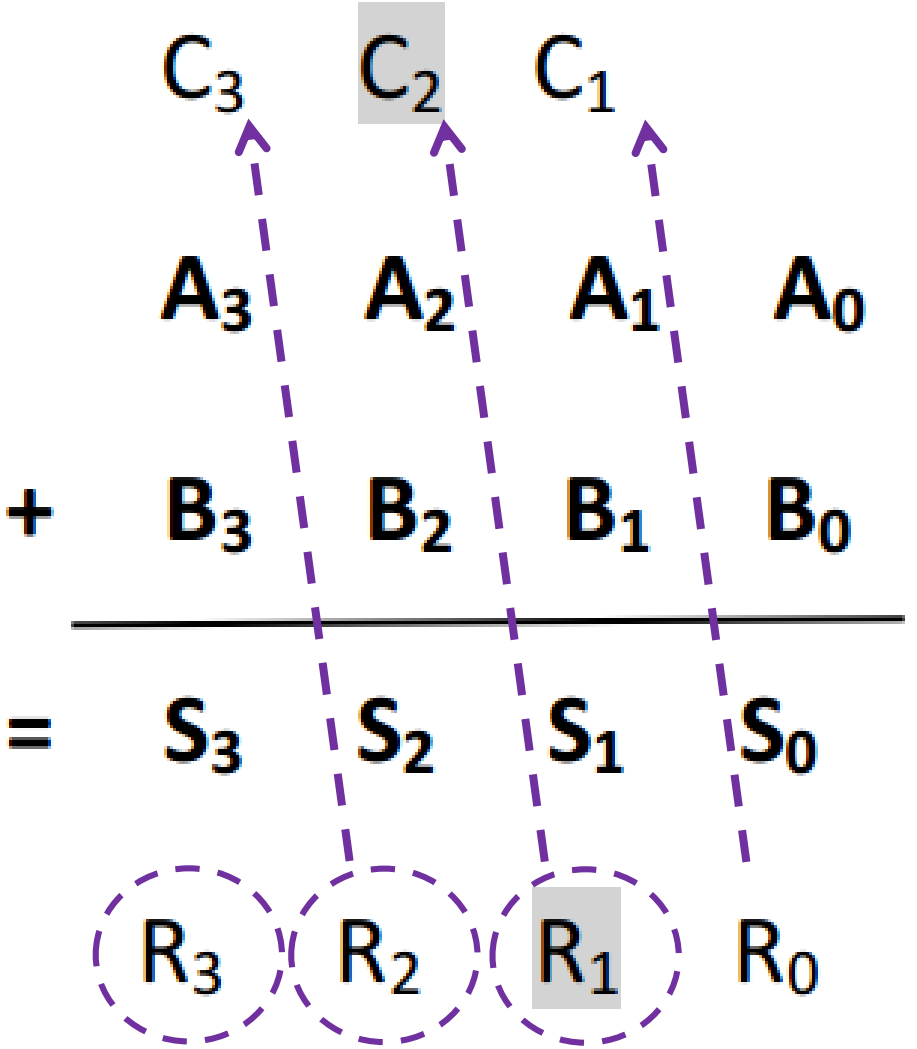
The **One-Bit Full Adder** is a circuit that allows to sum 2 binary digits with input carry.

When we sum 2 binary numbers, we work by column.

We sum the numbers (2 by 2) according to their weights (from right to left).

At the end of each addition we obtain a sum and an output carry.

This carry will be added to the sum of the next row and will become an output carry



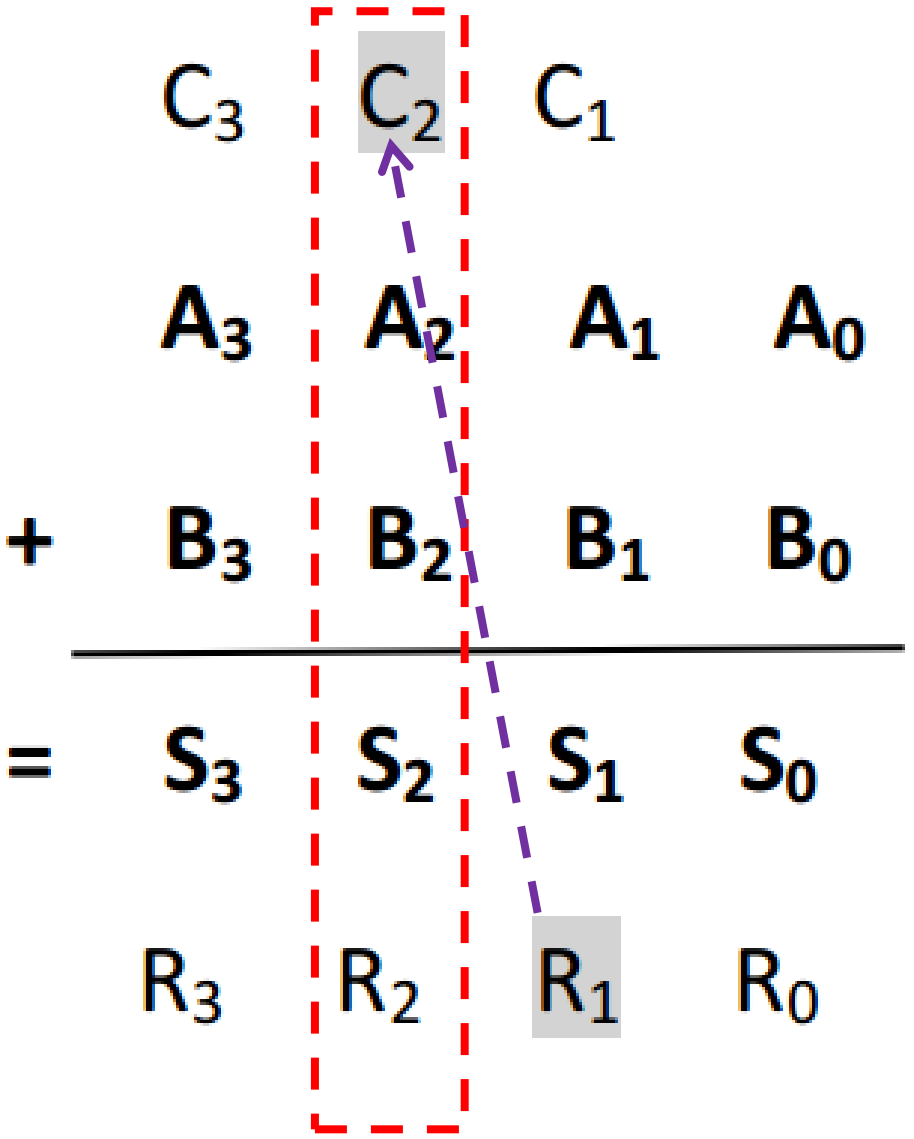
The adder

Full Adder Circuit

There is never an intput carry on the first column (least significant bit(LSB)).

For this row we need a half adder.

For all other columns we need full adders.



The adder

$$R(a,b,c) = ab + bc + ac$$

ab / c	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A	B	C	S	R
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

$$S(a,b,c) = /a/bc + /ab/c + abc + a/b/c$$

$$S(a,b,c) = /a(/bc + b/c) + a(bc + /b/c)$$

$$S(a,b,c) = /a(b \text{ xor } c) + a\overline{(b \text{ xor } c)} = a \text{ xor } b \text{ xor } c$$

The adder

$$a \oplus b = \bar{a}.b + a.\bar{b}$$

$$\overline{a \oplus b} = \bar{a}.\bar{b} + a.b$$

ab / c	00	01	11	10
0		1		1
1	1		1	

A	B	C	S	R
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

$$S = \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc + abc + ab + a\bar{b}\bar{c}$$

$$S = \bar{a}\left(\bar{b}c + b\bar{c}\right) + a\left(bc + \bar{b}\bar{c}\right)$$

$$S = \bar{a}(b \oplus c) + a\left(\overline{b \oplus c}\right)$$

$$S = a \oplus b \oplus c$$

The adder

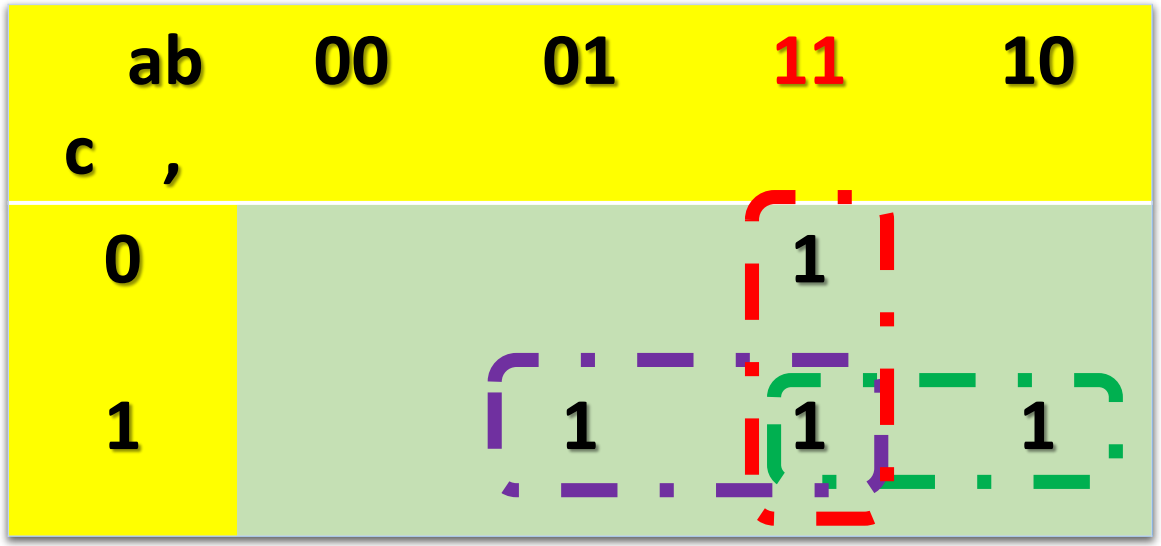
$$a \oplus b = \bar{a}.b + a.\bar{b}$$

$$\overline{a \oplus b} = \bar{a}.\bar{b} + a.b$$

ab c ,	00	01	11	10
0				
1				

A	*B*	*C*	S	R
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

The adder



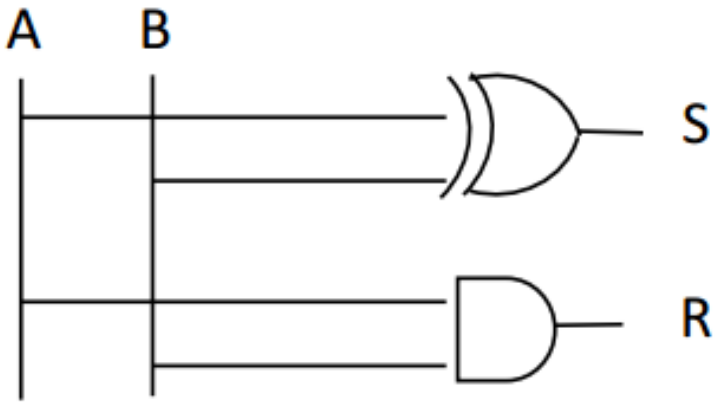
R = AB+ AC + BC

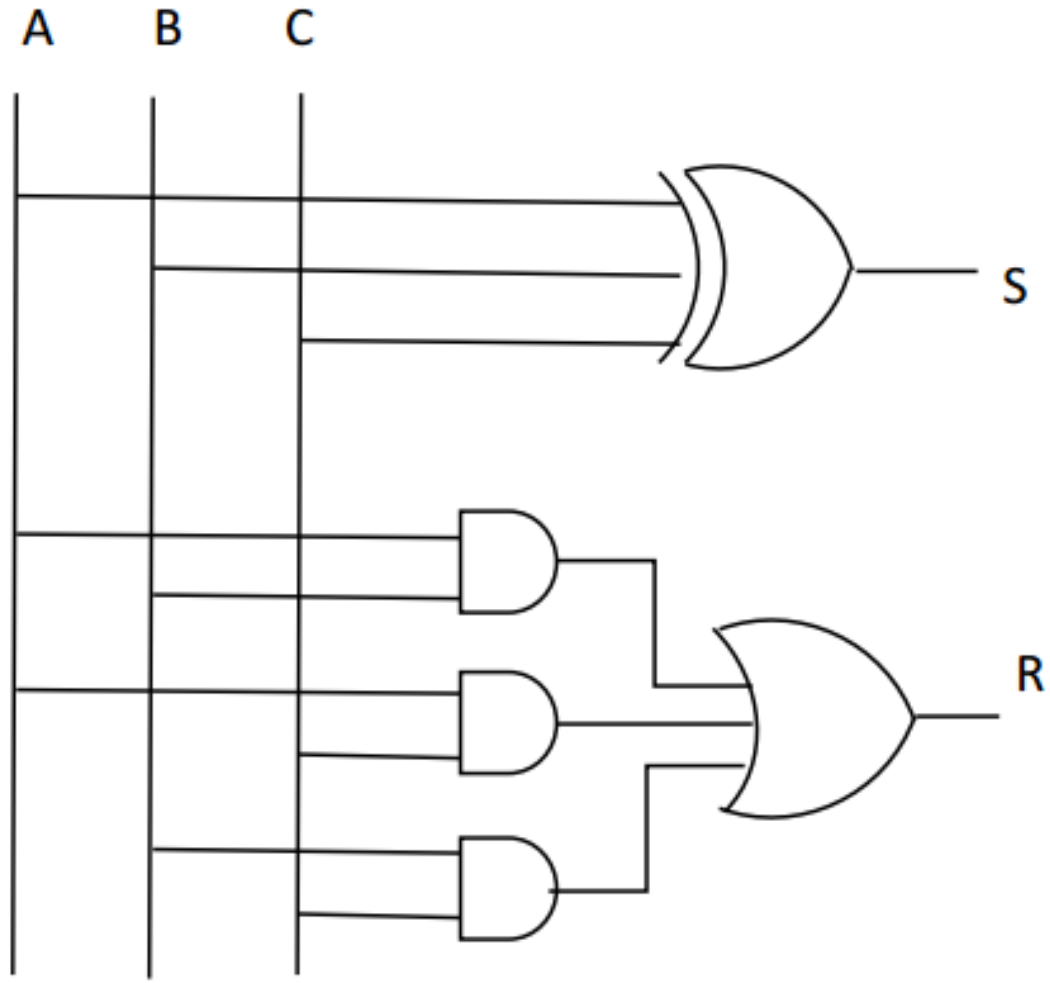
A	B	C	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The adder

$S = a \oplus b \oplus c$

R = AB+ AC + BC



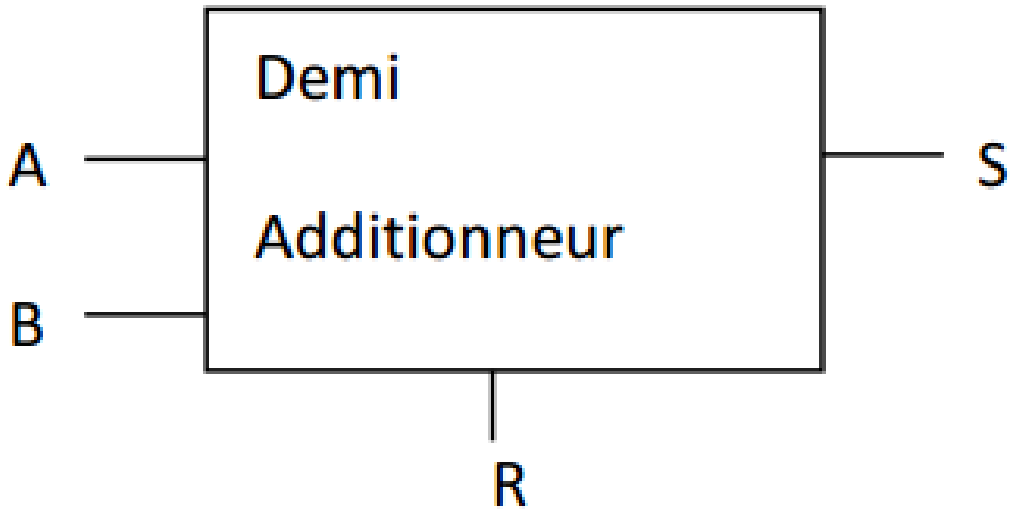


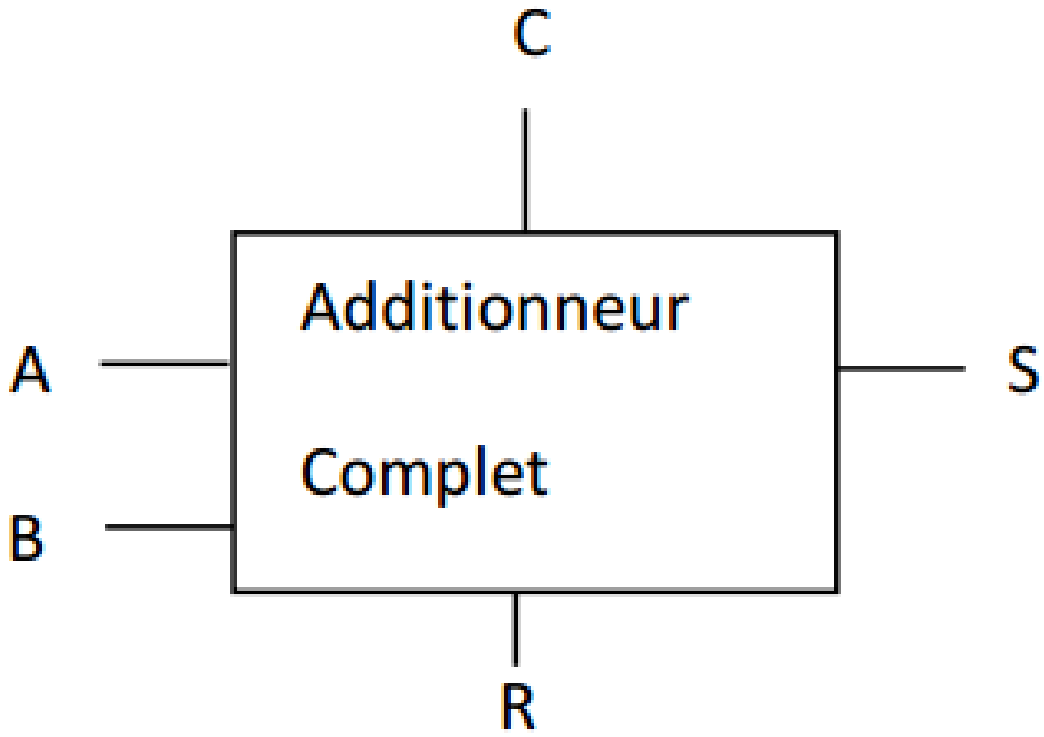
The adder

Functional circuits

A functional circuit is a component named by the function it represents.

It has the inputs and outputs of that function.





4-bit Adder circuit

To create a 4-bit adder, we must connect 4 **one-bit adders**: a half adder and three full adders.

The connection is made by the carry "$C_i = R_{i-1}$"

This circuit adds two numbers :

A3 A2 A1 A0

et **B3 B2 B1 B0**

The result is : **S3 S2 S1 S0**

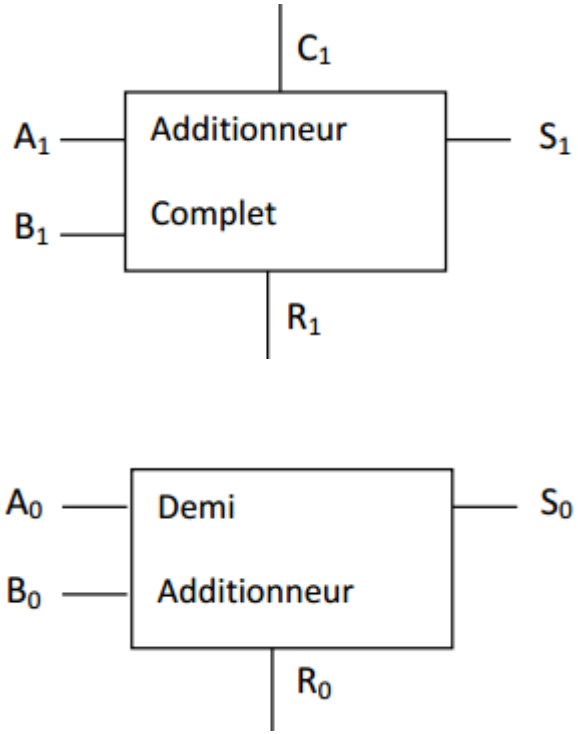
and the final carry is: **R3**

4-bit Adder circuit

To create a 4-bit adder, we must connect 4 **one-bit adders**:

a half adder and three full adders.

The connection is made by the carry "$C_i = R_{i-1}$"



This circuit adds two numbers :

A3 A2 A1 A0

et **B3 B2 B1 B0**

The result is : **S3 S2 S1 S0**

and the final carry is: **R3**

4-bit Adder circuit

The R_i is the output carry

The C_i is the input carry

$C_i = R_{i-1}$

$A_1+B_1+C_1 = S_1$ and Carry R_1 $(C_1 = R_0)$

$A_2+B_2+C_2 = S_2$ and Carry R_2 $(C_2 = R_1)$

$$\begin{array}{rcccc} & C_3 & C_2 & C_1 & \\ & A_3 & A_2 & A_1 & A_0 \\ + & B_3 & B_2 & B_1 & B_0 \\ \hline = & S_3 & S_2 & S_1 & S_0 \\ & R_3 & R_2 & R_1 & R_0 \end{array}$$

4-bit Adder circuit

To create a 4-bit adder, we must connect 4 **one-bit adders**:

a half adder and three full adders.

The connection is made by the carry "$C_i = R_{i-1}$"

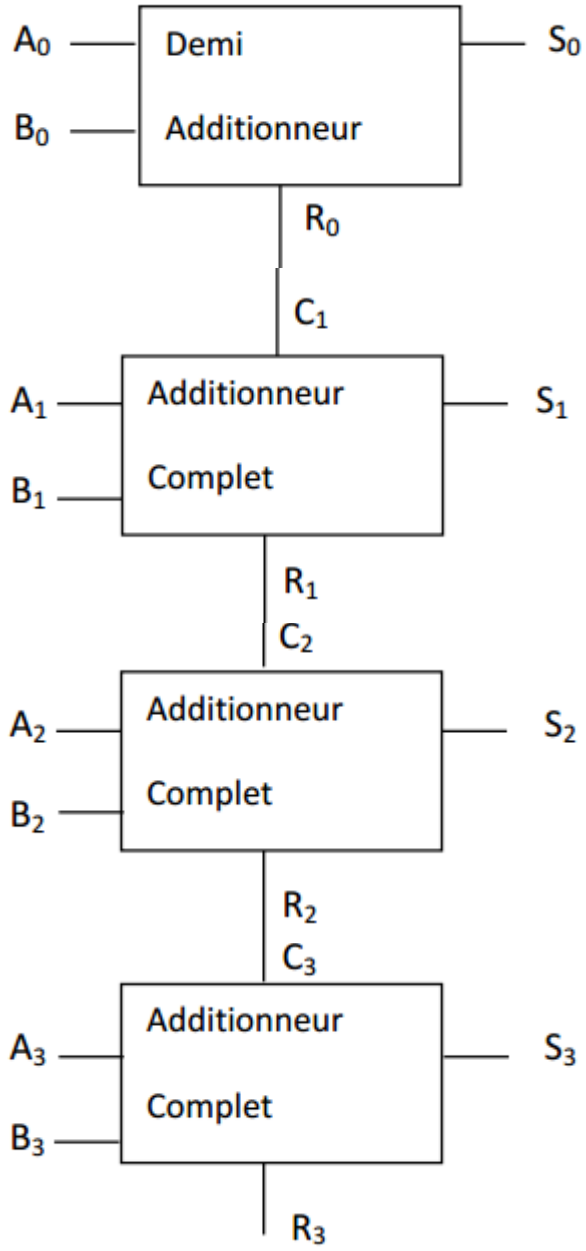
This circuit adds two numbers :

A3 A2 A1 A0

et **B3 B2 B1 B0**

The result is : **S3 S2 S1 S0**

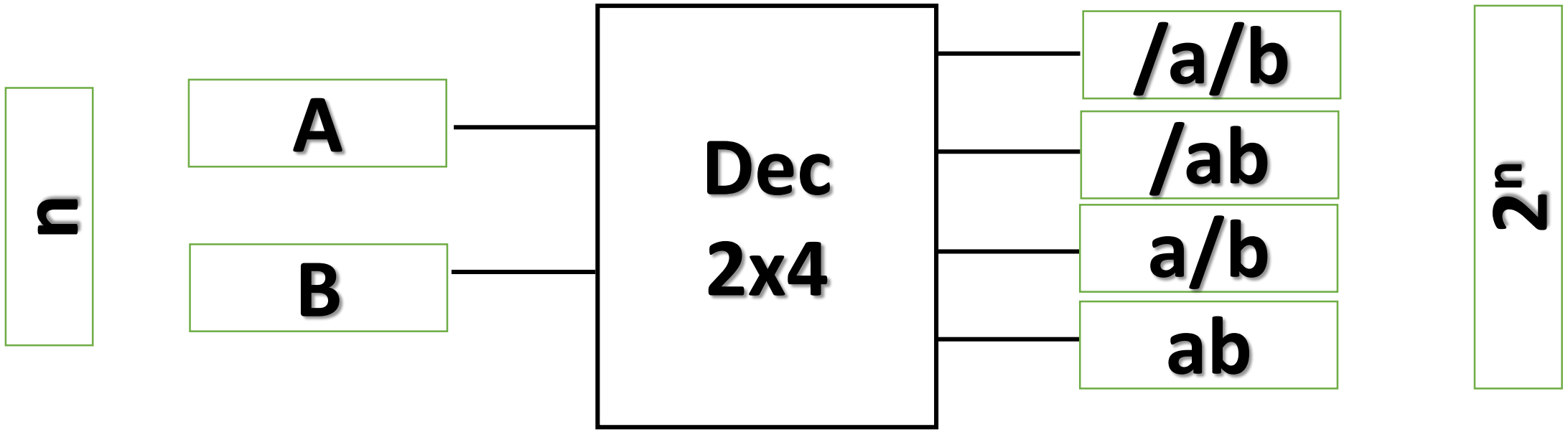
and the final carry is: **R3**



The decoder (DEC)

A decoder is a combinational circuit which has n inputs and 2n outputs of which only one is equal to 1

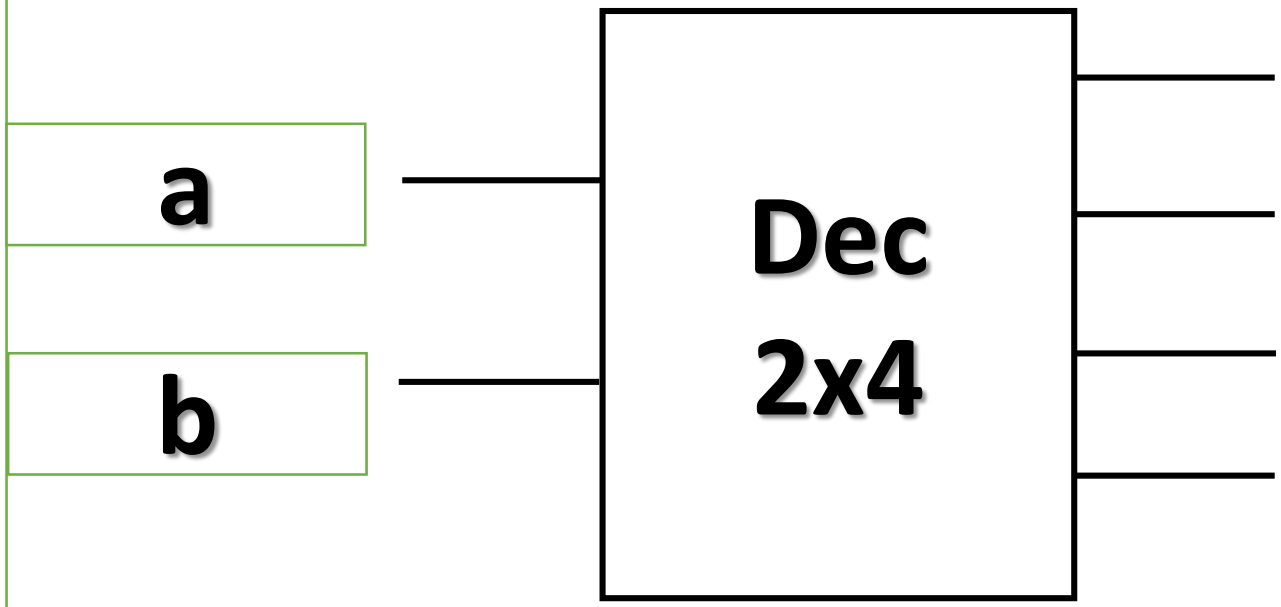
The following example represents a 2x4 decoder

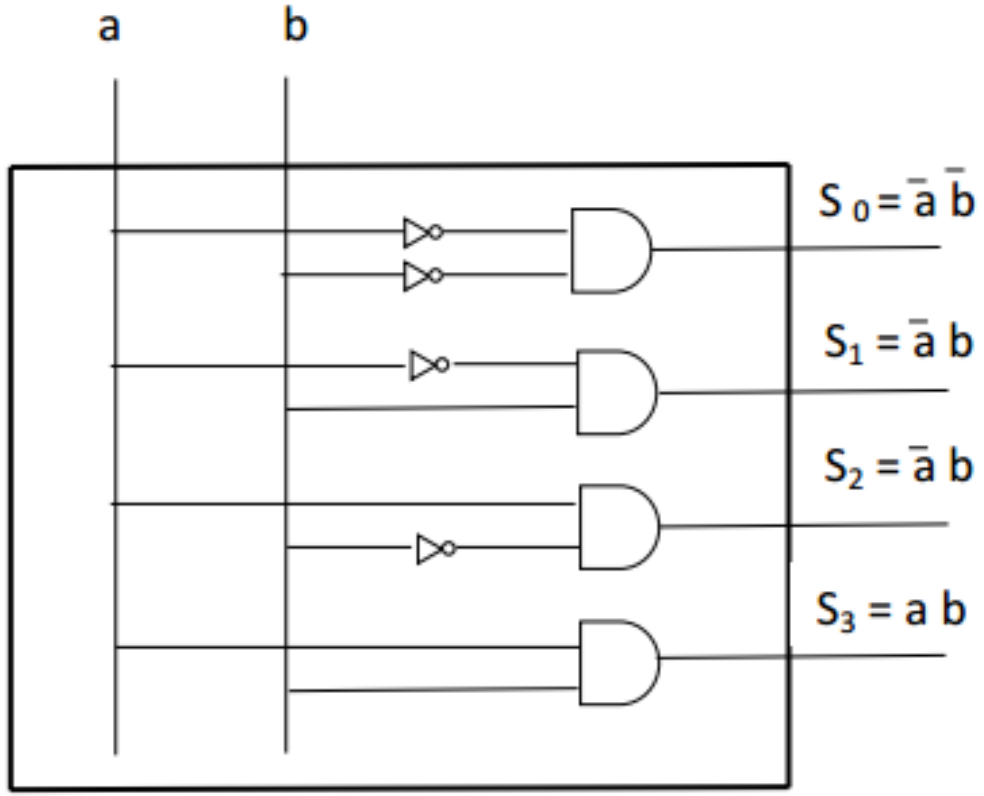


The decoder (DEC)

A decoder is a combinational circuit which has n inputs and 2n outputs of which only one is equal to 1

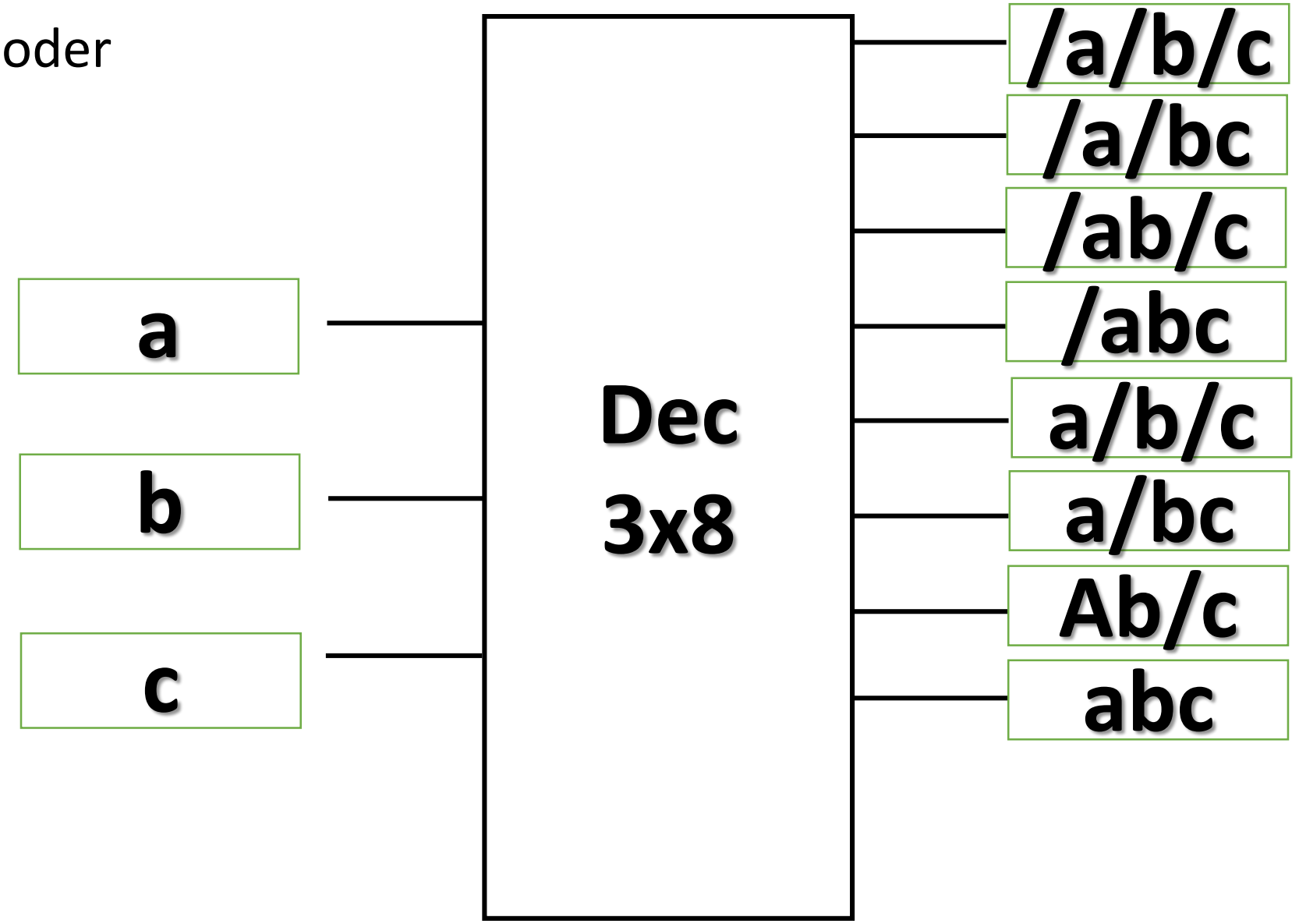
The following example represents a 2x4 decoder





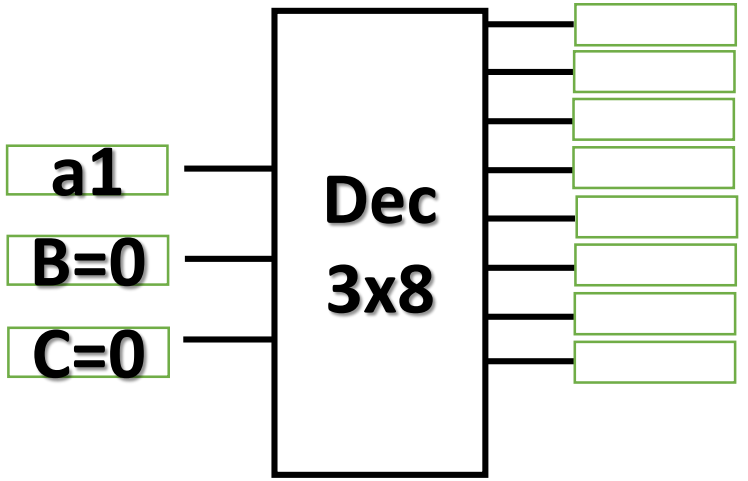
The decoder (DEC)

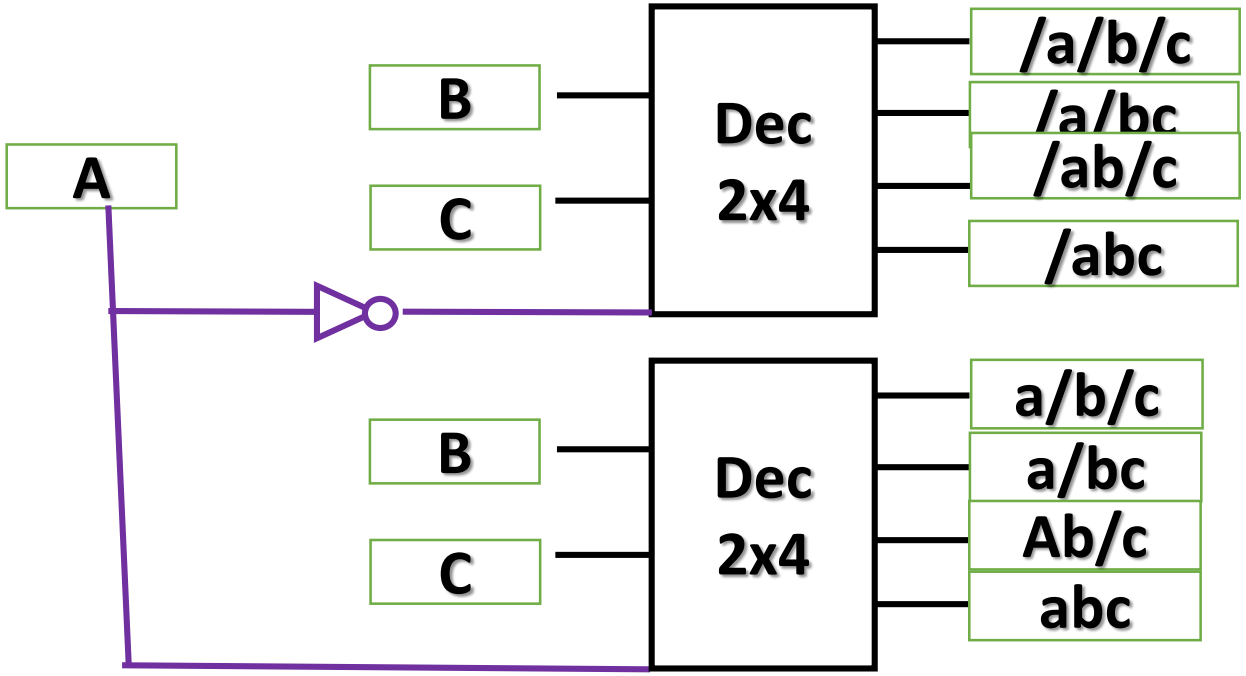
3x8 decoder



The decoder (DEC)

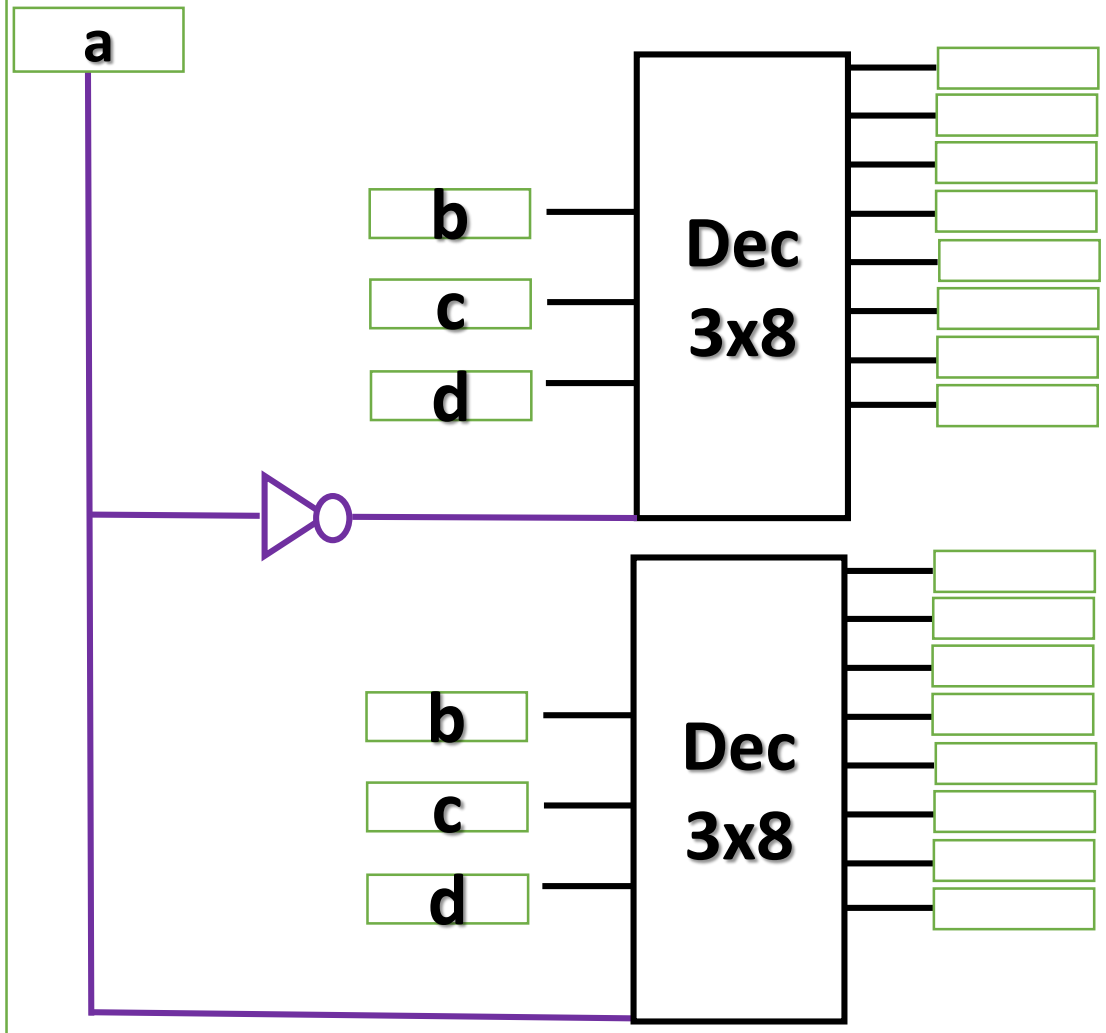
3x8 decoder using 2x4





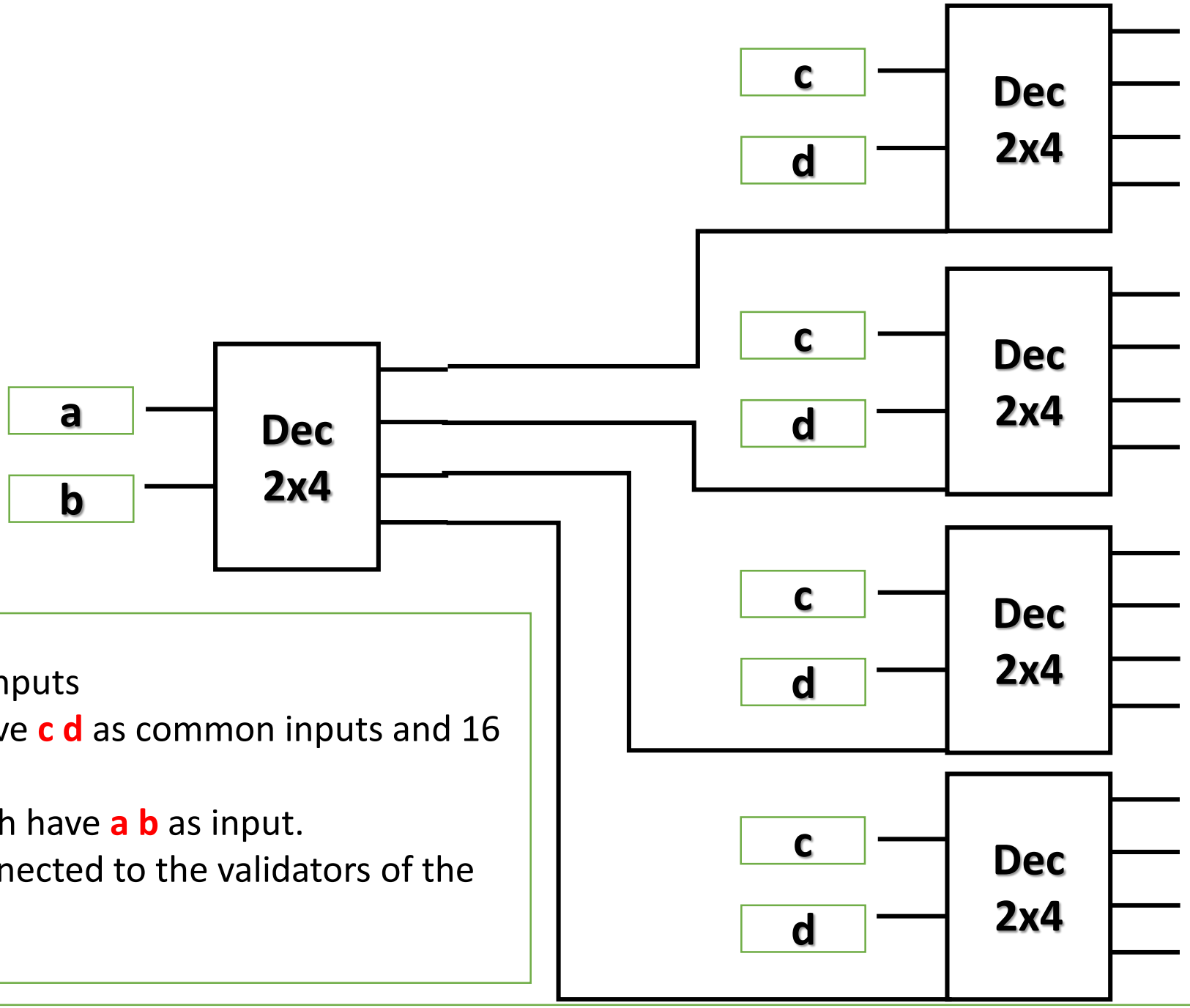
The decoder (DEC)

4x16 decoder using 3x8



The **DEC 4x16** will have **a b c d** as inputs
We will use **2 DEC** which have **b c d** as common inputs and 16 **2x8** outputs
a is used to enable one decoder at a time
$V1 = \bar{a} \ \ and \ \ V2 = a$
Si **a = 0** then **V1 = 1** the **first** DEC will be active
Si **a = 1** then **V2 = 1** the **second** DEC will be active
So **only one** output among the 16 will be equal to **1**
(That of validated DEC)

4x16 DEC using 2x4 DEC



The **DEC 4x16** will have **a b c d** as inputs
We will use **4 DEC of 2x4** which have **c d** as common inputs and 16 **4x4** outputs.
We will use also **1 DEC of 2x4** which have **a b** as input.
the outputs of this DEC will be connected to the validators of the other DECs (4 previous DECs).

Realize a function of n variables using a DEC p x 2^p

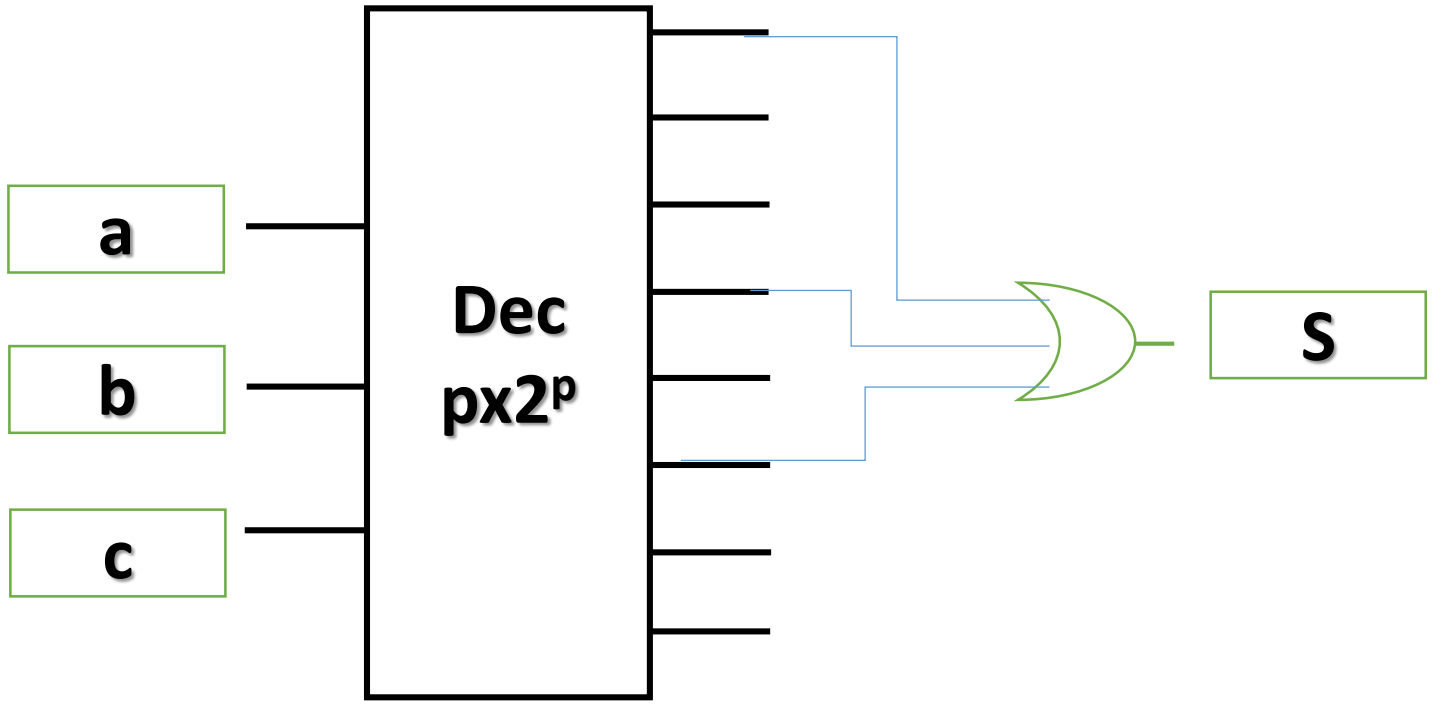
1) p = n

Each decoder output corresponds to a **min term**. We will make the logical sum **(OR)** of all the decoder outputs corresponding to the **min terms** for which the function is equal to 1

Example : n = 3 et P = 3

$$S(a, b, c) = \bar{a}\bar{b}\bar{c} + \bar{a}bc + a\bar{b}c$$

A	B	C	S
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



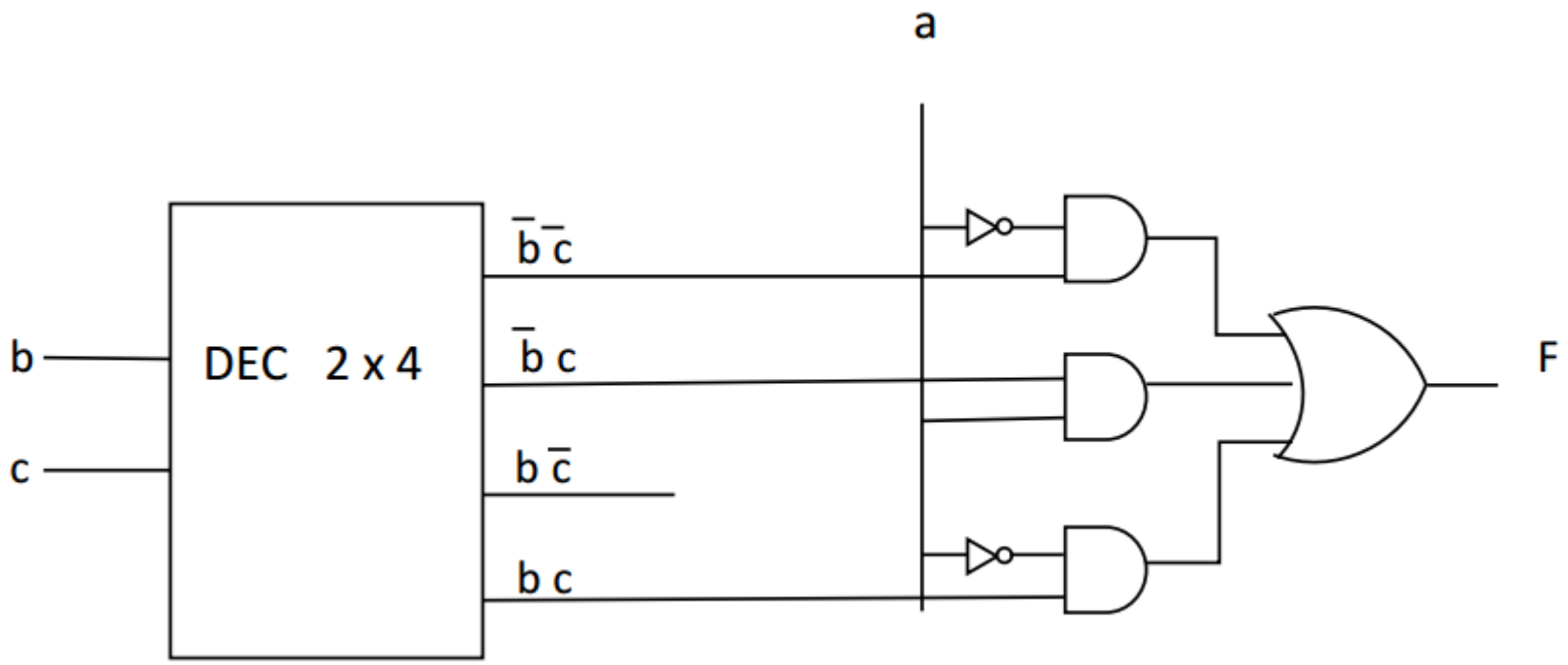
Realize a function of n variables using a DEC p x 2^p

2) p < n

P variables will be the **decoder inputs**. The remaining **(n-p)** variables will be **outside** the decoder. The function will be formed by the **combination** of the **decoder outputs** and **external variables**.

Example : **n = 3** et **P = 2**

F(abc) = $\bar{a}\bar{b}\bar{c} + \bar{a}bc + a\bar{b}c$



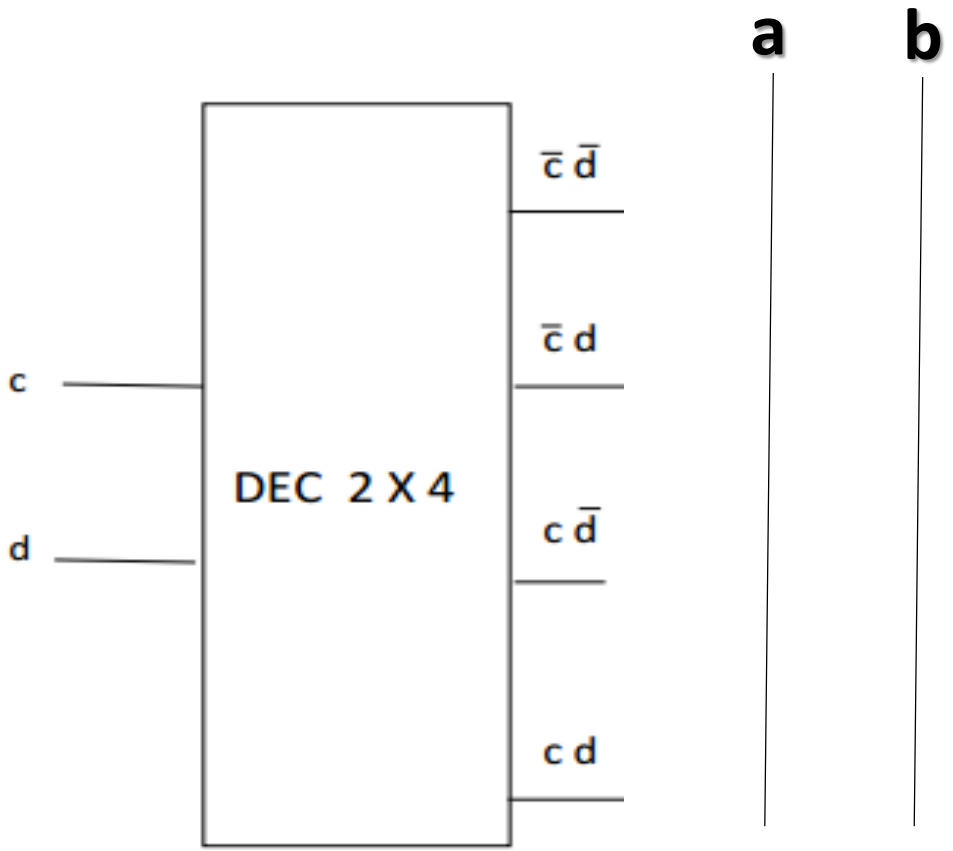
Realize a function of n variables using a DEC p x 2^p

Example 2 : **n = 4** et **p = 2**

$$F(a, b, c, d) = a\bar{b}cd + \bar{a}bd + a\bar{c}$$

If we put **a and b** outside the DEC, the **entries** of the DEC 2 X 4 will be **c and d**.

c and d are **inseparable**, It is therefore necessary to make **c and d** appear in each term.



Realize a function of n variables using a DEC p x 2^p

Exemple 2 : **n = 4** et **P = 2**

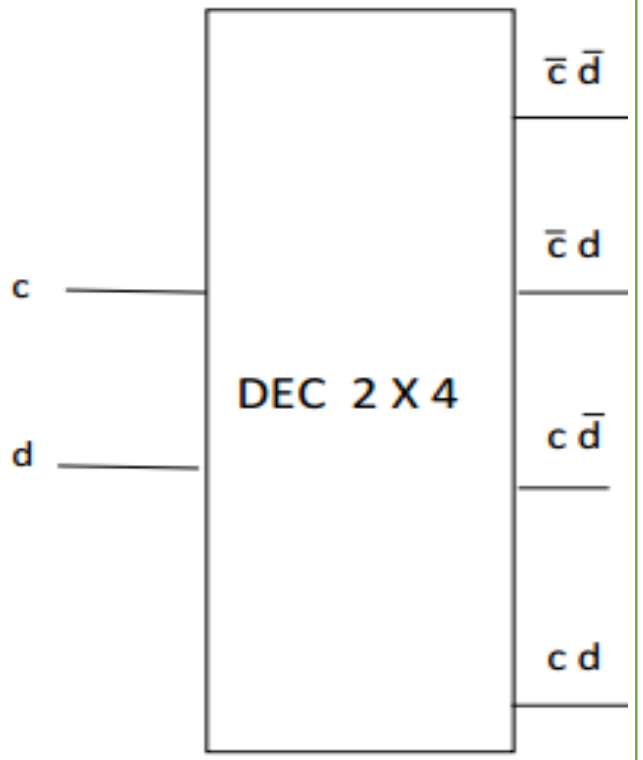
$$F(a,b,c,d) = a\bar{b}cd + \bar{a}bd + a\bar{c}$$

$$F(abc) = a\,\bar{b}\,c\,d + \bar{a}\,b\,d(c + \bar{c}) + a\,\bar{c}\,(d + \bar{d})$$

$$F(abc) = a\,\bar{b}\,c\,d + \bar{a}\,b\,c\,d + \bar{a}\,b\,\bar{c}\,d + a\,\bar{c}\,d + a\,\bar{c}\,\bar{d}$$

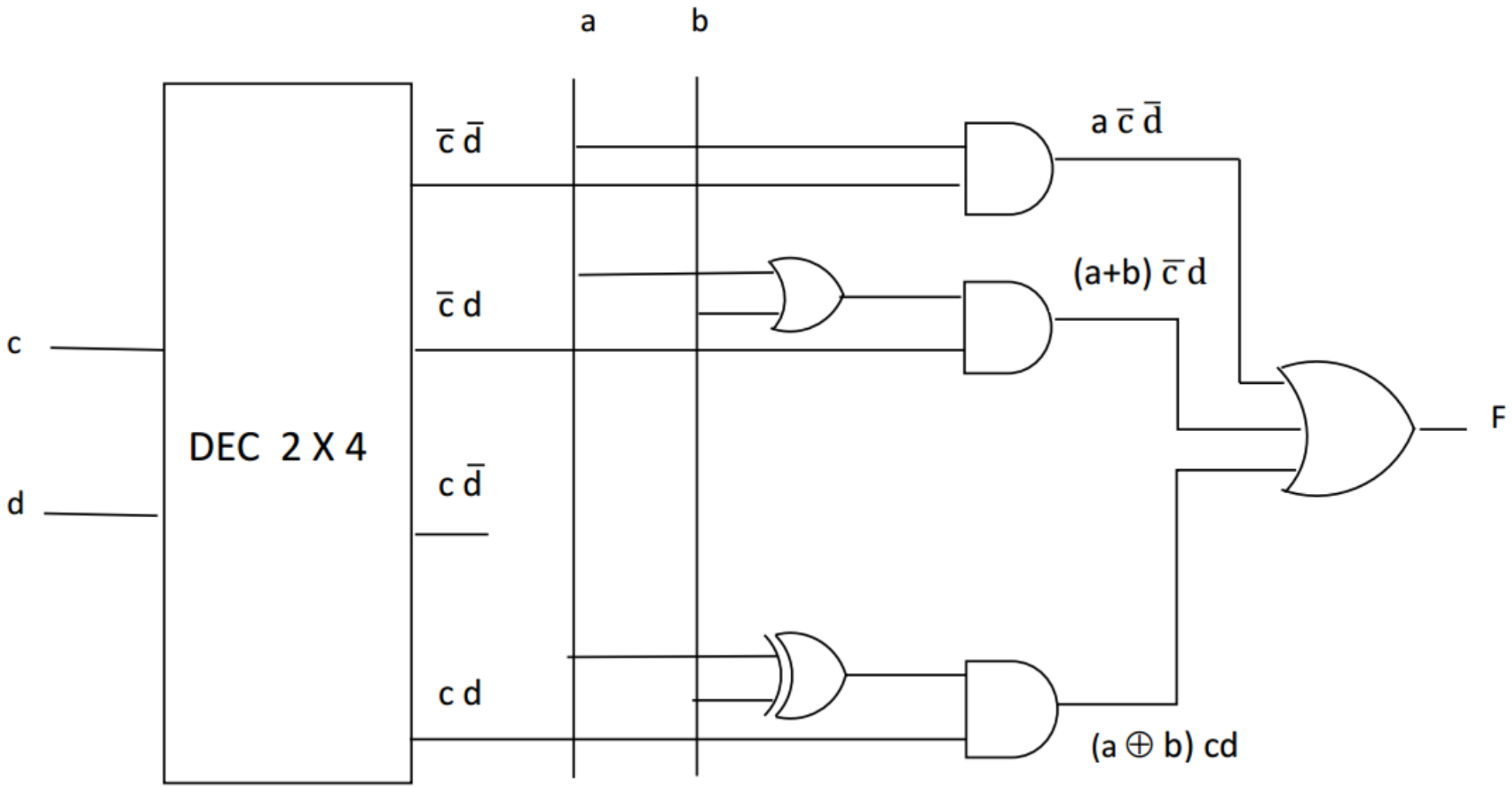
$$F(abc) = c\,d\,(a\,\bar{b} + \bar{a}\,b) + \bar{c}\,d\,\underbrace{(\bar{a}\,b + a)} + a\,\bar{c}\,\bar{d}$$

$$F(abc) = c\,d\,(a \oplus b) + \bar{c}\,d\,\underbrace{(a + b)} + a\,\bar{c}\,\bar{d}$$



Realize a function of n variables using a DEC p x 2^p

$$F(abc) = c\,d\,(a \oplus b) + \bar{c}\,d\,\underbrace{(a + b)} + a\,\bar{c}\,\bar{d}$$



Realize a function of n variables using a DEC p x 2^p

Example 3 : **n = 4 et p = 2**

If a term does not contain **c and d**. We only use external variables.

For example : $\mathbf{F(a, b, c, d) = ab + b\bar{c}d + \bar{a}c}$

The first term **(a b)** does not contain **c and d**, so we leave it.

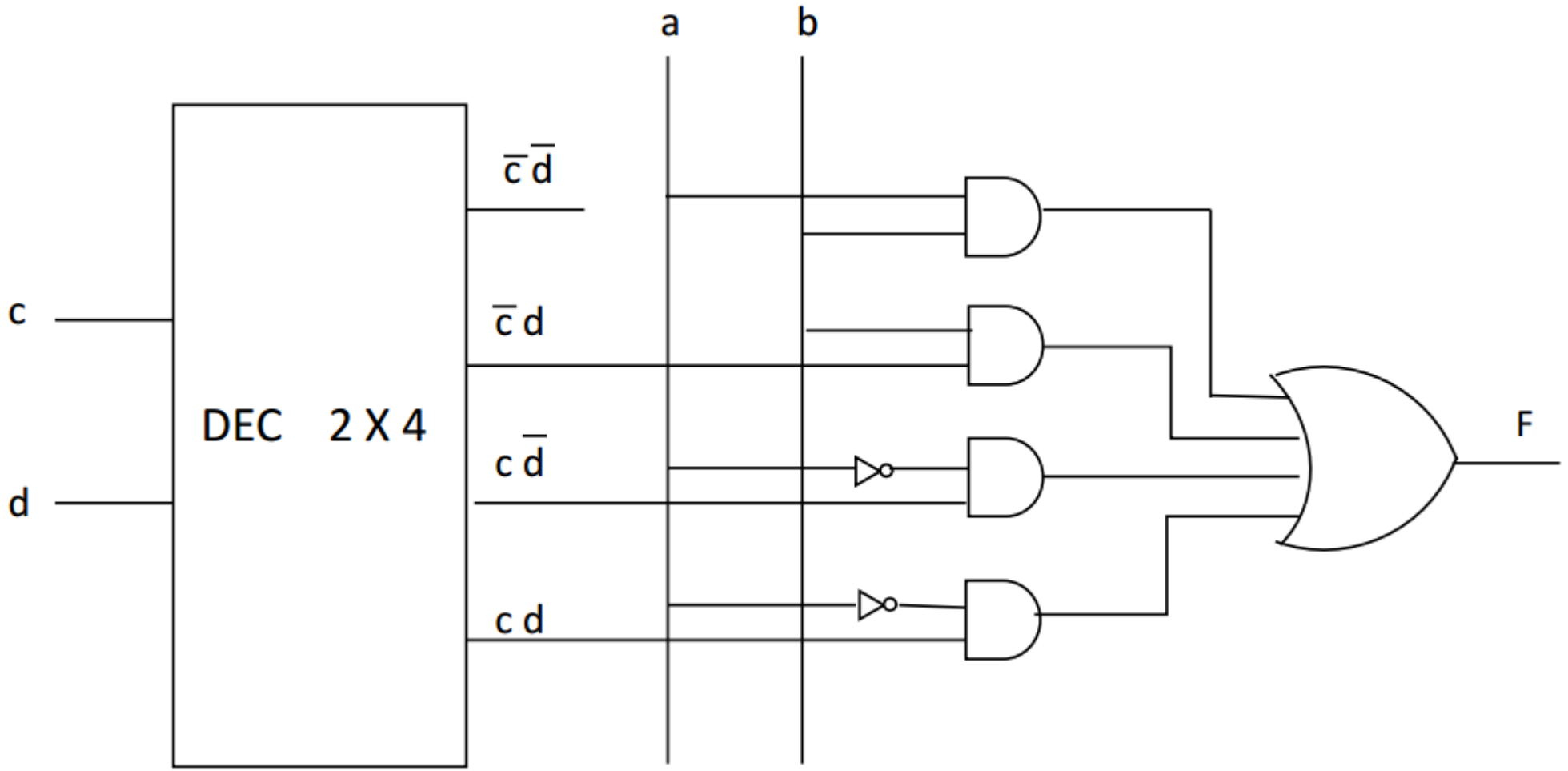
The second term **(a c)** contains **c** but not **d** so we transform it:

$\bar{a}c = \bar{a}c(\bar{d} + d)$

$\mathbf{F(a, b, c, d) = ab + b\bar{c}d + \bar{a}c\bar{d} + \bar{a}cd}$

Realize a function of n variables using a DEC p x 2^p

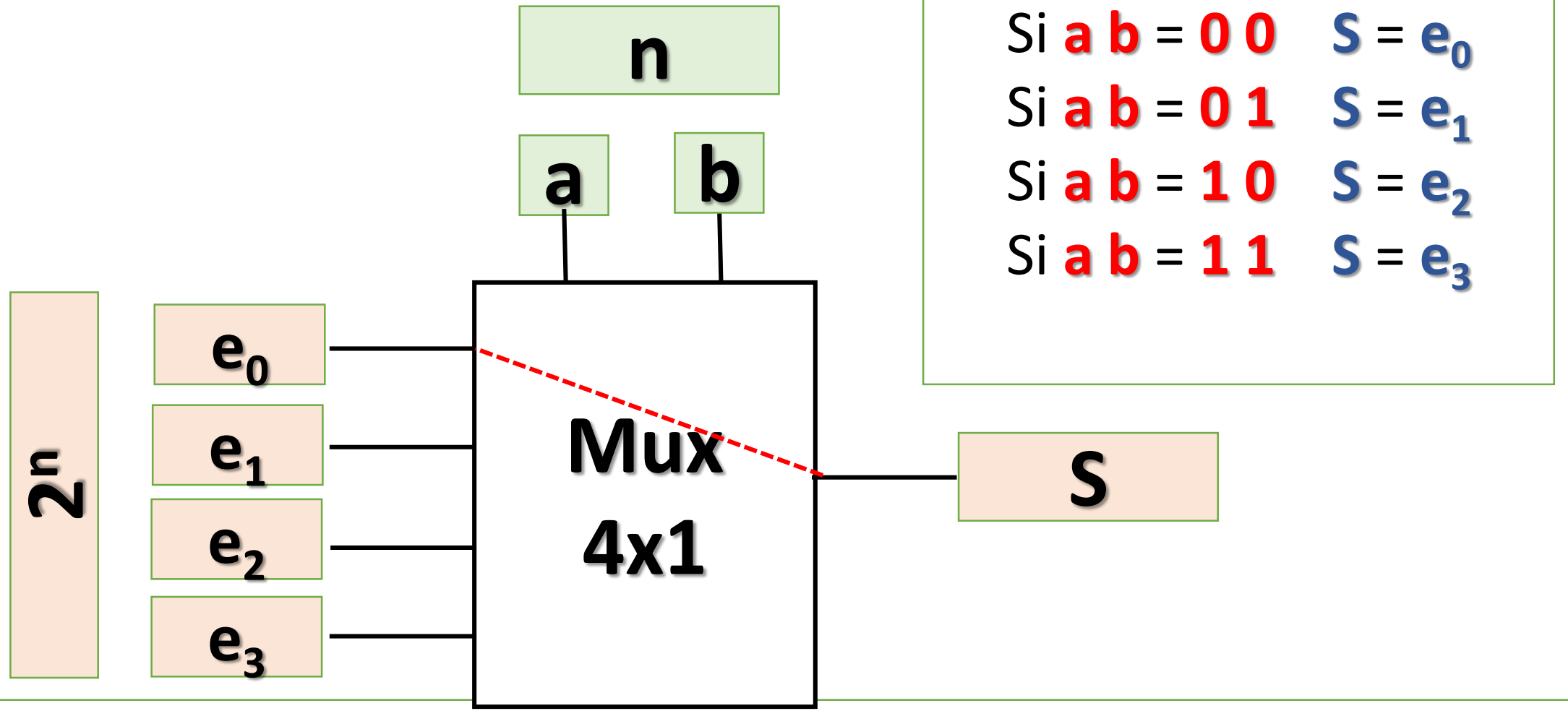
F(a b c d) = a b + b $\bar{c}$d + /a c



The Multiplexer (MUX)

A **multiplexer** is a combinational circuit that has **2n inputs**, **one output** and **n selection lines**.

The following example represents a **4x1** multiplexer

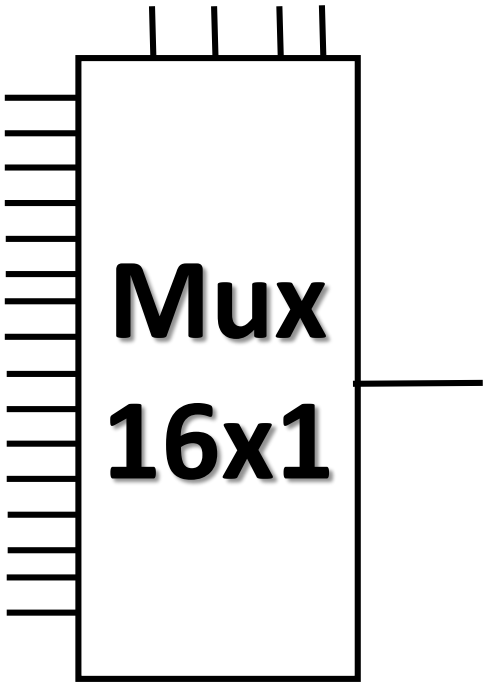


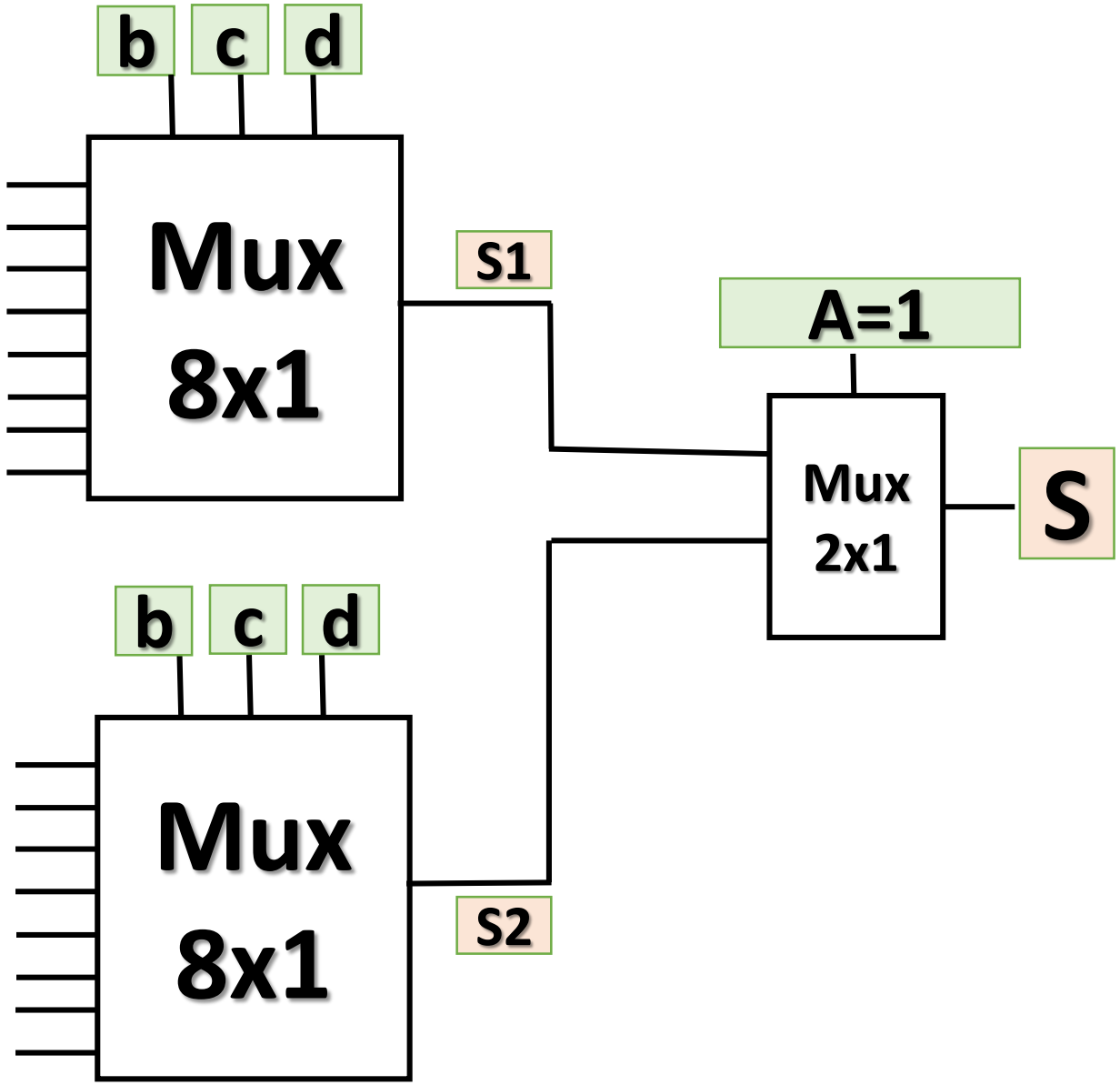
Realize a MUX 16x1(2^4x1) using a MUX 8x1(2^3x1)

The MUX 16x1 should have **16 inputs**, **one output** and **4 selection lines a b c d**.

We use **2 MUX 8x1**. each MUX has **one output** and **3 common selection lines b c d** and a **2x1 MUX** which has as selection line a

If **a = 0** then **S = S1**
If **a = 1** then **S = S2**





Realize a function of n variables using a MUX 2^px1

1/ p = n

Each MUX entry represents a value of the function

Example : n = 3 et P = 3

$$\mathbf{F(a,b,c)} = \bar{a}\,\bar{b}\,\bar{c} + \bar{a}\,b\,c + a\,\bar{b}\,c$$

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

Realize a function of n variables using a MUX 2^px1

P < n

Example : N = 4 et p = 2

F(abcd) = ad + bd + cd + abc

We take c d as selection lines and a b remain outside

F(abcd) = ad(c+/c) + bd(c+/c) + cd +

abc(d+$\bar{d}$)

F(abcd) = acd + a$\bar{c}$d + bcd + b$\bar{c}$d + cd + abcd + abc$\bar{c}$

F = cd(a+b+1+ab) + $\bar{c}$d(a+b) + c$\bar{d}$(ab)

F = cd + $\bar{c}$d(a+b) + c$\bar{d}$ (ab)