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 02

Dr. HASSANI DJAMEL

MACHINE STRUCTURE 02

SUMMARY

CHAPTER 01 : COMBINATORIAL LOGIC

Introduction	**1**
1.1. Arithmetic circuits	**1**
1.2. Addition of two binary numbers	**2**
1.2.1. Half adder	**2**
1.2.2. Full adder	**2**
1.3. Binary subtractor	**4**
1.4. Comparators	**6**
1.5. Decimal / bcd coding	**7**
1.6. BCD (8421) coding in exess 3 code	**8**
1.7. Multiplexers	**9**
1.7.1. Using multiplexers to implement logic functions	**11**
1.7.2. Association of multiplexers	**13**
1.8. Demultiplexer	**13**
1.8.1. Association of demultiplexers	**15**
1.9. Data transmission	**15**
1.9.1. Parallel-serial conversion	**15**
1.9.2. Series-parallel conversion	**16**
1.9.3. Serial data transmission	**16**

CHAPTER 02 : SEQUENTIAL LOGIC

Introduction	**17**
2.1. Asynchronous and synchronous sequential circuits	**17**
2.1.1. Asynchronous sequential circuit	**17**
2.1.2. Synchronous sequential circuit	**18**
2.1.3. Clock	**18**
2.1.4. Synchronization	**18**
2.2. Flip-Flops	**19**
2.2.1. RS flip-flop	**19**
2.2.2. Synchronous RS flip-flop	**21**
2.2.3. RS flip-flop application: bounce-free switch	**22**
2.2.4. Flip-flop with direct asynchronous input	**23**
2.2.5. RS Master - Slave flip-flop	**24**
2.2.6. T flip-flop	**25**
2.2.7. JK flip-flop	**26**
2.2.8. D flip-flop	**27**
2.3. Counters / down counters	**28**
2.3.1. Asynchronous counters	**28**
2.3.2. Propagation delay in asynchronous counters	**29**
2.3.3. Synchronous counters	**30**
2.3.4. Down counters asynchronous	**31**

2.3.5. Counters with modulo less than 2^n (incomplete cycle) **32**
2.3.6. BCD counters mounted in cascades **34**
2.4. Registers **35**
2.4.1. Characteristics of a register **35**
2.4.2. Memorization register **35**
2.4.3. Shift register **36**
2.4.4. Parallel input - parallel output shift register **36**
2.4.5. Serial input and serial output shift register **37**
2.4.5.1. Shift right **38**
2.4.5.2. Shift left **38**
2.4.6. Serial input - parallel output shift register **38**
2.4.7. Parallel input - serial output shift register **39**
2.4.8. Circular shift register **39**

CHAPTER 03 : LOGIC INTEGRATED CIRCUITS

3.1. Introduction **40**
3.2. Integrated circuit manufacturing process **40**
3.3. Housing and pinouts **41**
3.4. Identification of an integrated circuit **42**
3.5. Classification of integrated circuits **42**
3.5.1. Classification by field of use **42**
3.5.2. Classification by number of transistors per package **43**
3.6. Technology families **43**
3.6.1. The ttl family **45**
3.6.2. The cmos family **45**
3.7. Characteristic parameters **45**
3.7.1. Electrical parameters **45**
3.7.2. Dynamic parameters **47**
3.7.3. Comparison between ttl and cmos per gate **48**
3.8. Wiring and real realization of a logigram **49**
3.8.1. Input not connected **49**
3.8.1. Examples **50**
3.9. Basic logic gates **52**

Bibliography

CHAPTER 01

COMBINATORIAL LOGIC

CHAPTER 01

COMBINATORIAL LOGIC

INTRODUCTION

Combinational logic circuits are a fundamental element of digital electronics and computing. They are used to perform information processing operations without internal memory, meaning they produce outputs solely based on the current inputs, without regard to previous inputs or internal state. Here's a more detailed introduction:

Combinational logic circuits are composed of interconnected logic gates. These logic gates, such as AND, OR, NOT, XOR, NAND, NOR, etc., are electronic devices that take one or more binary inputs (0 or 1) and produce a binary output based on their specific logical function. For example, an AND gate produces an "1" output only when all its inputs are "1."

The key characteristic of combinational circuits is that they do not retain internal state or memory. Each change in inputs immediately results in a corresponding change in outputs, following the rules of Boolean logic. This means that combinational logic circuits are used for instantaneous operations, such as binary addition, multiplication, data selection, and more.

The applications of combinational logic circuits are vast and diverse, ranging from calculators and computers to automated control systems, telecommunications, and consumer electronics. They form the foundation of digital system design and are essential for information processing in the modern world.

In summary, combinational logic circuits are electronic components that perform data processing operations based solely on the current values of inputs, without retaining internal state. They are crucial for building digital systems and have a wide range of applications in electronics and computing.

1.1. ARITHMETIC CIRCUITS

An arithmetic circuit, also known as a combinational arithmetic circuit, is a type of combinational logic circuit designed specifically to perform arithmetic operations on binary numbers. These circuits are used to carry out operations such as addition, subtraction, multiplication, division, and other fundamental mathematical operations.

An arithmetic circuit typically consists of multiple combinational modules, such as adders, subtractors, multipliers, dividers, etc. These modules are interconnected in a way that executes the desired arithmetic operations based on the input signals. In this chapter we consider techniques that perform these operations with binary numbers.

1.2. ADDITION OF TWO BINARY NUMBERS

1.2.1. Half adder

Addition and subtraction are two basic arithmetic operations. Let's start with the addition of two binary numbers; subtraction will be studied in the next paragraph. As in decimal, we must therefore take into account a possible carry. In base 2, the addition of two bits is written as:

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

Table 1.1 : *Truth table of a half adder*

The logic function is:

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

The logic circuit is:

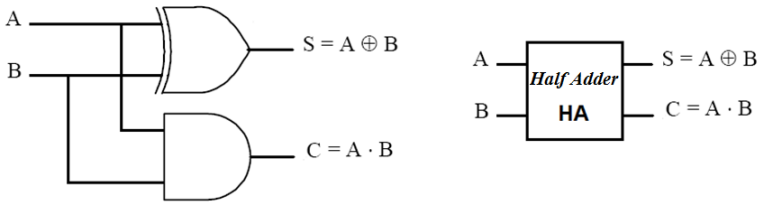


Figure 1.1: *half adder.*

This setup is called a half adder because it does not take into account any previous carry.

1.2.2. Full adder

If a previous carry is to be taken into account, the previous circuit must be completed. This is the case, for example, when an adder stage is to be set up to handle single-byte words, with parallel adders.

A_n	B_n	C_{n-1}	S_n	C_n
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 1.2 : *Truth table*

Karnaugh table for S_n and C_n :

S_n

$C_{n-1} \setminus A_nB_n$	00	01	11	10
0		1		1
1	1		1	

C_n

$C_{n-1} \setminus A_nB_n$	00	01	11	10
0			1	
1		1	1	1

Table 1.3 : *Karnaugh table*

$$C_n = A_nB_n + B_nC_{n-1} + A_nC_{n-1}$$

$$\boldsymbol{C_n = A_nB_n + C_{n-1}(A_n + B_n)}$$

$$S_n = \bar{A}_nB_n\bar{C}_{n-1} + A_n\bar{B}_n\bar{C}_{n-1} + \bar{A}_n\bar{B}_nC_{n-1} + A_nB_nC_{n-1}$$

$$S_n = C_{n-1}(\bar{A}_n\bar{B}_n + A_nB_n) + \bar{C}_{n-1}(\bar{A}_nB_n + A_n\bar{B}_n)$$

$$\boldsymbol{S_n = C_{n-1} \oplus (A_n \oplus B_n)}$$

with:

$$\bar{A}_n\bar{B}_n + A_nB_n = \overline{\bar{A}_nB_n + A_n\bar{B}_n} = \overline{A_n \oplus B_n}$$

The logic diagram or the implementation of the logic equations of C_n and S_n can be produced in several ways.

a. Using logic gates :

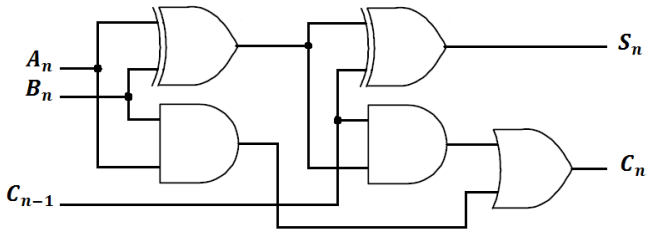
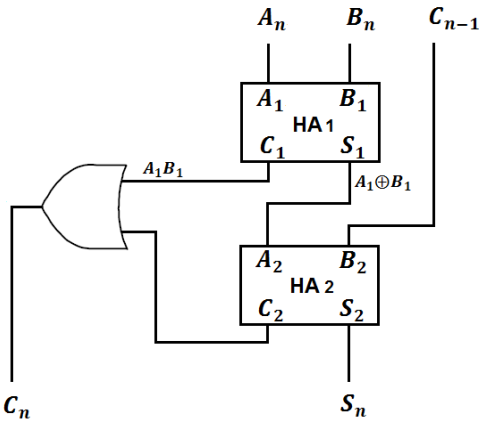


Figure 1.2: *Full adder with logic gates*

b. Using two half adders:



$$S_n = A_2 \oplus B_2$$
$$C_n = A_1B_1 + A_2B_2$$

with:

$$A_2 = A_1 \oplus B_1$$
$$B_2 = C_{n-1}$$
$$A_1 = A_n$$
$$B_1 = B_n$$

From where:

$$\boldsymbol{S_n = A_n \oplus B_n \oplus C_{n-1}}$$

$$\boldsymbol{C_n = A_nB_n + (A_n \oplus B_n)C_{n-1}}$$

Figure 1.3: *Full adder with half adder.*

c. Symbol :

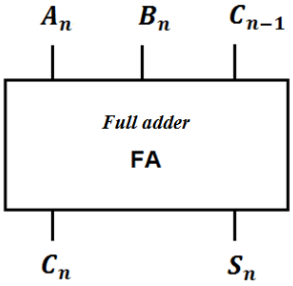


Figure 1.4 : *Full adder*

Two binary numbers can be added using several adders in parallel.

Example :

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

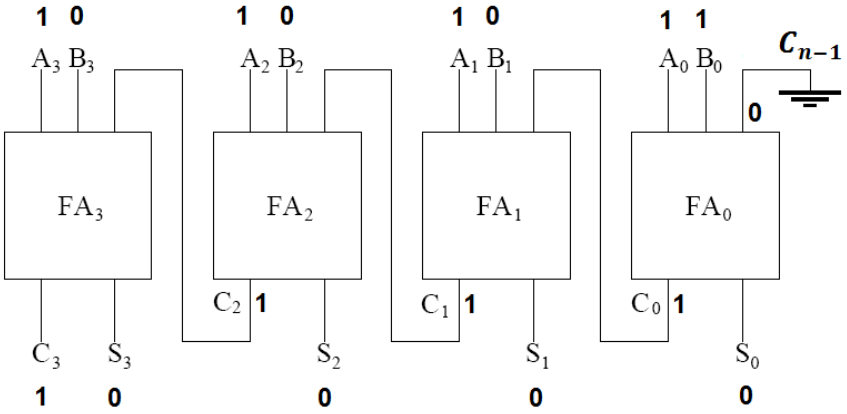


Figure 1.5 : *4-bit parallel adder*

1.3. BINARY SUBTRACTOR

The 2's complement transforms a binary subtraction into an addition. However, you could also subtract directly, using a half-subtractor.

A	*B*	*D*	*E*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 1.4 : *Truth table of a half-subtractor.*

The logic function is:

$$D = \bar{A}B + A\bar{B} = A \oplus B$$

$E = \bar{A}B$: Emprunt

The logic circuit is:

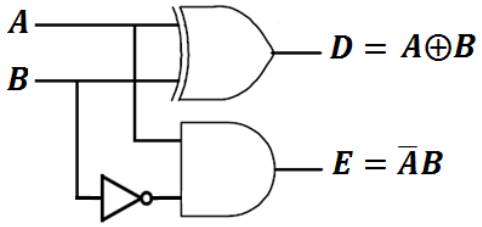


Figure 1.6: *Half-subtractor.*

The full subtractor requires an additional stage to take account of any previous borrowing.

The complete binary subtractor, taking into account any previous borrowing, consists of two half-subtractors, plus an OR gate.

A_n	B_n	E_{n-1}	D_n	E_n
0	0	0	0	0
0	1	0	1	1
1	0	0	1	0
1	1	0	0	0
0	0	1	1	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1

Table 1.5 : *Truth table*

Table de Karnaugh table for D_n and E_n :

D_n

$E_{n-1} \setminus A_nB_n$	00	01	11	10
0		1		1
1	1		1	

E_n

$E_{n-1} \setminus A_nB_n$	00	01	11	10
0		1		
1	1	1	1	

Table 1.6 : *Karnaugh table*

$$E_n = \bar{A}_n B_n + B_n E_{n-1} + \bar{A}_n C_{n-1}$$

$$\boldsymbol{E_n = \bar{A}_n B_n + E_{n-1}(\bar{A}_n + B_n)}$$

$$D_n = \bar{A}_n B_n \bar{E}_{n-1} + A_n \bar{B}_n \bar{E}_{n-1} + \bar{A}_n \bar{B}_n E_{n-1} + A_n B_n E_{n-1}$$

$$D_n = \bar{E}_{n-1}(\bar{A}_n B_n + A_n \bar{B}_n) + E_{n-1}(\bar{A}_n \bar{B}_n + A_n B_n)$$

$$\boldsymbol{D_n = E_{n-1} \oplus (A_n \oplus B_n)}$$

with :

$$\bar{A}_n \bar{B}_n + A_n B_n = \overline{\bar{A}_n B_n + A_n \bar{B}_n} = \overline{A_n \oplus B_n}$$

a. Using logic gates *:*

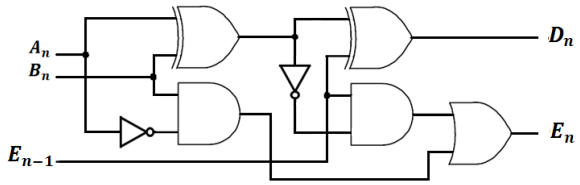


Figure 1.7 : *Full subtractor with logic gates*

b. Using two half subtractor*:*

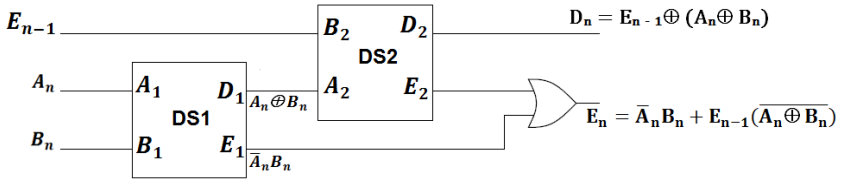


Figure 1.8 : *Full subtractor with half subtractors*

With :

$$\begin{aligned}\bar{A}B + E(\overline{A \oplus B}) &= \bar{A}B + E(\bar{A}\bar{B} + AB)\\ &= \bar{A}(B + BE) + ABE\\ &= \bar{A}(A + E) + ABE\\ &= \bar{A}B + E(\bar{A} + AB)\\ &= \bar{A}B + E(\bar{A} + B)\end{aligned}$$

1.4. COMPARATORS

At the very least, comparing two words means checking whether they are identical. If you only want to compare two bits, you can use the simple NEXOR gate: its output is 1 if the two inputs are similar, 0 otherwise.

Let's compare two 4-bit words, $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$. The circuit uses 4 NEXOR gates, whose outputs lead to an AND: the AND output provides a 1 when the two words are identical.

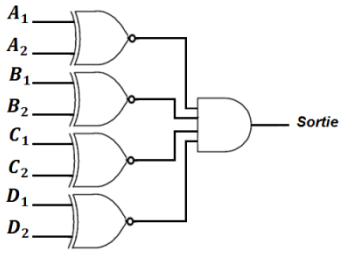


Figure 1.9 : *Comparator of two 4-bit words*

More elaborately, the comparator must indicate not only equality but, in the case of inequality, superiority or inferiority (A = B, A > B or A < B). Let's write the truth table corresponding to these three 2-bit comparison functions. The function C must be equal to 1 if and only if A > B, the function D equal to 1 if A < B and the function E equal to 1 if A = B. Which gives us:

A	*B*	*C (A > B)*	*D (A < B)*	*E (A = B)*
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Table 1.7: *Truth table*

We deduce the logical expressions of C, D and E :

$$C = A\bar{B}$$
$$D = \bar{A}B$$
$$E = \overline{A \oplus B} = \overline{A\bar{B} + \bar{A}B} = \overline{C + D}$$

The following diagram shows a logic block comparing two bits A and B:

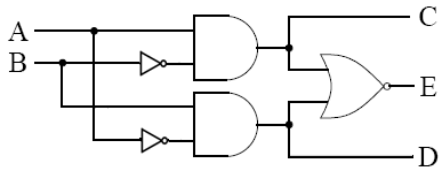


Figure 1.10 : *Two-bit comparator.*

In microcomputing, to compare two words, we generally proceed by subtraction: we subtract one word from the other (using the 2's complement). In fact, for a single bit, the difference A - B is zero and there is no borrowing if A = B, but the difference A - B = 1 is, without borrowing if A > B, with borrowing if A < B :

A	*B*	*D*	*E*	*Conclusion*
0	0	0	0	Egalité
0	1	1	1	A < B
1	0	1	0	A > B
1	1	0	0	Egalité

Table 1.8: *Truth table*

1.5. DECIMAL / BCD CODING

Let's look at the example of a BCD decimal decoder. The truth table of this decoder is very simple:

Decimal	*BCD*			
	B_3	B_2	B_1	B_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 1.9: *Truth table*

From this table, we deduce the necessary logic functions, which are obvious (we're talking about logical OR here!):

$$B_0 = 1 + 3 + 5 + 7 + 9$$
$$B_1 = 2 + 3 + 6 + 7$$
$$B_2 = 4 + 5 + 6 + 7$$
$$B_3 = 8 + 9$$

The following diagram shows the logic diagram of a BCD decimal decoder:

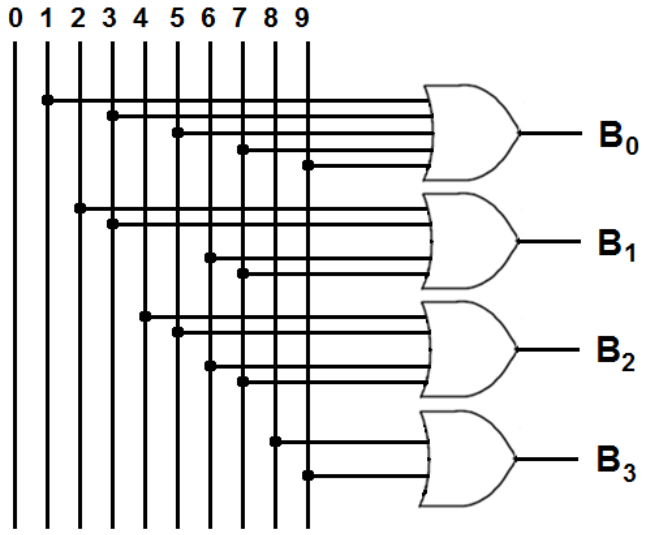


Figure 1.11 : *Decimal / BCD coding*

1.6. BCD (8421) CODING IN EXESS 3 CODE

Let's take a look at a DCB-decimal decoder. The truth table of this decoder is as follows:

BCD 8421				*EXESS 3*			
A	***B***	***C***	***D***	***W***	***X***	***Y***	***Z***
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Table 1.10: *Truth table*

Note: Binary combinations of ABCD that exceed1001 are considered indeterminate states.

Karnaugh's tables :

CD \ AB	00	01	11	10
00			Φ	1
01		1	Φ	1
11		1	Φ	Φ
10		1	Φ	Φ

$$W = A + BD + BC$$

CD \ AB	00	01	11	10
00		1	Φ	
01	1		Φ	1
11	1		Φ	Φ
10	1		Φ	Φ

$$X = B\overline{C}\,\overline{D} + \overline{B}D + \overline{B}C$$

CD \ AB	00	01	11	10
00	1	1	Φ	1
01			Φ	
11	1	1	Φ	Φ
10			Φ	Φ

$$Y = \overline{C}\,\overline{D} + CD = \overline{C \oplus B}$$

CD \ AB	00	01	11	10
00	1	1	Φ	1
01			Φ	
11			Φ	Φ
10	1	1	Φ	Φ

$$Z = \overline{D}$$

Table 1.12 : *Karnaugh's tables*

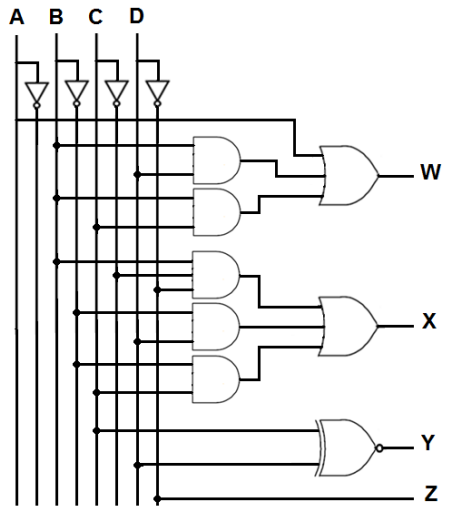


Figure 1.13 : *BCD / Exess 3 encoder*

1.7. MULTIPLEXERS

Multiplexing consists in selecting a channel for the information to be transmitted, when multiple channels are available, and connecting it to a single output line.

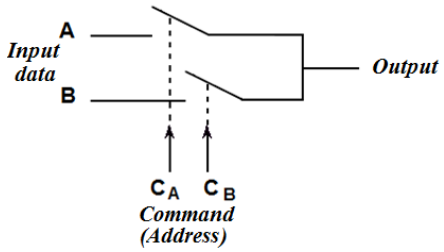


Figure 1.14 : *Multiplexer*

It can be seen that the command calls upon the input controls which are responsible for selecting the input data. The following figure shows the multiplexer diagrams:

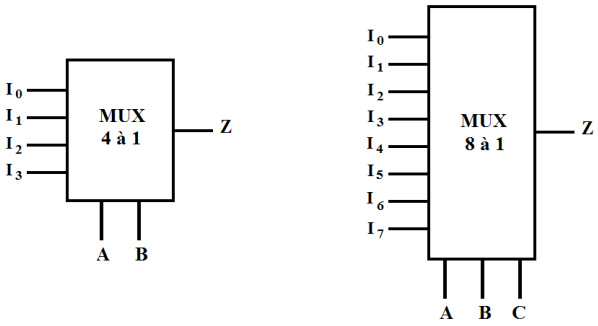


Figure 1.15 : *Multiplexer diagram*

- ***4 to 1 multiplexer (MUX 4 to 1)***

The logic function is:

$$Z = \bar{A}\bar{B}I_0 + \bar{A}BI_1 + A\bar{B}I_2 + ABI_3$$

Si	*AB = 00*	*la sortie*	$Z = I_0$
Si	*AB = 01*	*la sortie*	$Z = I_1$
Si	*AB = 10*	*la sortie*	$Z = I_2$
Si	*AB = 11*	*la sortie*	$Z = I_3$

The logic circuit is:

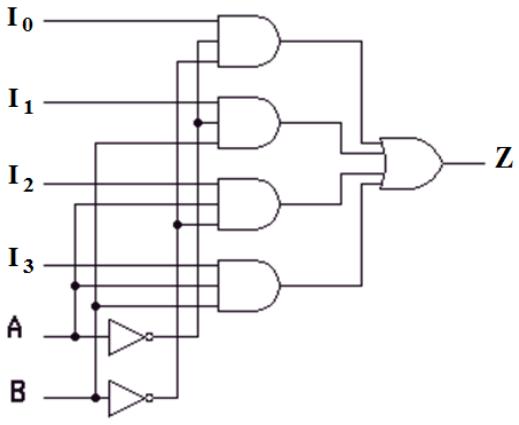


Figure 1.16 : *Logic circuit of a 4 to 1 multiplexer*

- ***8 to 1 multiplexer (MUX 8 to 1)***

The logic function is:

$$Z = \bar{A}\bar{B}\bar{C}I_0 + \bar{A}\bar{B}CI_1 + \bar{A}B\bar{C}I_2 + \bar{A}BCI_3 + A\bar{B}\bar{C}I_4 + A\bar{B}CI_5 + AB\bar{C}I_6 + ABCI_7$$

Note: If the control inputs ABC = 011, the output is Z = I_3. The other outputs are selected in the same way.

1.7.1. Using multiplexers to implement logic functions

Multiplexers can be used to implement logic functions. For example, a 4-to-1 multiplexer can perform a three-variable function without adding logic gates.

Example :

Let be the following logic function:

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

For example, we take a and b as control variables and c as input:

$$f(a,b,c) = \bar{a}\bar{b} + ac$$
$$f(a,b,c) = \bar{a}\bar{b} + ac(b + \bar{b})$$
$$f(a,b,c) = \bar{a}\bar{b}(1) + abc + a\bar{b}c + \bar{a}b(0)$$

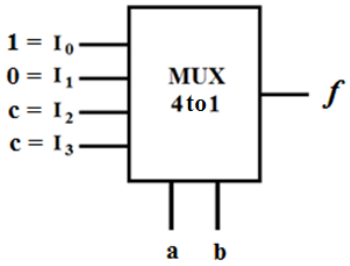


Figure 1.17 : *MUX 4 to 1 for a function of three variables*

An 8 to 1 multiplexer can be used to realize a four (04) variable function without gate addition. Three of the variables can be used as control inputs, while the remaining variable is used as input data.

Example:

Given a function L(ABCD) with four (04) variables, implement in the Karnaugh table below.

CD \ AB	00	01	11	10
00	1		1	1
01	1			
11	1	1		1
10		1	1	

Table 1.12 : *Karnaugh table*

Implement this function using A, B and C as control inputs and D as input to MUX 8 to 1.

We simplify only the variable used as input:

$$L = \bar{A}\bar{B}\bar{C}(1) + \bar{A}BC(1) + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{B}CD + ABC\bar{D} + \bar{A}B\bar{C}(0)$$
$$0 \quad 3 \quad 6 \quad 4 \quad 1 \quad 5 \quad 7 \quad 2$$

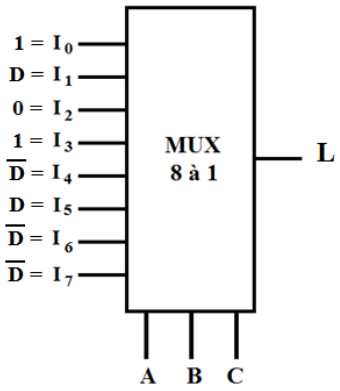


Figure 1.18 : *MUX 8 to 1 for four-variable functions*

The function described in the previous paragraph can be implemented using a 4 to 1 MUX by adding logic gates (using two variables as control inputs and two variables as input data).

CD \ AB	**00**	**01**	**11**	**10**
00	1	0	1	1
01	1	0	0	0
11	1	1	0	1
10	0	1	1	0
	I_0	I_1	I_3	I_2

Table 1.13 : *Karnaugh table*

C \D	***0***	***1***
0	1	1
1	0	1

AB = 00

$I_0 = \bar{C} + D$

C \D	***0***	***1***
0	0	0
1	1	1

AB = 01

$I_1 = C$

C \D	***0***	***1***
0	1	0
1	0	1

AB = 10

$I_2 = \bar{C}\bar{D} + CD = \bar{C} \oplus D$

C \D	***0***	***1***
0	1	0
1	1	0

AB = 11

$I_3 = \bar{D}$

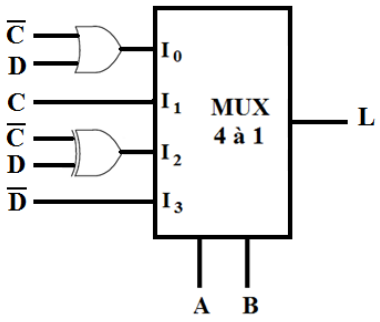


Figure 1.19 : *MUX 4 to1 for four-variable functions*

1.7.2. Association of multiplexers

Integrated circuits are available for 2, 4, 8 or 16 input multiplexers. To create higher-order multiplexers, it may be necessary to cascade several multiplexers. The following figure shows how a 32-input multiplexer can be created from four 8-input multiplexers and a 4-input multiplexer.

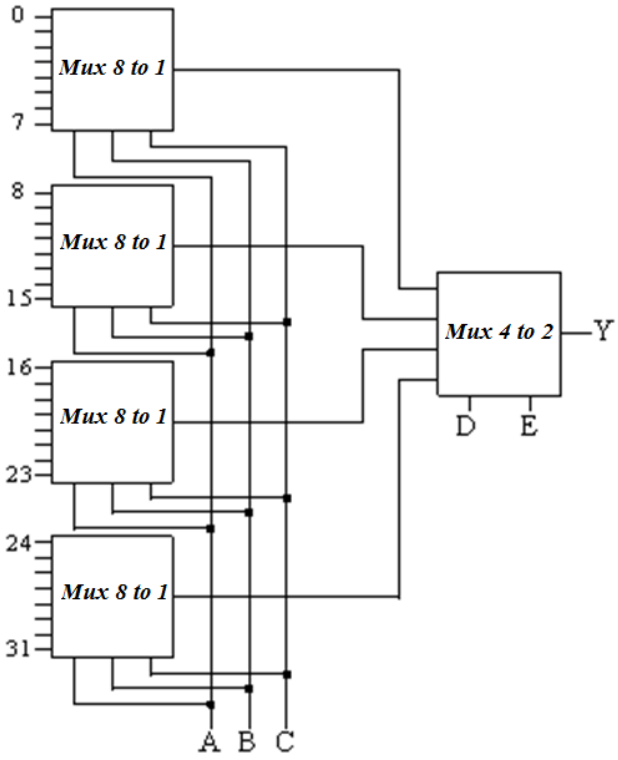


Figure 1.20 : *Mux 32 to 1 by association of multiplexers*

1.8. DEMULTIPLEXER

A demultiplexer consists in routing a single data input to 2^n outputs, in other words, it plays the opposite role to a multiplexer, allowing information to be routed to one of the outputs according to the values of the control inputs.

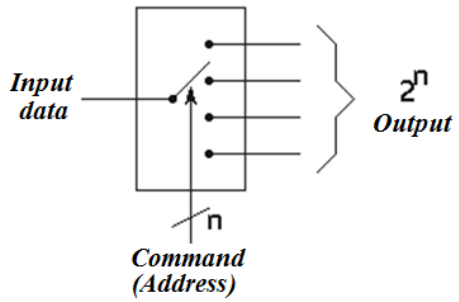


Figure 1.21 : *Démultiplexer*

To be able to select this output you also need addressing lines: the code carried by these lines (address) identifies the output line to use. Consider a demultiplexer with four output lines ($Y_0Y_1Y_2Y_3$). Two address lines (AB) are required. The truth table of this circuit is given by the following table:

B	A	Y0	Y1	Y2	Y3	Produit
0	0	D	0	0	0	$\bar{A}\bar{B}D$
0	1	0	D	0	0	$A\bar{B}D$
1	0	0	0	D	0	$\bar{A}BD$
1	1	0	0	0	D	ABD
0	0	0	0	0	0	
0	1	0	0	0	0	
1	0	0	0	0	0	
1	1	0	0	0	0	

Table 1.14: *Truth table*

From this table we deduce the following flowchart:

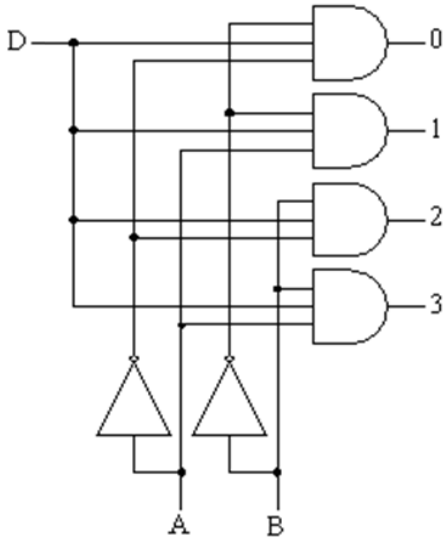


Figure 1.22 : *Logic circuit of a 4 to 1 demultiplexer*

1.8.1. Association of demultiplexers

As with multiplexers, several demultiplexers can be cascaded to form a higher-order demultiplexer. The following figure shows how a 32-output demultiplexer can be created from 4 x 8-output demultiplexers and 1 x 4-output demultiplexer.

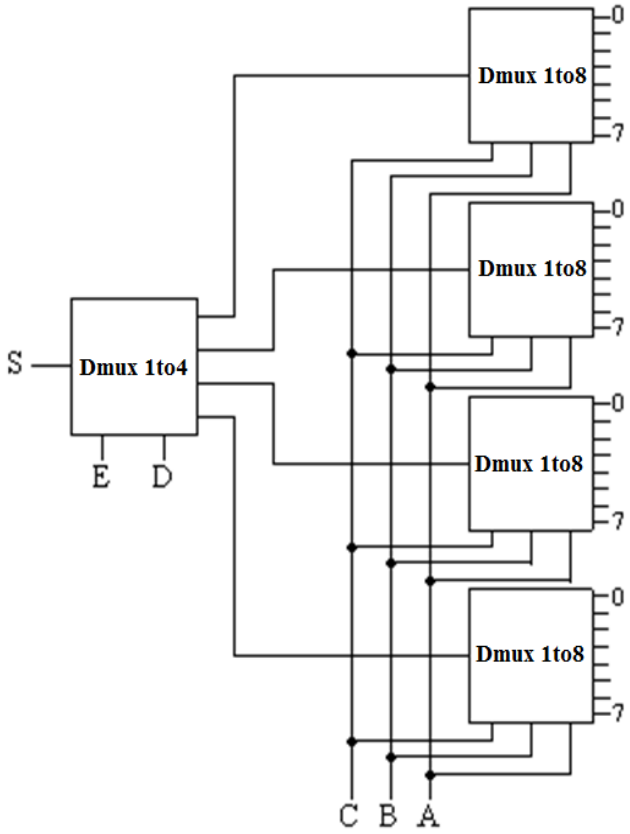


Figure 1.23 : *Mux 32 to 1 by association of demultiplexers*

1.9. DATA TRANSMISSION

1.9.1. Parallel-serial conversion

Consider a 4-bit word (X_0 X_1 X_2 X_3) present in parallel on the inputs of a multiplexer. Assuming that address lines A and B are connected to the outputs of a counter of period T, we'll get the following truth table as a function of time:

t	B	A	Y
T	0	0	X_0
2T	0	1	X_1
3T	1	0	X_2
4T	1	1	X_3
5T	0	0	X_0

Table 1.15: *Truth table of a parallel-serial conversion*

Bits X0, X1, X2 and X3 are found in time series on the multiplexer's Y output.

1.9.2. Series-parallel conversion

Consider a 4-bit word (X0 X1 X2 X3) connected in series at the input of a demultiplexer. Assuming that address lines A and B are connected to the outputs of a counter of period T, we'll have the following truth table as a function of time:

t	Input	address		Output			
		B	A	Y_0	Y_1	Y_2	Y_3
T	$X_0X_1X_2X_3$	0	0	X_0	0	0	0
2T	$X_0X_1X_2X_3$	0	1	X_0	X_1	0	0
3T	$X_0X_1X_2X_3$	1	0	X_0	X_1	X_2	0
4T	$X_0X_1X_2X_3$	1	1	X_0	X_1	X_2	X_3
5T	$X_0X_1X_2X_3$	0	0	X_0	0	0	0

Table 1.16: *Truth table of a series -parallel conversion*

Bits X0, X1, X2 and X3 are found in parallel on the demultiplexer's Y outputs.

1.9.3. Serial data transmission

The combination of a multiplexer and a demultiplexer makes it easy to transmit multiplexed data on a single line:

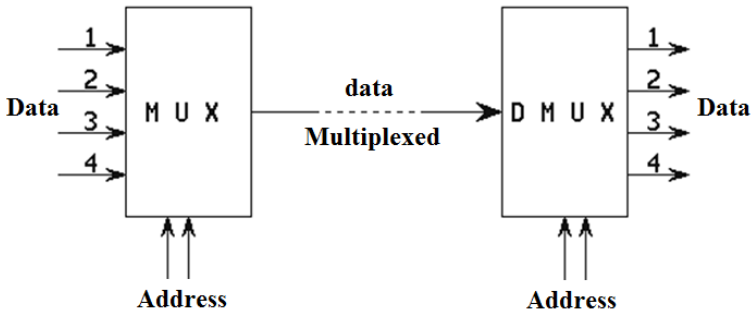


Figure 1.24 : *Serial data transmission*

CHAPTER 02

SEQUENTIAL LOGIC

CHAPTER 02

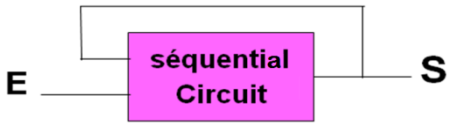
SEQUENTIAL LOGIC

INTRODUCTION

Combinatorial circuits were studied in the previous chapter: each state of the input variables corresponds to a single state of the output variables, and vice versa. With sequential circuits, this is no longer the case: one state of the input variables can correspond to several different states of the output variables, because the circuit remembers its previous states. Complex sequential circuits are built using combinational circuits and elementary sequential circuits.

2.1. ASYNCHRONOUS AND SYNCHRONOUS SEQUENTIAL CIRCUITS

In sequential circuits, there is feedback: output signals depend not only on inputs, but also on the previous state of the system. The circuit remembers inputs and previous states: it has a memory of the past.



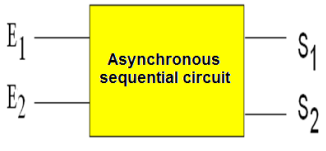
$$S_{t+1} = f(E, S_t)$$

Figure 2.1: *Sequential circuit.*

2.1.1. Asynchronous sequential circuit

An asynchronous sequential circuit is one whose outputs depend on the change of state of the inputs, taking into account the previous state. The asynchronous circuit has no clock as input variable, and the output is modified as soon as there is a change in the state of the inputs, independently of the clock signal.

Example: Assume a circuit that changes state at every transition $0 \rightarrow 1$



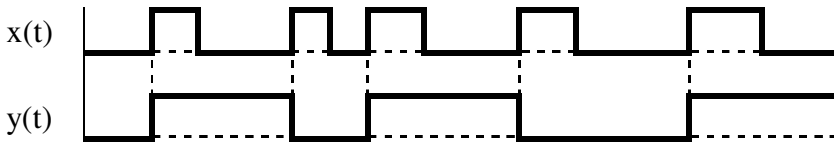


Figure 2.2: *Chronogram of an asynchronous circuit*

2.1.2. Synchronous sequential circuit

In a synchronous sequential system, for the output to respond to the input, a periodic clock signal must be associated with the input. Inputs will only be enabled if the clock signal is "1". In synchronous circuits, the output only changes after a clock signal. Synchronous circuits are simpler to synthesize and analyze.

Example:

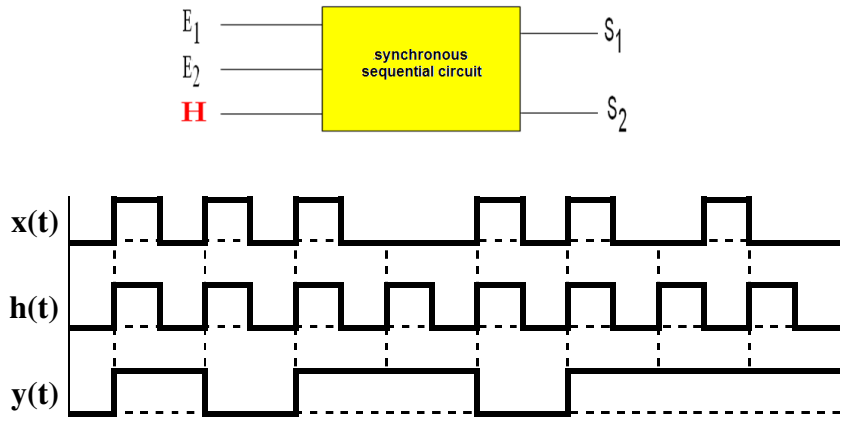


Figure 2.3: *Chronogram of a synchronous system*

2.1.3. Clock

A clock is a logic variable that passes indefinitely and regularly from a low level to a high level (succession of 0s and 1s). This variable is often used as an input for synchronous sequential circuits. The clock is denoted by h or Clk.

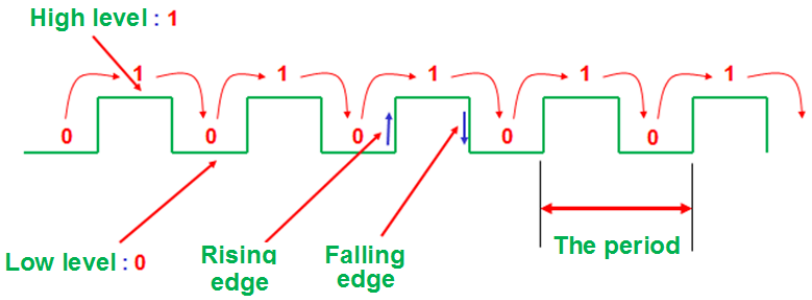


Figure 2.4: *Clock signal*

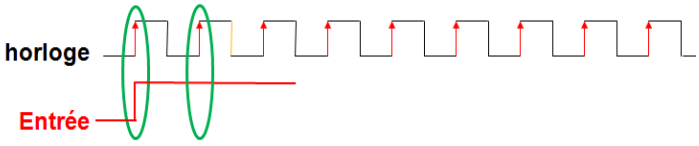
2.1.4. Synchronization

The synchronization clock signal, often called the clock signal or simply "clock", is a crucial element in electronic and digital systems. It takes two main forms: the rising edge and the falling edge.

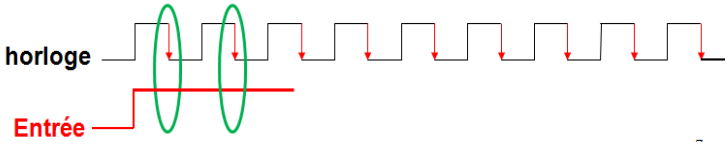
When a clock signal is toggled on the rising edge, it changes from low (0) to high (1). On the falling edge, on the other hand, the signal changes from high (1) to low (0). These rising-edge and falling-edge transitions are used as time marks to synchronize electronic operations in a system. For example, in a processor, instructions can be executed on the rising or falling edge of the clock, thus guaranteeing synchronized task execution.

The choice between rising and falling edges depends on system design and synchronization requirements. They are essential for maintaining order and consistency in electronic operations, and contribute to the stability and reliability of digital systems.

Synchronization on rising edge



Synchronization on falling edge



Synchronization on one level

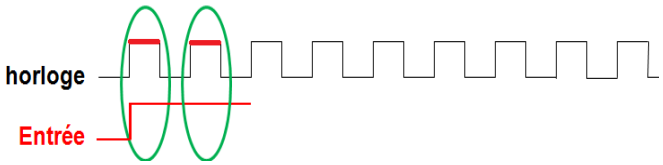


Figure 2.5: *Synchronization clock signals*

2.2. FLIP-FLOPS

Flip-flops are circuits with two stable states. Once a state has been acquired, the circuit locks on to this state, which it then retains until another command is received.

There are four basic types of flip-flop: RS, T, JK and D, with the complementary "master-slave" formula. Flip-flop applications include all sequential logic setups, such as counters and registers....

2.2.1. RS flip-flop

The simplest flip-flop structure is the RS flip-flop, these two letters coming from "Set" and "Reset" for set to "1" and set to "0". The inputs are R and S, and the outputs are Q and $\bar{Q}$.

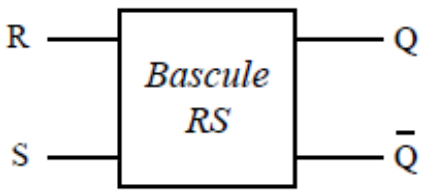


Figure 2.6: *Symbol of the RS flip-flop*

R	***S***	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	x
1	1	1	x

Truth table condensed

R	***S***	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	x

x : impossible state

Table 2.1: *Truth table*

Karnaugh's table

Q_n\ RS	00	01	11	10
0	0	1	x	0
1	1	1	x	0

$$Q^+ = S + \overline{R}.Q \qquad (1)$$

$$\overline{Q^+} = R + \overline{Q}\,\overline{S} \qquad (2)$$

RS flip-flop logic diagram:

a - With NOR gates

$$(1) \rightarrow Q^+ = S + \overline{R}.Q$$

$$\overline{Q^+} = \overline{S + \overline{R + \overline{Q}}}$$

$$(2) \rightarrow \overline{Q^+} = R + \overline{Q}\,\overline{S}$$

$$Q^+ = \overline{\overline{Q^+}} = \overline{R + \overline{Q}\,\overline{S}} = \overline{R + \overline{Q + S}}$$

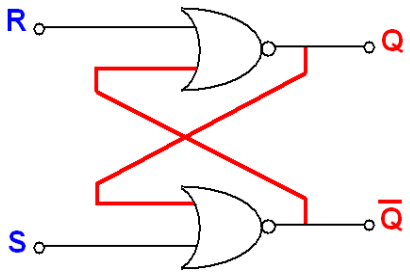


Figure 2.7: *RS flip-flop with NOR gates.*

b - With NAND gates

$$(1) \rightarrow Q^+ = S + \overline{R}.Q$$
$$Q^+ = \overline{\overline{Q^+}} = \overline{\overline{S}.Q.\overline{R}}$$

$$(2) \rightarrow \overline{Q^+} = R + \overline{Q}\,\overline{S}$$
$$\overline{Q^+} = \overline{\overline{Q^+}} = \overline{\overline{R}.\overline{Q}.\overline{S}}$$

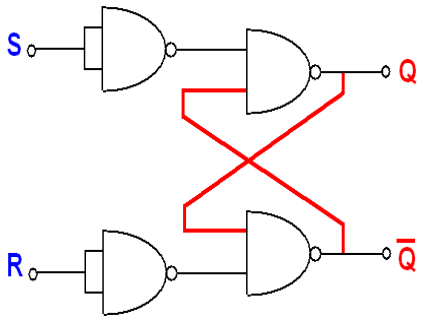


Figure 2.8 : *RS flip-flop with NAND gates.*

2.2.2. Synchronous RS flip-flop

In an asynchronous flip-flop, commands applied to the inputs immediately trigger the corresponding change of state. In a synchronous flip-flop, the execution of the command only occurs with the clock pulse.

R	S	H	Q^+	
Φ	Φ	0	Q	*Clearly, the flip-flop can only change state if the clock input is high.*
0	1	1	1	
1	0	1	0	
1	1	1	x	

Table 2.2 : *Truth table.*

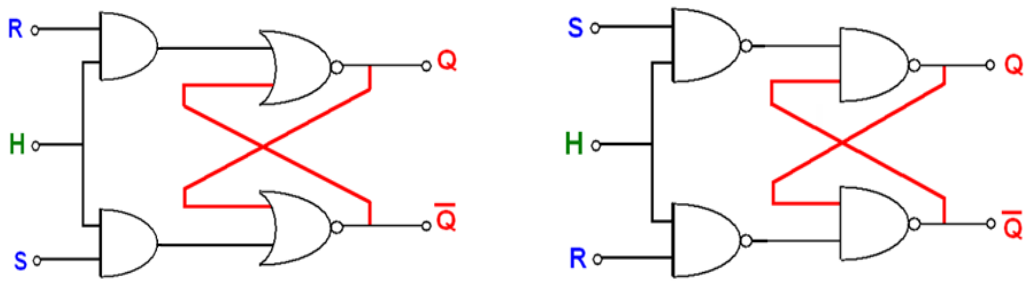


Figure 2.9: *Internal structure of a synchronous RS flip-flop*

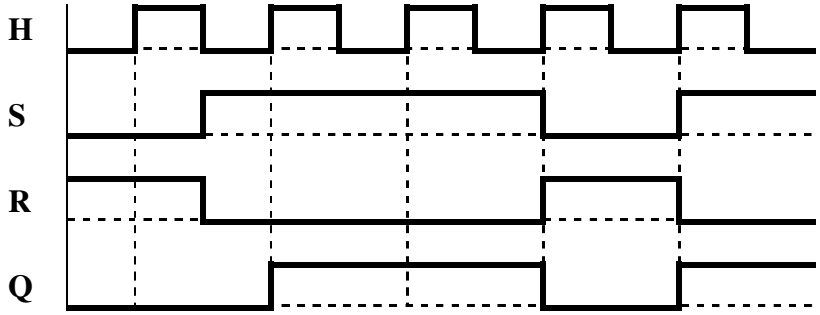


Figure 2.10: *Chronogram of a synchronous RS flip-flop*

2.2.3. RS flip-flop application: bounce-free switch

A very useful application for the RS rocker is the bounce-free switch. It is virtually impossible to make a mechanical switch without bouncing. This is illustrated in the following figure:

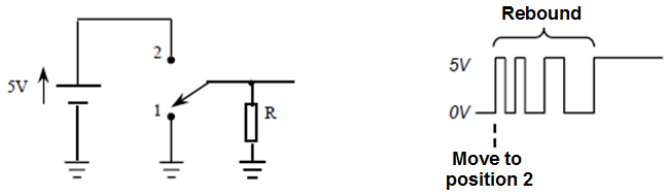


Figure 2.11: *Rebound phenomenon*

The effects of bouncing that occur when the contacts of a switch or relay are closed can be eliminated by using an RS flip-flop, as shown in the following diagram:

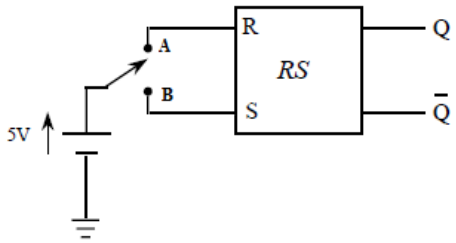


Figure 2.12 : *Debounce Circuit*

The anti-bounce circuit is an interesting application of the RS flip-flop in the case of push-buttons: the contact is normally in position A, and is used to apply 5V (logic high state) to input B, from which the push-button is in the open position. The duration of the pulse is equal to the time the operator keeps the pushbutton open. Once the button is released, the internal mechanism returns the contact to position A. The problem encountered is that the contact bounces back when it reaches position B at the start of the pulse and input A at the end of the pulse, leading to undesirable pulses that can distort the changeover in a logic system. The following figure shows typical pulses that can occur during an operating cycle.

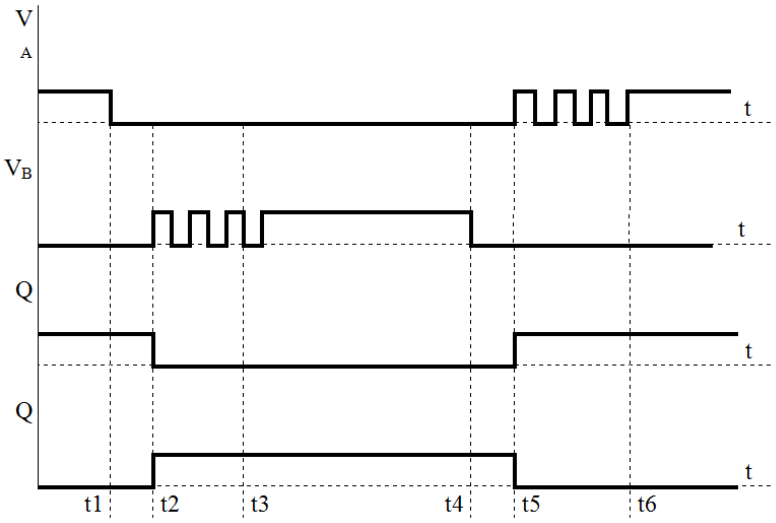


Figure 2.13 : *Operating cycle*

1. At the start of the process the contact is in position A such that $V_A = 5V$ and $V_B = 0V$ so S = 1 and R = 0 according to the RS flip-flop truth table: $Q = 1$ and $\overline{Q} = 0$;
2. At time t1, the operator pushes the button, the contact moves from position A to B. As soon as the contact leaves position A, the voltage $V_A = 0V$ and $V_B = 0V$ until instant t_2 when the contact reaches position B, i.e. $V_A = 0V$ and $V_B = 5V$. In the interval $[t_1\ t_2]$, $S = R = 0$, so the flip-flop retains the previous state $Q = 1$;
3. At time t_2, $R = 1$ and $S = 0$ therefore $Q = 0$;
4. Between t_2 and t_3, the contact will bounce several times to position B. During the bounce we have either $R = 1$ and $S = 0$ or $R = S = 0$, so the output will remain $Q = 0$;
5. At time t_4, the operator releases the pushbutton and the contact leaves position B;
6. Between t_4 and t_5, $R = S = 0$ so there is conservation of state $Q = 0$;
7. At time t_5, $S = 1$ and $R = 0$ therefore $Q = 1$;
8. Between t_5 and t_6, we have bounces at position A, i.e. $S = 1$ and $R = 0$ or $S = R = 0$ so the last state is preserved $\overline{Q} = 0$.

Note: For the flip-flop to change from a high state (1) to a low state (0), the push-button must remain in position B for longer than the propagation time through the two flip-flop gates.

2.2.4. Flip-flop with direct asynchronous input

In practice, it is often desirable to be able to set the flip-flop to "1" ($Q=1$) or "0" ($Q=0$) independently of the S, R and H inputs. The following figure illustrates a circuit in which the "Preset" (Pr) and "Clear" (Cl) signals have priority over all inputs.

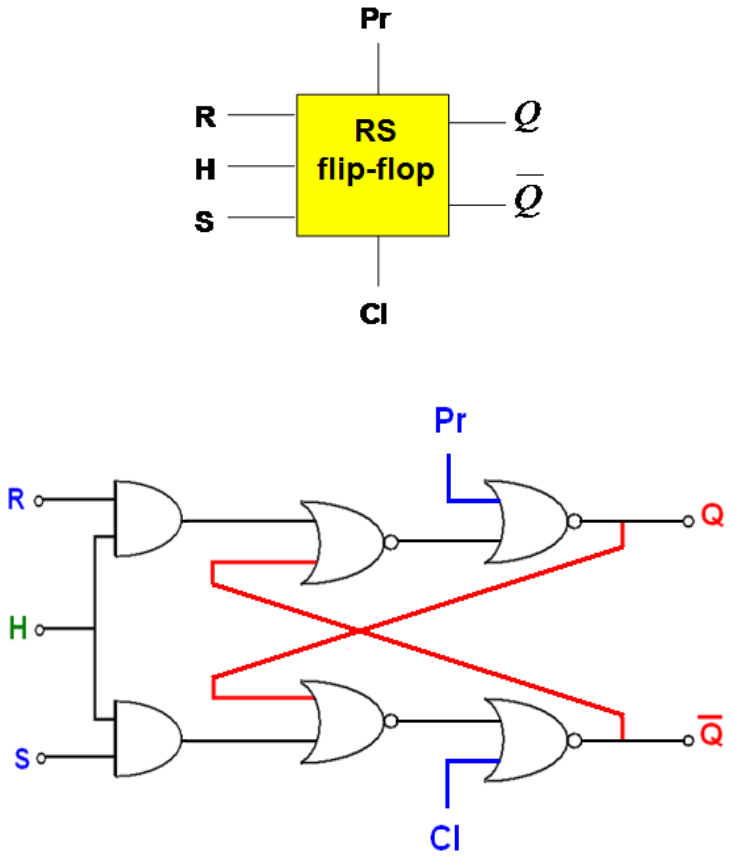


Figure 2.14: *Direct asynchronous input flip-flop*

- When *Pr = Cl = 0* these signals have no effect on the flip-flop. The flip-flop state is determined by the H, S and R inputs.

- When *Pr = 1* and *Cl = 0* in this case *Q = 1* independently of the other inputs H, S and R. Similarly if *Pr = 0* and *Cl = 1* the flip-flop will be set to zero, i.e. *Q = 0* independently of the inputs H, S and R.

The Pr and Cl inputs are called asynchronous inputs because the flip-flop changes state independently of the clock. Note that inputs Pr and Cl cannot be set to "1" simultaneously.

2.2.5. RS Master - Slave flip-flop

Cascading several flip-flops is widely used for applications such as shift registers. In the "master-slave" flip-flop, two flip-flops are cascaded: the first registers the situation corresponding to the inputs and then, on the clock pulse, transmits it to its "slave", the second flip-flop. Inherently, this arrangement involves a delay in the input-output transmission. So the "master-slave" flip-flop consists of two flip-flops connected in series, with the outputs sent back to the inputs, as shown in the following figure:

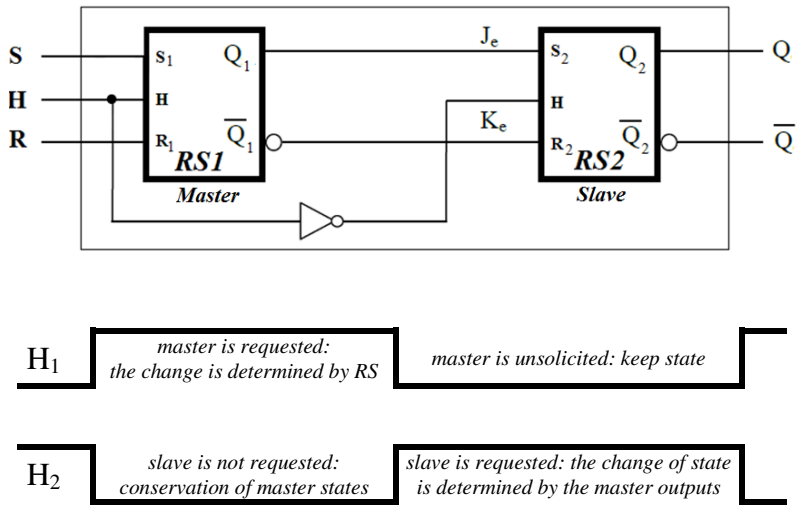


Figure 2.15: *Master-slave flip-flop*

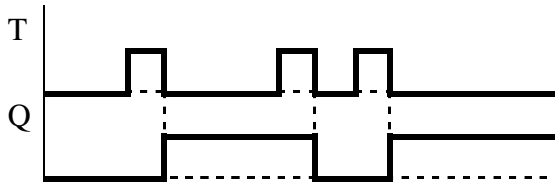
- When the clock is high, H_1 is high, and the S and R inputs are used to change Q_1 of the master flip-flop. However, during this time, clock H_2 is in the low state, so the slave flip-flop does not change state.
- When the clock is low, H_1 is low and the R and S inputs have no effect on the master flip-flop. However, clock H_2 is in the high state and the slave flip-flop responds to its R_2 and S_2 inputs, so the Q output of the master-slave flip-flop changes state only after clock H returns to its low state.

2.2.6. T flip-flop

The T flip-flop switches to the opposite situation with each new input pulse. With a single T input, its logic equation is : $Q^+ = Q \oplus T$

T	***Q_n***	***Q_{n+1}***
0	0	0
0	1	1
1	0	1
1	1	0

Truth table



Chronogram

Table 2.3 *: T flip-flop.*

Logic function :

$$Q^+ = Q\overline{T} + \overline{Q}T$$
$$Q^+ = Q \oplus T$$

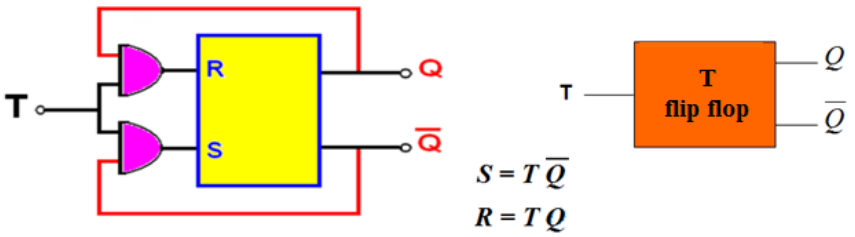


Figure 2.16 : *T flip-flop.*

Note: The T flip-flop is a frequency divider by 2, applying the clock directly to the input.

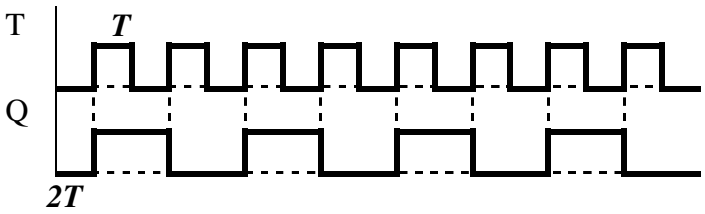


Figure 2.17: *Chronogram of a frequency divider*

2.2.7. JK flip-flop

A "1" at the input of J or K only reacts like an RS flip-flop, however it is possible to apply a "1" to both inputs simultaneously, in this case the flip-flop changes state like a T flip-flop.

J	*K*	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table 2.4 : *Truth table.*

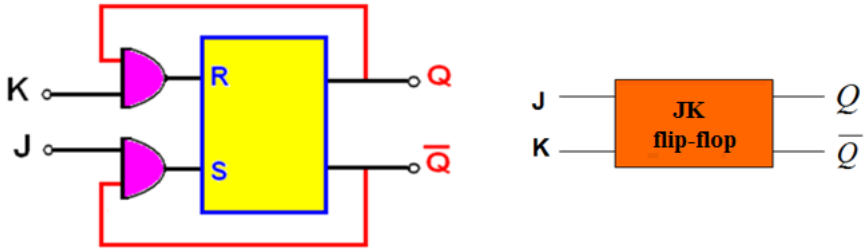


Figure 2.18 : *JK flip-flop*

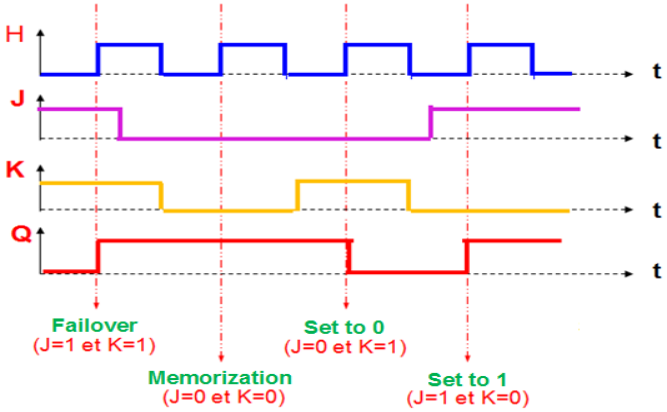


Figure 3.19: *Chronogram of a JK flip-flop rising edge*

The following figure shows the symbolic representation of JK flip-flops with Preset and Clear inputs, one triggered by the rising edge and the other by the falling edge of the control signal.

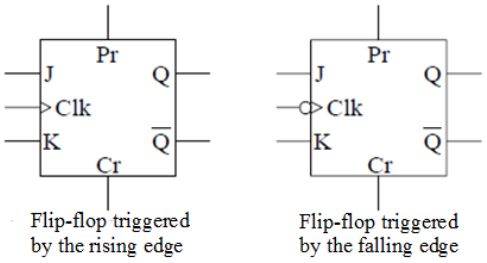


Figure 2.20: *JK flip-flop with Preset and Clear inputs*

2.2.8. D flip-flop

This is a synchronous flip-flop with a single D input and two outputs. It is used to copy the input at each clock pulse and conserve it.

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Table 2.5 : *Truth Table.*

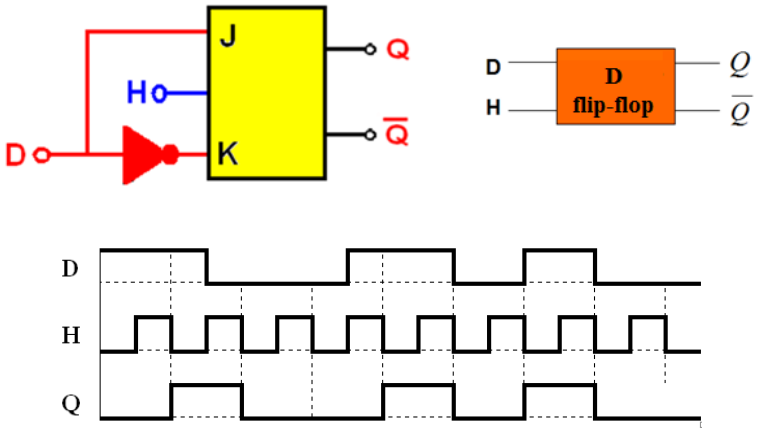


Figure 2.21 : *Chronogram of a D flip-flop.*

2.3. COUNTERS / DOWN COUNTERS

Counters are important digital electronic circuits. They are sequential logic circuits because timing is an important function, and because they require memory properties. Counters generally perform several functions in a single measuring device. In addition to counting pulses to measure quantities, they are often used to control the sequencing of measurement processes, and to visualize measurement results. The main features of digital counters are as follows:

1. maximum counting capacity;
2. direct counting or down counting;
3. synchronous or asynchronous operation;
4. permanent operation or automatic stop.

2.3.1. Asynchronous counters

The propagation counter (asynchronous counters) comes from the fact that the three flip-flops don't all change state at the same time when the clock signal transitions, i.e. the output of each flip-flop acts as the clock signal for the next. Flip-flop A switches when the clock pulses arrive. The bascule B must wait for the output A passes from "1" to "0" to be triggered. Flip-flop C has to wait for output B to respond, and so on for the rest of the flip-flops, giving a delay between each flip-flop's response. We can also use this counter as a frequency divider, so each of these three JK flip-flops halves the frequency of the signal feeding its Clk input.

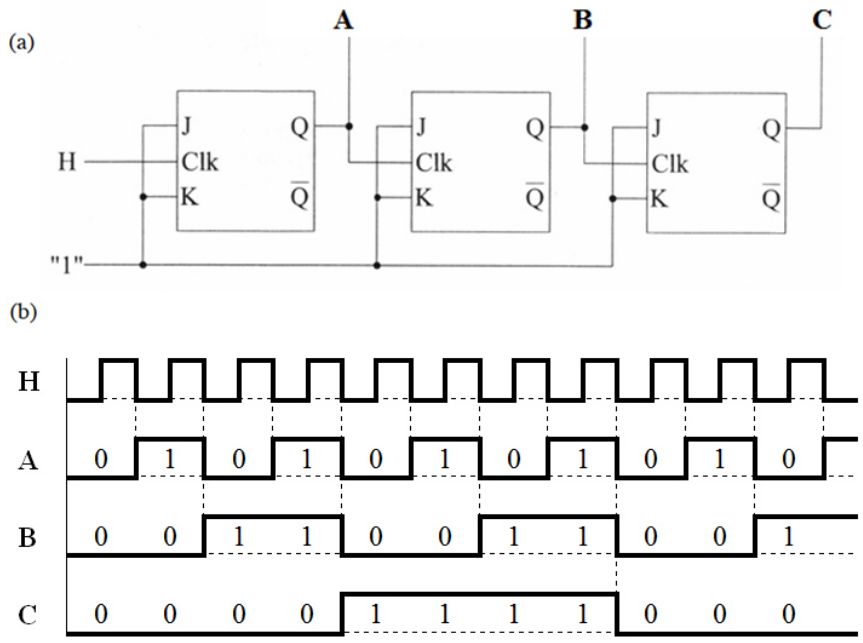


Figure 2.22: *3-bit asynchronous counter synchronized on falling edge*

The same counter can be realized using rising-edge synchronized flip-flops and $\bar{Q}$ outputs instead of Q.

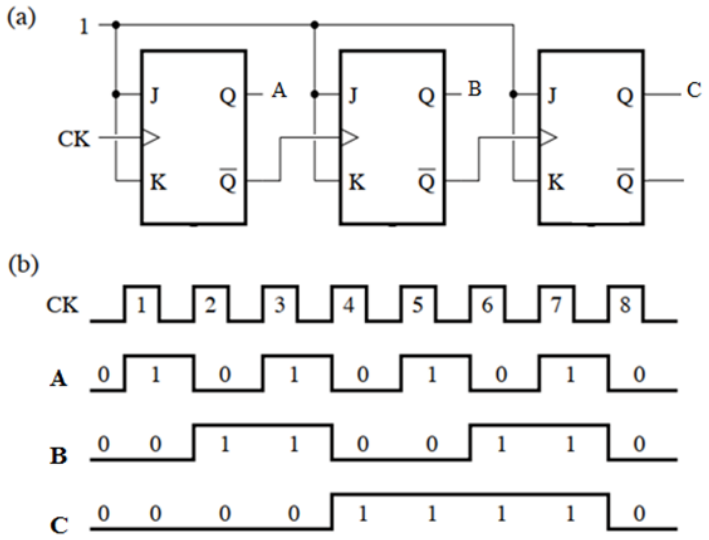


Figure 2.23: *3-bit asynchronous counter synchronized on rising edge*

Note that all counters are generally associated with a common "RAZ" reset line.

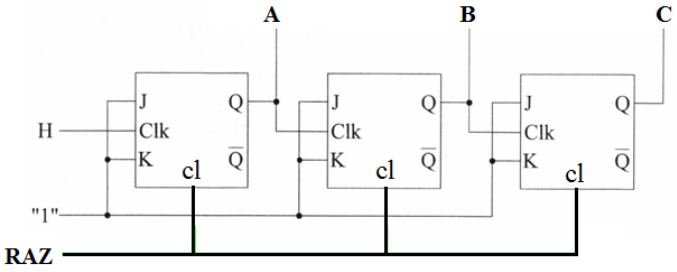


Figure 2.24: *Asynchronous counter with common "RAZ" reset line*

2.3.2. Propagation delay in asynchronous counters

Propagation counters are the simplest type of binary counters, but they have a major drawback due to the fundamental principle of their operation. Each flip-flop is triggered by the output transition of the flip-flop preceding it. Because of the propagation delay (t_R) introduced by each flip-flop, which add up so that the N^{th} flip-flop only changes state ($N.t_R$) units of time after the arrival of the clock pulse.

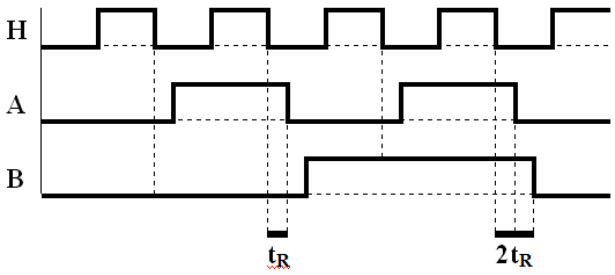


Figure 2.25: *Waveforms of a 3-bit propagation counter*

Asynchronous counters are not very useful at very high frequencies, especially when they have a large number of bits. Their simplicity makes them particularly suitable for applications where the limiting frequency is not a critical factor.

2.3.3. Synchronous counters

The problems caused by asynchronous counters are due to the propagation delays of cascaded flip-flops, which do not change state simultaneously with the clock pulses. This is why we use synchronous counters (parallel counters), in which all the flip-flops are triggered simultaneously (in parallel) by the input clock pulses. The clock pulses are applied to all the flip-flops at the same time. The principle of synchronous counters is as follows:

- All flip-flops receive the same clock pulse

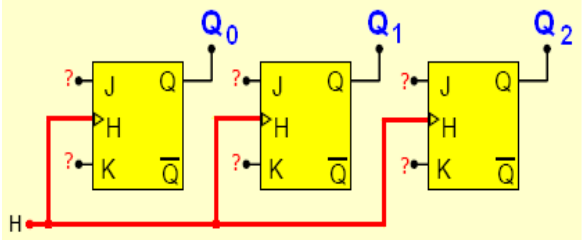


Figure 2.26 : *Clock applied to all three flip-flops at the same time*

- At the stroke of the clock, all the flip-flops change state at the same time, no transient states
- What should we do with the J's and K's of the flip-flops to make them count? To do this, we use the transition table.

We know the counting sequence, so at the stroke of the clock, we know for each flip-flop whether it should change state or not. From this we can deduce the J and K to be applied to each flip-flop.

1. **Flip-flop A** changes state at each transition of clock H:

H	C	B	A
↓	0	0	0
↓	0	0	1
↓	0	1	0
↓	0	1	1
↓	1	0	0
↓	1	0	1
↓	1	1	0
↓	1	1	1
↓	0	0	0

2. **Flip-flop B** changes state at each transition of clock H, if ***(A =1)*** :

H	C	B	A
↓	0	0	0
↓	0	0	1
↓	0	1	0
↓	0	1	1
↓	1	0	0
↓	1	0	1
↓	1	1	0
↓	1	1	1
↓	0	0	0

3. **Flip-flop C** changes state at each transition of clock H, if ***(A . B =1)*** :

H	C	B	A
↓	0	0	0
↓	0	0	1
↓	0	1	0
↓	0	1	1
↓	1	0	0
↓	1	0	1
↓	1	1	0
↓	1	1	1
↓	0	0	0

Therefore, for flip-flops activated by the falling edge of the H clock signal, we obtain the following synchronous counter:

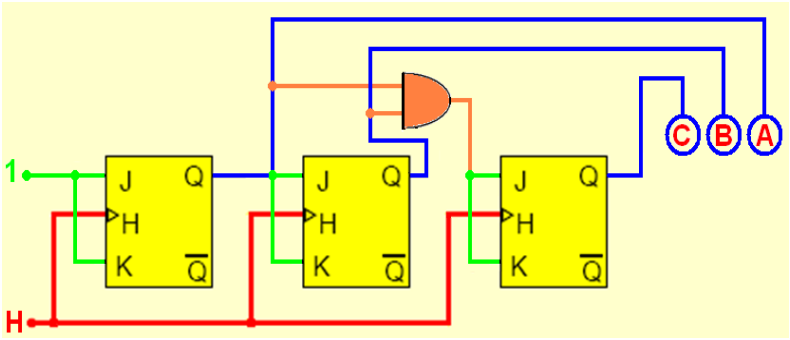


Figure 2.27: *Block diagram of a 3-bit parallel counter*

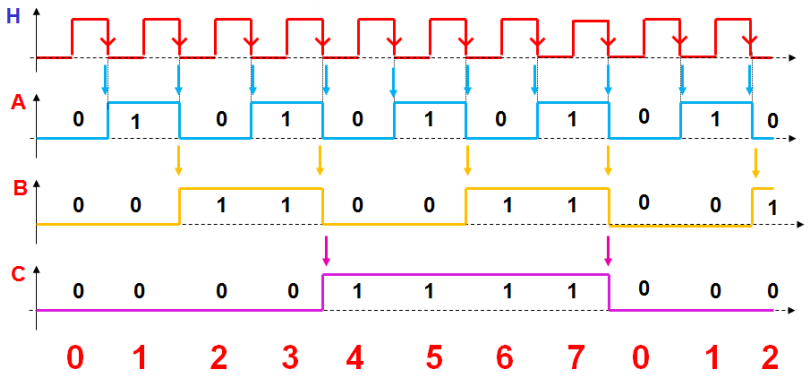


Figure 2.28 : *Chronogram of a 3-bit parallel counter*

2.3.4. Down counters asynchronous

So far, we've seen counters that increment their contents by one with each new pulse. There are also counters that decrement their contents. These are known as decounters. The following diagram shows a module 8 countdown timer with 3 JK flip-flops, a chronogram of its operation and a state diagram. For a falling-edge clock signal, simply use the $\bar{Q}$ output of each flip-flop to trigger the next one.

	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0

Table 2.6: *Counting table*

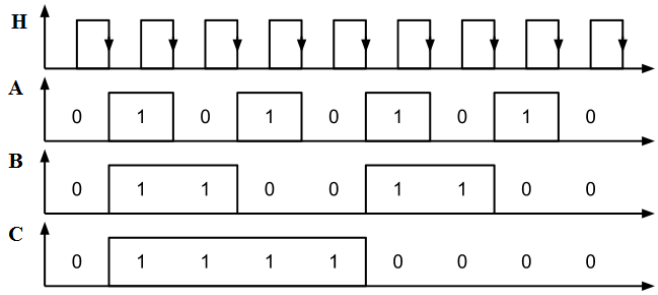


Figure 2.29 : *Chronogram of a 3-bit down counter*

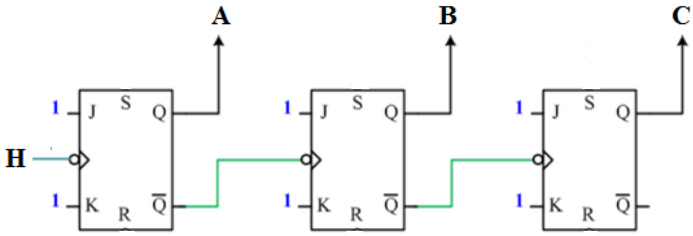


Figure 2.30 : *Block diagram of a 3-bit down counter*

2.3.5. Counters with modulo less than 2^n (incomplete cycle)

It is possible to modify the elementary counter to obtain a counter with a modulo lower than 2n (with icomplete cyclne), by allowing it to skip a certain section of the binary number sequence. One of the most common methods of achieving this is illustrated in the following figure, which shows a modulo 6 counter:

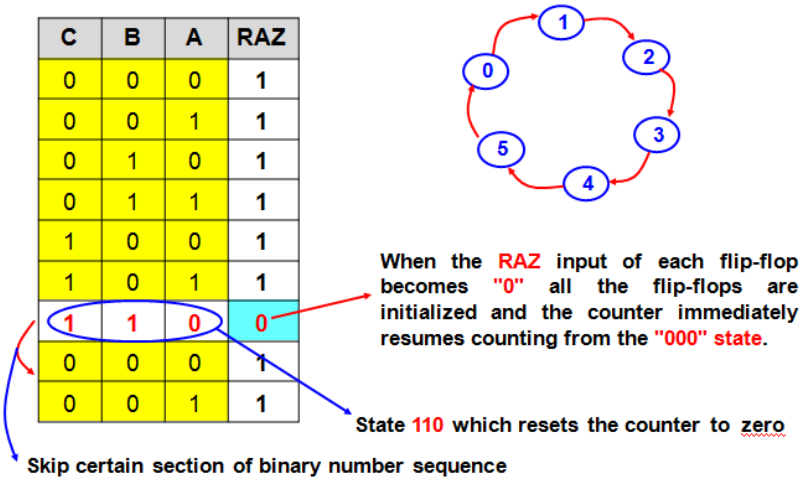


Figure 2.31: *Principle of a modulo 6 counter*

The following figure shows a binary counter modulo 8 that can display the sequence of numbers from "000" to "111", but the NAND gate disturbs this sequence as follows:

- The output of the NAND gate is connected to the reset input RAZ of each flip-flop. As long as this output remains in state "1", the counter is not affected.
- When it becomes "0", all the flip-flops are initialized and the counter immediately starts counting again from state "000".
- The inputs of the NAND gate are the outputs of flip-flops B and C, so the output of this gate changes to "0" when we have ($B = C = 1$). This condition occurs when the counter goes from state "101" to state "110" (on the sixth pulse).
- The low level of the NAND gate immediately sets the counter to "000". As soon as the flip-flops have been set to "0", the NAND gate output returns to "1", since the condition ($B = C = 1$) no longer exists.

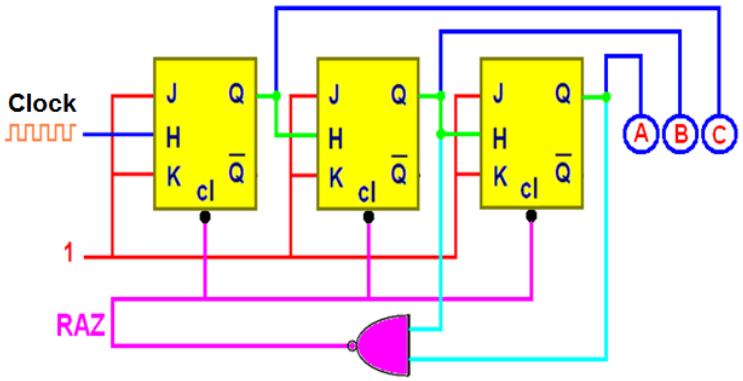


Figure 2.32: *Block diagram of a modulo 6 counter*

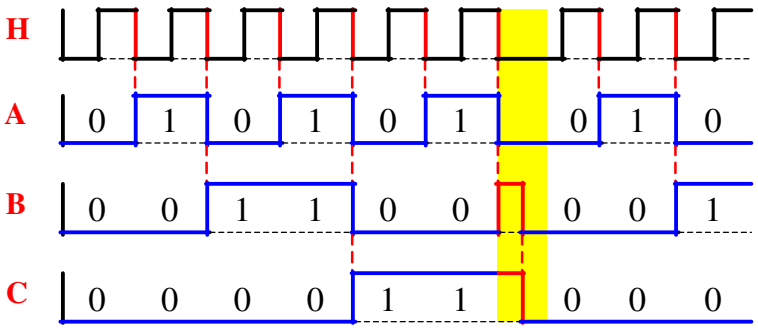


Figure 2.33 : *Chronogram of a modulo 6 counter*

Example :

The following figure shows a modulo 10 counter, with the NAND gate disrupting the counting sequence from "0000" to "1111" when the counter reaches state 1010 (on the tenth pulse). The low level of the NAND gate immediately sets the counter to "0000". As soon as the flip-flops have been set to "0", the NAND gate output returns to "1", and the counter resumes counting.

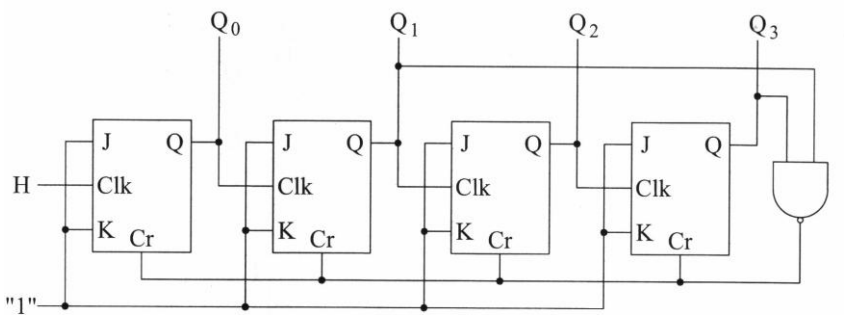


Figure 2.34 : *Modulo 10 counter*

2.3.6. BCD counters mounted in cascades

BCD counters are used to count pulses and display the result in decimal notation. A single BCD counter can count from "0" to "9" and then be reset to "0". To count larger decimal values, we can cascade BCD counters, as shown in the following diagram:

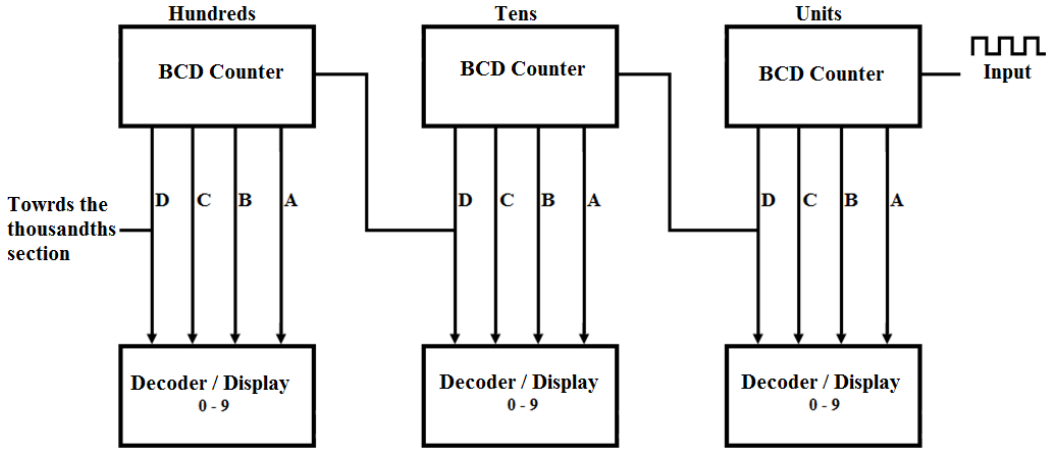


Figure 2.35: *BCD counter mounted in cascades*

The operation of this assembly is as follows:

1. At the start, all counters are initialized, so the decimal number displayed is (**000**).
2. As pulses arrive, the BCD units counter is increased by "1" with each pulse. After "9" pulses, the tens and hundreds BCD counters are still at "0" and the units counter is at "9" (binary number 1001), so the decimal number displayed is (**009**).
3. On the tenth input pulse, the units counter is recycled to "0", causing the output of flip-flop D to change from "1" to "0". This transition acts as a clock signal for the tens counter, which is incremented after "10" pulses, so that the decimal value displayed is (**010**).
4. As further pulses arrive, the units counter is incremented with each pulse, and each time it is recycled to "0", the tens counter increases by "1". So, after "99" pulses, the tens counter is at "9" like the units counter. The decimal number displayed is therefore (**099**).

5. On the hundredth pulse, the units counter is recycled to "0", which also causes the tens counter to be recycled to "0". The tens counter's D flip-flop output affects a transition from "1" to "0", which acts as a clock signal for the hundreds counter, causing it to increment. So after "100" pulses, we read the decimal digit (**100**).
6. This process continues until the "999th" pulse. The decimal number displayed is therefore (**999**). At the "1000th" pulse, all counters are recycled to "0", and we read the decimal digit (**000**).

We can easily see that an assembly can be modified to represent any number of decimal digits, by adding new stages. For example, to count to (**999999**), we need six BCD counters with their decoder and display. In general, we need one BCD counter per decimal digit.

2.4. REGISTERS

A register is a synchronized sequential circuit in the central processing unit that stores a certain number of bits, using a set of flip-flops synchronized by the same clock signal. Registers are widely used in digital processing systems to perform operations such as temporary storage (buffering), shifting, rotation and so on.

2.4.1. Characteristics of a register

Every register is characterized by:

- **Capacity:** number of bits of the binary word it can store.
- **Write or load mode:** depends on the number of entries:
  - Serial writing: bit-by-bit generation on the first flip-flop, with transmission via a single wire. At each clock stroke, a new bit is present, and those already loaded are shifted by one rank.
  - Parallel writing: global generation of the N-bit word, with transmission via an N-bit bus (N wires).
- **Read mode:** the information stored in a shift register can be read in the same way, in series or in parallel:
  - Serial reading: bit by bit exploitation of the word (a single output).
  - Parallel reading: global exploitation of the word (N outputs).

Registers can be classified into two categories: memorization (buffer) registers and shift registers.

2.4.2. Memorization register

A memorization register (or data register) is a register in which the various stages are independent of each other, although certain signals act on all the stages, such as reset to 0 and reset to 1. This register stores binary information in the form of an N-bit word. The information is loaded (written), then stored and made available for reading.

For example, a register of 4 D flip-flops stores 4 bits of information. The register stores the states of inputs E_0, E_1, E_2 and E_3. These states are kept until the next control signal

W. The stored data can be read on the outputs Q_0, Q_1, Q_2 and Q_3 in coincidence with a validation signal R.

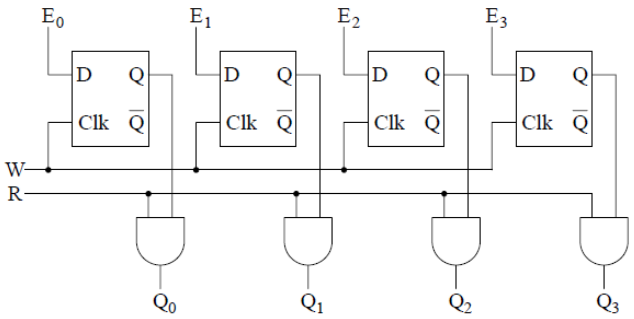


***Figure 2.36** : Memorization register*

2.4.3. Shift register

A shift register is a register that can shift its contents right or left. In a shift register, the flip-flops are interconnected so that the logic state of the output of the flip-flop of rank (i) can be transmitted to the flip-flop of rank (i+1) (or i-1) when a clock signal is applied to all the flip-flops.

This type of register is mainly used as a dynamic information memory; the shift function consists of sliding the information from each elementary cell into another adjacent elementary cell. Depending on the mode of access, write (input) and read (output), serial or parallel. There are several types of shift register:

1. Parallel input/parallel output register.
2. Register with serial input and serial output
3. Register with serial input and parallel output.
4. Register with parallel input and serial output.
5. Circular shift register.

2.4.4. Parallel input - parallel output shift register

The parallel input/parallel output register (parallel load register) is a register that can load information on *N* bits at the same time:

- The *N* D flip-flops change states at the same time.
- Each D_i flip-flop takes the value of information i.
- It has a loading input *ch* (*ch = 0*: memory state; *ch = 1*: loading).

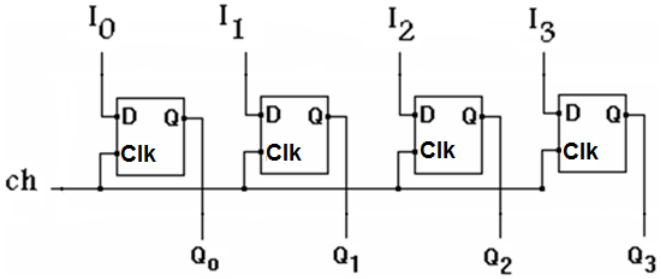


***Figure 2.37:** Parallel input and parallel output shift register*

2.4.5. Serial input and serial output shift register

The information bits are presented sequentially bit by bit at the input of the first flip-flop, and propagate through the register with each clock pulse, exiting at the last flip-flop. For example, load binary word 0001:

Word 0001				
1st Pulse	1			
2nd Pulse	0	1		
3rd Pulse	0	0	1	
4th Pulse	0	0	0	1
5th Pulse	0	0	0	0

The following diagram shows the timing of the loading of a binary word "0001" into a 4-bit shift register. The information bits are loaded into the D input of the first flip-flop. Assuming that all the register's flip-flops are reset to zero, the register's output will be "0000".

Inject the first bit of information, a "1", at the data input, and send a clock pulse: the outputs will display "1000". Inject a "0" at the data input, send a second clock pulse: the outputs will now display "0100". On the third pulse, the display will be "0010" and on the fourth, "0001". The binary word "0001" will have been loaded into the register bit by bit. This operation constitutes serial loading.

Note that with each clock pulse, the register shifts the data to the right: this register could be called a serial-load right-shift register.

On the fifth pulse, the outputs will display "0000"; the "1" on the right is shifted out of the register and lost.

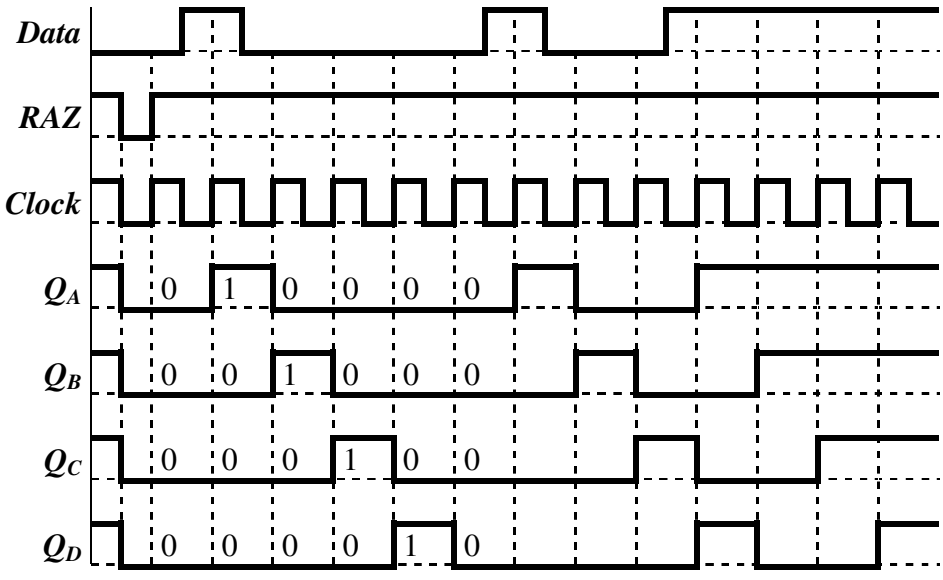


Figure 2.38: *chronogram of the loading of a binary word 0001 into a 4-bit shift register*

2.4.5.1. Shift right

Such a register is called a left-serial-in, right-serial-out register. In this case, the input of the D flip-flop of rank *(i)* must be connected to the output of the flip-flop of rank *(i-1)*.

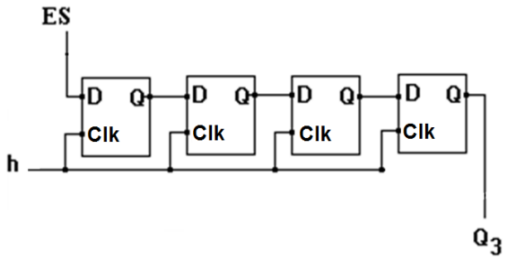


Figure 2.39: *Right shift register.*

2.4.5.2. Shift left

In this case the input of flip-flop D of rank *(i)* must be connected to the output of flip-flop of rank *(i+1)*.

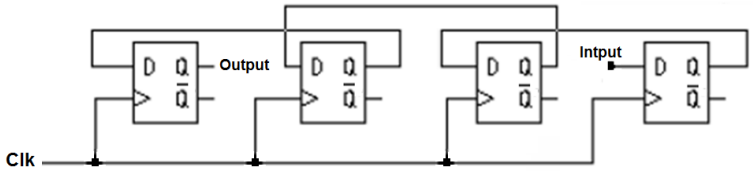


Figure 2.40 *: Left shift register.*

2.4.6. Serial input - parallel output shift register

This type of register transforms temporal coding (succession of bits over time) into spatial coding (information stored in static memory). The register then acts as a buffer. The following figure shows an example of a 4-bit register with serial input and parallel output, using D flip-flops.

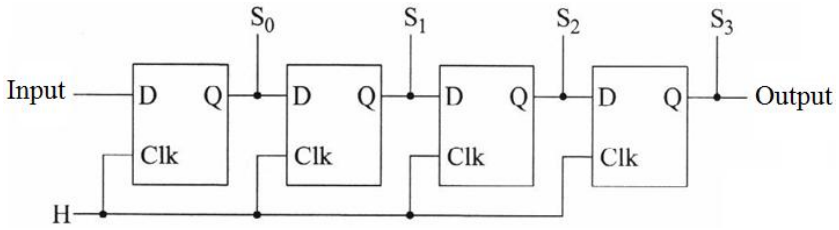


Figure 2.41: *Serial input - parallel output shift register.*

The R (Read) signal is not obligatory, it just allows you to control the reading of the outputs at the same time, so as to avoid reading at the time of loading.

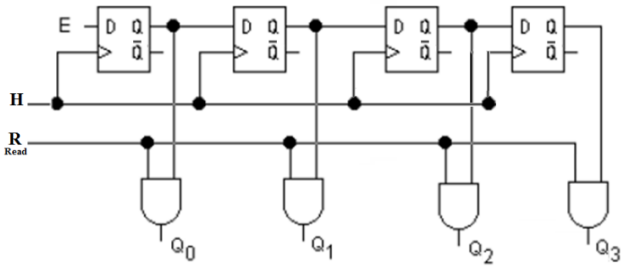


Figure 2.42: *Serial input - parallel output shift register with R input.*

2.4.7. Parallel input - serial output shift register

This type of register transforms spatial coding (information stored in static memory) into temporal coding (succession of bits over time). The following figure shows an example of a 2-bit register with parallel input and serial output using D flip-flops.

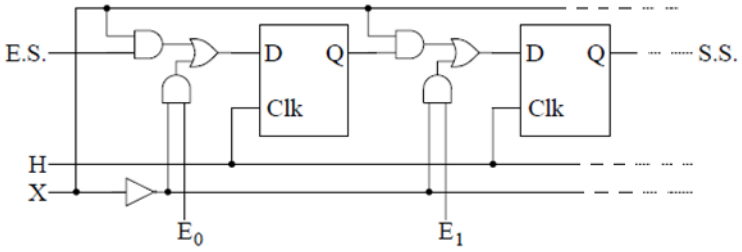


Figure 2.43: *Shift register parallel input serial output.*

2.4.8. Circular shift register

The circular shift register is a register that performs a left shift by sending the output of the last flip-flop to the input of the first flip-flop. The shift can be a right shift (circular right) or a left shift (circular left).

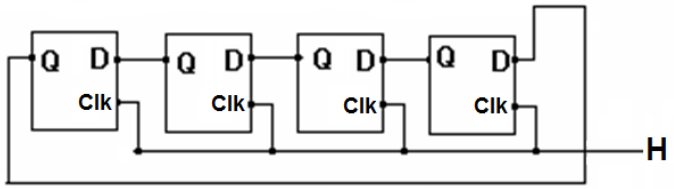


Figure 2.44: *Circular shift register.*

CHAPTER 03

LOGIC INTEGRATED CIRCUITS

CHAPTER 03

LOGIC INTEGRATED CIRCUITS

3.1. INTRODUCTION

In 1958, American Jack Kilby (1923 - 2005), then employed by Texas Instruments, created the very first integrated circuit. The integrated circuit is an electronic component made up of a set of discrete semiconductor-based components, such as diodes and transistors, assembled on a substrate. This combination of elementary functions enables the realization of many complex analog or digital functions, such as operational amplifiers, logic gates or microcontrollers. Since its creation, the integrated circuit has become increasingly miniaturized, thanks to numerous technological advances. As a result, they have discreetly invaded our everyday objects. This major invention was the starting point for the development of electronics and computing. Thanks to this revolutionary process, large volumes of data and logic functions could now be processed with maximum efficiency.

In this chapter, we'll focus on gaining a better understanding of the properties of logic ICs. To limit the scope of our study, we'll take a general look at the two main families, TTL and CMOS, without examining the nuances between the many sub-families.

3.2. INTEGRATED CIRCUIT MANUFACTURING PROCESS

The basic material for the manufacture of IC (Integrated Circuit) is pure silicon. Wafers half a millimeter thick are cut from a 30 cm diameter bar. The electronic circuits are built on the surface of this wafer by photolithoengraving. This requires dozens of steps:

- Oxidation of the surface,
- Deposition of a sensitive photo varnish,
- Transfer of the circuit design to the surface of this varnish,
- Varnish exhibition no covered to UV or X radiation,
- Removal of varnish sensitized by a corrosive agent,
- Doping of silicon surfaces in places that are no longer covered with oxide

The process is repeated to superimpose several layers of circuitry. The circuits are tested, then the wafer is cut into chips, also known as dies. Each chip is then placed in a housing that protects it mechanically and provides the pins to connect the circuit to the printed circuit boards.

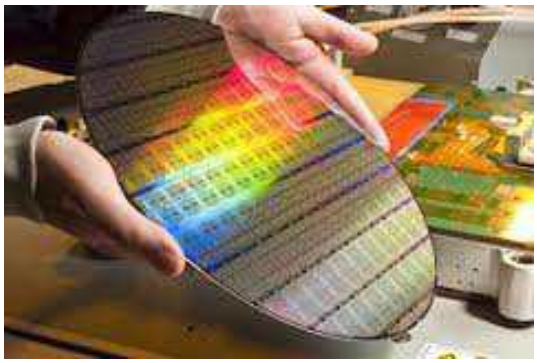


Figure 3.1: *Pure silicon wafers.*

3.3. HOUSING AND PINOUTS

Integrated circuits generally take the form of rectangular, black, solid packages, fitted on one or more sides, or even on one face, with 'tabs' for making electrical connections to the outside of the package. These components are soldered onto a printed circuit, or plugged, for disassembly purposes, into brackets which are themselves soldered onto a printed circuit.

The first packages were in Dual In Line Package (DIL or DIP) format. This format is still used today for circuits with only a small number of components.

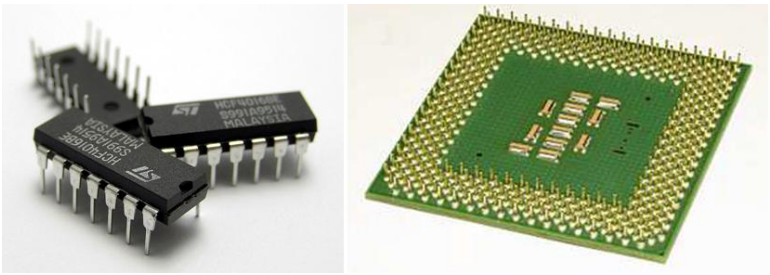


Figure 3.2: *Housing and pinouts*

A recessed notch indicates the location of pin 1, which is to the left of this notch. Pin 2 is below; the other pins are numbered counter-clockwise.

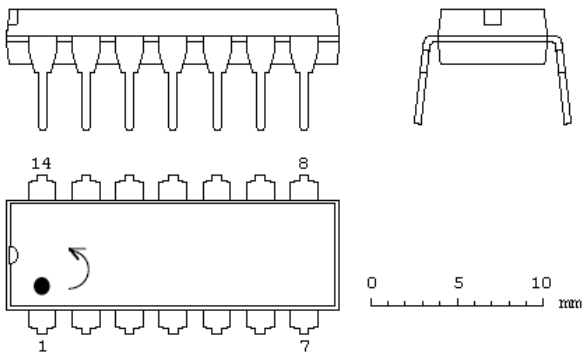


Figure 3.3 : *Pinout*

3.4. IDENTIFICATION OF AN INTEGRATED CIRCUIT

Integrated circuits are identified by their serial number, which is engraved on the surface of the circuit. This serial number contains information about the manufacturer, the type of circuit, the date of manufacture and other important information. It is important to know the serial number of an integrated circuit to ensure that it is compatible with the electronic system in which it will be used.

For example, in the following figure the reference of the left circuit is 4029, and the reference of the right circuit is 74S113.

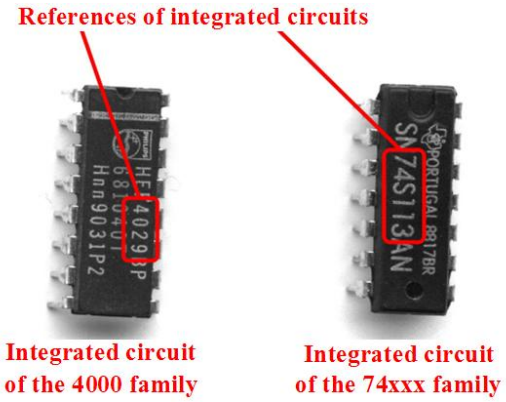


Figure 3.4 : *Identification of an integrated circuit*

To find out the function of an integrated circuit for which the reference is known, you must consult the catalog or the manufacturer's technical sheet for possible details on this or that particular version.

3.5. CLASSIFICATION OF INTEGRATED CIRCUITS

Integrated circuits are classified according to their characteristics, field of use, electrical performance and manufacturing technologies, as well as the number of transistors per package.

3.5.1. Classification by field of use

The classification according to their field of employment can be as follows:

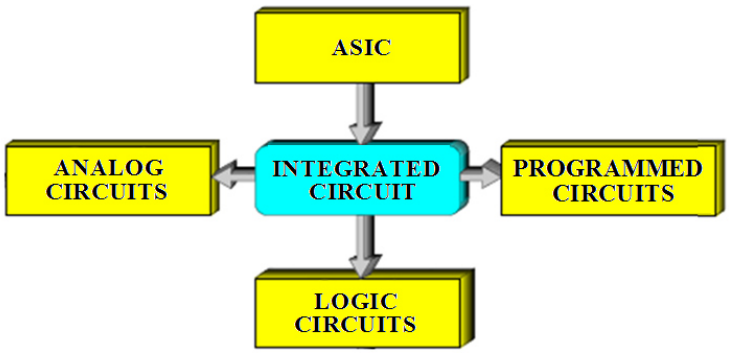


Figure 3.5 : *Fields of employment*

1. **ASIC:** Application Specific Integrated Circuit ce qui signifie : These are "made-to-order" integrated circuits. They can integrate analog and logic structures, but are expensive on a small scale.
2. **Programmed Integrated Circuits:** they require virtual information (a program) governing their operation. These are generally logic circuits (μ Processor, EPLD...) but they can also be analog, thanks to DSP (Digital Signal Processing).
3. **Analog integrated circuits:** Analog ICs are used to process analog signals such as sound, light and radio waves. They are used in devices such as audio amplifiers, temperature sensors and motor control circuits. Analog ICs are designed to work with continuous signals, meaning that signals are measured as continuous rather than discrete values.
4. **Logic integrated circuits:** are non-programmed integrated logic structures. The simplest digital integrated circuits are logic gates (AND, OR and NOT), the most complex are microprocessors and the densest are memories. They are used to process digital signals such as computer data. They are used in devices such as computers, cell phones and games consoles. Digital ICs are designed to work with binary signals, meaning that signals are measured in discrete values, such as 0 and 1.

3.5.2. Classification by number of transistors per package

The number of components integrated in a standard-size IC represents its integration scale, in other words, its component density. It is classified as follows:

- **SSI** (**Small Scale Integration**): It has less than 100 transistors per cm^2, approximately 10 gates.
- **MSI** (**Medium Scale Integration**): It contains fewer than 500 transistors or has between 10 and 100 gates.
- **LSI** (**Large Scale Integration**): the number of transistors per cm^2 is between 500 and 300 000 or has more than 100 gates.
- **VLSI** (**Very Large Scale Integration**): It contains more than 300 000 transistors per cm^2.
- **VVLSI** (**Very Very Large Scale Integration**): It contains more than 1 500 000 transistors per cm^2.

3.6. TECHNOLOGY FAMILIES

Over the course of the industrial development of integrated circuits, several families have evolved, each with its own structure. Initially, the DTL family, short for Diode-Transistor Logic, enabled the simple construction of a NAND gate. Further modifications, notably the replacement of the input diodes by a multi-emitter transistor, led to the basic structure of the TTL family, short for Transistor-Transistor Logic.

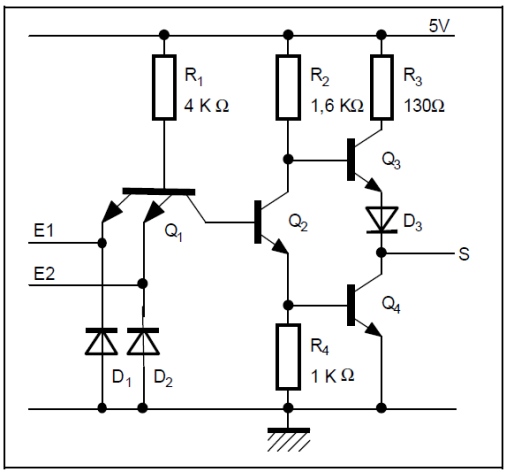


Figure 3.6: *Internal structure of a 2-input NAND gate in TTL family*

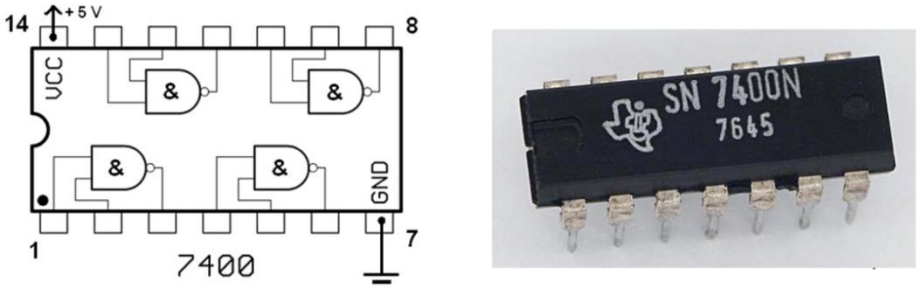


Figure 3.7: *Integrated circuit of a NAND gate with 2 inputs in TTL family*

The TTL family, based on bipolar technology, held a dominant position in current applications until the emergence of CMOS (Complementary Metal Oxide Semiconductor) circuits, based on more recent MOS transistor technology.

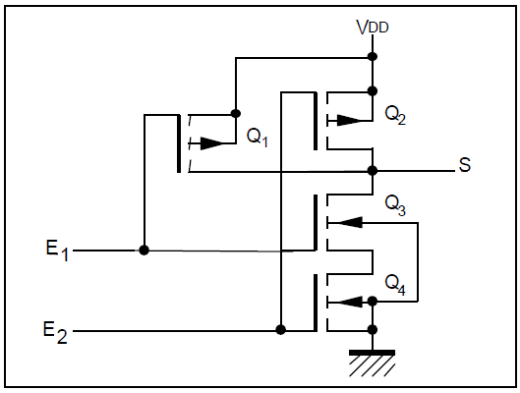


Figure 3.8: *Internal structure of a 2-input NAND gate in CMOS family*

For identical logic operation, each technology offers different electrical (voltages, currents, power) and temporal (speed) performances.

1. A logic family is characterized by its electrical parameters:
    - the range of supply voltages and the tolerance allowed on this value ;
    - the voltage range associated with a logic level, input or output;
    - currents for each logic level, input or output;

- the maximum current that can be extracted from a logic gate and the current absorbed at the input;
- maximum power consumption, which often depends on operating frequency.

2. The main dynamic performances are:

- the rise (low-high transition) and fall (high-low transition) times of signals at the output of a gate;
- signal propagation times between the input and output of a logic gate.

3.6.1. The TTL family

The TTL (Transistor-Transistor Logic) family, one of the oldest, requires a supply voltage of exactly 5 volts. Its response (or propagation) time is good: around 10 ns for the standard version. On the other hand, its power consumption is high (10 mW) and its operating frequency doesn't exceed around 25 MHz. Many TTL variants subsequently appeared:

- 74 : TTL Standard (10ns, 10 mW)
- 74 H : High Speed (6ns, 22 mW)
- 74 L : Low Power (33ns, 1 mW)
- 74 S : Schottky (3ns, 19 mW)
- 74 LS : Low Power Schottky (10ns, 2 mW)
- 74 AS : Advanced Schottky (1.5ns, 8 mW)
- 74 ALS : Advanced Low Powe Schottky (4ns, 1 mW)
- 74 F : Fast (3.7ns, 5.5 mW)

3.6.2. The CMOS family

The CMOS family has more flattering characteristics: a supply voltage generally between 3 and 18 volts and much lower power consumption, of the order of 0.1mW. The disadvantages of the first logic CMOS (long switching times, limited frequency, etc.) were soon remedied by the HC (High Speed) and HCT (High Speed CMOS TTL compatible) versions, which offer very short response times (15 ns) and allow high frequencies. Noise immunity is also very good.

3.7. CHARACTERISTIC PARAMETERS

Each logic family is characterized by electrical parameters such as power supply and consumption, and dynamic performances such as propagation delay.

3.7.1. Electrical parameters

- **V_{CC}** (supply voltage) : voltage level required to supply the circuit :
  - ✓ **In TTL:** the power supply must be continuous and stable V_{cc} = 5V +/- 5%
  - ✓ **In CMOS:** the choice of supply voltage is wider from 3V to 18V. But dynamic performance degrades at low supply voltage levels.

- **VIH (min) - *input voltage HIGH level*:** voltage level required to have a logic 1 at input.
- **VIL (max) - *input voltage LOW level*:** voltage level required to have a logic 0 at input.
- **VOH (min) - *output voltage HIGH level:*** voltage level of the output of a logic circuit corresponding to logic state 1.
- **VOL (max) - *output voltage LOW level:*** voltage level of the output of a logic circuit corresponding to logic state 0.
- **IIH - *HIGH level input current:*** the current that flows through an input terminal when a high level voltage is applied to that input.
- **IIL - *LOW level input current:*** the current that flows through an input terminal when a low level voltage is applied to that input.
- **IOH - *HIGH level output current:*** the current flowing through an output terminal placed at logic level 1 under specified load conditions.
- **IOL - *LOW level output current:*** the current flowing through an output terminal placed at logic level 0 under specified load conditions.
- **VT - *The switching voltage:*** (T for threshold), corresponds to the input voltage for which the output changes state.

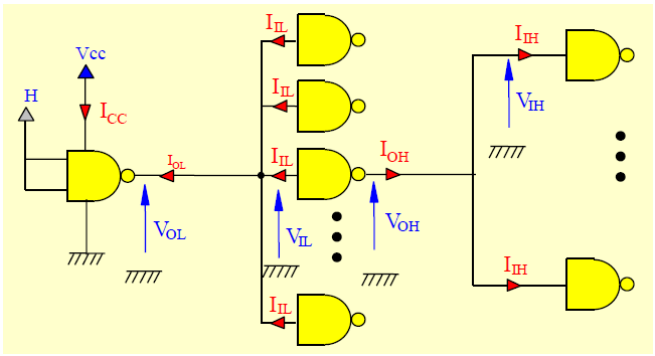


Figure 3.9: *Conventions for voltage and currents*

- **Fanout :** Normally, the output of a logic output must be able to drive several logic inputs. Fanout (also known as load factor) is defined as the maximum number of logic inputs a gate can drive. For example, when a logic gate is indicated as having an output of 10, this means that it can drive 10 logic inputs. If this number is exceeded, there is no guarantee that the output logic level voltages will be correct.
  - ✓ **In TTL:** the output can deliver IOH(max)= 400µA at logic high and absorb IOL(max)=16mA at logic low. The fanout is therefore 10 in TTL.
  - ✓ **In CMOS:** input currents are less than 1µA and outputs can carry more than 1mA. Output is limited not by input-output currents, but by input parasitic capacitances (5pF), which reduce switching times.

- **Noise immunity:** Noise is an untimely (or random) variation of a physical quantity around a given value (permanent, expected or nominal). Noise immunity is the safety margin that can be observed without causing an undesired change in logic state: it is the maximum amplitude of the spurious signal to be superimposed on the input signal to cause a change in output state.

  High state noise sensitivity margin:

  $$VNH = VOH_{(min)} - VIH_{(min)}$$

  Low state noise sensitivity margin:

  $$VNL = VOL_{(max)} - VIL_{(max)}$$

  - ✓ **In TTL :** noise immunity is : 0,4 V

    VOH(min) = 2,4 VOL(max) = 0,4 V
    VIH(min) = 2 V VIL(max) = 0,8 V

  - ✓ **In CMOS :** noise immunity is : $0,4*V_{cc}$

    $VOH(min)=0,95*V_{cc}$ $VOL(max)=0,05*V_{cc}$
    $VIH(min)=0,55*V_{cc}$ $VIL(max)=0,45*V_{cc}$

- **Interfacing:** this is the linking of one system, via its outputs, to another, via its inputs, with different electrical characteristics. The interface circuit is adapted to receive information from the upstream circuit and transmit it in a compatible manner to the downstream circuit.

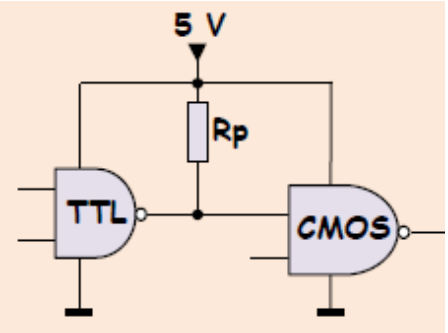


Figure 3.10 : *TTL / CMOS interfacing*

- In the case of TTL and CMOS circuits, the CMOS output can easily drive a TTL input. On the other hand, a TTL output delivers a VOH(min) voltage of **2.4V**, which is insufficient for a CMOS input (**VIHmin=0.55*5=2.75V**). The most common solution to ensure compatibility is to use a pull-up resistor Rp between **1KΩ** and **10KΩ**.

3.7.2. Dynamic parameters

- **Propagation time:** When a logic function changes the state of an output, this is not instantaneous, and a logic signal passing through a circuit always undergoes a delay. For simplicity's sake, we'll use the example of a logic inverter, comparing

the evolution of the input voltage level with its evolution at the output. However, this problem applies to all gates. Two propagation delays are defined:

- ✓ **tpHL** : delay to go from logic level 1 to logic level 0.
- ✓ **tpLH** : delay to go from logic level 0 to logic level 1.

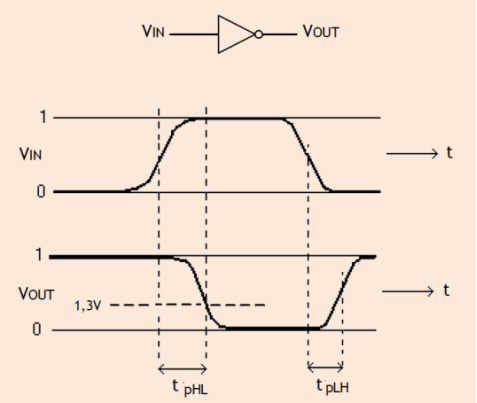


Figure 3.11: *Propagation time*

To ensure the measurement of these durations, a voltage reference is set by the manufacturers (1.3 V in TTL) for the start and end of propagation.

- ✓ **In TTL :** the propagation time varies depending on the subfamily from 10ns (TTL "N") to 1.5ns (TTL "AS");
- ✓ **In CMOS :** propagation time varies with V_{cc} supply level. Speed increases as V_{cc} increases.

3.7.3. Comparison between TTL and CMOS per gate

- CMOS power consumption is almost nil at low switching frequencies. But it reaches that of TTL around 1MHz. Each time it switches, the function calls up a current pulse (di/dt) on the supply pins. To ensure that V_{cc} does not vary due to wiring inductances (L.di/dt), decoupling capacitors (0.1µF) are placed on the power supply of each integrated circuit.

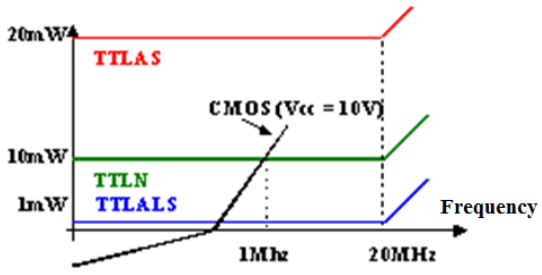


Figure 3.12 : *CMOS consumption as a function of frequency*

- To see the difference between these two families and use them safely, we'll try to summarize the main features of some of them.

Parameters'	TTL (Series 74XX)	CMOS (Series 40XX)
supply voltage (Vcc)	5V ±5%	from 3 to 18 V
VL	0.8V	U < 30% of V_{cc}
VH	2V	U > 70% of V_{cc}
VSL	0.4V	Approximately 0,4V
VSH	2.7V	Approximately V_{cc}
Response time	10 ns	8 to 10 ns
Consumption per door	1mW	< 1mW
Consumption for 5 MHz	1.5 mW	1,5 mW
Maximum frequency	30 MHz	10MHz
Fanout	10	50 mini

Table 3.1: *Comparison table*

3.8. WIRING AND REAL REALIZATION OF A LOGIGRAM

First of all, you need to know the exact pinout and internal structure of an integrated circuit (internal layout of the gates), so you'll need to consult the Mémotech électronique book. Wiring a logigram means making the link between the pinout of an integrated circuit and the symbol of the logic function you're about to perform. To do this, you need :

- Draw the flow chart.
- Identify the number of circuits and gates to be used.
- Identify the connections between the different doors (symbols); to facilitate the work you can list the pins used on the logic diagram and follow this numbering by interconnecting the different pins.
- terminals 7 and 14 are used to supply voltage to the integrated circuit. Terminal 7 must be connected to the minus (-) of a power supply (or 0V) and terminal 14 must be connected to the plus (+) of a power supply ($+V_{cc}$).

3.8.1. Input not connected

In digital electronics, a used input must not be left unconnected, hence the need for a resistor called a ground pull-up resistor, which forces the input to logic 0 when not connected to $+V_{cc}$.

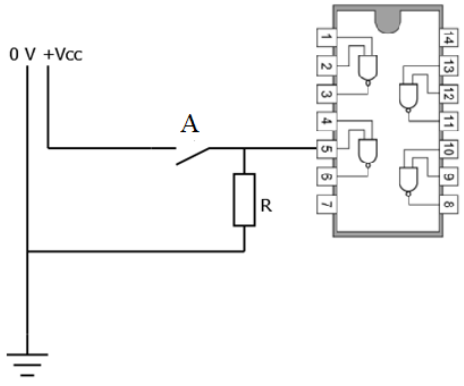


Figure 3.13 : *Entrée logique connectée à un circuit*

- If contact A is open, the input is connected through resistor R, so it is forced to 0.
- If contact A is closed, the voltage across the resistor is V_{cc}, which is applied to the input. The input is then in logic state 1.
- the role of resistor R is to prevent a short-circuit when contact A is closed.

3.8.1. Examples

Example 01 : Consider the following logical function:

$$f = \bar{A}.C + \bar{B}$$

1. The diagram of the function f with the logic gates:

$$f = \overline{\overline{\bar{A}C}.B}$$

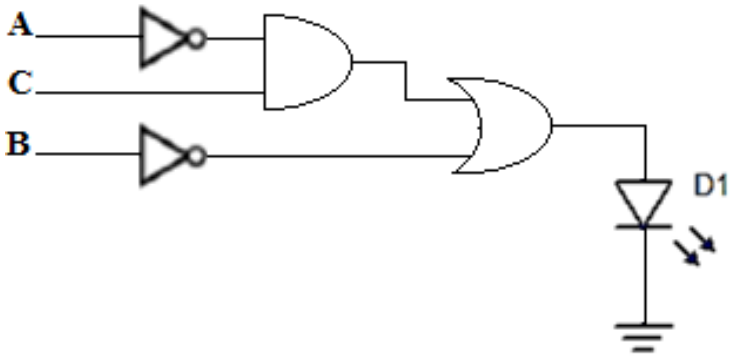


Figure 3.14 : *Schéma de la fonction f avec les portes logiques*

2. The diagram of the function f with NAND logic gates only

$$f = \overline{\overline{\bar{A}C}.B}$$

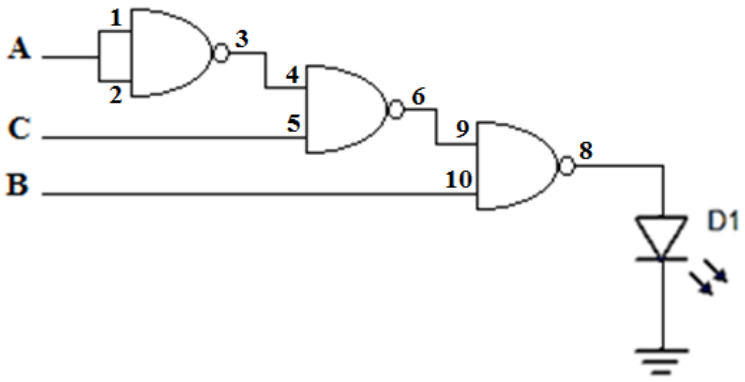


Figure 3.15 : *The f function with NAND logic gates only*

3. Wiring the f function using the TTL 7400 integrated circuit: $f = \overline{\overline{\bar{A}C}.B}$

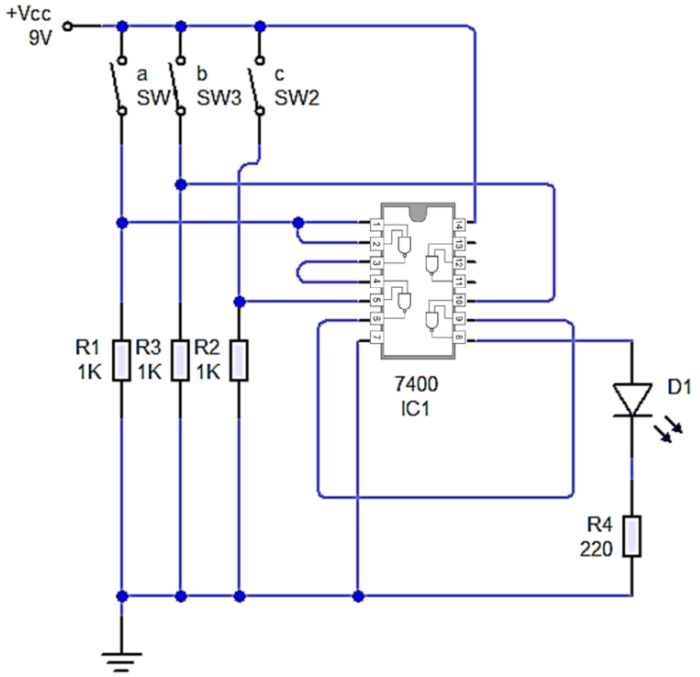


Figure 3.16: *Wiring of function f with a TTL 7400 integrated circuit*

Example 02 :

1. Establish the S function corresponding to the wiring diagram with the integrated circuit:

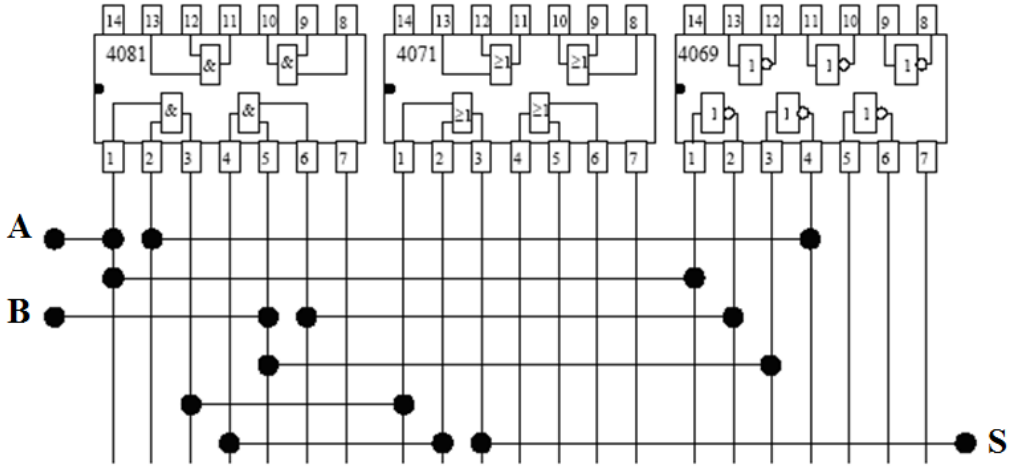


Figure 3.17 : *Wiring the S function with integrated circuits*

The logical function is : $S = \bar{A}.B + A.\bar{B} = A \oplus B$

2. Establish the diagram of the function S based on the logic gates :

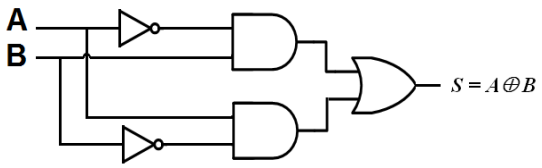
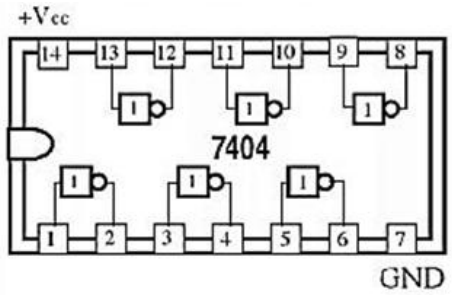
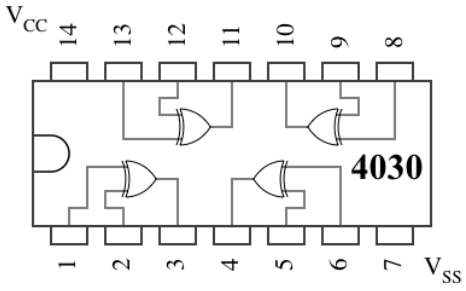
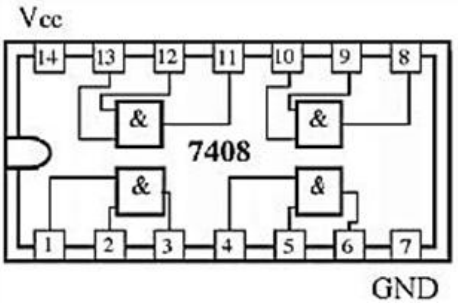
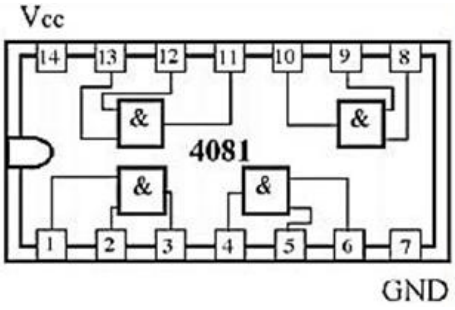
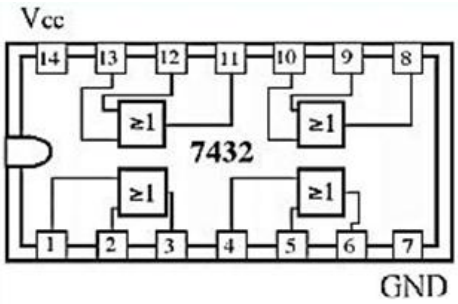
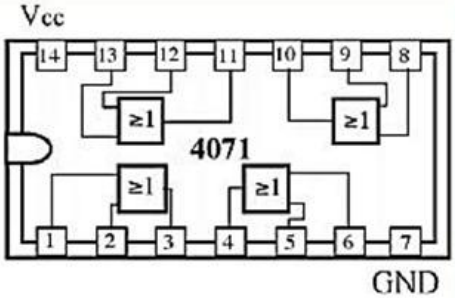
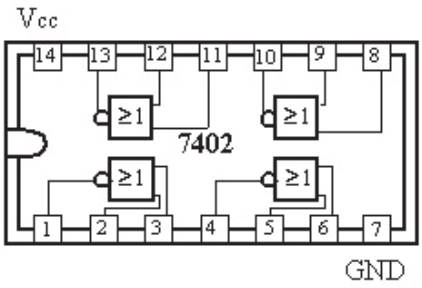
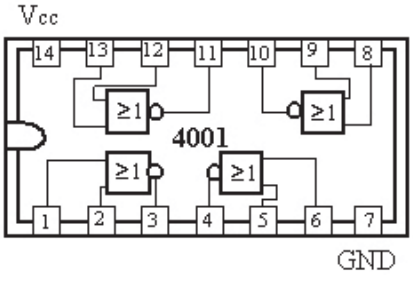
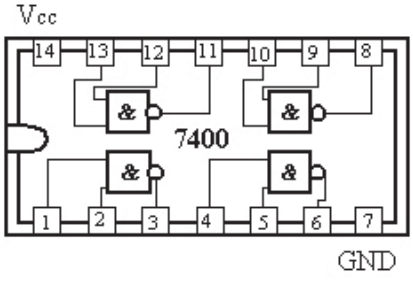
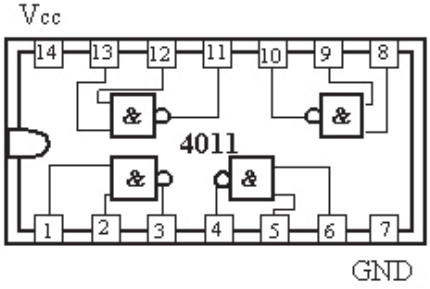
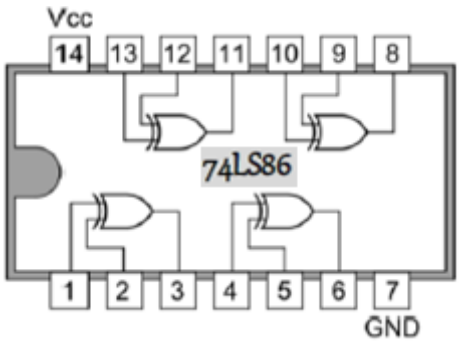
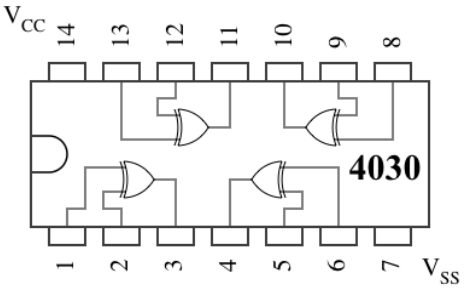
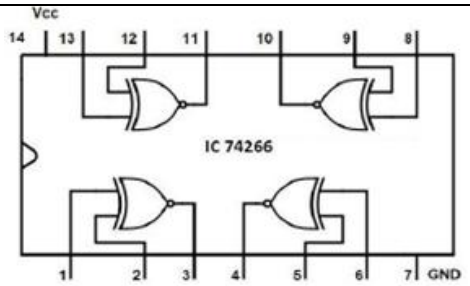


Figure 3.18 : *Flowchart of the S function with logic gates*

3.9. BASIC LOGIC GATES

Operator	*TTL integrated circuits*	*CMOS integrated circuits*
NOT		
AND		
OR		
NOR		
NAND		

XOR	Vcc 14 13 12 11 10 9 8 74LS86 1 2 3 4 5 6 7 GND	V_{CC} 14 13 12 11 10 9 8 4030 1 2 3 4 5 6 7 V_{SS}
NXOR	Vcc 14 13 12 11 10 9 8 IC 74266 1 2 3 4 5 6 7 GND	

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.