ESATIC
ECOLE SUPÉRIEURE AFRICAINE DES TIC

2022-2023

ELECTRONIQUE NUMERIQUE

Cours destiné aux étudiants
en Licence 1 SRIT

Ano KOUADJO

Enseignant – Chercheur à l'ESATIC

Email: ano.kouadjo@esatic.edu.ci

Cel: +2250757140002 / +2250171491111

Plan de L'ECUE

- ❑ Présentation générale
- ❑ Systèmes de numération et codes
- ❑ Algèbre de Boole
- ❑ fonctions combinatoires
- ❑ Systèmes séquentiels

Présentation de L'ECUE

Ce module entre dans la formation Licence 1 afin de permettre aux étudiants d'acquérir les outils nécessaires pour comprendre, analyser et modéliser les systèmes numériques.

L'objectif est donc de maitriser le fondement de la logique combinatoire et séquentiel ainsi que leurs utilisations dans la modélisation et la résolution des problèmes dont fait face un futur technicien.

Objectifs du l'ECUE

- ✓ Comprendre les notions de système de numération et maitriser les opérations arithmétiques binaires
- ✓ Connaitre les concepts de base de l'Algèbre de Boole et des Portes Logiques
- ✓ Maitriser les méthodes de simplification des circuits logiques nécessaires à la synthèse des systèmes combinatoires
- ✓ Comprendre le fonctionnement des circuits séquentiels tels que les bascules, les compteurs et les registres.

Progression de l'ECUE

Séance	Contenu
1	Présentation du module, Systèmes de numération
2	Conversions entre bases, Arithmétique
3	Codes numériques
4	Opérateurs et opérations de base
5	Expressions Booléennes et tables de vérité
6	Logique positive et logique négative, Formes canoniques
7	Simplifications algébriques
8	Conception de circuits combinatoires en utilisant les tables de vérité
9	Minimisation par la méthode de Karnaugh
10	Additionneur, Comparateur
12	Multiplexeur/Demux, Codeur/Décodeur , Transcodeur
13	Circuits séquentiels, La bascule,
14	Conception de compteurs, Les registres à décalage

CHAPITRE 1

Systèmes de numération et codes

Plan du Chapitre

- ✓ Objectif du chapitre
- ✓ Systèmes de numération
- ✓ Changement de base
- ✓ Opérations arithmétiques binaires
- ✓ Opérations arithmétiques hexadécimales
- ✓ Les codes

Objectifs du Chapitre

- ✓ Ce chapitre traite en détail les différents systèmes de numération : **systèmes décimal, binaire, octal et hexadécimal** ainsi que les méthodes de **conversion** entre ces systèmes de numération.
- ✓ Nous étudierons également les **opérations arithmétiques** sur les nombres binaires signés, après avoir introduit la notion du **complément à 2** d'un nombre binaire.
- ✓ Nous terminons ce chapitre par l'étude de quelques codes numériques tels que les codes **B.C.D, Gray et A.S.C.I.I**

Systèmes de numération

Nous avons pris l'habitude de représenter les nombres en utilisant dix symboles différents: 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9.

Ce système est appelé le système **décimal** (déci signifie 10).

Il existe cependant d'autres systèmes de numération. Les plus utilisés en électronique numérique sont:

- **Le système binaire (bi: base 2)**
- **le système octal (oct: base 8)**
- **le système hexadécimal (hexa: base 16)**

Dans un système de numération : le nombre de symboles distincts est appelé **base** du système de numération.

Représentation polynomiale

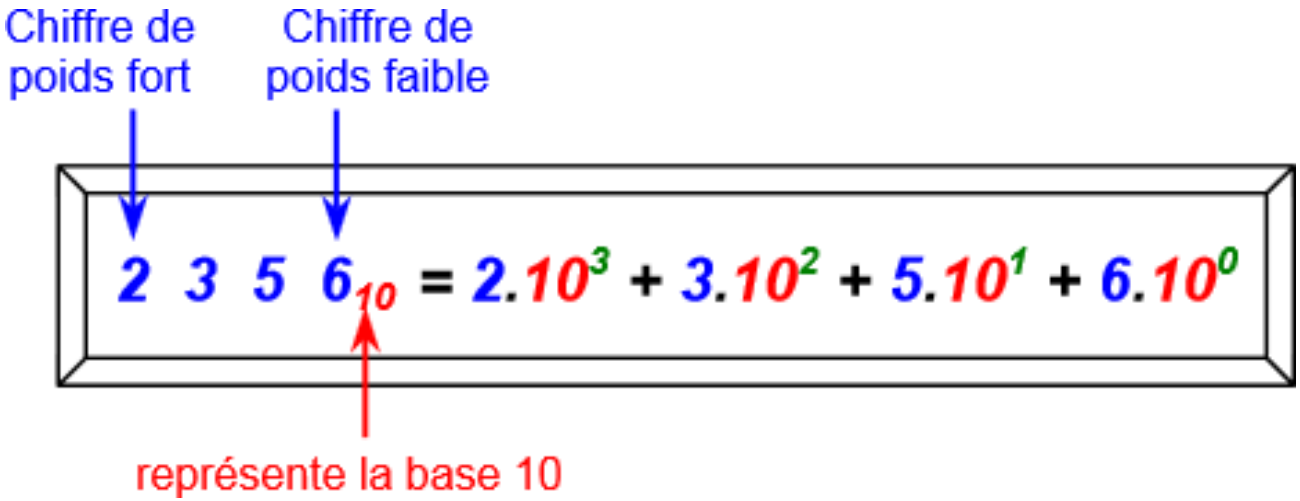
Tout nombre N peut se décomposer en fonction des puissances entières de la base de son système de numération. Cette décomposition s'appelle la forme polynomiale du nombre N et qui est donnée par :

$$N = a_n.b^n + a_{n-1}.b^{n-1} + \ldots\ldots a_1.b^1 + a_0.b^0 + a_{-1}.b^{-1} + \ldots. a_{-m}.b^{-m}$$

- b: base du système de numération, il représente le nombre de chiffres différents qu'utilise ce système de numération.
- a_i: un chiffre (ou digit) parmi les chiffres de la base du système de numération.
- i: Rang du chiffre a_i.

- Système décimal (base 10)

Ce système comprend 10 symboles (chiffres) qui sont {0,1,2,3,4,5,6,7,8,9}.
N'importe quelle combinaison de ces symboles nous donne un nombre.

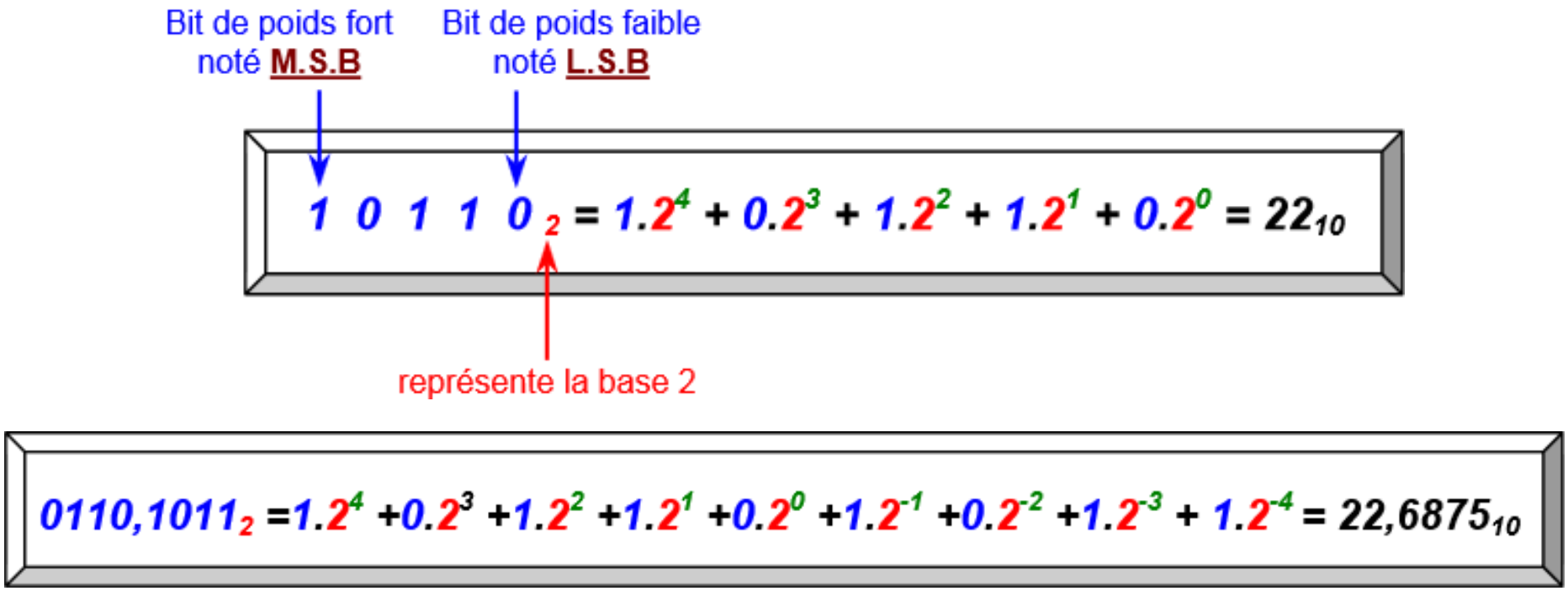


$4\ 2\ 9\ ,\ 6\ 5\ 7_{10} = 4.10^2 + 2.10^1 + 9.10^0 + 6.10^{-1} + 5.10^{-2} + 7.10^{-3}$

- Système binaire (base 2)

Dans ce système de numération, il n'y a que deux chiffres possibles {0,1} qui sont souvent appelés bits "binary digit ".

Exemples :



- Système octal (base 8)

Ce système octal ou à base 8, comprend 8 chiffres qui sont {0,1,2,3,4,5,6,7}. Les chiffres 8 et 9 n'existent pas dans cette base.

Exemples :

$$6057_8 = 6.8^3 + 0.8^2 + 5.8^1 + 7.8^0 = 3119_{10}$$

$$6057,14_8 = 6.8^3 + 0.8^2 + 5.8^1 + 7.8^0 + 1.8^{-1} + 4.8^{-2} = 3119_{10}$$

Ajouter la partie fractionnaire

- Système hexadécimal (base 16)

Le système hexadécimal ou base 16 contient seize éléments qui sont {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}.

Exemples :

$$N_1 = 356_{16} = 3.16^2 + 5.16^1 + 6.16^0 = 854_{10}$$

$$N_2 = 2AF_{16} = 2.16^2 + 10.16^1 + 15.16^0 = 687_{10}$$

$$N_3 = 81,B_{16} = 8.16^1 + 1.16^0 + 11.16^{-1} = 129,6875_{10}$$

Résumé

- Dans une base X , on utilise X symboles distincts pour représenter les nombres.
- La valeur de chaque symbole doit être strictement inférieure à la base X.
- Chaque nombre dans une base X peut être écrit sous sa forme polynomiale .

CODES	BASE	SYMBOLES	UTILISATIONS
Décimal	*10*	*0, 1, 2, 3, 4, 5, 6, 7, 8, 9*	*Usage courant*
Binaire	*2*	*0, 1*	*C'est le code le plus utilisé en électronique numérique* *Ex : '0' = 0 volt '1' = 5 volts*
Octal	*8*	*0, 1, 2, 3, 4, 5, 6, 7*	
Hexadécimal	*16*	*0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F*	*Utilisé en informatique pour simplifier l'écriture des nombres binaires.* *Ex :* $(110001110101)_2 = (C75)_{16}$

Changement de base

Il s'agit du processus de conversion d'un nombre écrit dans une base b1 à une autre base b2

- Conversion d'un nombre en base b →base 10

La valeur décimale d'un nombre N, écrit dans une base b, s'obtient par sa forme polynomiale

Exemples:

$$N_1 = 101101_2 = 1.2^5 + 0.2^4 + 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0 = 45_{10}$$

$$N_2 = 6734_8 = 6.8^3 + 7.8^2 + 3.8^1 + 4.8^0 = 3548_{10}$$

$$N_3 = A73_{16} = 10.16^2 + 7.16^1 + 3.16^0 = 2675_{10}$$

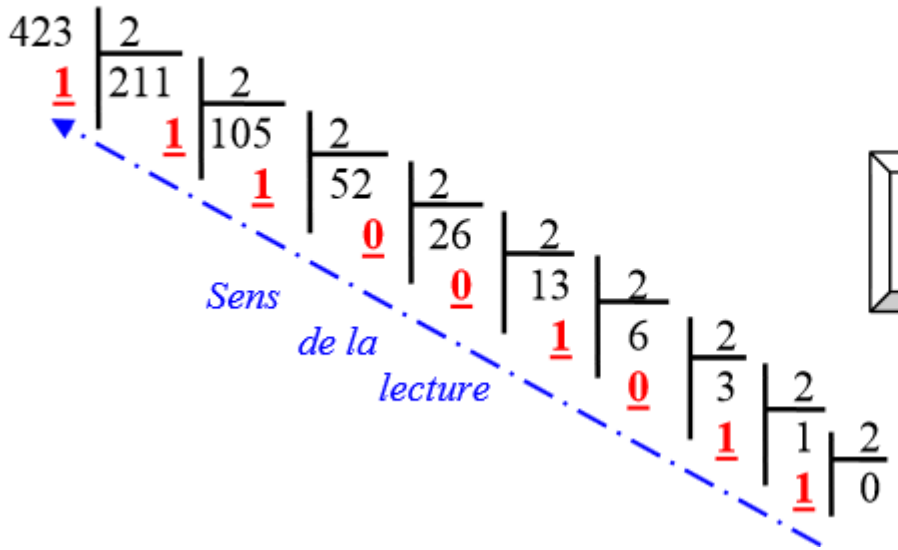
- Conversion d'un nombre en base 10 →base b

Le principe consiste à faire des divisions successives du nombre par la base b , et prendre le reste des divisions dans l'ordre inverse.

Exemples

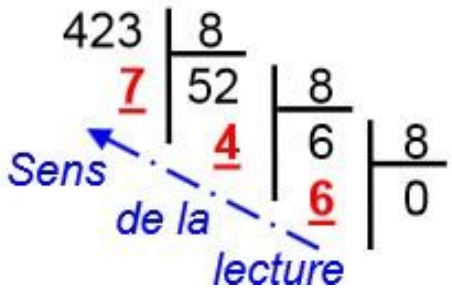
Convertir le nombre décimal $(423)_{10}$ en nombre binaire, octal et hexadécimal

a/ Conversion décimale binaire



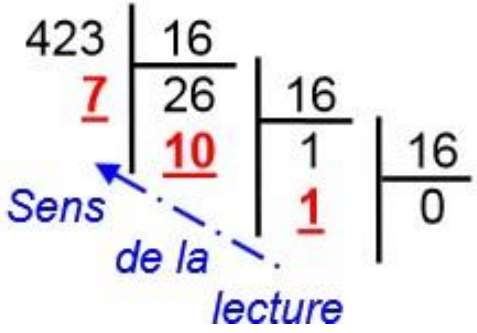
Résultat : $423_{10} = 110100111_2$

b/ Conversion décimale octale



Résultat : $423_{10} = 647_{8}$

c/ Conversion décimale hexadécimale



Résultat : $423_{10} = 1A7_{16}$

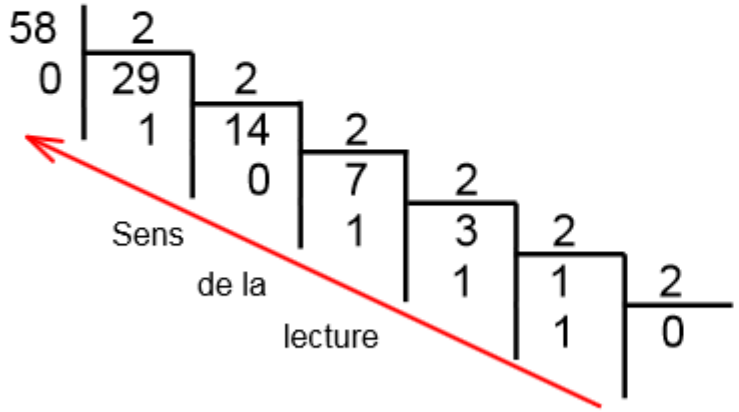
d) Cas des nombres réels

- Pour convertir un nombre décimal à virgule dans une base b quelconque, nous effectuons la division successive par b de la partie entière et on multiplie la partie fractionnaire du nombre à convertir par la base b et on note sa partie entière

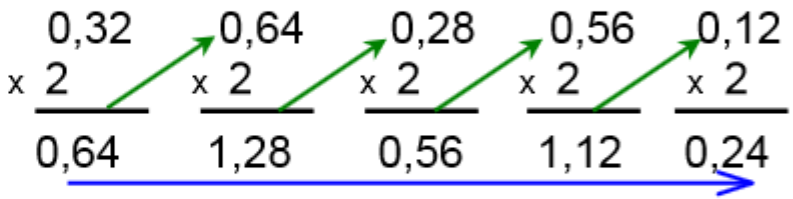
Exemple:

Conversion du nombre $58,32_{10}$ en base 2.

Recherche de la partie entière par divisions successives par 2



Recherche de la partie fractionnaire par multiplications successives par 2



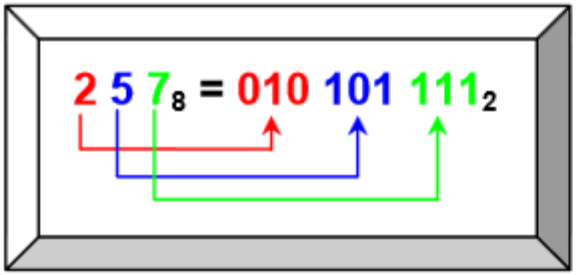
Pour la partie décimale, on s'arrête quand le nombre de bits permet de respecter la précision souhaitée.

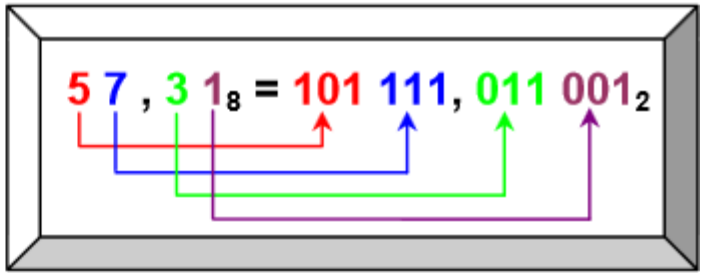
Résultat : $58,32_{10} = 111010,01010\ldots_2$

- Autres conversions : Octal-binaire, Hexadécimal-binaire, et vice versa

a- La conversion octal-binaire s'obtient en remplaçant chaque chiffre du nombre octal par son équivalent binaire écrit sur trois bits.

Exemples:



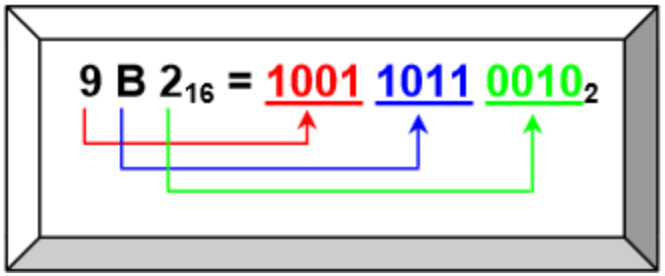


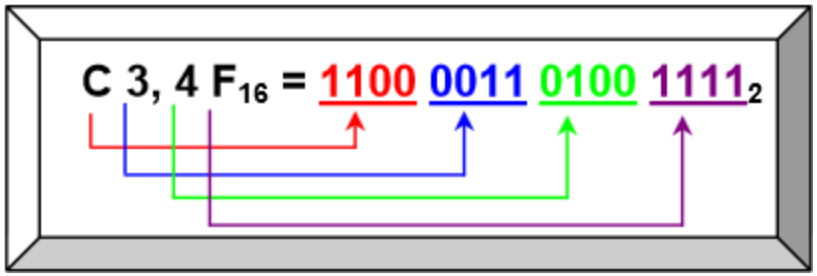
Inversement, pour convertir un nombre binaire en un nombre octal, il faut regrouper les bits du nombre binaire par trois en allant vers la gauche à partir de la virgule pour la partie entière, et vers la droite pour la partie fractionnaire, puis chaque groupe est remplacé par le chiffre octal correspondant.

- b- La conversion hexadécimal-binaire

s'obtient en remplaçant chaque chiffre du nombre hexadécimal par son équivalent binaire sur quatre bits.

Exemples:



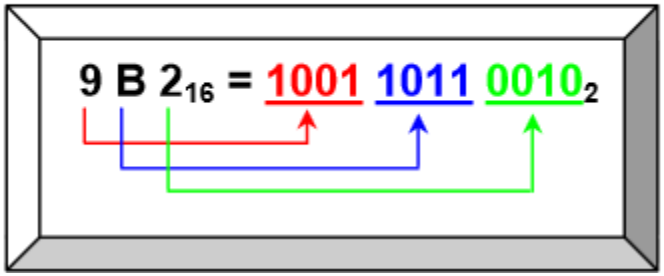


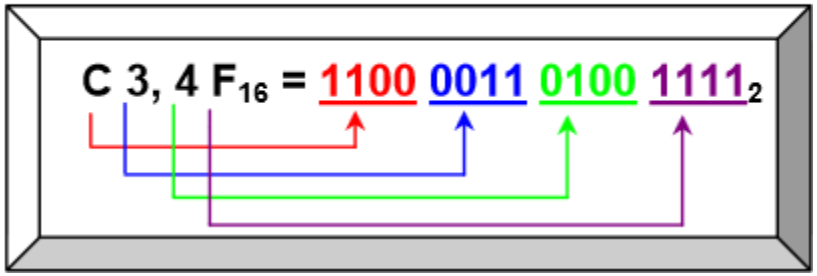
Inversement, pour convertir un nombre binaire en un nombre hexadécimal, il faut regrouper les bits du nombre binaire par quatre en allant vers la gauche à partir de la virgule pour la partie entière, et vers la droite pour la partie fractionnaire, puis chaque groupe est remplacé par le chiffre hexadécimal correspondant.

- b- La conversion hexadécimal-binaire

s'obtient en remplaçant chaque chiffre du nombre hexadécimal par son équivalent binaire sur quatre bits.

Exemples:





Inversement, pour convertir un nombre binaire en un nombre hexadécimal, il faut regrouper les bits du nombre binaire par quatre en allant vers la gauche à partir de la virgule pour la partie entière, et vers la droite pour la partie fractionnaire, puis chaque groupe est remplacé par le chiffre hexadécimal correspondant.

Opérations arithmétiques binaires

Avant d'effectuer des opérations arithmétiques sur les nombres binaires, il faut définir au préalable, la représentation des nombres binaires signés. Le codage des nombres négatifs se fait le plus souvent en notation complément à 2

- **Représentation en complément à 2**

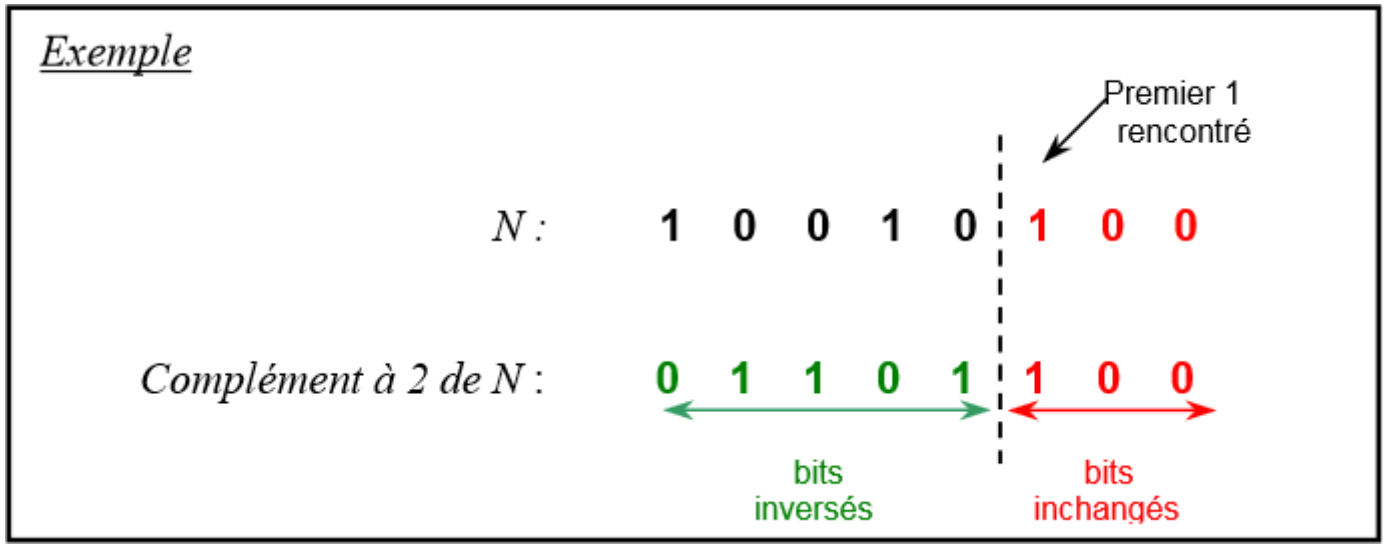
Un nombre binaire signé est écrit en notation complément à 2 comme suit:

- Si le nombre est positif, il est écrit en nombre binaire pur avec un bit de signe 0, représenté par le bit de poids le plus fort.

- Si le nombre est négatif, il possède un bit de signe 1, mais il est écrit en notation en complément à 2. Le complément à 2 d'un nombre binaire est obtenu en changeant chaque 0 par 1 et chaque 1 par 0 (cette étape est appelée complément à 1) et en ajoutant 1 au bit de poids le plus faible.

- Méthode pratique pour le calcul du complément à 2:

Cette méthode consiste à examiner chaque bit du nombre binaire, en commençant par celui de poids le plus faible (L.S.B). Tous les zéros à partir du L.S.B (s'il y en a) jusqu'au premier 1 rencontré seront conservés et tous les bits suivants seront inversés.



- Le tableau ci-dessous donne tous les nombres binaires écrits en notation en complément à 2 sur 4 bits :

Valeur décimale	Nombre binaire en complément à 2
$7=2^3-1$	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
$-8=-2^3$	1000

D'après le tableau ci-dessus, on remarque :

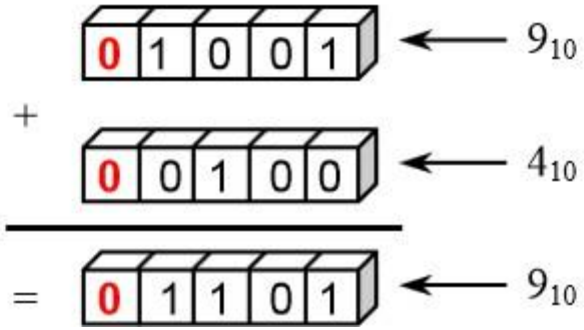
- En notation en complément à 2, les nombres positifs sont représentés avec un bit de signe 0 et les nombres négatifs par un bit de signe 1.
- En notation en complément à 2 et avec n bits, on représente les nombres signés compris dans l'intervalle $[-2^{n-1},2^{n-1}-1]$.

Addition binaire

La méthode d'addition pour des nombres binaires signés, consiste à écrire les nombres positifs en binaire avec un bit de signe 0, et à remplacer les nombres négatifs par leur complément à 2 avant l'addition. Si le résultat est positif, il est en notation binaire, s'il est négatif, il est en notation complément à 2.

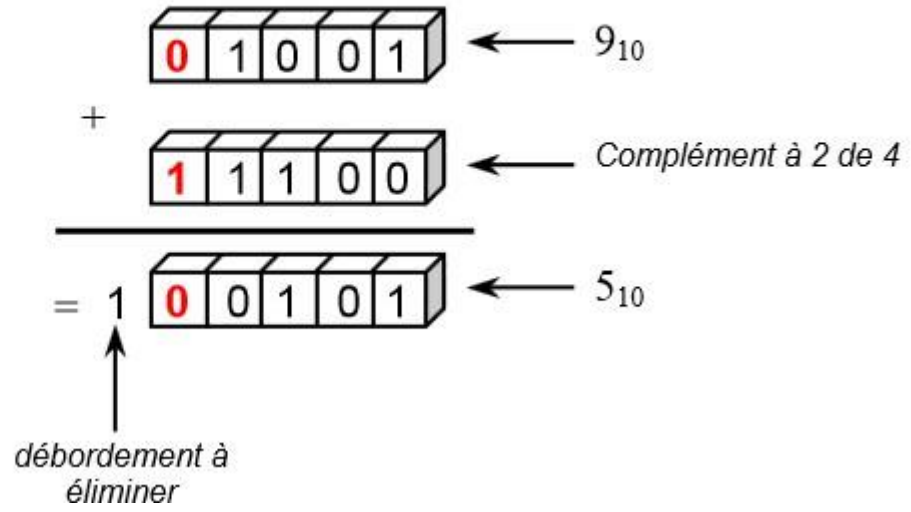
En voici quelques exemples, où les nombres binaires sont écrits avec n=5 bits.

a/ $9_{10} + 4_{10} = ?$



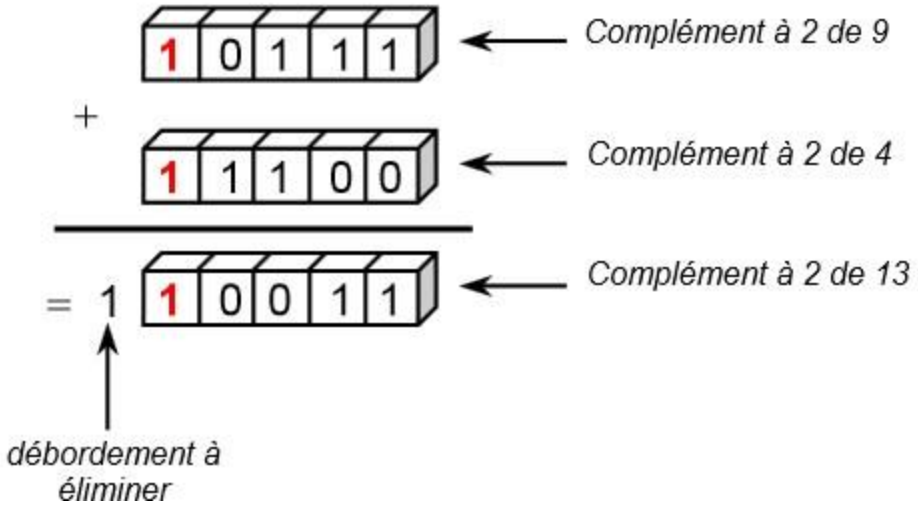
b/ $9_{10} - 4_{10} = ?$

Comme -4_{10} est un nombre négatif, il faut le remplacer par son complément à 2.



c/ $-9_{10} - 4_{10} = ?$

Comme -9_{10} et -4_{10} sont des nombres négatifs, il faut les remplacer par leurs compléments à 2 respectifs.



Le bit de signe est 1, donc le résultat de la somme est négatif, Il faut le complémenter à 2 pour trouver sa valeur absolue.

Remarque: Le débordement sur le (n+1) ème bit du résultat de l'addition est toujours ignoré, car la taille des nombres binaires est limitée à n bits uniquement.

Opérations arithmétiques hexadécimales

Addition hexadécimale

La procédure proposée est la suivante :

- Additionner les deux chiffres hexadécimaux comme des chiffres décimaux, en remplaçant mentalement les lettres par leurs équivalents décimaux.
- Si la somme ≤ 15, inscrire directement le chiffre hexadécimal.
- Si la somme est égale ou supérieur à 16, soustraire 16 et reporter 1 sur le rang à gauche.

Exemples:

```
   5 3        5 4        3 A F        C A F E
+  8 8     +  A 8     +  4 3 C     +  C A F E
```

Soustraction hexadécimale

On peut soustraire les nombres hexadécimaux en utilisant la même méthode que celle pour les nombres décimaux.

Exemples:

```
   C 5 B        1 5 C E        B 4 C 9
-  5 9 2     -    7 D 5     -  9 D 7 A
=  6 C 9     =  0 D F 9     =  1 7 4 F
```

LES CODES

- Codage : opération qui établit une correspondance entre un ensemble source (nombre, caractère, symbole) vers un ensemble destination (but) contenant des combinaisons de 0 et de 1.

L'action de faire correspondre à des nombres, des lettres ou des mots, un groupe spécial de symboles s'appelle codage. On distingue deux types de codes: les codes numériques (codes B.C.D, Gray...) et alphanumériques (code A.S.C.I.I, …).

Code BCD

- chaque chiffre décimal (0,1, . . . , 9) est codé en binaire sur 4 bits
- Code pondéré avec les poids 1,2,4,8,10,20,40,80,100, . . .
- Plus facile pour coder des grands nombre, il est surtout utilisé pour l'affichage des nombres.

Le code B.C.D est un code pondéré qui représente chaque chiffre décimal par son équivalent binaire sur 4 bits, comme le montre le tableau suivant:

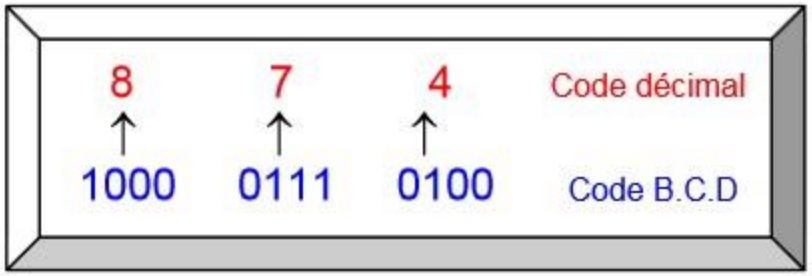
Code décimal	Code B.C.D
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Code BCD

On note que les codes 1010, 1011, 1100, 1101, 1110, 1111 sont des combinaisons interdites, on les appelle des pseudo-tétrades.

Le codage en B.C.D d'un nombre décimal se fait en remplaçant chaque chiffre individuel de ce nombre par son équivalent binaire sur 4 bits.

Exemple:



On note bien la facilité relative avec laquelle on passe de ce code au nombre décimal correspondant et vice versa.

Il y a une méthode qui permet d'effectuer l'addition sur des nombres en B.C.D méthode.

Code de Gray

Le code Gray (ou code binaire réfléchi) est un code qui appartient à la catégorie des codes dit à distance minimale, du fait qu'une représentation codée en Gray ne diffère de celle qui la précède que d'un élément binaire et d'un seul, comme le montre le tableau 1.3.

Code décimal	Code binaire	Code Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Code décimal, binaire et Gray

Code de Gray (suite)

- Ce code est utilisé dans les tableaux de Karnaugh***, dans des*** circuits d'entrée/sortie, et dans certains convertisseurs analogique/numérique.

- Il ne convient pas pour l'arithmétique binaire.

Codage des caractères

- Les caractères englobent : les lettres alphabétiques (A-Z a-z) , les chiffres (0,…9) , et les autres symboles : ponctuations, caractères spéciaux (> , ; / : ….) .
- Le codage revient à créer une table de correspondance entre les caractères et les nombres.
- Le codage le plus utilisé est le ASCII (American Standard Code for Information Interchange)

Code ASCII

Dans ce codage chaque caractère est représenté sur 7 bits .

Avec 7 bits on peut avoir $2^7 = 128$ combinaisons

Chaque combinaison représente un caractère

Exemple :

Le code 65 $(1000001)_2$ correspond au caractère **A**

Le code 97 (1100001) correspond au caractère **a**

Le code 58 (0111010)correspond au caractère **:**

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

- **Table ASCII Etendu**
  - 8 bits pour représenter 256 caractères (0 à 255)
  - Code les caractères accentués : à, è,...etc.
  - Compatible avec ASCII
- **Code Unicode (mis au point en 1991)**
  - 16 bits pour représenter 65 536 caractères (0 à 65 535)
  - Compatible avec ASCII
  - Code la plupart des alphabets : Arabe, Chinois, ...
  - On en a défini environ 50 000 caractères pour l'instant

Code ASCII Etendu

DECIMAL VALUE	➡	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
⬇	HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	BLANK (NULL)	►	SP	0	@	P	‘	p	Ç	É	á	▒	└	╨	∝	≡
1	1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	3	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	7	BEL	↨	’	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	8	BS	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	9	HT	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
10	A	LF	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	B	VT	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
12	C	FF	FS	,	<	L	\	l	¦	î	£	¼	╝	╠	▄	∞	n
13	D	CR	GS	—	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	²
14	E	♫	RS	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	∈	■
15	F	☼	US	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	ⁿ

CHAPITRE 2

Algèbre de Boole

Plan du chapitre

- Objectif du chapitre
- Opérateurs de l'algèbre de Boole
- Théorèmes de l'algèbre de Boole
- Représentation d'une fonction logique
- Simplification des expressions logiques

Objectif du chapitre

Pour étudier d'une manière systématique les circuits numériques, on utilise une algèbre différente de l'algèbre classique, dite **algèbre de Boole**, du nom du mathématicien anglais, inventeur de ce concept (George Boole 1815-1864).

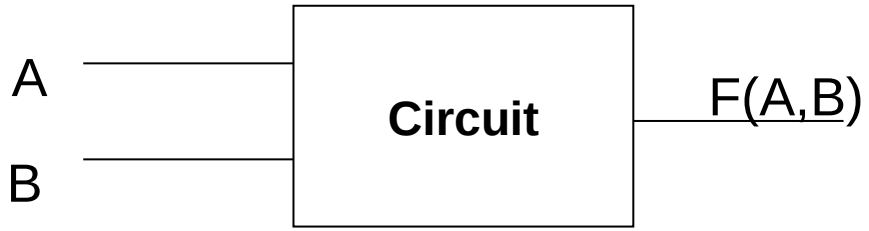
Nous proposons dans ce chapitre les lois, règles et théorèmes

de l'algèbre de Boole, nécessaires et suffisants pour la compréhension du fonctionnement de ces circuits numériques.

Nous étudierons également la simplification des expressions booléennes en utilisant les règles de l'algèbre de Boole et les **diagrammes de Karnaugh**

Introduction

- Les machines numériques sont constituées d'un ensemble de **circuits électroniques**.
- Chaque circuit fournit une **fonction logique** bien déterminée (addition, comparaison ,….).



La fonction F(A,B) peut être : la somme de A et B , ou le résultat de la comparaison de A et B ou une autre fonction.

- Pour concevoir et réaliser ce circuit on doit avoir un modèle mathématique de la fonction réalisée par ce circuit.
- Ce modèle doit prendre en considération le système binaire.
- Le modèle mathématique utilisé est celui de Boole.

Introduction

- George Boole est un mathématicien anglais (1815-1864).
- Il a fait des travaux dans lesquels les fonctions(expressions) sont constitués par des variables qui peuvent prendre les valeurs ‘OUI’ ou ‘NON’ .
- Ces travaux ont été utilisés pour faire l’étude des systèmes qui possèdent deux états qui s’exclus mutuellement :
  - Le système peut être uniquement dans deux états E1 et E2 tel que E1 est l’opposé de E2.
  - Le système ne peut pas être dans l’état E1 et E2 en même temps
- Ces travaux sont bien adaptés au Système binaire (0 et 1).

Exemple de systèmes à deux états

- Un interrupteur est ouvert ou non ouvert (fermé)
- Une lampe est allumée ou non allumée (éteinte)
- Une porte est ouverte ou non ouverte (fermée)

- **Remarque :**

On peut utiliser les conventions suivantes :

OUI → VRAI (true)
NON → FAUX (false)

OUI → 1 (Niveau Haut)
NON → 0 (Niveau Bas)

Définitions et conventions

- **Niveau logique : Lorsqu'on fait l'étude d'un système logique il faut bien préciser le niveau du travail.**

Niveau	Logique positive	Logique négative
H (Hight) haut	1	0
L (Low) bas	0	1

Exemple :

Logique positive :

lampe allumée : 1

lampe éteinte : 0

Logique négative

lampe allumée : 0

lampe éteinte : 1

Variable logique (booléenne)

- Une variable logique (booléenne) est une variable qui peut prendre soit la valeur 0 ou 1 .
- Généralement elle est exprimée par un seul caractère alphabétique en majuscule (A , B, S , …)
- **Exemple :**
  - Une lampe : allumée L = 1
    éteinte L = 0
  - interrupteur ouvert : I =1
    fermé : I =0

Fonction logique

- C'est une fonction qui relie N variables logiques avec un ensemble d'opérateurs logiques de base.

- Dans l'Algèbre de Boole il existe trois opérateurs de base :
  - Le produit logique, dit aussi **ET**. Le symbole de cette opération est (.)
  - L'addition logique, dite aussi **OU**. Le symbole de cette opération est (+).
  - L'inverse logique, dite aussi **NON**. Le symbole de cette
- opération est (--).

Fonction logique

La valeur d'une fonction logique est égale à 1 ou 0 selon les valeurs des variables logiques.

Si une fonction logique possède N variables logiques → 2^n combinaisons → la fonction possède 2^n valeurs.

Les 2^n combinaisons sont représentées dans une table qui s'appelle table de vérité (TV).

Exemple d'une fonction logique

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

La fonction possède 3 variables → 2^3 combinaisons

$$F(0,0,0) = \overline{0}.\overline{0}.0 + \overline{0}.0.0 + 0.\overline{0}.0 + 0.0.0 = 0$$
$$F(0,0,1) = \overline{0}.\overline{0}.1 + \overline{0}.0.1 + 0.\overline{0}.1 + 0.0.1 = 1$$
$$F(0,1,0) = \overline{0}.\overline{1}.0 + \overline{0}.1.0 + 0.\overline{1}.0 + 0.1.0 = 0$$
$$F(0,1,1) = \overline{0}.\overline{1}.1 + \overline{0}.1.1 + 0.\overline{1}.1 + 0.1.1 = 1$$
$$F(1,0,0) = \overline{1}.\overline{0}.0 + \overline{1}.0.0 + 1.\overline{0}.0 + 1.0.0 = 0$$
$$F(1,0,1) = \overline{1}.\overline{0}.1 + \overline{1}.0.1 + 1.\overline{0}.1 + 1.0.1 = 1$$
$$F(1,1,0) = \overline{1}.\overline{1}.0 + \overline{1}.1.0 + 1.\overline{1}.0 + 1.1.0 = 0$$
$$F(1,1,1) = \overline{1}.\overline{1}.1 + \overline{1}.1.1 + 1.\overline{1}.1 + 1.1.1 = 1$$

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

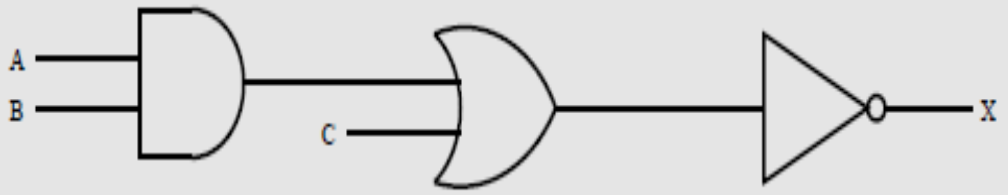
Une table de vérité

Logigramme

- Un logigramme est un schéma illustrant l'expression d'une fonction logique sans tenir compte des constituants technologiques.

Exemple :

$X=\overline{A.B+C}$ $\longrightarrow$



Opérateurs de l'algèbre de Boole

Operateur ET « AND »

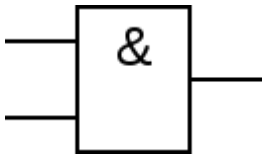
Si deux variables logiques A et B sont combinées par la multiplication logique (ET), le résultat s'exprime ainsi :

$$X = A.B$$

Cela se traduit par l'expression suivante :
Si A est vraie ET B est vraie alors X est vraie.

A	B	X=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité de l'opérateur ET



Operateur OU « OR »

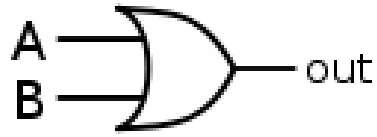
L'addition logique de deux variables A et B donne :

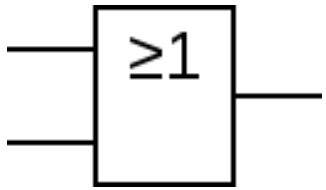
$$X = A+B$$

Cela se traduit par l'expression suivante : X est vrai si au moins A ou B est vrai.

A	*B*	*X=A+B*
0	0	0
0	1	1
1	0	1
1	1	1

Table de vérité de l'opérateur OU





Operateur inverseur

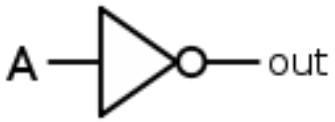
L'opérateur inverseur, contrairement aux opérateurs précédents, est un opérateur à une seule variable. Le résultat X de l'opérateur inverseur sur une variable booléenne A donne son complément. On note :

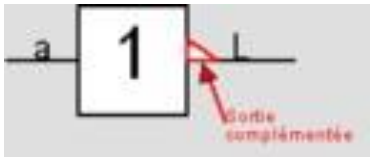
$$X = \bar{A}$$

La table de vérité de l'opérateur inverseur est donnée par le tableau suivant.

A	$X=\bar{A}$
0	1
1	0

Table de vérité de l'opérateur inverseur

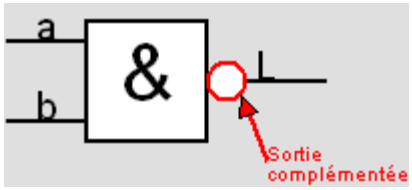




Operateur NON ET « NAND »

- La fonction **NON-ET** (*NAND* en anglais) est un opérateur logique de l'algèbre de Boole. À deux opérandes, qui peuvent avoir chacun la valeur VRAI ou FAUX, il associe un résultat qui a lui-même la valeur VRAI seulement si au moins l'un des deux opérandes a la valeur FAUX.

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

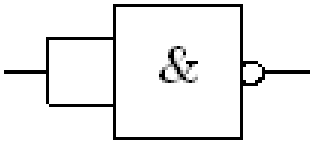


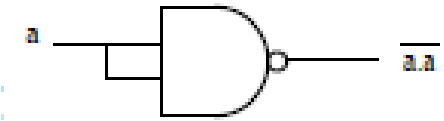
Entrées		Sortie
a	**b**	**L**
0	0	1
0	1	1
1	0	1
1	1	0

Operateur NON ET « NAND »

- NON

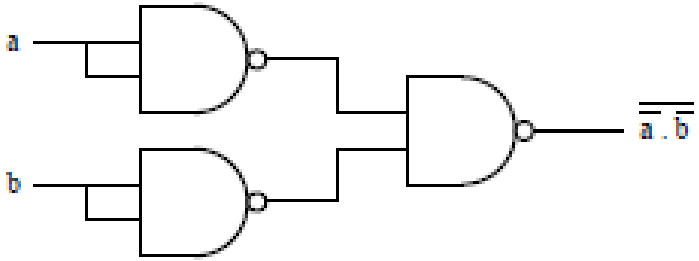
$$S = \overline{a \cdot a} = \bar{a}$$

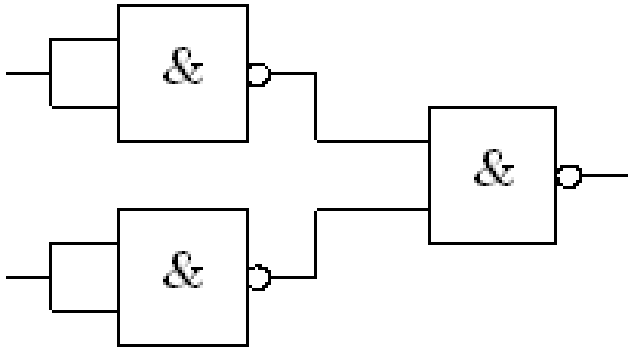




- OU

$$S = \overline{\overline{(a \cdot a)} \cdot \overline{(b \cdot b)}} = \overline{\overline{(a)} \cdot \overline{(b)}} = \overline{\overline{(a)}} + \overline{\overline{(b)}} = a + b$$

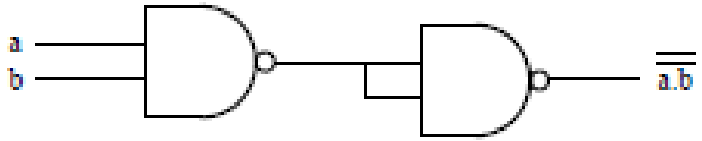


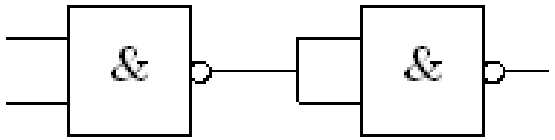


Operateur NON ET « NAND »

- ET

$$S = \overline{\overline{(a \cdot b)} \cdot \overline{(a \cdot b)}} = \overline{\overline{(a \cdot b)}} = a \cdot b$$

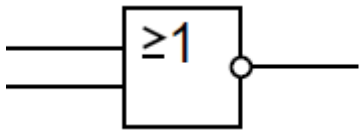


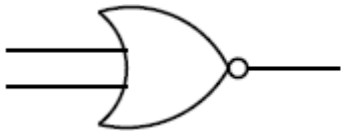


Operateur Non OU « NOR »

- La fonction **NON-OU** (*NOR* en anglais) est un opérateur logique de l'algèbre de Boole. À deux opérandes, qui peuvent avoir chacun la valeur VRAI ou FAUX, il associe un résultat qui a lui-même la valeur VRAI seulement si les deux opérandes ont la valeur FAUX.
- Cette fonction logique correspond aux mots français *ni... ni*, car la phrase

ni A ni B est vrai seulement si les phrases A et B sont tous les deux faux.

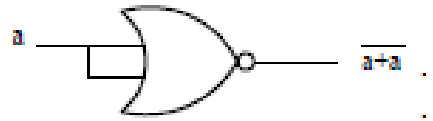




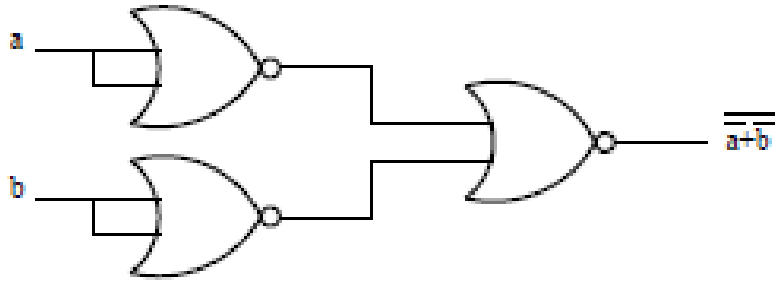
$$S=\overline{A+B}$$

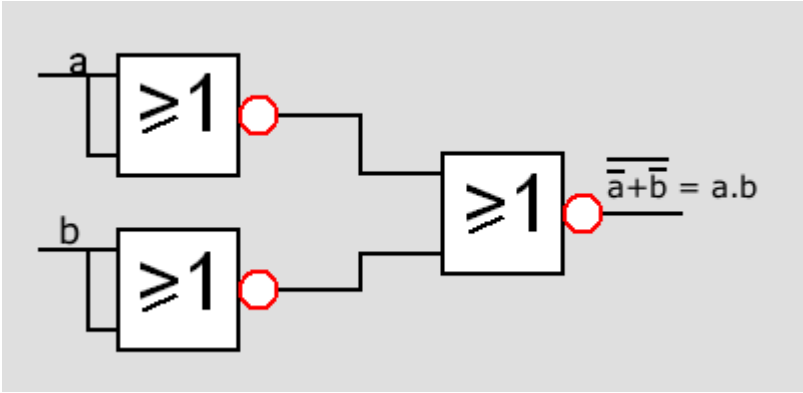
Operateur NON OU « NOR »

- NON

NON : 

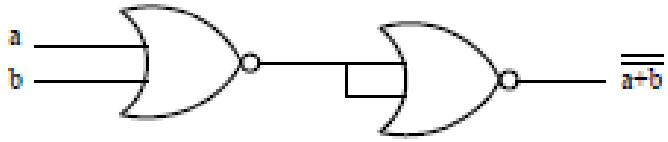
- ET

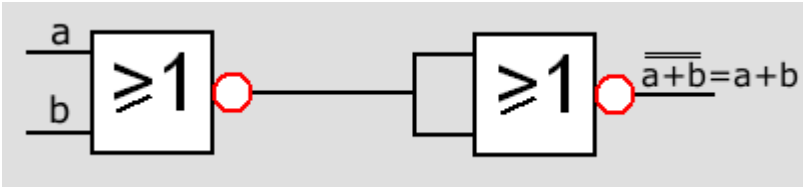




Operateur NON OU « NOR »

- OU

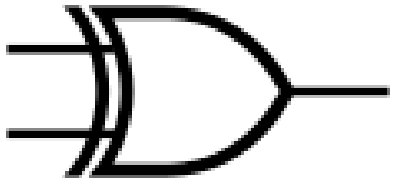


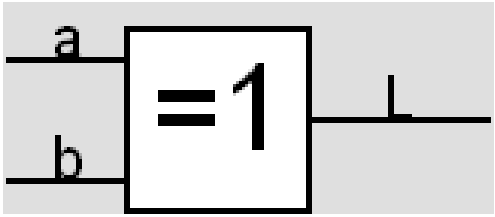


OU exclusif (XOR)

- La fonction **OU exclusif**, souvent appelée **XOR** (e**X**clusive **OR**) ou **disjonction exclusive**, est un opérateur logique de l'algèbre de Boole. À deux opérandes, qui peuvent avoir chacun la valeur VRAI ou FAUX, il associe un résultat qui a lui-même la valeur VRAI seulement si les deux opérandes ont des valeurs distinctes.

  •Cet opérateur est très utilisé en électronique, en informatique, et aussi en cryptographie du fait de ses propriétés intéressantes. Son symbole est traditionnellement un signe "PLUS" dans un cercle : « ⊕ », et parfois "FOIS" dans un cercle : « ⊗ ».





OU exclusif (XOR)

- Appelons A et B les deux opérandes considérés. Convenons de représenter leur valeur ainsi :
- 1 = VRAI

0 = FAUX

- L'opérateur XOR est défini par sa table de vérité, qui indique pour toutes les valeurs possibles de A et B la valeur du résultat R :

Table de vérité de XOR		
A	**B**	**R = A ⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

ET inclusif (XNOR)

- En informatique, l'opérateur logique ET inclusif appelé également Coïncidence (ou équivalence logique) ainsi que NON-OU exclusif peut se définir par la phrase suivante:
- « La sortie est VRAI si et seulement si les deux entrées sont identiques »

On peut donc noter qu'il s'agit de la négation du OU exclusif souvent noté XNOR. On le nomme parfois (bien qu'abusivement) *identité*.

- Son symbole est traditionnellement un point ("DOT" en anglais) dans un cercle : « ⊙ ».

ET inclusif (XNOR)

- Appelons A et B les deux opérandes considérés. Convenons de représenter leur valeur ainsi :
  - 1 = VRAI
  - 0 = FAUX
- L'opérateur XNOR est défini par sa table de vérité, qui indique pour toutes les valeurs possibles de A et B la valeur du résultat S :

Entrée		**Sortie**
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Théorèmes de l'algèbre de Boole

Théorème de commutativité	$A + B = B + A$ $A.B = B.A$
Théorème d'associativité	$A + (B + C) = (A + B) + C = A + B + C$ $A.(B.C) = (A.B).C = A.B.C$
Théorème de distributivité	$A.(B + C) = A.B + A.C$ $A + (B.C) = (A + B).(A + C)$
Théorème d'idempotence	$A + A = A$ $A.A = A$
Théorème de complémentation	$A + \overline{A} = 1$ $A.\overline{A} = 0$
Théorème des constantes	$A.1 = A$ $A + 0 = A$ $A + 1 = 1$ $A.0 = 0$

Théorèmes de l'algèbre de Boole

Théorème d'absorption	$A + A.B = A$ $A.(A + B) = A$
Théorème d'allègement	$A+\overline{A}.B = A+B$ $A.(\overline{A} + B) = A.B$
Théorème de Morgan	$\overline{A.B} = \overline{A} + \overline{B}$ $\overline{A+B} = \overline{A} . \overline{B}$ *Ce théorème peut être généralisé à plusieurs variables :* $\overline{A.B.....Z} = \overline{A} + \overline{B} +.....+ \overline{Z}$ $\overline{A+B+....+Z} = \overline{A}. \overline{B}.....\overline{Z}$

Représentation d'une fonction logique

- **Représentation algébrique**

Formes	*Exemples*
Somme Une fonction est écrite sous la forme de somme, si elle est constituée de plusieurs termes reliés entre eux par l'opérateur OU (appelée aussi forme disjonctive).	$X1=A+B$ $X2=A+B.\overline{C}$ $X3=A.(\overline{B}+C)+B.D.(\overline{A+C})$
Produit Une fonction est écrite sous la forme de produit, si elle est constituée de plusieurs facteurs reliés entre eux par l'opérateur ET (appelée aussi forme conjonctive).	$X1=A.B,$ $X2=A(B+C).(D+E)$ $X3=(A+\overline{B}).(D+C)$
Somme canonique (Σ.Π) Une fonction logique est écrite sous forme de somme canonique si toutes les variables figurent dans chaque terme et si, dans chacun de ces termes, toutes les variables sont reliées entre elles par l'opérateur ET. Ces termes se désignent sous le nom de **mintermes**.	*Soient les fonctions à 3 variables :* $X_1=\overline{A}.B.C+\overline{A}.\overline{B}.C$ $X_2=A.B.C+\overline{A}.\overline{B}.\overline{C}+A.\overline{B}.C$
Produit canonique (Π.Σ) Une fonction logique est écrite sous forme de produit canonique si toutes les variables figurent dans chaque produit et si, dans chacun de ces termes, elles sont toutes reliées entre elles par l'opérateur OU. Ces produits se désignent sous le nom de **maxtermes**.	*Soient les fonctions à 3 et à 4 variables suivants :* $X1=(A+\overline{B}+C+D).(\overline{A}+B+\overline{C}+\overline{D})$ $X_2=(\overline{A}+\overline{B}+\overline{C}).(\overline{A}+B+C).(\overline{A}+\overline{B}+C)$

- **Représentation par une table de vérité**

Une expression logique X(A, B, C...), fonction des variables A, B, C..., peut être représentée par une table de vérité. Cette table donne les valeurs que peut prendre X, suivant les différentes combinaisons des variables A, B, C...

Exemple
: Soit X(A,B,C) une fonction logique à trois variables, représentée par la table de vérité suivante:

C	B	A	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1ère forme canonique: Somme canonique (ΣΠ)

X est vrai si

C=0 et B=0 et A=0 *(ligne-0)*
ou bien
C=0 et B=1 et A=0 *(ligne-2)*
ou bien
C=1 et B=0 et A=1 *(ligne-5)*
ou bien
C=1 et B=1 et A=1 *(ligne-7)*

Par conséquent X est vrai (X=1) si :

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

C'est un développement en somme de produits chaque terme "produit" s'appelle minterme.

2ème forme canonique: Produit canonique (ΠΣ)

En s'intéressant aux 0 de X, on peut écrire de la même manière :

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

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

En appliquant le théorème de Morgan on obtient :

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

C'est un développement en produit de sommes chaque terme "somme" s'appelle maxterme.

Simplification des expressions logiques

- La simplification d'une expression logique, consiste à réduire cette expression à sa forme la plus simple mais équivalente, c'est à dire à un nombre minimal de termes et à un nombre minimal de variables dans chaque terme.

- Nous exposons dans ce qui suit deux méthodes de simplification :
  - Simplification algébrique.
  - Simplification graphique par le diagramme de Karnaugh

Simplification

Exemple-1:

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

or $X + \overline{X}.Y = X + Y$

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

Exemple-2:

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

(0 sous $\overline{A}.A.\overline{D}$, 0 sous $\overline{A}.D.\overline{D}$, 0 sous $B.D.\overline{D}$)

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

Diagramme de Karnaugh

- Le diagramme de Karnaugh d'une table de vérité ou d'une fonction logique à N variables, est constitué d'un rectangle divisé en 2^N cases. Chaque case du diagramme correspond à l'une des 2^N combinaisons possibles des N variables (2^N mintermes).
- L'ordre des variables en abscisse et en ordonnée est choisi de telle sorte qu'entre deux cases adjacentes, il n'y a qu'une seule variable qui change de valeur (on codifie le tableau selon le code de Gray).

- Table de vérité à deux variables

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

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

- Table de vérité à trois variables

C	B	A	X
0	0	0	1
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	0

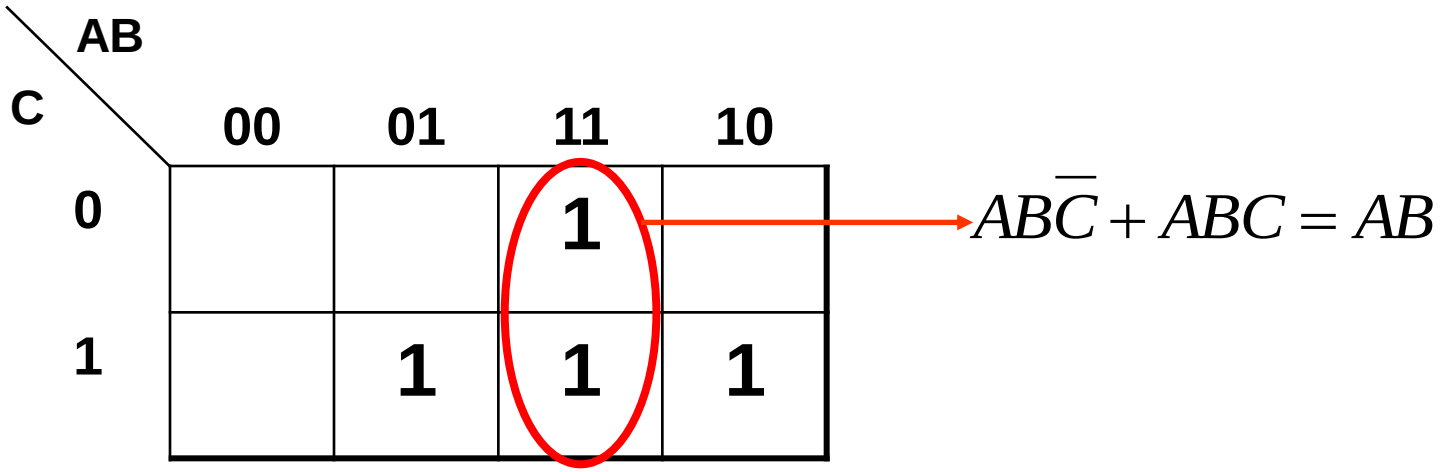
	$\overline{A}.\overline{B}$	$\overline{A}.B$	$A.B$	$A.\overline{B}$
$\overline{C}$	1	0	0	0
C	1	1	0	0

Méthode de simplification (Exemple : 3 variables)

•L'idée de base est d'essayer de regrouper (faire des regroupements) les cases adjacentes qui comportent des 1 (rassembler les termes adjacents).

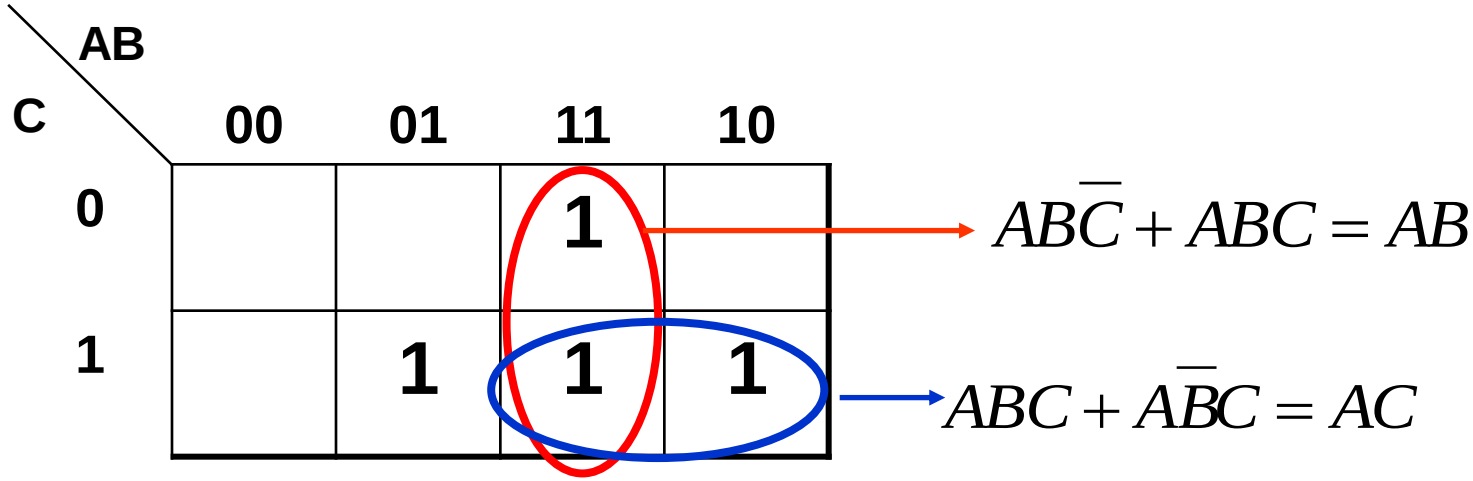
•Essayer de faire des regroupements avec le maximum de cases (16,8,4 ou 2)

•Dans notre exemple on peut faire uniquement des regroupements de 2 cases .

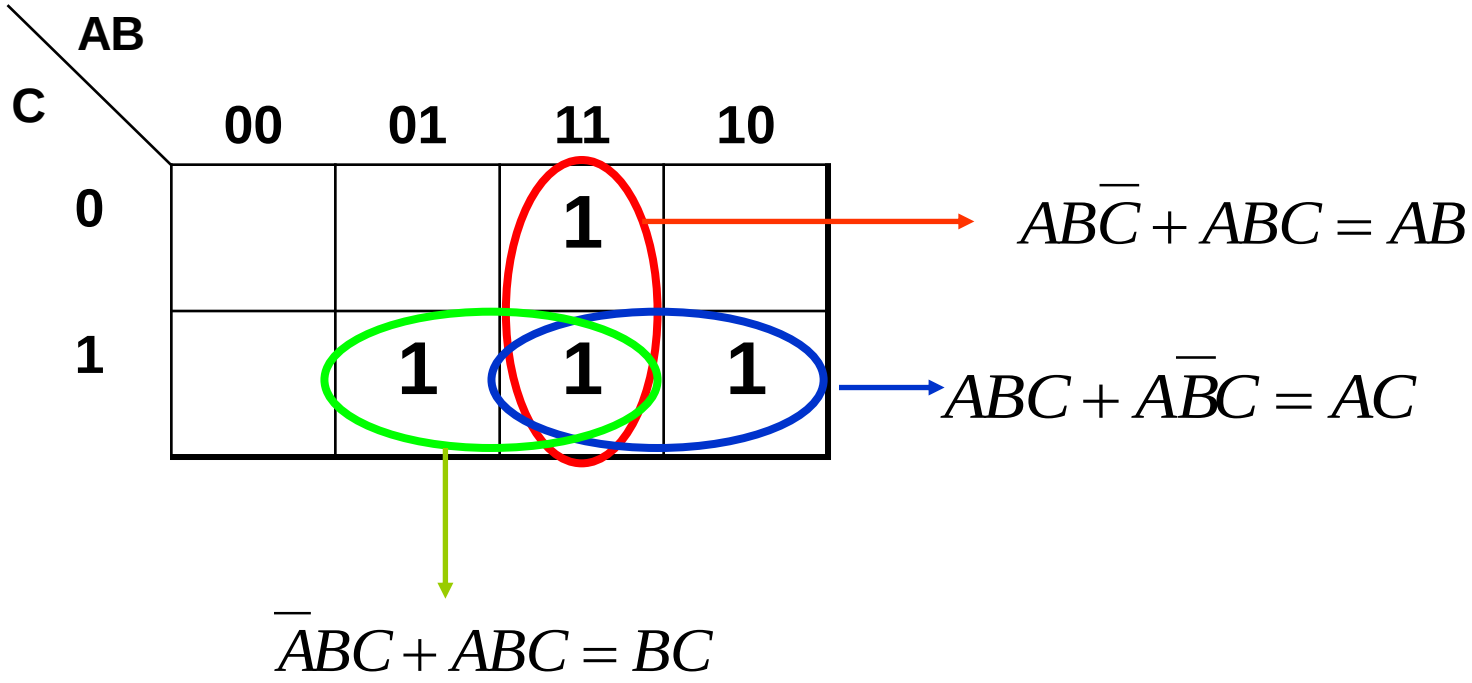


•Puisque il existent encore des cases qui sont en dehors d'un regroupement on refait la même procédure : former des regroupements.

•Une case peut appartenir à plusieurs regroupements



•On s’arrête lorsque il y a plus de 1 en dehors des regroupements
•La fonction final est égale à la réunion (somme) des termes après simplification.



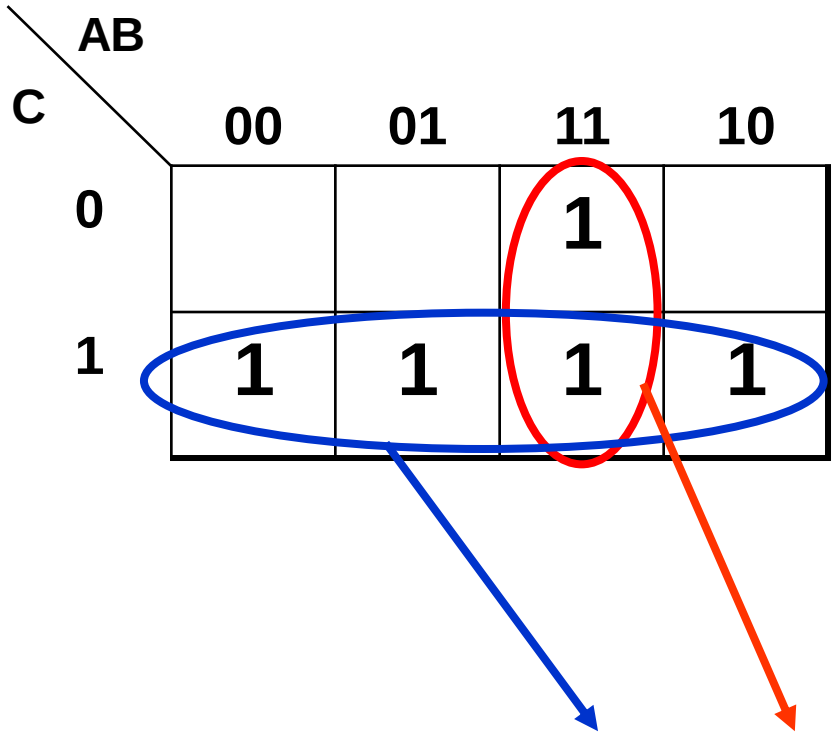
$$F(A,B,C) = AB + AC + BC$$

- **Méthode de simplification**

Donc , en résumé pour simplifier une fonction par la table de karnaugh il faut suivre les étapes suivantes :

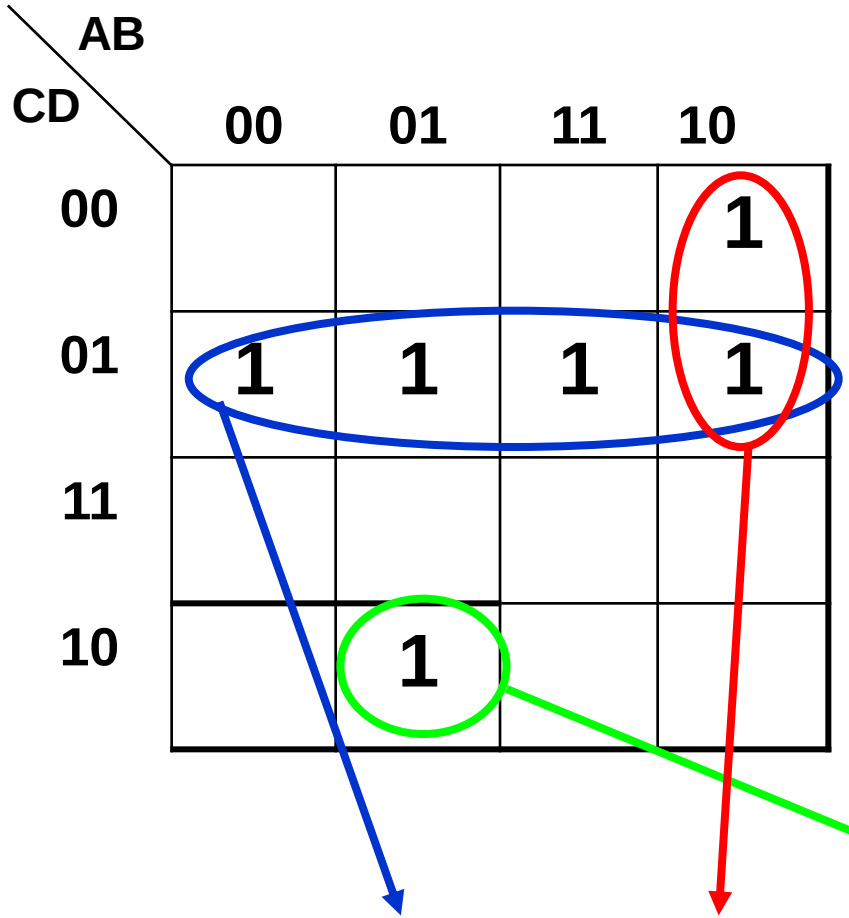
1. Remplir le tableau à partir de la table de vérité ou à partir de la forme canonique.
2. Faire des regroupements : des regroupements de 16,8,4,2,1 cases (Les même termes peuvent participer à plusieurs regroupements) .
3. Dans un regroupement :
   - Qui contient un seule terme on ne peut pas éliminer de variables.
   - Qui contient deux termes on peut éliminer une variable (celle qui change d'état).
   - Qui contient 4 termes on peut éliminer 2 variables.
   - Qui contient 8 termes on peut éliminer 3 variables.
   - Qui contient 16 termes on peut éliminer 4 variables.
5. L'expression logique finale est la réunion (la somme) des groupements après simplification et élimination des variables qui changent d'état.

Exemple 1 : 3 variables



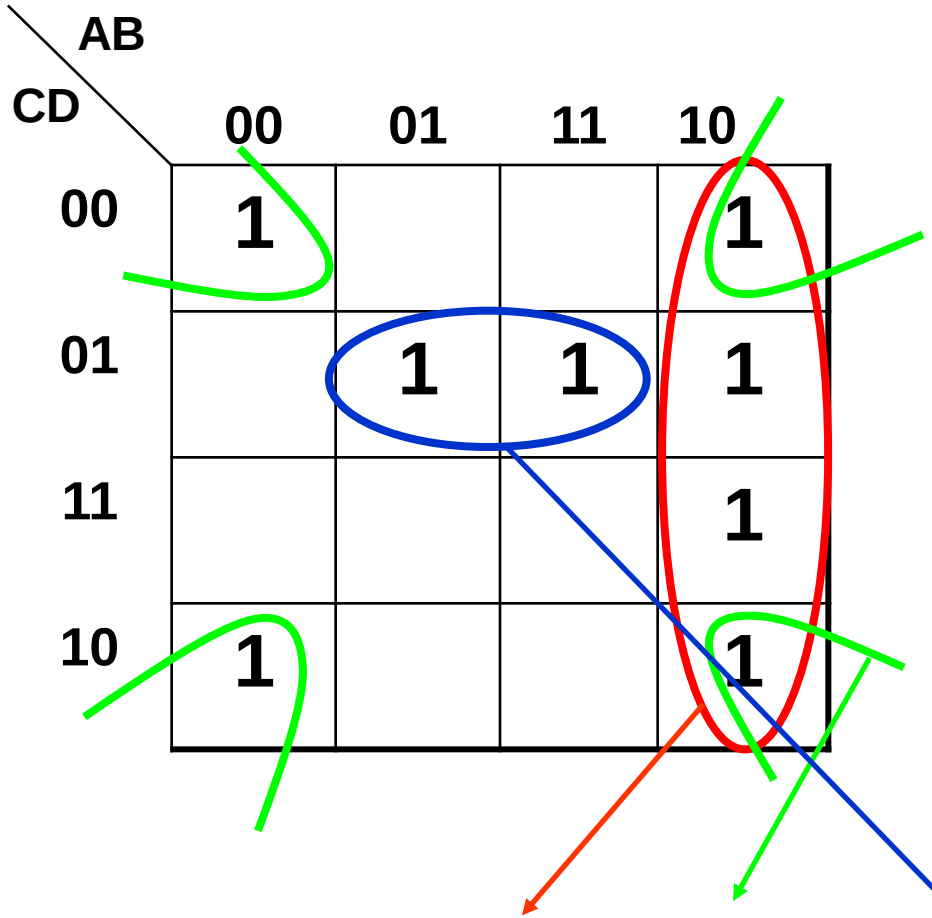
$$F(A,B,C) = C + AB$$

Exemple 2 : 4 variables



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

Exemple 3 : 4 variables



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

Cas d'une fonction non totalement définie

• Examinons l'exemple suivant :

Une serrure de sécurité s'ouvre en fonction de quatre clés A, B, C D. Le fonctionnement de la serrure est définie comme suite :

S(A,B,C,D)= 1 si au moins deux clés sont utilisées

S(A,B,C,D)= 0 sinon

Les clés A et D ne peuvent pas être utilisées en même temps.

•On remarque que si la clé A et D sont utilisées en même temps l'état du système n'est pas déterminé.

•Ces cas sont appelés cas impossibles ou interdites → comment représenter ces cas dans la table de vérité ?.

•Pour les cas impossibles ou interdites

il faut mettre un X dans la T.V .

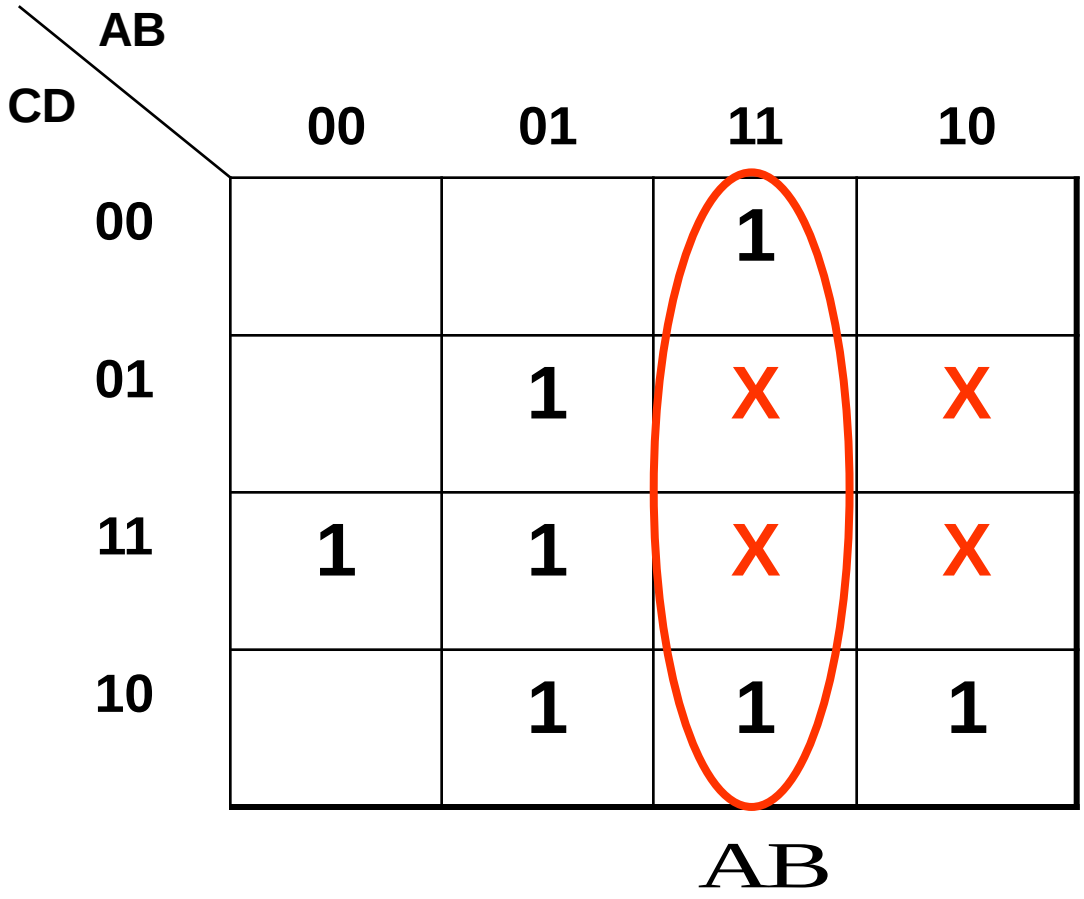
•Les cas impossibles sont représentées

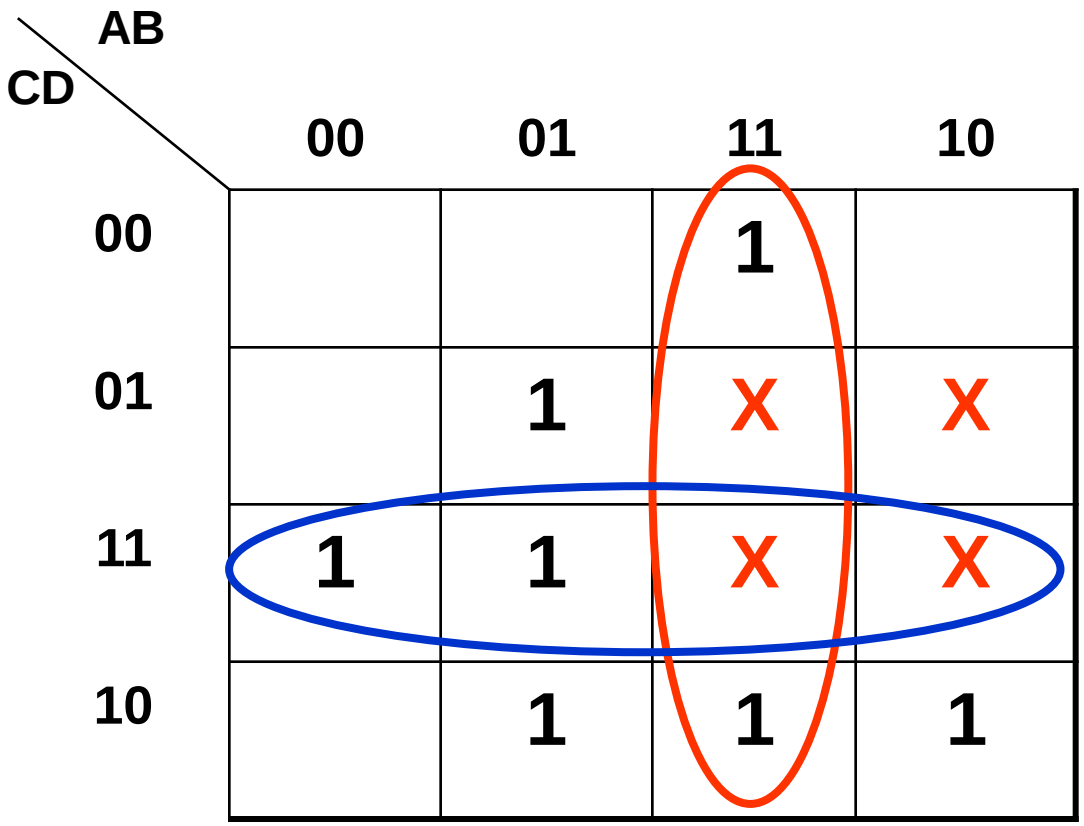
aussi par des X dans la table de karnaugh

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

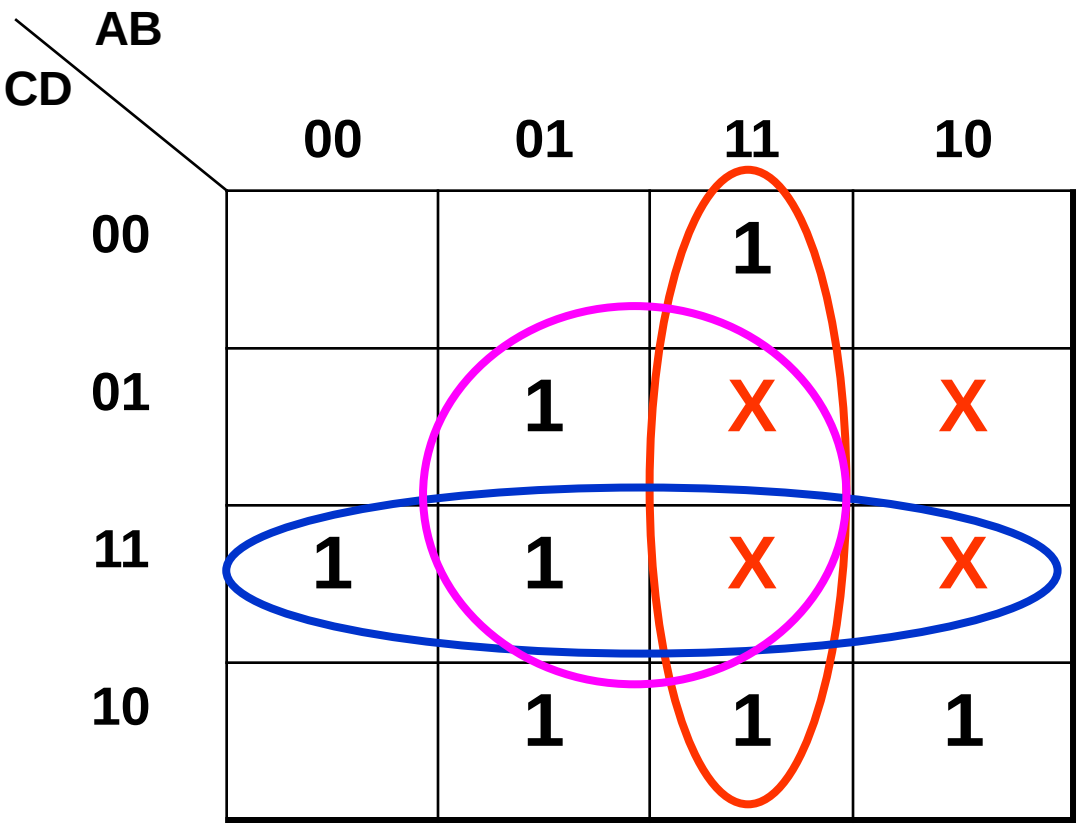
A	B	C	D		S
0	0	0	0		0
0	0	0	1		0
0	0	1	0		0
0	0	1	1		1
0	1	0	0		0
0	1	0	1		1
0	1	1	0		1
0	1	1	1		1
1	0	0	0		0
1	0	0	1		X
1	0	1	0		1
1	0	1	1		X
1	1	0	0		1
1	1	0	1		X
1	1	1	0		1
1	1	1	1		X

- Il est possible d'utiliser les X dans des regroupements :
  - Soit les prendre comme étant des 1
  - Ou les prendre comme étant des 0
- Il ne faut pas former des regroupement qui contient uniquement des X

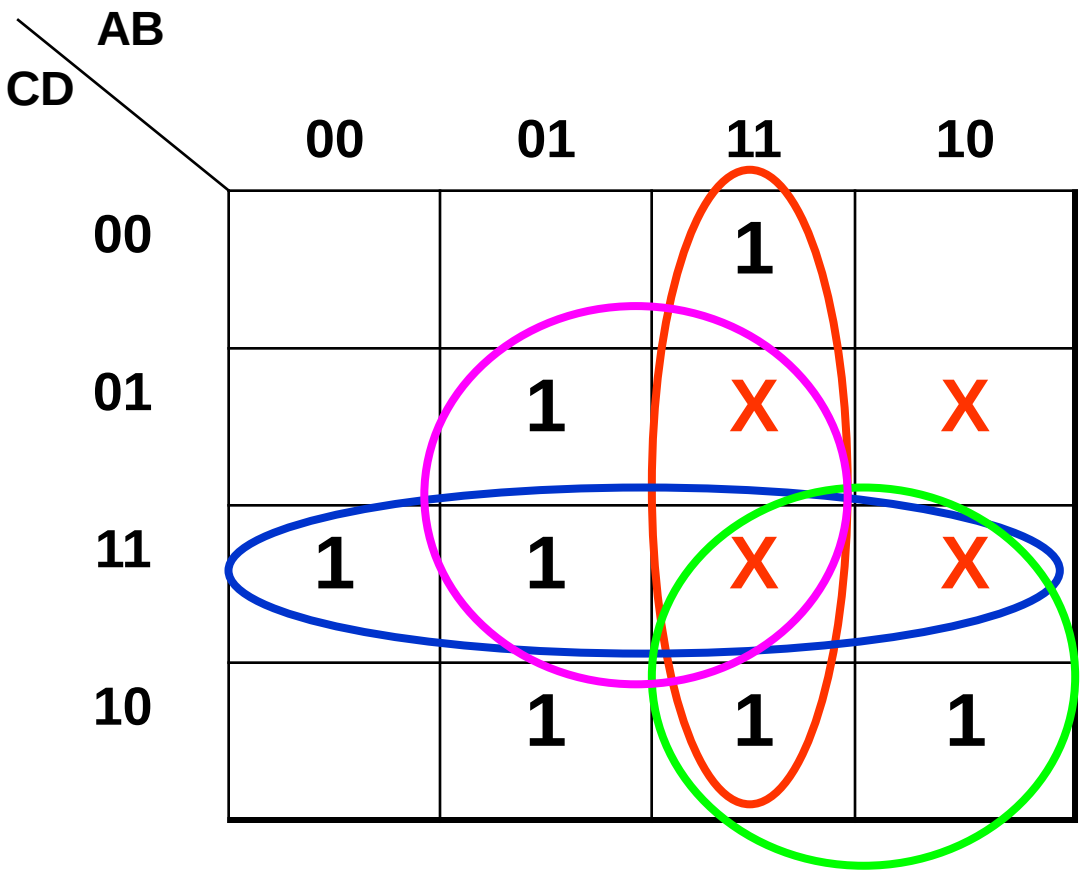




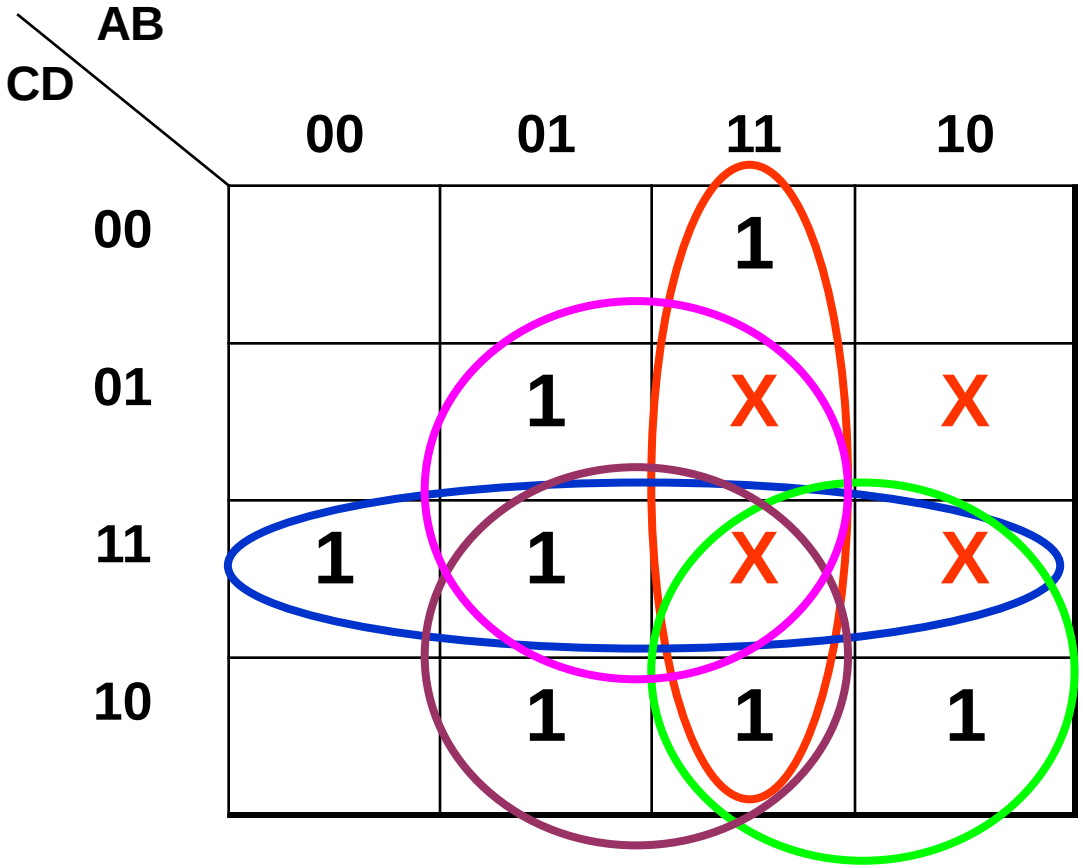
$$AB + CD$$



$$AB + CD + BD$$



$$AB + CD + BD + AC$$



$$AB + CD + BD + AC + BC$$

CHAPITRE 3

Fonctions combinatoires

Plan du chapitre

- ✓ Additionneur
- ✓ Comparateur
- ✓ Multiplexeur/Démultiplexeur
- ✓ Codeur/Décodeur
- ✓ Transcodeur

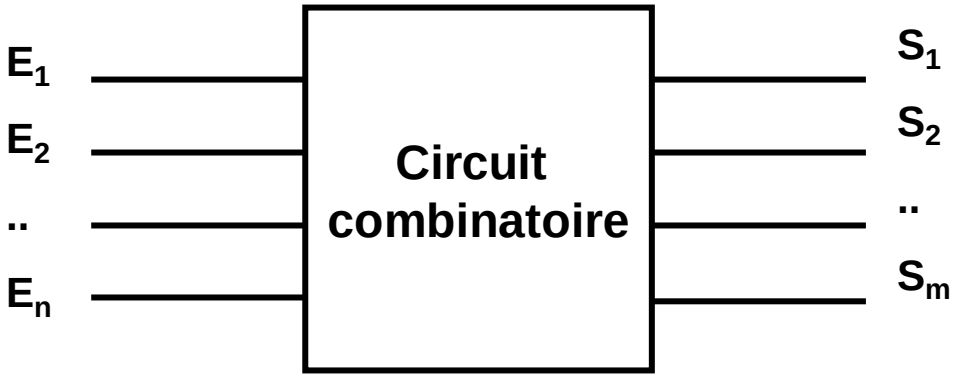
Objectifs du chapitre

- connaitre la structure de quelques circuits combinatoires souvent utilisés (demi additionneur , additionneur complet,……..).

- utiliser des circuits combinatoires pour concevoir d'autres circuits plus complexes.

Introduction

- ✓ **La logique** est une technique élaborée par l'humain pour développer des outils dédiés à diverses fonctions.
- ✓ Un système logique est dit ***combinatoire*** si l'état de sa sortie ne dépend que de l'état de son entrée.



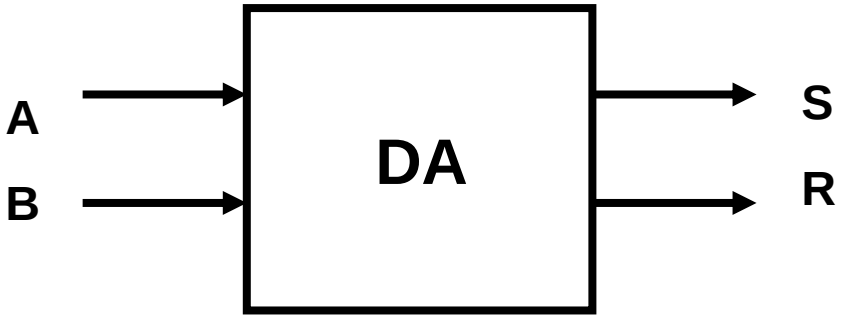
- $S_i=F(E_i)$
- $S_i=F(E_1,E_2,....,E_n)$

- ✓ La logique combinatoire permet de construire des opérateurs :
  - ❑ **Logiques**
  - ❑ **Arithmétiques**

Additionneur

- ***Demi additionneur (Half Add ou HA)***

- Le demi additionneur est un circuit combinatoire qui permet de réaliser la somme arithmétique de deux nombres A et B chacun sur un bit sans tenir compte de la retenue de l'étage précédent. .
- A la sortie on va avoir la somme S et la retenu R (Carry).



Pour trouver la structure (le schéma) de ce circuit on doit en premier dresser sa table de vérité

½ ADD

- En binaire l'addition sur un seul bit se fait de la manière suivante:

$$\begin{cases} 0+0 = 00 \\ 0+1 = 01 \\ 1+0 = 01 \\ 1+1 = 10 \end{cases}$$

➢ La table de vérité associée :

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

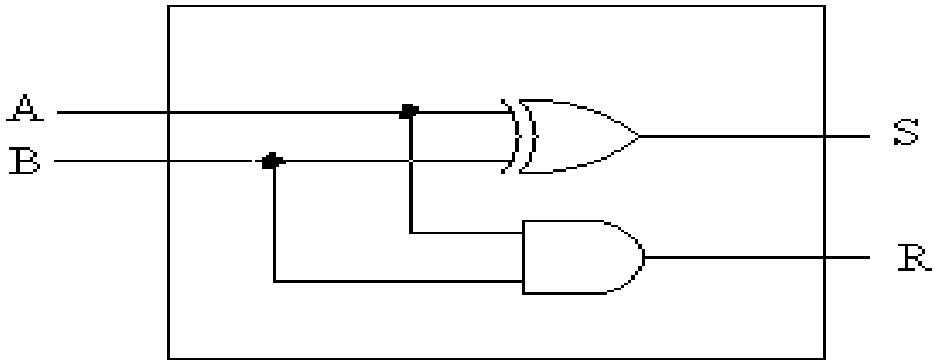
½ ADD

- **Expressions des sorties**

$$R = A.B$$

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

- **Logigramme**



- ***Additionneur complet (Full Add ou FA) :***

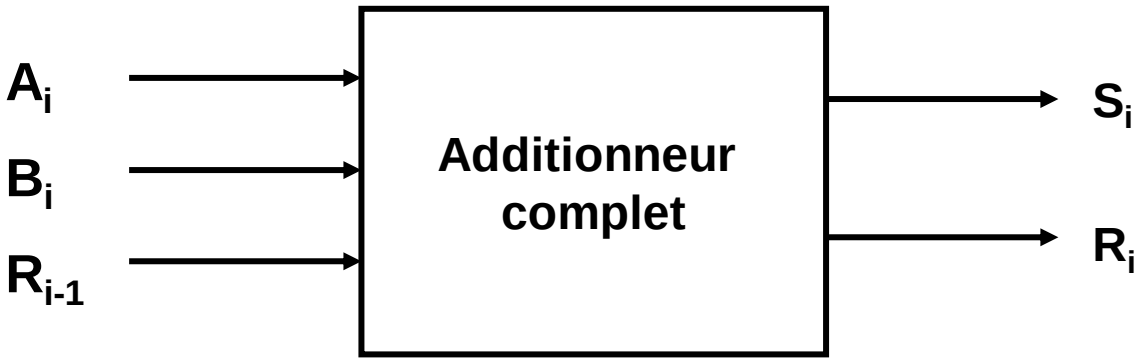
- En binaire lorsqu'on fait une addition il faut tenir en compte de la retenue entrante.

$$\begin{array}{cc} & r_{i-1} \\ & a_i \\ + & b_i \\ \hline r_i & s_i \end{array}$$

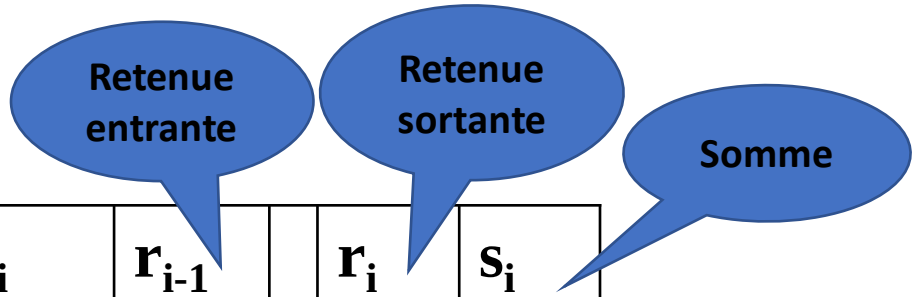
$$\begin{array}{ccccc} r_4 & r_3 & r_2 & r_1 & r_0 = 0 \\ & a_4 & a_3 & a_2 & a_1 \\ + & b_4 & b_3 & b_2 & b_1 \\ \hline r_4 & s_4 & s_3 & s_2 & s_1 \end{array}$$

- *Additionneur complet 1 bit*

- L'additionneur complet un bit possède 3 entrées :
  - A_i : le premier nombre sur un bit.
  - B_i : le deuxième nombre sur un bit.
  - R_{i-1} : la retenue entrante sur un bit.
- Il possède deux sorties :
  - S_i : la somme
  - R_i la retenue sortante



➢ Table de vérité



a_i	b_i	r_{i-1}		r_i	s_i
0	0	0		0	0
0	0	1		0	1
0	1	0		0	1
0	1	1		1	0
1	0	0		0	1
1	0	1		1	0
1	1	0		1	0
1	1	1		1	1

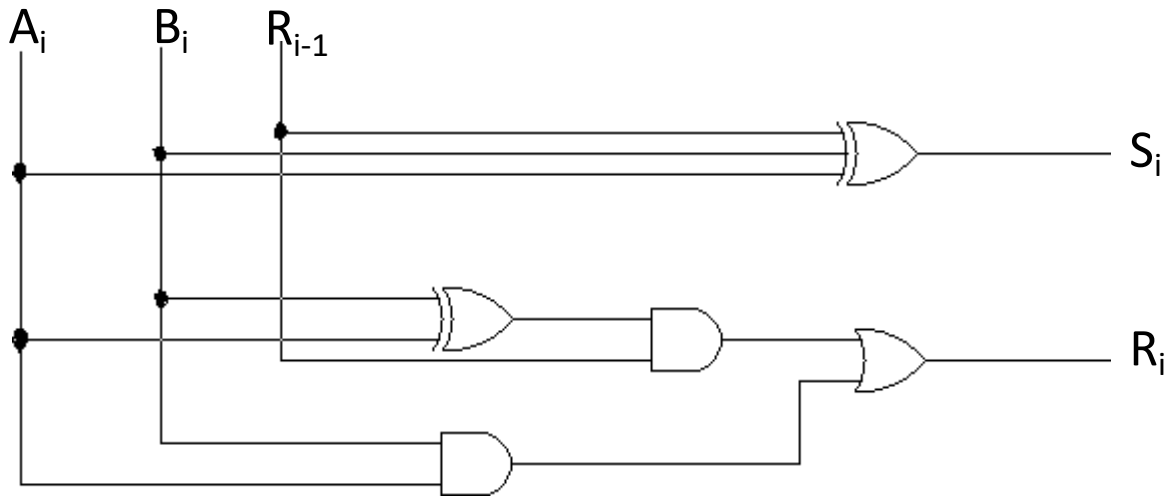
Table de vérité d'un additionneur complet sur 1 bit

➢ Expressions des sorties

$$S_i = R_{i-1} \oplus A_i \oplus B_i$$

$$R_i = R_{i-1} \bullet (A_i \oplus B_i) + A_i B_i$$

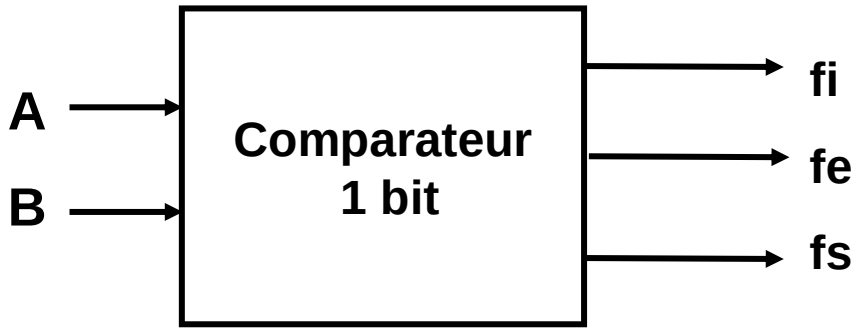
➢ Logigramme



- **Exercice (TD)**
  - Réaliser un additionneur complet en utilisant deux demi- additionneurs.
  - Réaliser un additionneur de deux mots à n bits

Comparateur

- C'est un circuit combinatoire qui permet de comparer deux nombres binaire A et B.
- Il possède 2 entrées :
  - A : sur un bit
  - B : sur un bit
- Il possède 3 sorties
  - fe = 1, fi = 0, fs = 0 si A=B (Egalité)
  - fe = 0, fi = 1, fs = 0 si A < B (Inférieur)
  - fe = 0, fi = 0, fs = 1 si A > B) (supérieur)



- **Table de vérité**

A	B		fs	fe	fi
0	0		0	1	0
0	1		0	0	1
1	0		1	0	0
1	1		0	1	0

- **Expressions des sorties**

$$fs = A.\overline{B}$$

$$fi = \overline{A}B$$

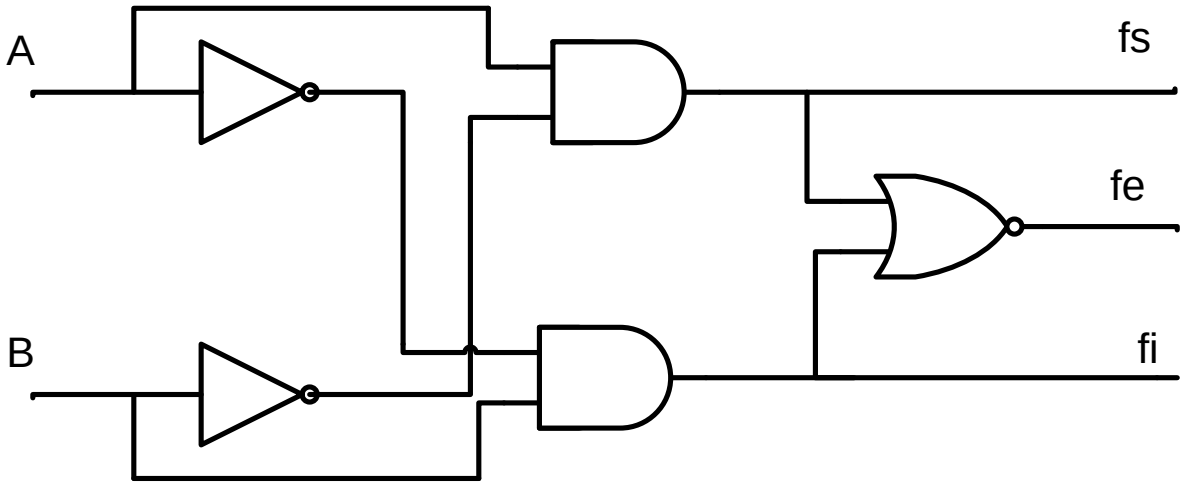
$$fe = \overline{A}\,\overline{B} + AB = \overline{A \oplus B} = \overline{fs + fi}$$

➢ Logigramme

$$fs = A.\overline{B}$$

$$fi = \overline{A}B$$

$$fe = \overline{fs + fi}$$

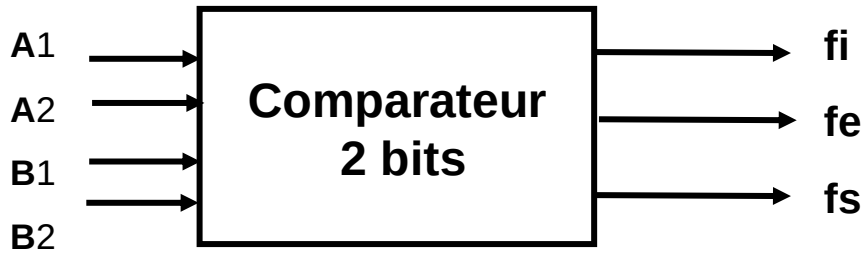


comparateur de deux mots à n bits

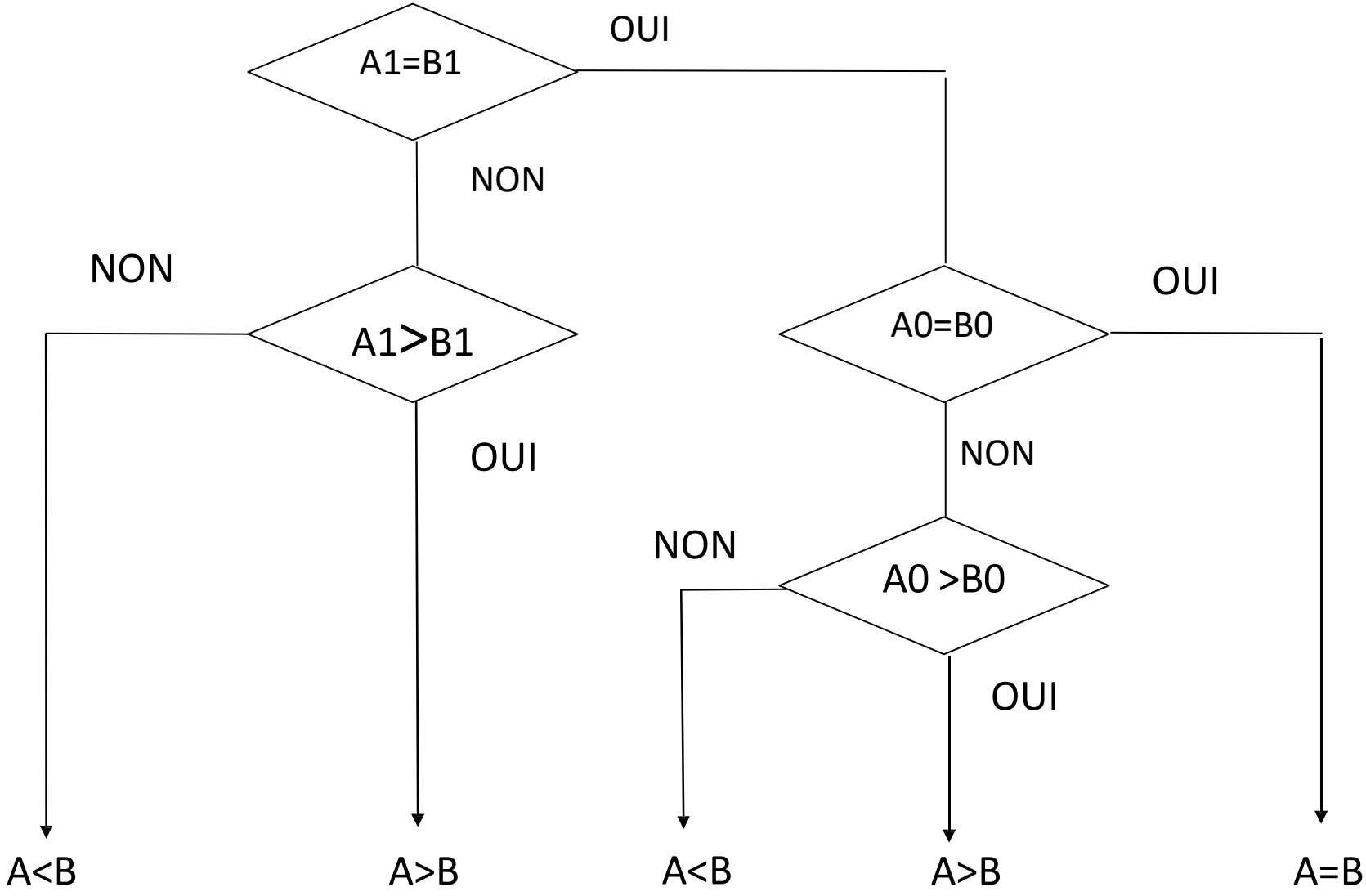
- Le principe consiste à comparer d'abord les bits les plus significatifs (Most Significant Bit ou M S B) . S'ils sont différents, il est inutile de continuer la comparaison. Par contre s'ils sont égaux, il faut comparer les bits de poids immédiatement inférieur et ainsi de suite.

➢ **Exemple: Comparateur 2 bits**

- Il permet de faire la comparaison entre deux nombres A (a_2a_1) et B(b_2b_1) chacun sur deux bits.



- ***comparateur de deux mots à 2 bits: principe***



- **Table de vérité**
- **Expressions des sorties**

- 1. A=B si

A1=B1 et A0=B0

$$fe = (\overline{A1 \oplus B1}).(\overline{A0 \oplus B0})$$

- 2. A>B si

A1 > B1 ou (A1=B1 et A0>B0)

$$fs = A1.\overline{B1} + (\overline{A1 \oplus B1}).(A0.\overline{B0})$$

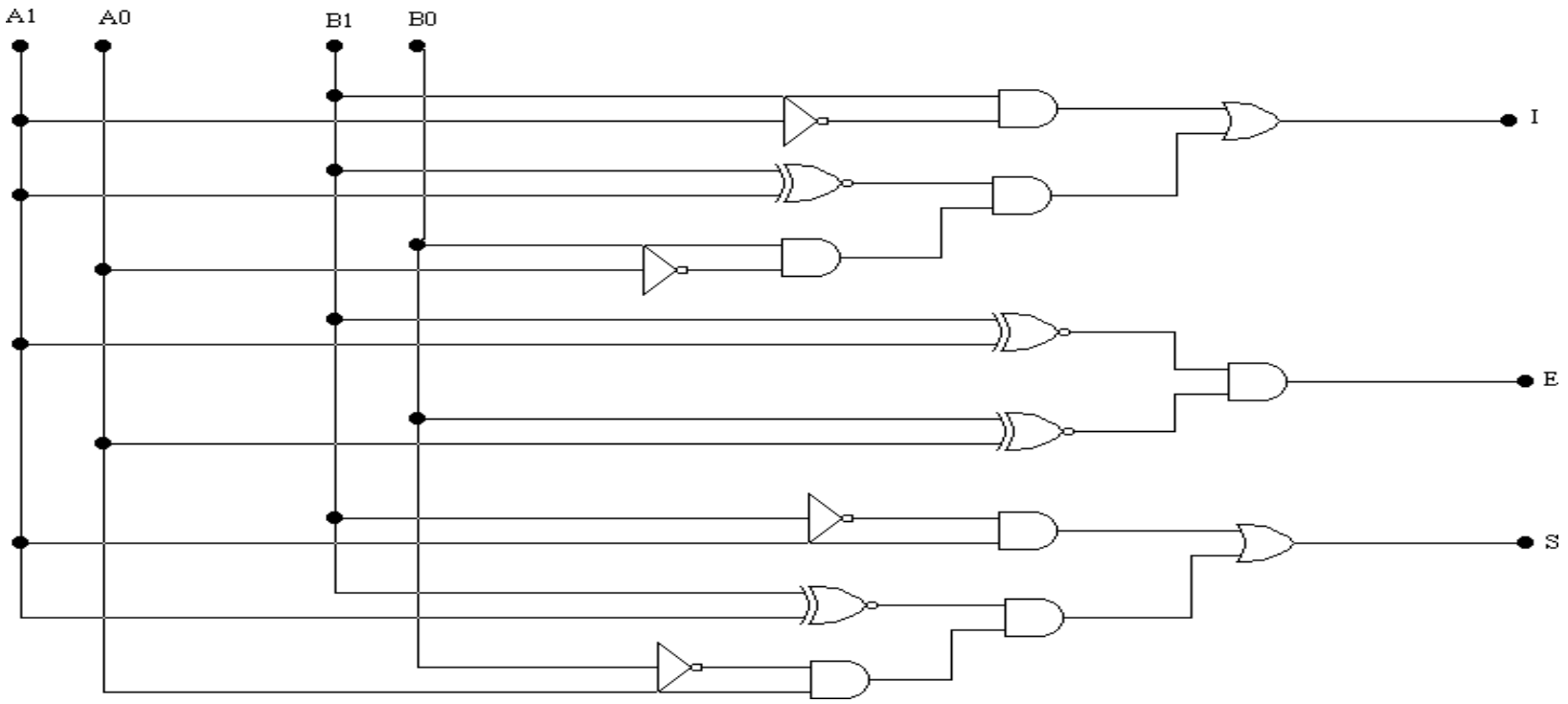
- 3. A<B si

A1 < B1 ou (A1=B1 et A0<B0)

$$fi = \overline{A1}.B1 + (\overline{A1 \oplus B1}).(\overline{A0}.B0)$$

A1	A0	B1	B0		fs	fe	fi
0	0	0	0		0	1	0
0	0	0	1		0	0	1
0	0	1	0		0	0	1
0	0	1	1		0	0	1
0	1	0	0		1	0	0
0	1	0	1		0	1	0
0	1	1	0		0	0	1
0	1	1	1		0	0	1
1	0	0	0		1	0	0
1	0	0	1		1	0	0
1	0	1	0		0	1	0
1	0	1	1		0	0	1
1	1	0	0		1	0	0
1	1	0	1		1	0	0
1	1	1	0		1	0	0
1	1	1	1		0	1	0

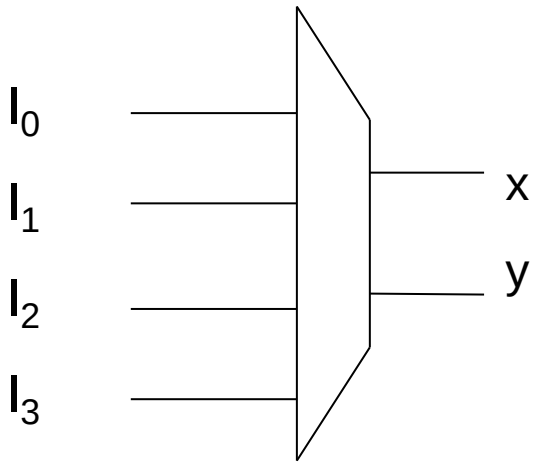
➢ Logigramme



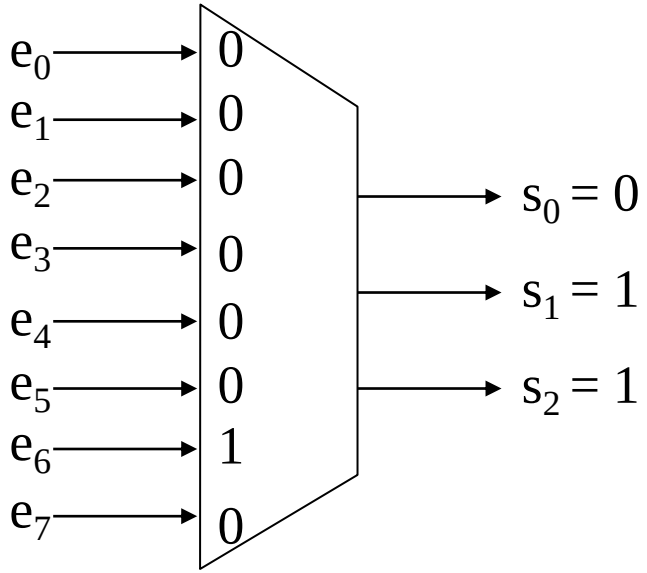
- **Exercice (TD)**
  - Réaliser un comparateur 2 bits en utilisant deux comparateurs 1 bit et des portes logiques.

Codeur et Décodeur

- ***Codeur ou encodeur*** *:* C'est un circuit à **N=2^n** entrées dont une seulement est active et qui délivre sur **M = n** sorties (en code binaire ou autre) le numéro de l'entrée active.
- **Codeur 4 vers 2** : Un codeur **4 vers 2** permet de valider en sortie le code correspondant à une entrée active parmi 4.



Encodeur 4→2



Encodeur 8→3

Codeur 4 vers 2

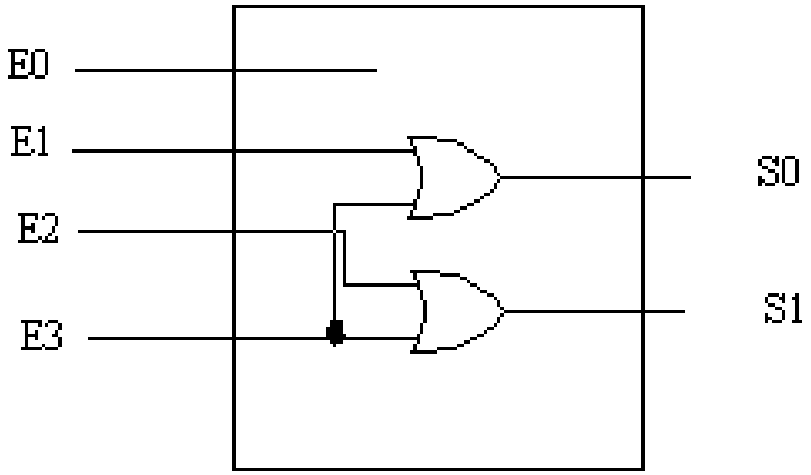
- **Table de vérité**

E_0	E_1	E_2	E_3		S_1	S_0
1	0	0	0		0	0
0	1	0	0		0	1
0	0	1	0		1	0
0	0	0	1		1	1

- **Expression des sorties**

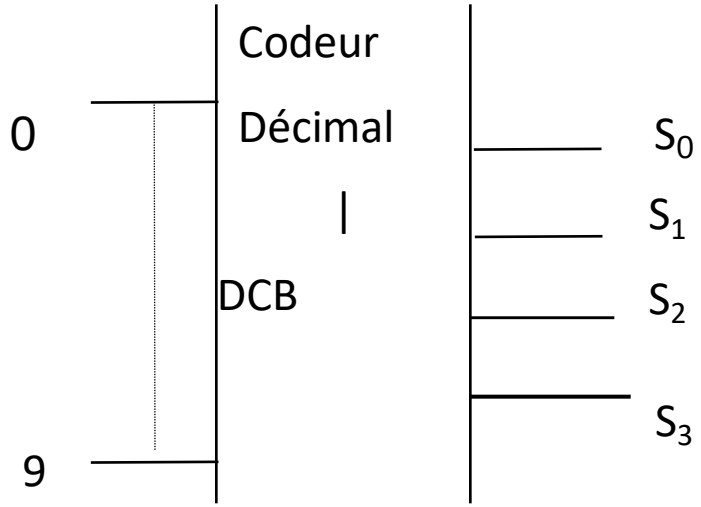
$$S_0 = E_1 + E_3$$
$$S_1 = E_2 + E_3$$

- **Logigramme**



Exemple: Codeur décimal-DCB

- Ce type de codeur possède 10 entrées (une pour chaque chiffre décimal) et 4 sorties qui correspondent au code DCB. Il s'agit d'un codeur **10 vers 4**



Codeur décimal-DCB

- **Table de vérité**

N	S_3	S_2	S_1	S_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

- **Expression des sorties**

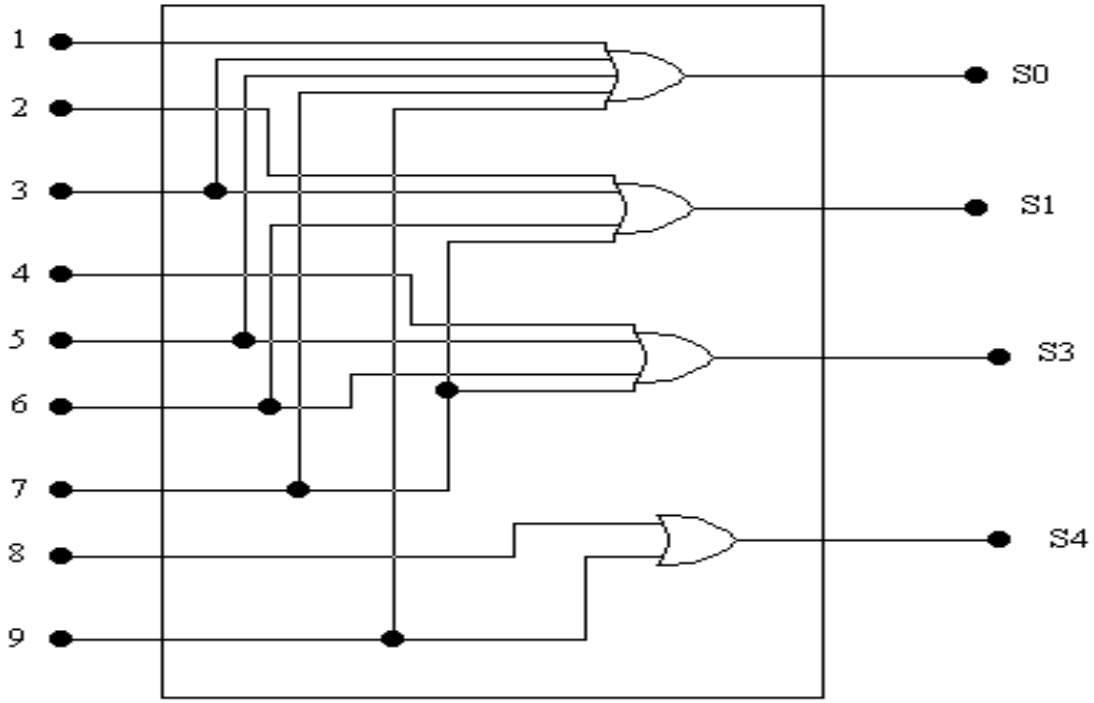
$$S_0 = E1 + E3 + E5 + E7 + E9$$

$$S_1 = E2 + E3 + E6 + E7$$

$$S_2 = E4 + E5 + E6 + E7$$

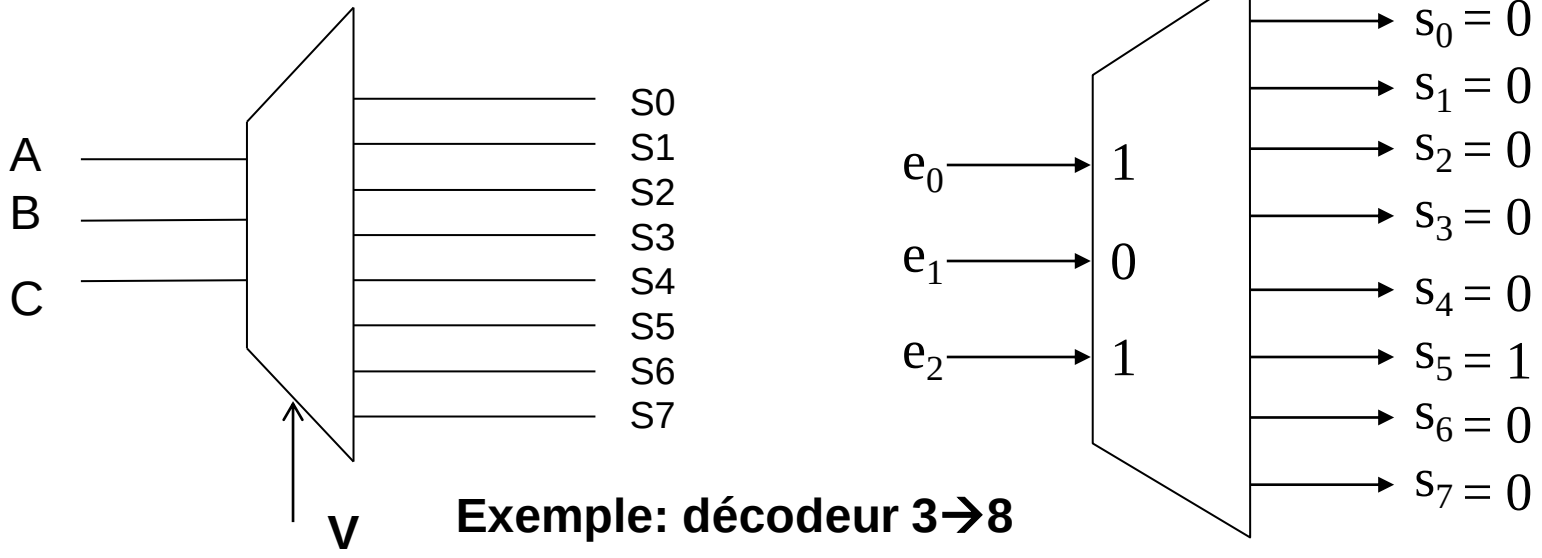
$$S_3 = E8 + E9$$

- **Logigramme**



Codeur et Décodeur

- ***Décodeur :*** Le décodeur est un circuit qui établit la correspondance entre un code d'entrée sur N bits et M lignes de sortie ($M = 2^N$). C'est un circuit à N entrées qui permet de sélectionner une sortie parmi M.
- **Décodeur 2 vers 4 :** Un décodeur 2 vers 4 valide la sortie correspondante à une des combinaisons présentes à l'entrée.



Exemple: décodeur 3→8

Décodeur 2 vers 4

- **Table de vérité**

Entrées		Sorties			
A_1	**A_0**	**S0**	**S1**	**S2**	**S3**
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

- **Expression des sorties**

$$S_0 = \overline{\overline{A_0} \bullet \overline{A_1}}$$

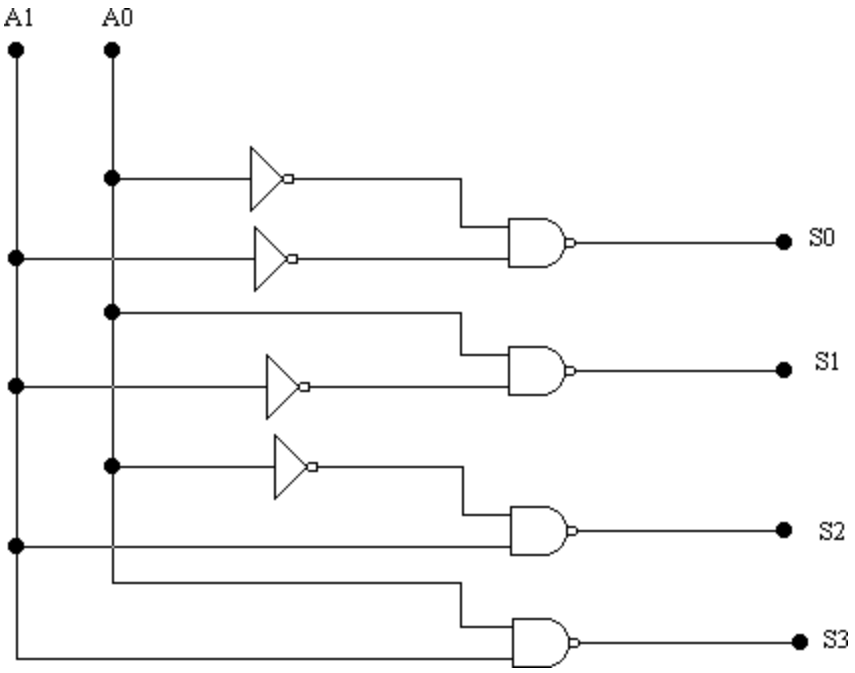
$$S_1 = \overline{A_0 \bullet \overline{A_1}}$$

$$S_2 = \overline{\overline{A_0} \bullet A_1}$$

$$S_3 = \overline{A_0 \bullet A_1}$$

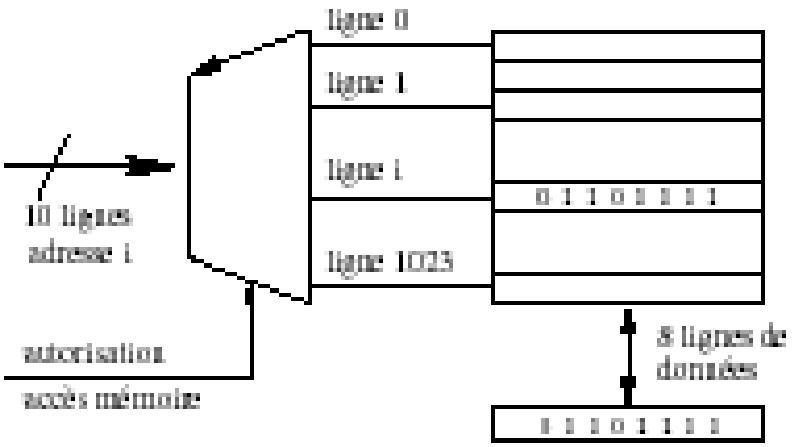
Décodeur 2 vers 4

- **Logigramme**



Applications des décodeurs

- **Adressage d'une mémoire**



- une mémoire est un tableau d'éléments binaires (divisés en lignes et colonnes) ;
- pour lire un mot mémoire, il faut lui envoyer le numéro de ligne souhaité (Adresse) ;
- souvent, le décodeur est interne à la mémoire.

Applications des décodeurs

- **Génération de fonction :** Toute fonction logique peut être réalisée à partir d'une combinaison de décodeur.
- Exemple : $F = \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C} + \overline{A}BC$

Remarque :

- Il n'est pas nécessaire de simplifier la fonction avant la réalisation.

Transcodeurs (convertisseurs)

- Un transcodeur est un Circuit à **m** entrées et **n** sorties qui convertit un nombre écrit dans un code C1 en un nombre écrit dans un code C2.

- **Exemple : transcodeur BCD-code Gray**
- C'est un circuit combinatoire qui à chaque combinaison en entrée (sur quatre bits) du code BCD fait correspondre en sortie son équivalent en code Gray.

Transcodeurs BCD-GRAY

- **Table de vérité**

	E_3	E_2	E_1	E_0		S_3	S_2	S_1	S_0
0	0	0	0	0		0	0	0	0
1	0	0	0	1		0	0	0	1
2	0	0	1	0		0	0	1	1
3	0	0	1	1		0	0	1	0
4	0	1	0	0		0	1	1	0
5	0	1	0	1		0	1	1	1
6	0	1	1	0		0	1	0	1
7	0	1	1	1		0	1	0	0
8	1	0	0	0		1	1	0	0
9	1	0	0	1		1	1	0	1

- **Expression des sorties**

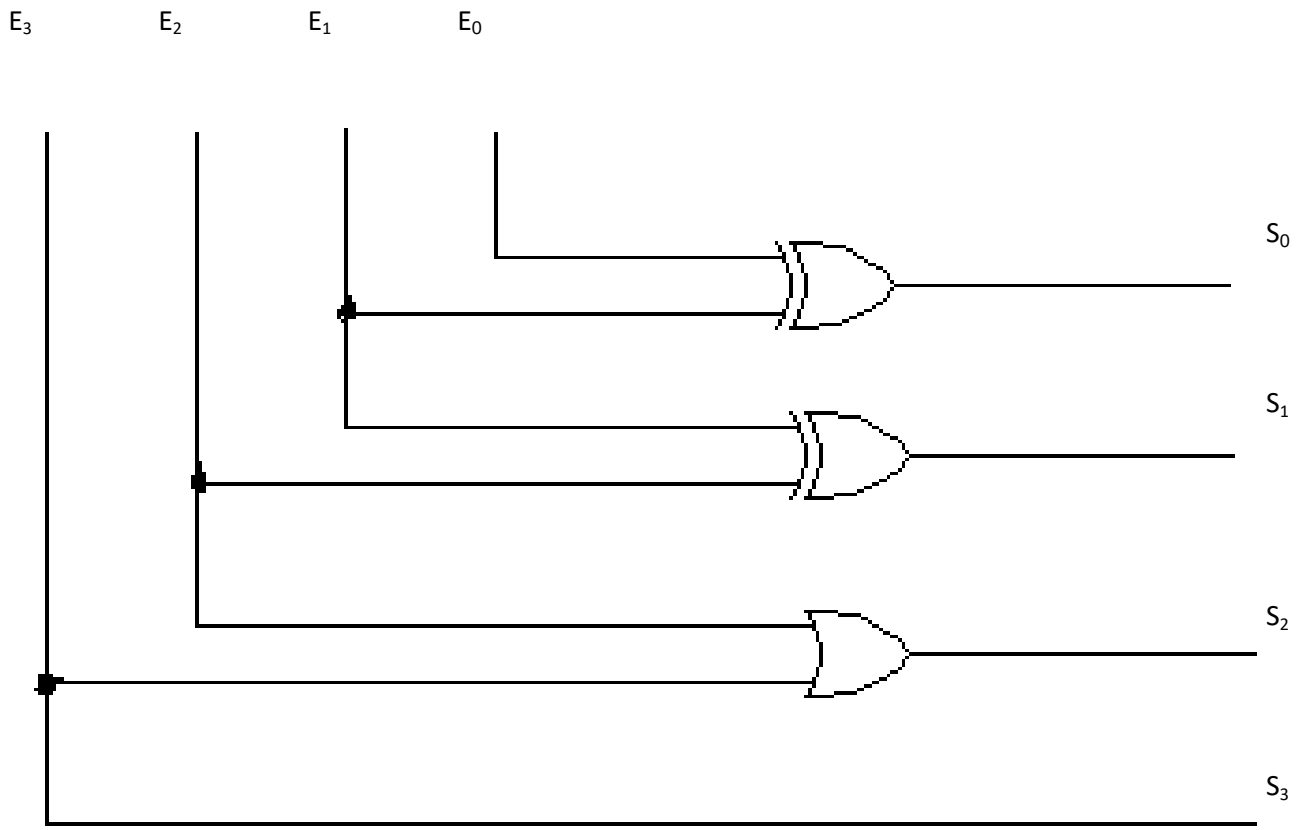
$$S_0 = E_0 \oplus E_1$$

$$S_1 = E_1 \oplus E_2$$

$$S_2 = E_2 \oplus E_3$$

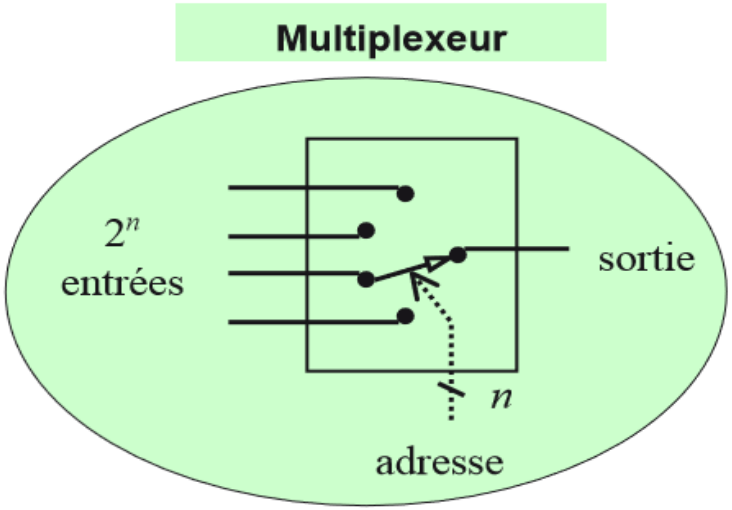
$$S_3 = E_3$$

➢ Logigramme d'un transcodeur DCB-code Gray

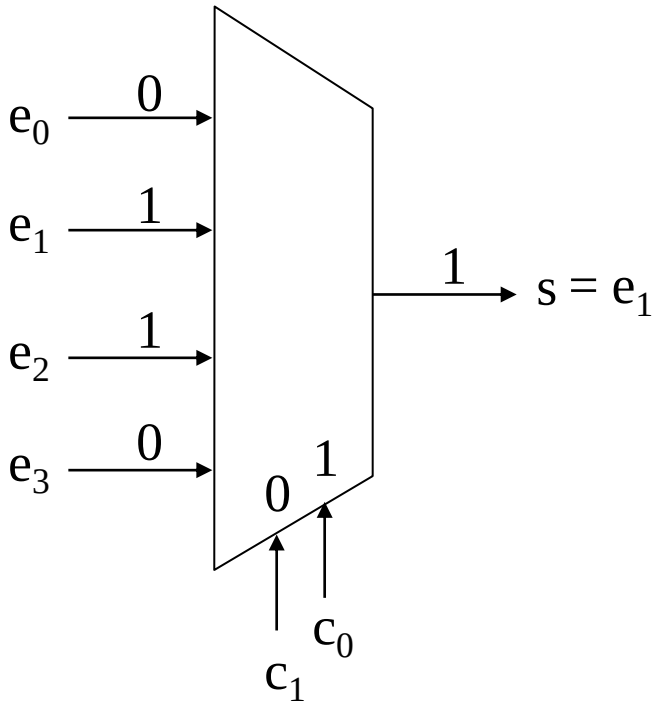


Multiplexeurs–démultiplexeurs

- *Multiplexeurs :* Un multiplexeur est circuit à 2^n entrées d'informations, n entrées de sélection, et une sortie. L'aiguillage des données d'entrée vers la sortie se fait à l'aide des entrées de sélection.



rôle : aiguiller un signal d'entrée parmi 2^n vers une sortie à l'aide de *n* bits d'adresse



▪ Mux 4 vers 1

- **Exemple : Multiplexeur à 4 entrées de données (MUX 4 vers 1)**

➢ **Table de vérité**

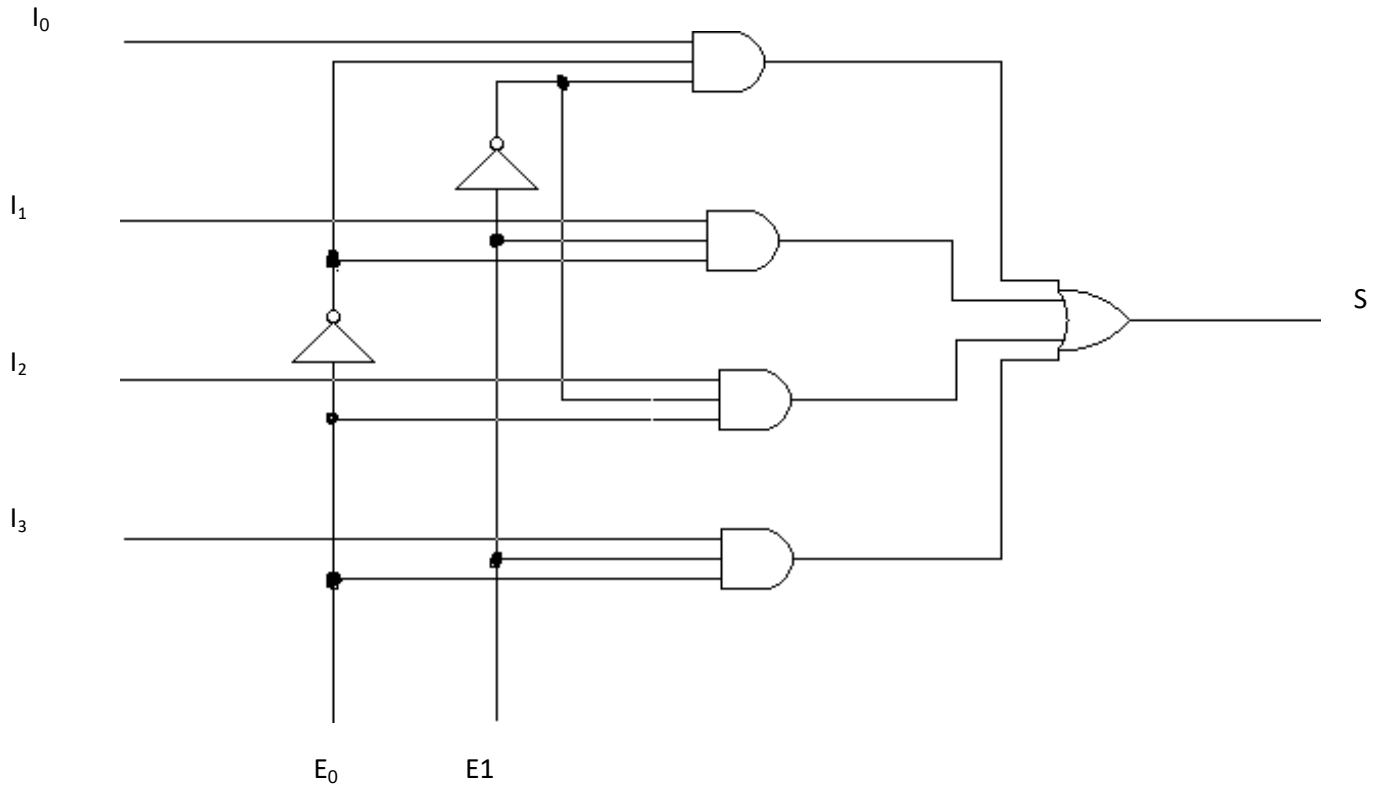
E_1	E_0	S
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

➢ **Expression de la sortie**

$$S = \overline{E_1}\,\overline{E_0}\,I_0 + \overline{E_1}E_0\,I_1 + E_1\,\overline{E_0}\,I_2 + E_1E_2\,I_3$$

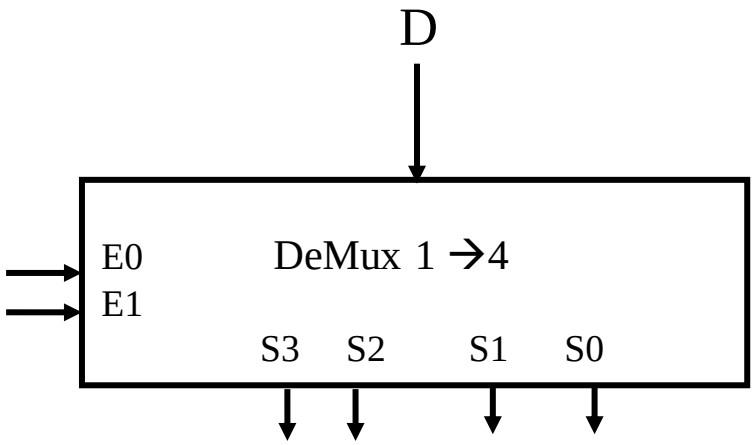
▪ Mux 4 vers 1

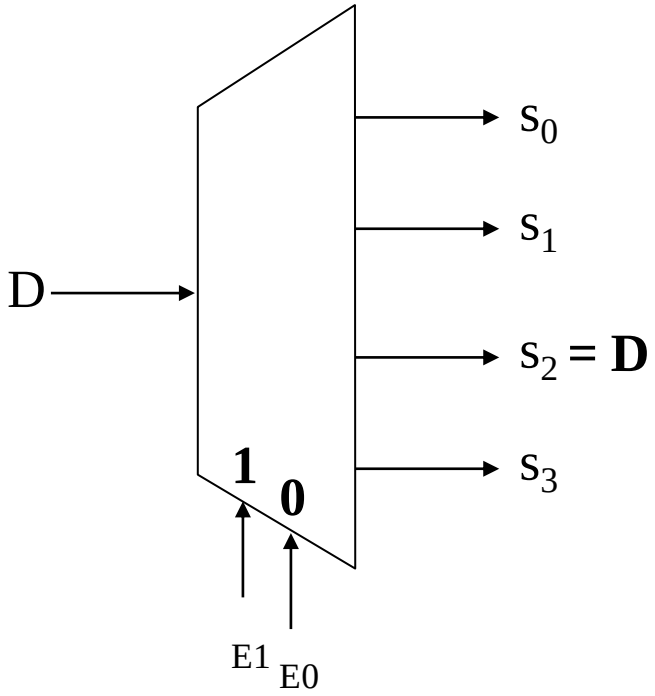
➢ Logigramme



Démultiplexeurs

- ***Démultiplexeur*** *:* C'est circuit logique à 2^n sorties, 1 entrée de données, n entrées de commande. Il reçoit les données d'entrée qu'il les dirige vers l'une des sorties.





DeMux 1 vers 4

Table de vérité

E1	E0		S3	S2	S1	S0
0	0		0	0	0	D
0	1		0	0	D	0
1	0		0	D	0	0
1	1		D	0	0	0

Expression des sorties

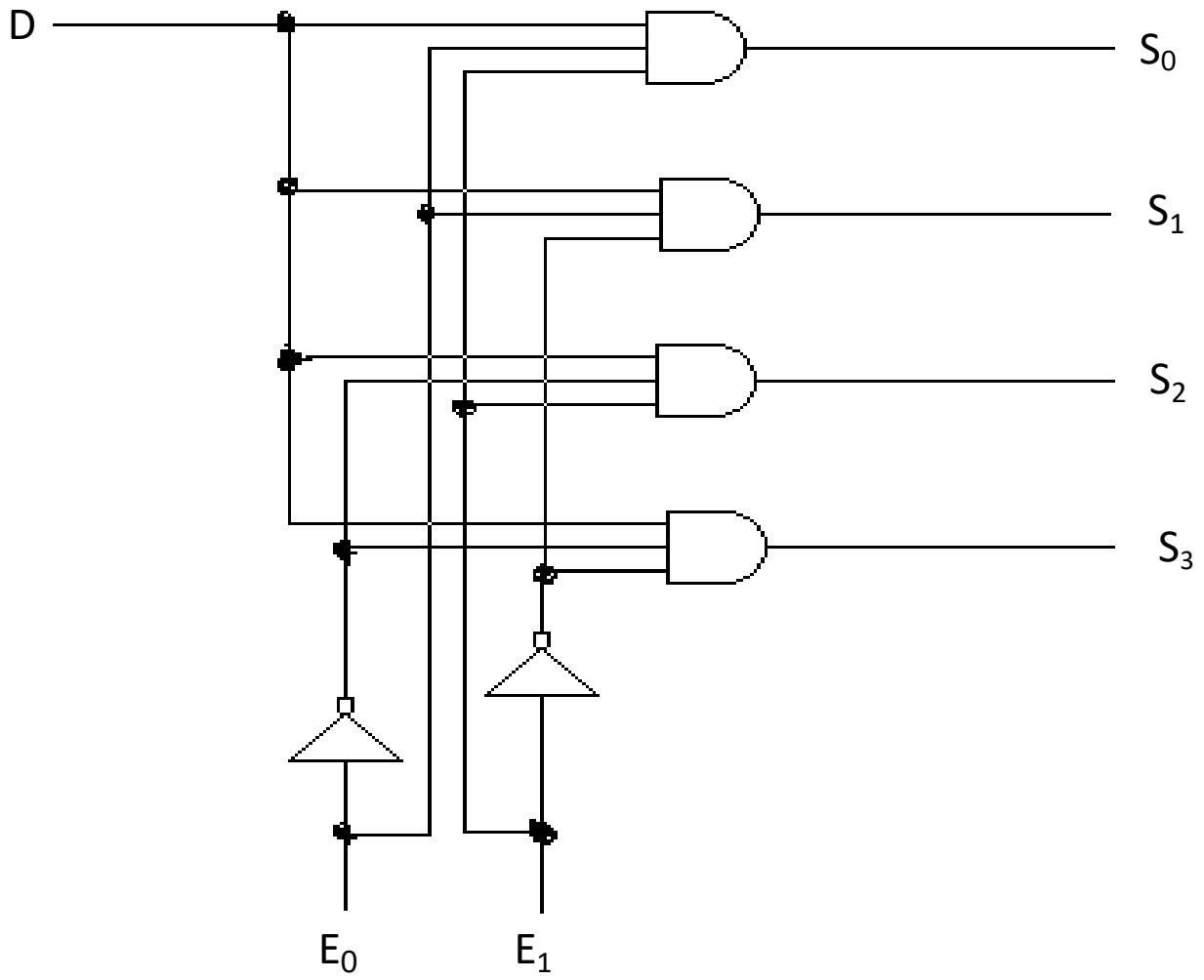
$$S0 = \overline{C1}.\overline{C0}.D$$

$$S1 = \overline{C1}.C0.D$$

$$S2 = C1.\overline{C0}.D$$

$$S3 = C1.C0.D$$

- **Logigramme d'un démultiplexeur à 4 sorties**



Chapitre 4
Logique séquentielle

Plan du chapitre

Introduction générale : La logique séquentielle		
I :Eléments de mémoire : latches et bascules		
	Latch RS	
	Latch et bascule D	
	Bascule T, Latch et bascule JK	
	Tables de transition des bascules	
II : Les registres et les compteurs		
	Les registres	
	Les compteurs asynchrones	
	Les compteurs synchrones	

Objectifs du chapitre

- **Connaitre et exploiter les méthodes de conception et d'analyse pour :**
  - ✓ Les systèmes séquentiels synchrones
  - ✓ Les systèmes séquentiels asynchrones

Introduction générale : La logique séquentielle

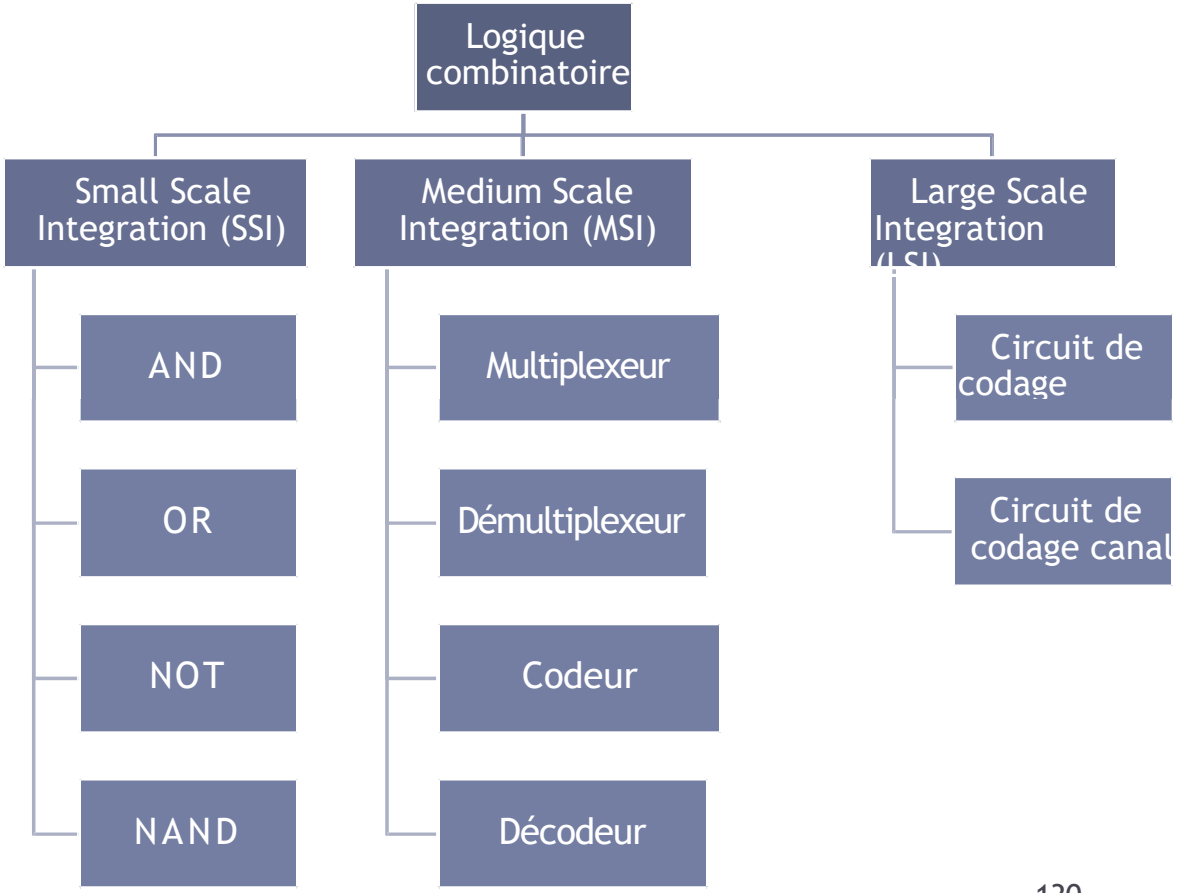
1. Comment connaitre la logique séquentielle ?
2. Comment construire la logique séquentielle ?
3. Eléments de base en logique séquentielle

Rappel de la logique combinatoire

- La sortie d'une fonction ne dépend que de ses entrées.
- ✓ Pour les mêmes entrées, on obtient les mêmes sorties.

- La logique combinatoire permet de construire des opérateurs :
- ✓ Logiques
- ✓ Arithmétiques

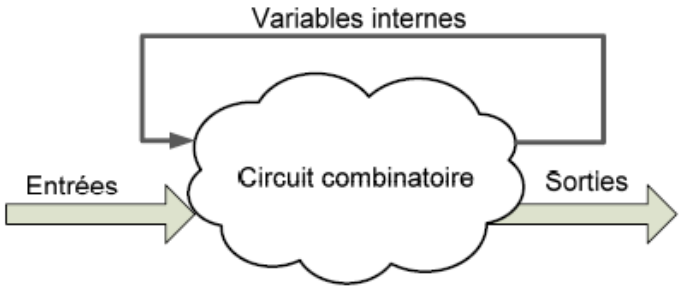
Rappel : Eléments de logique combinatoire



La logique séquentiel

- Un circuit séquentiel est un circuit numérique (logique) dont **l'état** à l'instant **t+1** est une fonction **des entrées** au même instant **t+1** et de **l'état précédent du système** (l'instant t)

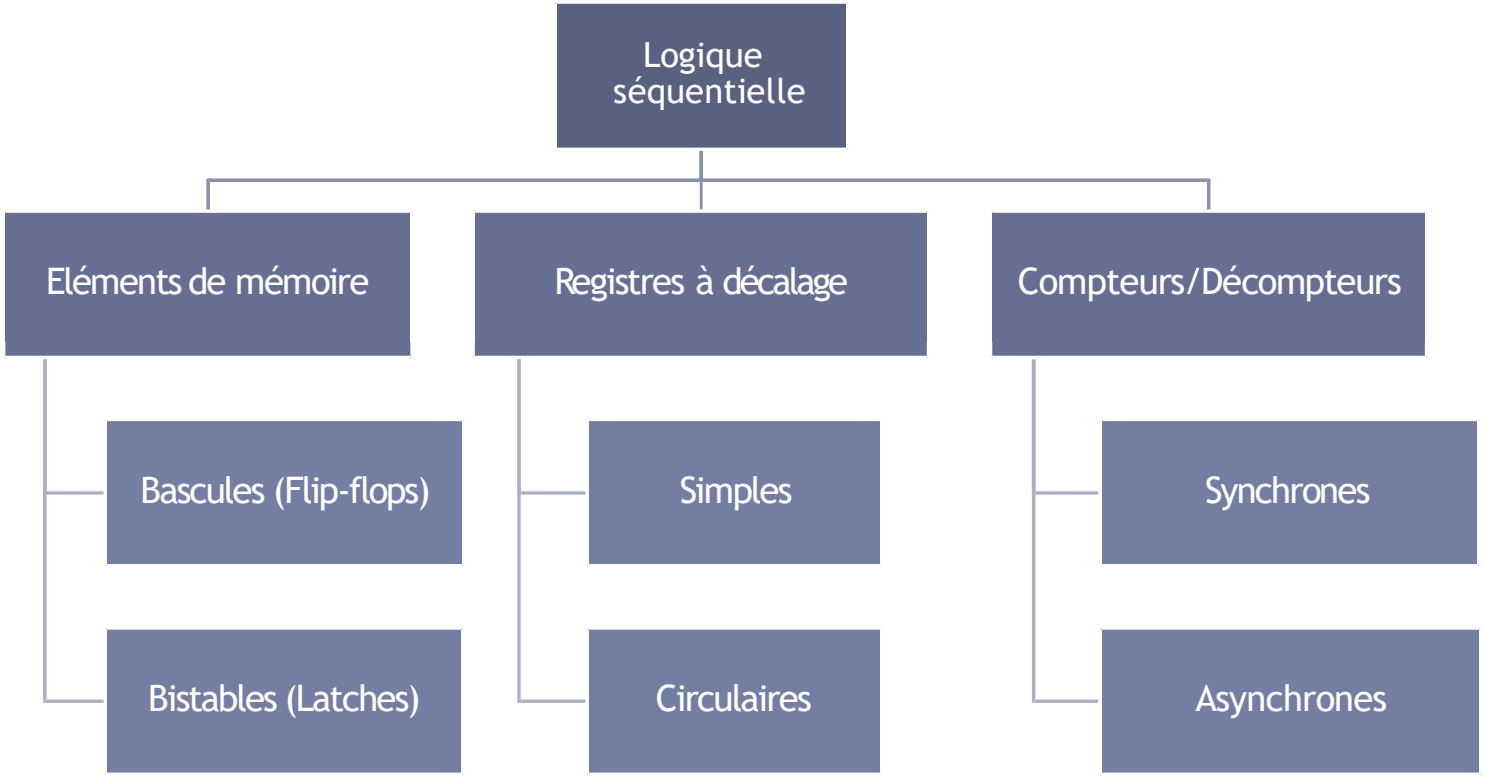
  - Circuit combinatoire calculant les variables internes
  - Rebouclage des variables internes



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

Grâce à la logique séquentielle, on peut réaliser des circuits de mémorisation et concevoir complètement un processeur.

Eléments de logique séquentielle



Eléments de logique séquentielle

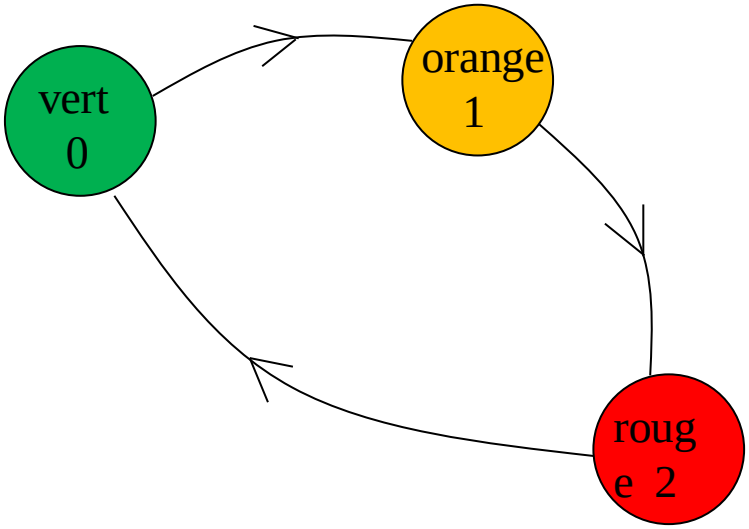
1 : **Un état d'un circuit** est une configuration des sorties de ce circuit.

- Les n états d'un circuit sont numérotés de 0 à n-1.
- Exemple : une bascule (définie plus loin) possède deux états; un circuit composé de 2 bascules à 4 états différents ; un circuit composé de m bascules a 2^m états différents.

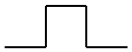
2 : **Une transition** est un changement d'état.

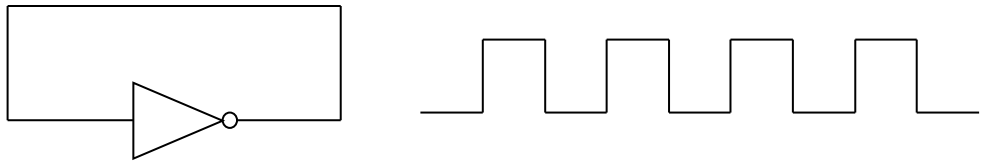
3 : **Le graphe** de transitions d'un circuit est un graphe dont les nœuds sont les états possible du circuit, les arcs les transitions possibles.

Exemple : Les feux tricolores



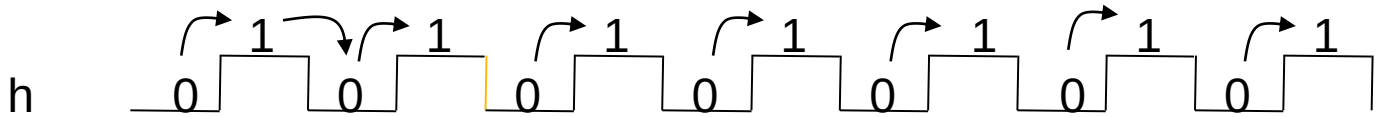
Système synchrone (Notion de l'horloge)

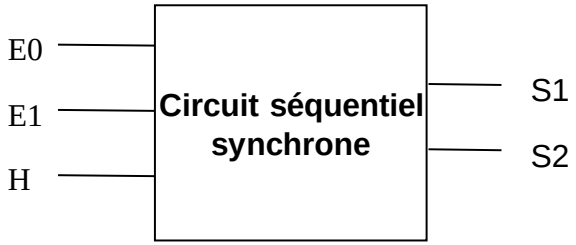
- L' horloge (ou oscillateur) est l'élément permettant l'introduction de la notion de temps dans les circuits.
- Elle est symbolisée par : 
- Une horloge permet d'obtenir un signal carré ayant une fréquence bien précise, constante au cours du temps et qui peut être élevée (plusieurs centaines de Mhz).
- L'oscillateur le plus simple est une simple porte inverseur bouclant sur elle-même :



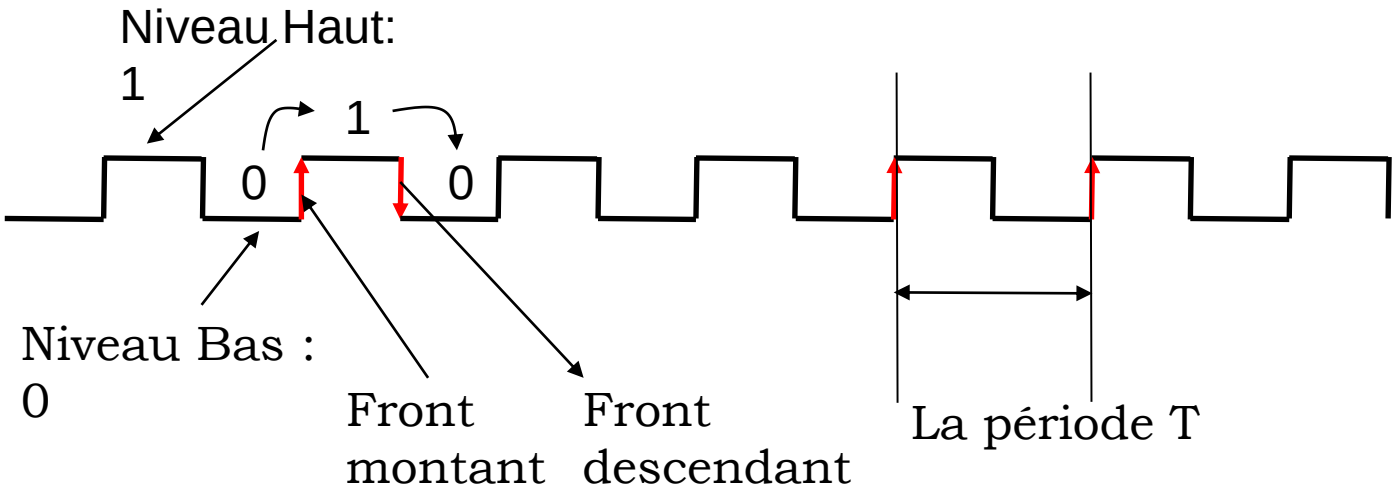
Système synchrone (Notion de l'horloge)

- Une horloge est donc une **variable logique** qui passe successivement de 0 à 1 et de 1 à 0 d'une façon périodique.
- Cette variable est utilisée souvent comme une entrée des circuits séquentiels → le circuit est dit **synchrone**.
- L'horloge est notée par h ou ck (clock).





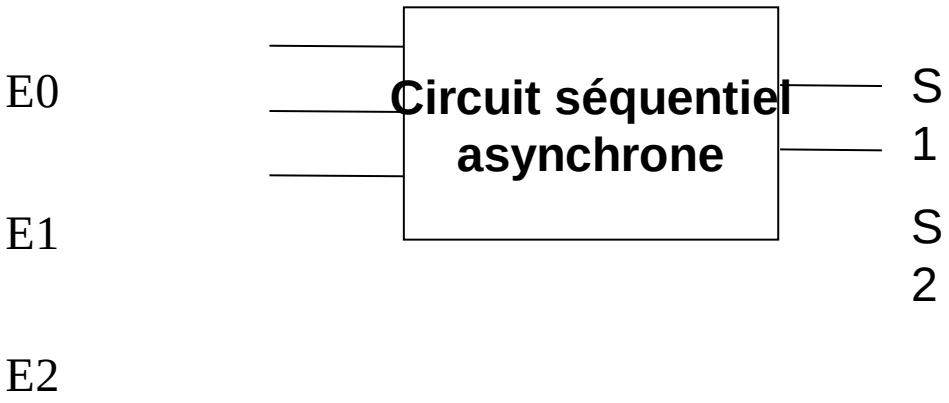
Description d'un signal d'horloge



- Changement d'état = front (transition)
  - 0 à 1 : front montant
  - 1 à 0 : front descendant

Les systèmes Asynchrones

- Lorsque un circuit séquentiel n'a pas d'horloge comme variable d'entrée ou si le circuit fonctionne indépendamment de cette horloge alors ce circuit est asynchrone.



Exemples de circuits séquentiels

- **Bistable (latch)**
  - Signal d'activation : niveau logique haut ou bas
  - Bistable actif : entrée transférée à la sortie
  - Bistable inactif : conservation de la valeur de la sortie jusqu'à ce que le signal devienne actif
- **Bascule**
  - Signal d'activation : transition de niveau
  - Bascule active : entrée transférée à la sortie
- **Compteur**
  - Circuit logique séquentiel ayant un fonctionnement cyclique
  - Fonctionnement piloté par une horloge
  - Résultat représenté sur n bits
  - Sur une transition du signal d'horloge (de '0' à '1'), un incrément est ajouté au résultat.

I : Eléments de mémoire : latches et bascules

1. **Latch RS**
2. Latch et bascule D
3. Bascule T
4. Latch et bascule JK
5. Tables de transition des bascules

Latch RS (1/2)

Un latch RS :

Fonction mémoire,

Réalisé par un opérateur logique qui peut stocker une information jusqu'à ce que cette information soit effacée par une autre information.

Stockage d'information : **"SET" (Mise à un)**

Effacement : **"RESET " (Mise à zéro).**

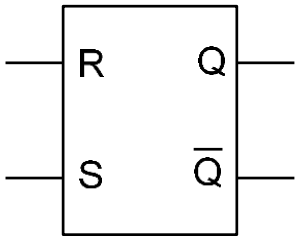
Fonctionnement

Mise à 1 de S (Set) : la sortie Q passe à 1.

Mise à 1 de R (Reset) : la sortie Q passe à 0.

R = S = 0 : maintien de l'état précédent des sorties.

R=S=1 : état interdit à l'entrée.



Latch RS (2/2)

- Soit Q^+ l'état défini lorsqu'il y a un changement à l'entrée, et Q l'état précédent.

Table de vérité

R	S	Q+
0	0	Q
0	1	1
1	0	0
1	1	X

Table de Karnaugh

Q \ RS	00	01	11	10
0	0	1	X	0
1	1	1	X	0

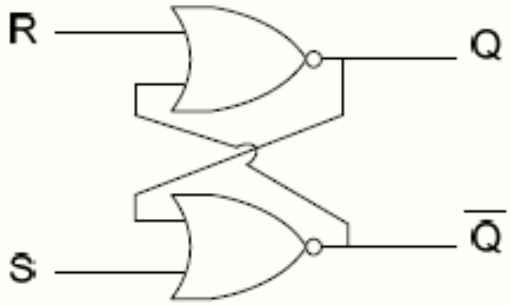
$$Q^+ = \overline{R}Q + S$$

Circuit du latch RS

- Avec des portes NOR

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

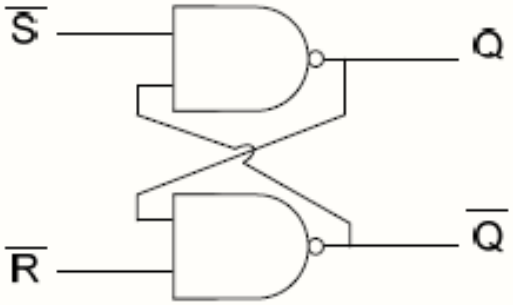
$$\overline{Q^+} = \overline{\overline{R}Q + S} = \overline{\left(R + \overline{Q}\right) + S}$$



- Avec des portes NAND

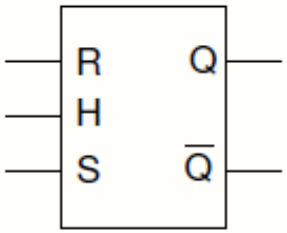
$$Q^+ = \overline{\overline{RQ + S}} = \overline{\overline{RQ}.\overline{S}}$$

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



Latch RS synchronisé (RSH)

- Si H=1 : fonctionnement du latch RS.
- Si H=0 : état précédent.



H	R	S	Q+
0	X	X	Q
1	0	0	Q
1	0	1	1
1	1	0	0
1	1	1	X

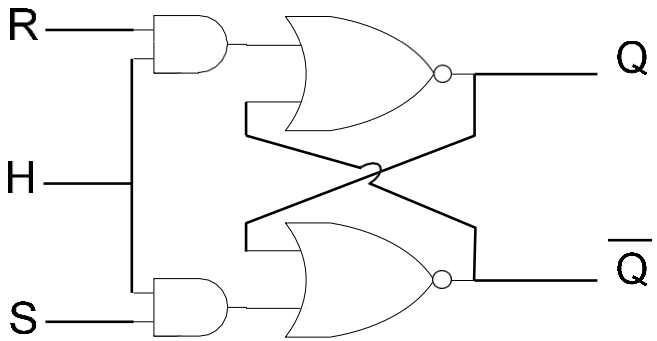
HQ \ RS	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	X	0
10	0	1	X	0

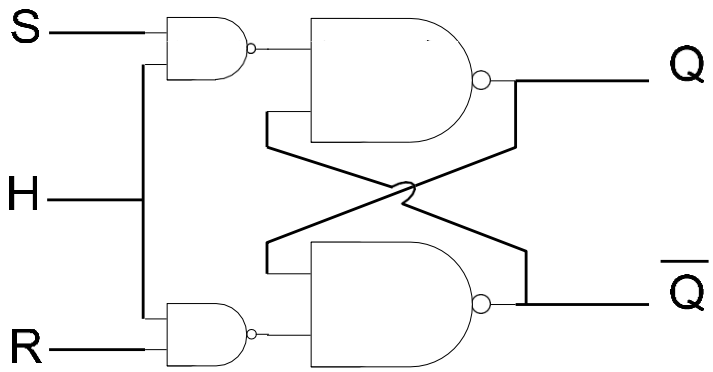
$$Q^+ = \overline{R}Q + SH + \overline{H}Q$$
$$= Q\left(\overline{R} + \overline{H}\right) + SH$$
$$= Q\overline{HR} + SH$$

Circuit du latch RSH

Il suffit de remplacer, dans l'équation de Q+ du latch RS :

- R par RH
- S par SH



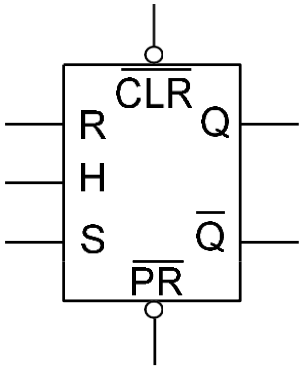


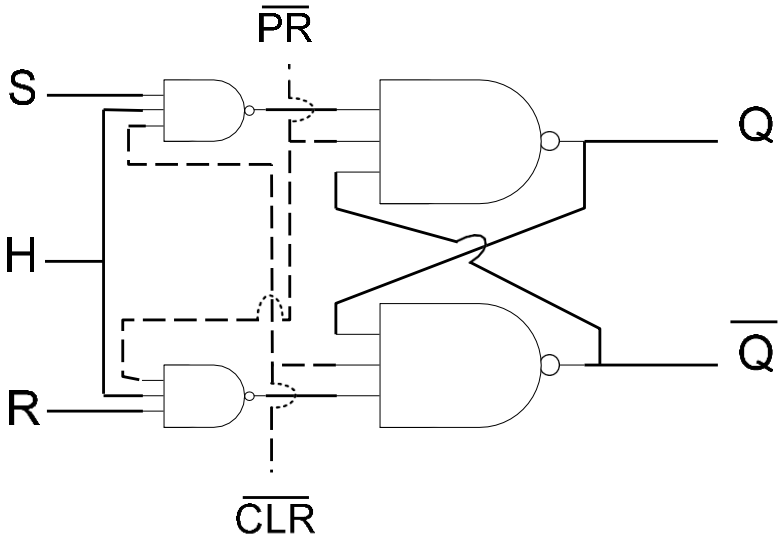
Initialisation du latch RSH

Entrées de forçage asynchrones (indépendantes de H)

- PR (preset) : forçage à 1
- CLR (clear) : forçage à 0

PR et CLR sont actives au niveau '0'





I : Eléments de mémoire : latches et bascules

1. Latch RS
2. **Latch et bascule D**
3. Bascule T
4. Latch et bascule JK
5. Tables de transition des bascules

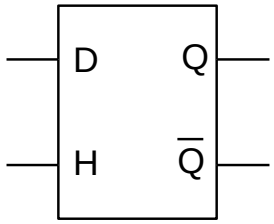
Latch D

Latch D

Entrée de données D

Entrée de validation H

2 sorties complémentaires Q et $\overline{Q}$



Fonctionnement sur niveau de H

H=0 : la sortie maintient son état quelque soit le niveau appliqué à D ($Q^+=Q$)

H=1 : la sortie Q recopie l'état de D ($Q^+=D$)

Table de vérité

D	H	Q+
x	0	Q
0	1	0
1	1	1

Table de Karnaugh

Q \ DH	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$$Q^+ = HD + Q\overline{H} + DQ$$

$$= HD + Q\left(\overline{H} + D\right)$$

Circuit du latch D

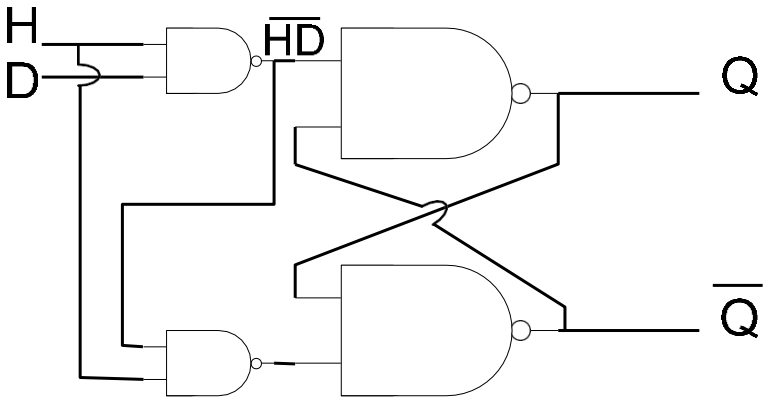
On a: $Q^+ = \overline{\overline{HD} + Q\left(\overline{H} + D\right)}$

$= \overline{\overline{HD}.\overline{Q\left(\overline{H} + D\right)}}$

$= \overline{\overline{HD}.\overline{Q\left(\overline{H\,\overline{D}}\right)}}$

Or : $H\,\overline{D} = H\,\overline{H} + H\,\overline{D} = H\left(\overline{H} + \overline{D}\right) = H.\overline{HD}$

Donc : $Q^+ = \overline{\overline{HD}.\overline{Q\left(\overline{H.\overline{HD}}\right)}}$



Bascule D

- Entrée Dcopiée à la sortie selon une horloge active sur une transition

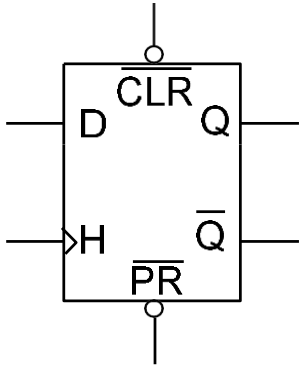
✓ Problème du latch :

- Sensible aux parasites
- Ne peut pas compter

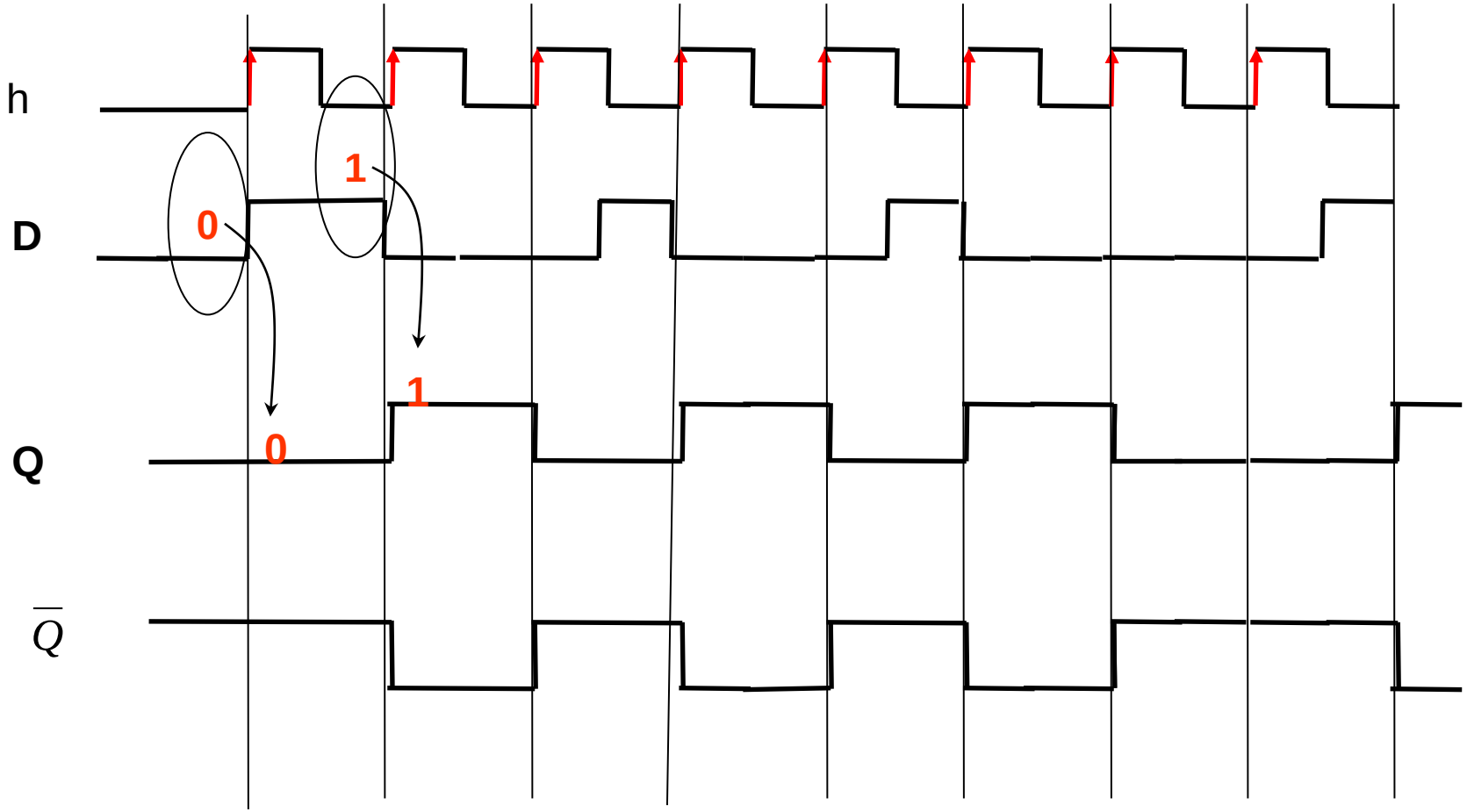
bascule D sur front montant

Au front montant, Q recopie D

PR	CLR	H	D	Q+	Mode
0	1	X	X	1	Asynchrone
1	0	X	X	0	
0	0	X	X	X	
1	1	0,1,↓	X	Q	Synchrone
1	1	↑	0	0	
1	1	↑	1	1	



Chronogramme d'une bascule D



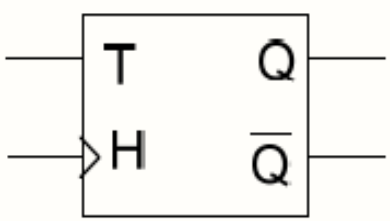
I : Eléments de mémoire : latches et bascules

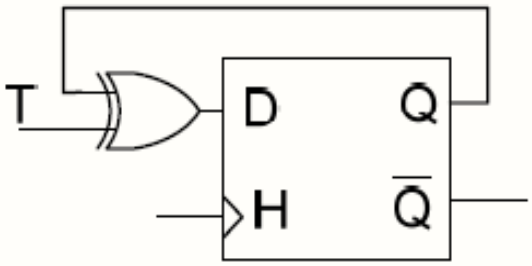
1. Latch RS
2. Latch et bascule D
3. **Bascule T**
4. Latch et bascule JK
5. Tables de transition des bascules

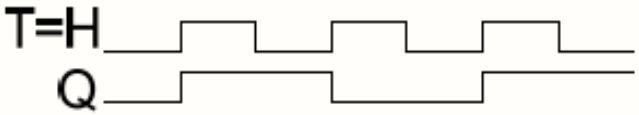
Bascule T

- Élément de mémoire à une seule entrée
- Lors d'une transition d'horloge T, la sortie sera inversée si T est actif et conservera son état sinon.

T	Q+	Mode
0	Q	Maintien
1	$\overline{Q}$	Basculement







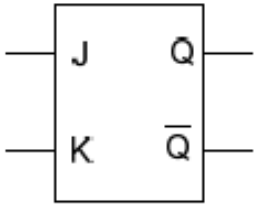
Bascule T fonctionne en diviseur de fréquence par 2.

I : Eléments de mémoire : latches et bascules

1. Latch RS
2. Latch et bascule D
3. Bascule T
4. **Latch et bascule JK**
5. Tables de transition des bascules

Latch JK

- Latch JK
  - 2 entrées synchrones J et K
  - 2 sorties complémentaires Q et $\overline{Q}$
- Fontionnement
  - J=K=0 : mémorisation
  - J=1 et K=0 : mise à 1
  - J=0 et K=1 : mise à 0
  - J=K=1 : mode basculement (commutation)



J	K	Q+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Q^+ = J\overline{Q} + Q\overline{K}$$

Circuit du latch JK

Latch JK : $Q^+ = J\overline{Q} + Q\overline{K}$

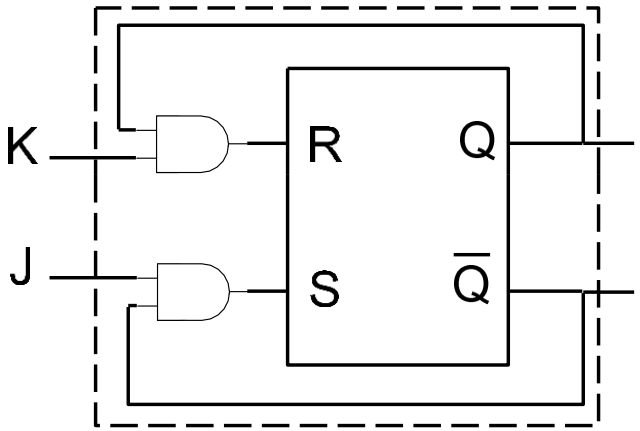
$= J\overline{Q} + Q\overline{Q} + Q\overline{K}$

$= Q\left(\overline{Q} + \overline{K}\right) + J\overline{Q}$

$= Q\overline{QK} + J\overline{Q}$

Latch RS : $Q^+ = \overline{R}Q + S$

Il suffit de remplacer S par $J\overline{Q}$ et R par QK.



I : Eléments de mémoire : latches et bascules

1. Latch RS
2. Latch et bascule D
3. Bascule T
4. Latch et bascule JK
5. **Tables de transition des bascules**

Rappel des bascules

Bascule JK (front montant de H)

J	K	Q	Q+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Bascule D (front montant de H)

D	Q	Q+
0	0	0
0	1	0
1	0	1
1	1	1

Bascule T (front montant de T)

T	Q	Q+
0	0	0
0	1	1
1	0	1
1	1	0

Table de transition

- t_{xy} : transition de la sortie du niveau x au niveau y provoquée par le front d'horloge
- 4 cas possibles : t_{00} , t_{01} , t_{10} , t_{11}
- Table de transition : ensemble des niveaux qui doivent être présents sur les entrées pour provoquer la transition voulue au front d'horloge.
- Conception des machines synchrones

	Etat présent Q	Etat futur Q+	Bascule D	Bascule T	Bascule JK	
					J	K
t_{00}	0	0	0	0	0	X
t_{01}	0	1	1	1	1	X
t_{10}	1	0	0	1	X	1
t_{11}	1	1	1	0	X	0

II :Les registres et les compteurs

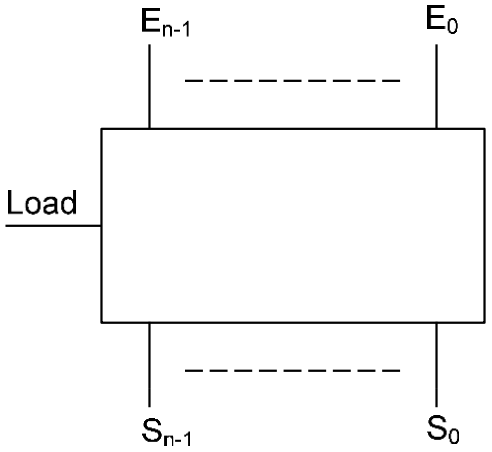
1. **Les registres**
2. Les compteurs asynchrones
3. Les compteurs synchrones

Registres de mémorisation

Principe

Si Load est active, les sorties recopient les entrées

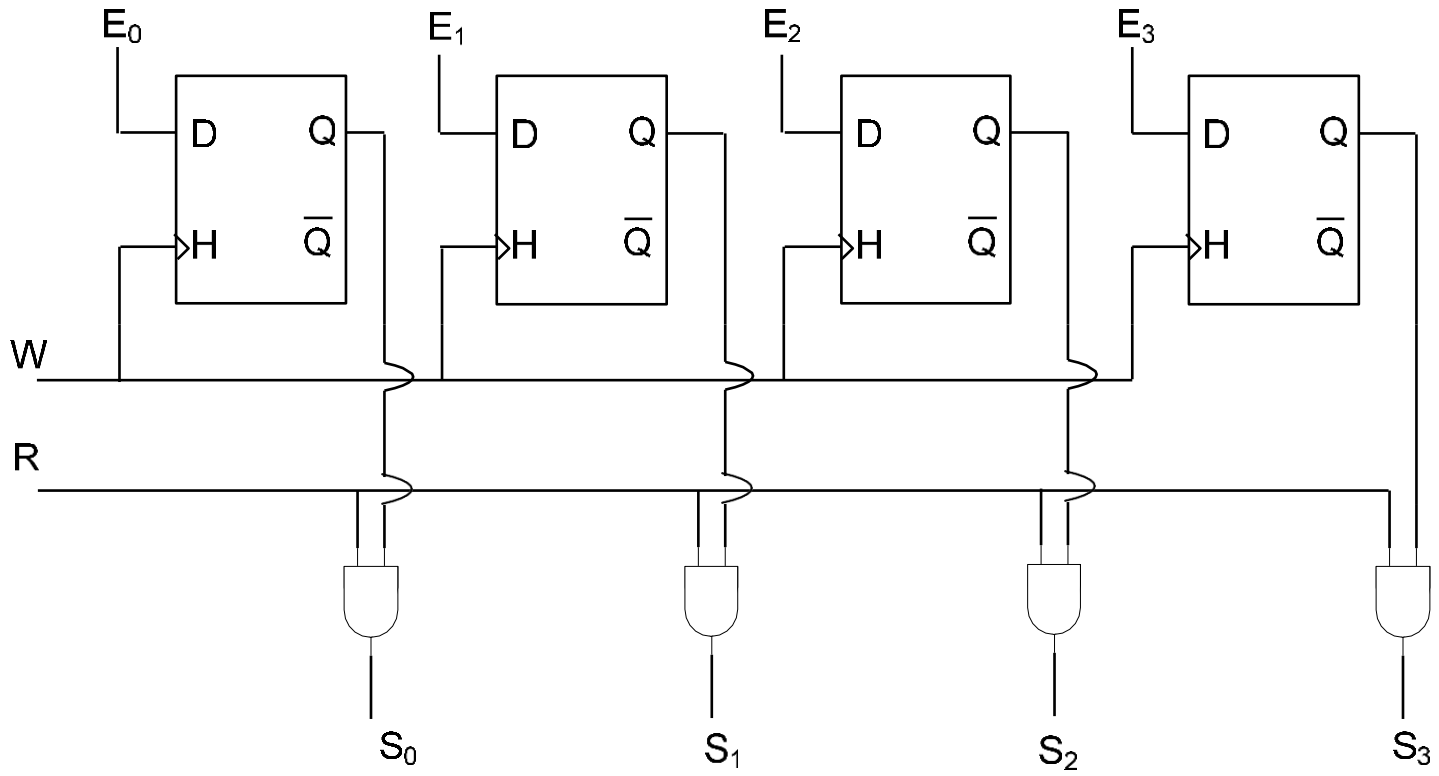
Sinon les sorties restent inchangées quelque soit les entrées.



Réalisation

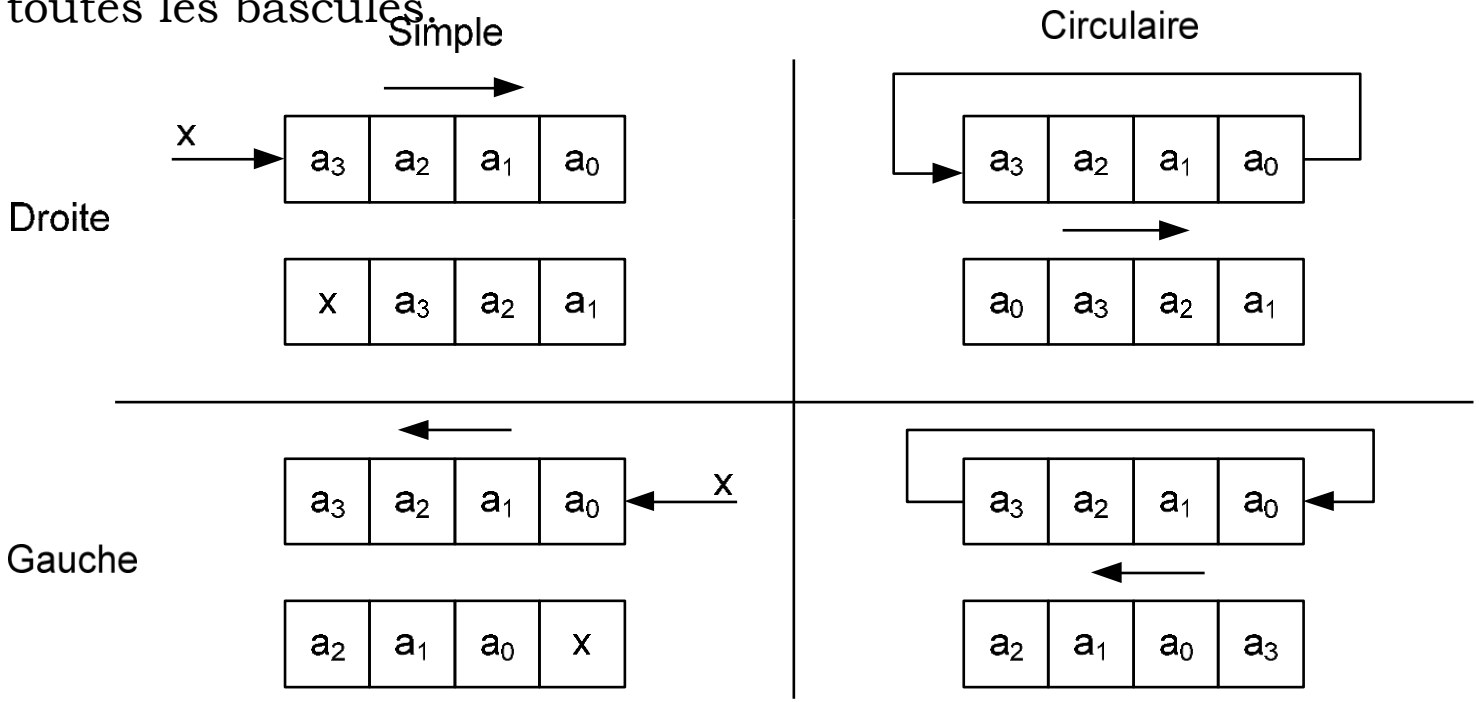
Au moyen de bascules de recopie (bascules D)

Registre de mémorisation de taille 4 bits

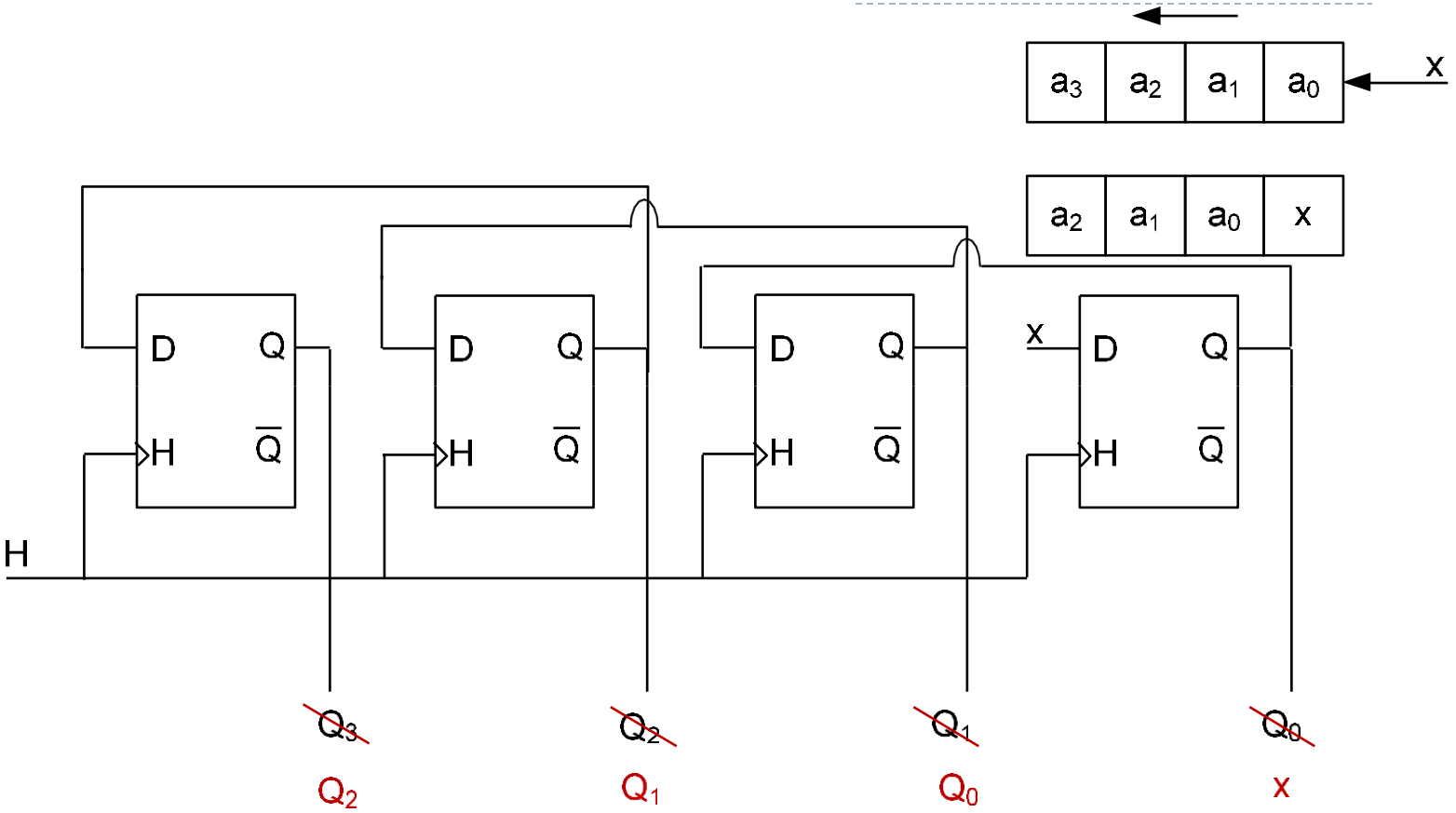


Registres à décalage

Un registre à décalage est constitué de bascules interconnectées de façon à ce que l'état de la sortie de l'étage *i* soit transmis à l'étage suivant (*i+1*) à chaque cycle d'horloge appliquée simultanément à toutes les bascules.



Registre à décalage simple gauche de taille 4 bits



Application : Réaliser un registre à décalage circulaire vers la droite de taille 4 bits.

Application des registres à décalage

- Retard numérique (Entrées série, sorties série)
- Convertisseur de données série/parallèle
- Interfaces USB, disque dur, télévision
- Simulation physique des files d'attente : chaînes de production

II :Les registres et les compteurs

1. Les registres
2. Les compteurs asynchrones
3. Les compteurs synchrones

Les compteurs

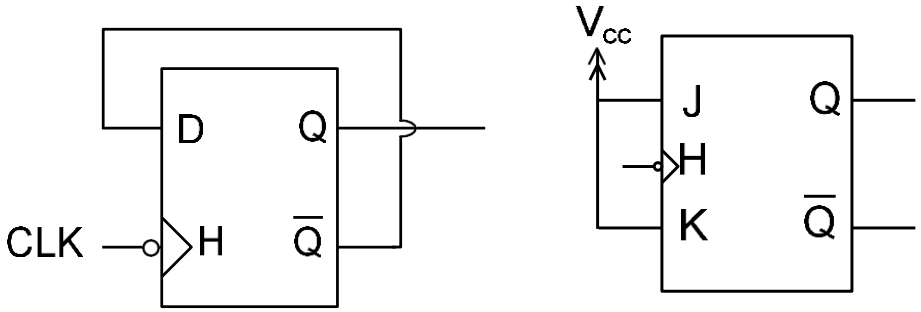
- Circuit logique séquentiel ayant un fonctionnement cyclique
  - Fonctionnement piloté par une horloge
  - Chaque état affiché par les sorties est appelé moment
  - Un compteur modulo N possède un cycle composé de N moments distincts

- Deux types :
  - Compteur asynchrone
  - Compteur synchrone

- Réalisation des compteurs basée sur des :
  - Bascules D
  - Bascules JK

Compteurs asynchrones

- Réalisation : mise en cascade de bascules D ou JK câblées en diviseur de fréquence par 2.



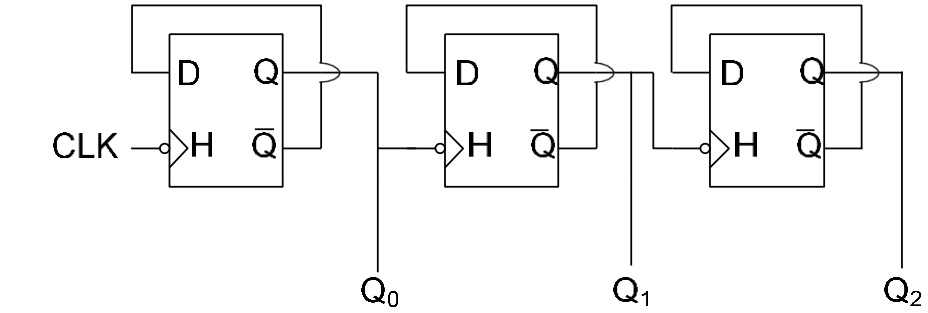
  - La sortie de chaque bascule pilote la bascule suivante
  - On obtient des <u>compteurs</u> si la sortie de rang n (Q_n) de la $n^{ième}$ <u>bascule</u> évolue sur le front <u>descendant</u> de la sortie précédente (Q_{n-1}).
  - On obtient des <u>décompteurs</u> si la sortie de rang n (Q_n) de la $n^{ième}$ <u>bascule</u> évolue sur le front <u>montant</u> de la sortie précédente (Q_{n-1}).

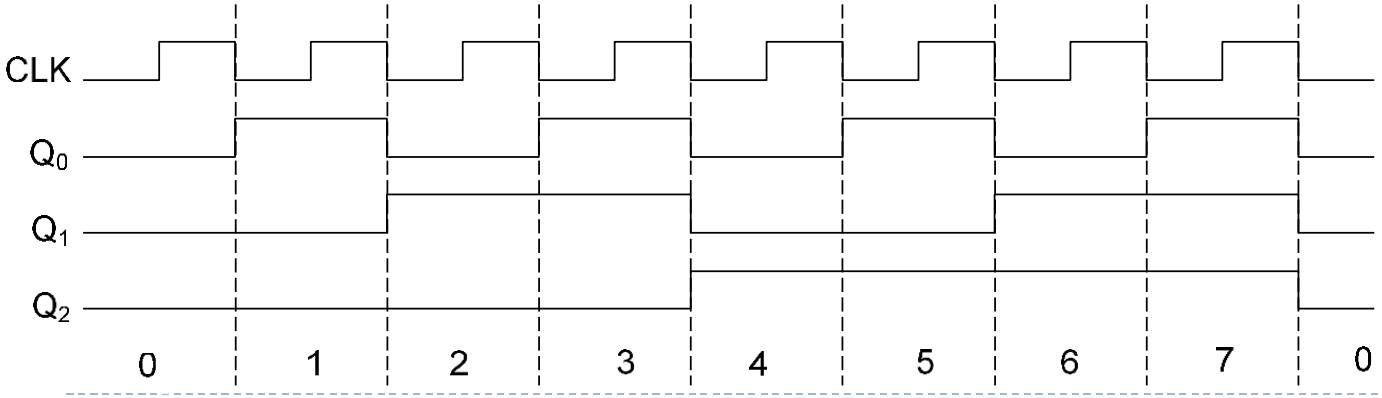
Compteur modulo 8

moment	Q_2	Q_1	Q_0
m_0	0	0	0
m_1	0	0	1
m_2	0	1	0
m_3	0	1	1
m_4	1	0	0
m_5	1	0	1
m_6	1	1	0
m_7	1	1	1

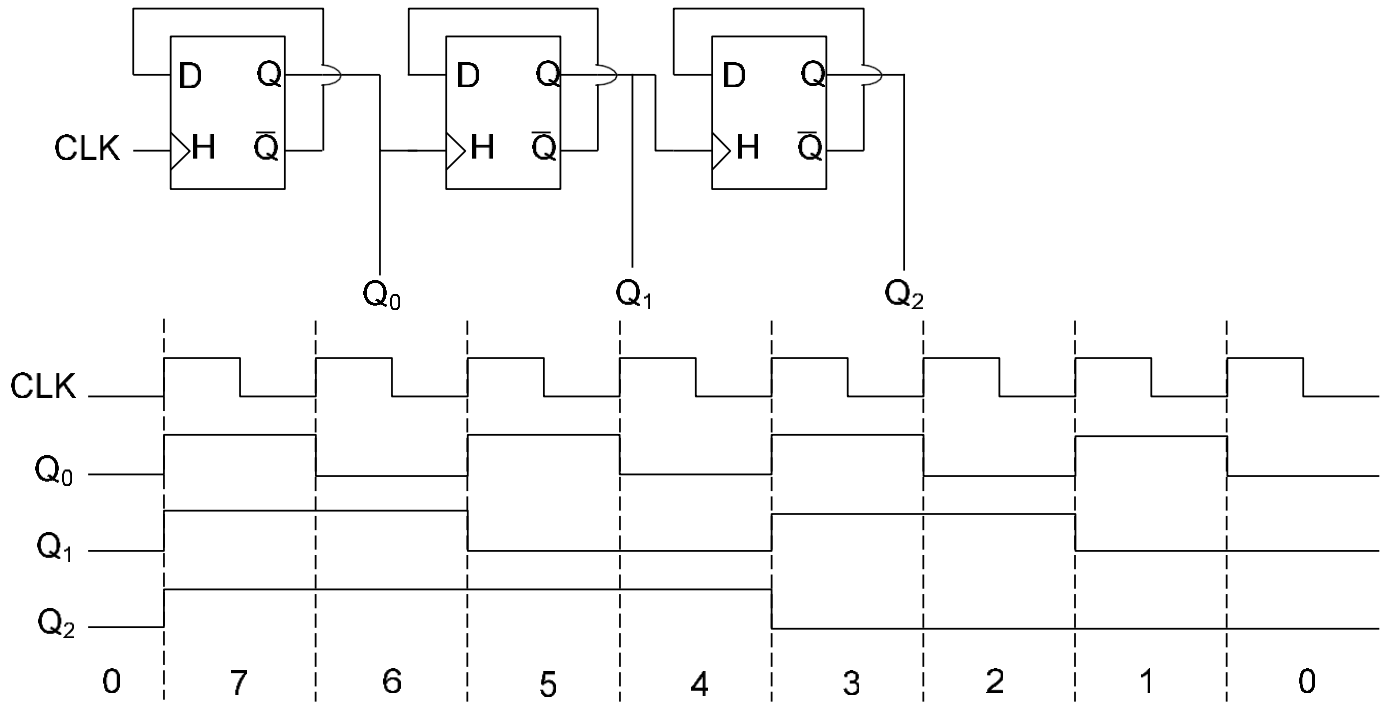
Table de séquencement

Logigramme et chronogramme





Décompteur asynchrone modulo 8



Compteur asynchrone à cycle incomplet (1/3)

- Compteur asynchrone modulo 2^m : cascade de m bascules câblées en diviseur de fréquence par 2 et compte de 0 à $2^m - 1$
- ✓ Compteur asynchrone à cycle incomplet :
  - Compteur modulo N avec $2^{m-1} < N < 2^m$
  - cascade de m bascules câblées en diviseur de fréquence par 2 et compte de 0 à N-1

- Exemple : compteur modulo 6
  - **m_6 est un moment fugitif**

moment	Q_2	Q_1	Q_0
m_0	0	0	0
m_1	0	0	1
m_2	0	1	0
m_3	0	1	1
m_4	1	0	0
m_5	1	0	1
m_6	1	1	0
m_0	0	0	0

Passage à forcer

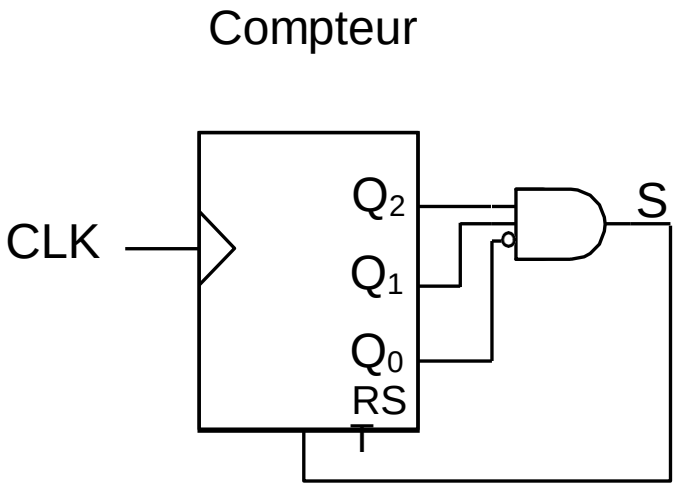
Compteur asynchrone à cycle incomplet (2/3)

- Détection et décodage d'un moment m_k
- ✓ Il suffit de définir une sortie S telle que :
  - S=1 lors de la présence d'un moment m_k

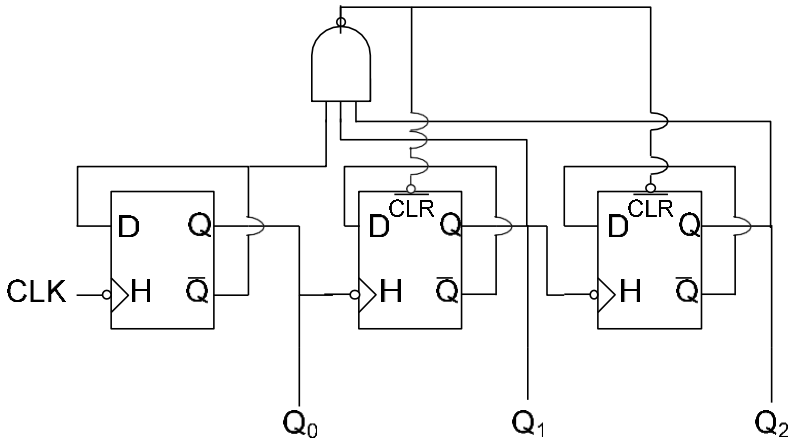
  S=0 sinon
- ✓ Exemple : moment m_6

Q_2	Q_1	Q_0	S
1	1	0	1
x	x	x	0

$$S = Q_2 Q_1 \overline{Q_0}$$



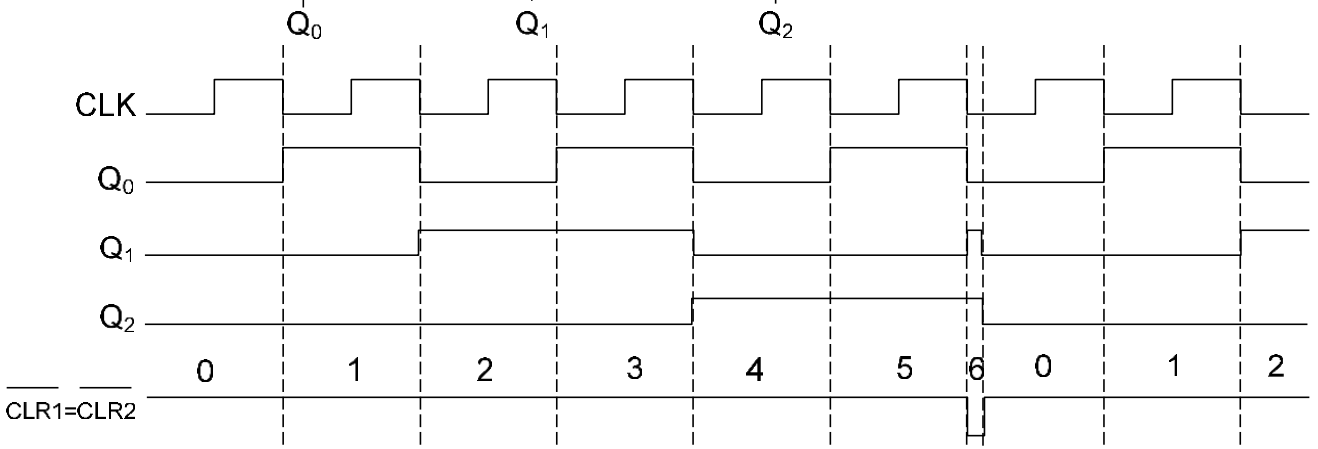
Compteur asynchrone à cycle incomplet (3/3)



Compteur modulo 6 : exploiter m_6 pour générer m_0

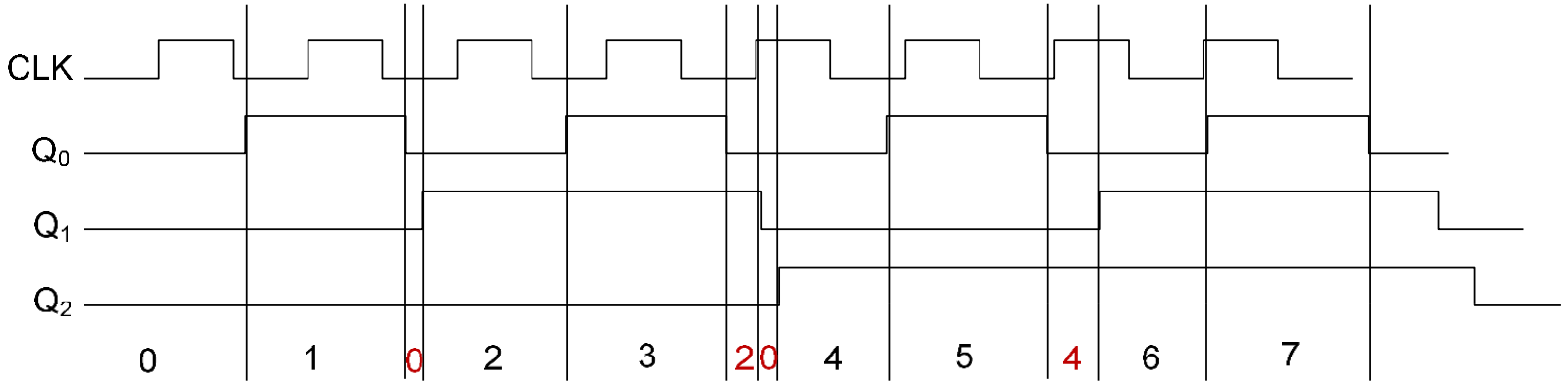
m_6 va servir de forçage à 0 des sorties Q_2 et Q_1 (Q_0 déjà à 0)

Si $Q_2Q_1Q_0$ doit activer CLR_2 et $\overline{CLR_1}$ afin de mettre Q_2 et Q_1 à 0.



Inconvénients des compteurs asynchrones

Temps de retard de basculement au temps de réponse des bascules



Moments transitoires (0, 2, 0, 4) qui causent une erreur lors du comptage

Cause : temps de propagation tp

Le temps de propagation maximal (propagation de l'information de l'horloge à la dernière sortie) correspond à n.tp avec n le nombre de bascules.

Ne pas utiliser des compteurs asynchrones à des fréquences élevées.

II :Les registres et les compteurs

1. Les registres
2. Les compteurs asynchrones
3. **Les compteurs synchrones**

Compteur synchrone

- Toutes les bascules sont pilotées par le même signal d'horloge
- Transitions des états : réalisées par la programmation des entrées JK, D ou T selon le type de bascule utilisée.

- Démarche de synthèse (méthode de Marcus)

1. Nombre de bascules JK, D ou T
2. Table des transitions complète du compteur
3. Expressions simplifiées des entrées JK, D ou T
4. Réalisation

❖ Réaliser un compteur synchrone de cycle (0, 3, 2, 7, 4, 0, …)

Réponses à la démarche (1/3)

1. 3 bascules T

Q	Q+	T
0	0	0
0	1	1
1	0	1
1	1	0

2. Table de transition

Q2	Q1	Q0	Q2+	Q1+	Q0+	T2	T1	T0
0	0	0	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1
0	1	0	1	1	1	1	0	1
1	1	1	1	0	0	0	1	1
1	0	0	0	0	0	1	0	0

Réponses à la démarche (2/3)

3. Expression de T0

Q2 \ Q1Q0	00	01	11	10
0	1	X	1	1
1	0	X	1	X

$$T_0 = \overline{Q_2} + Q_1$$

Expression de T1

Q2 \ Q1Q0	00	01	11	
0	1	X	0	
1	0	X	1	

$$T_1 = \overline{Q_2}\,\overline{Q_1} + Q_2 Q_1 = Q_1 \oplus \overline{Q_2}$$

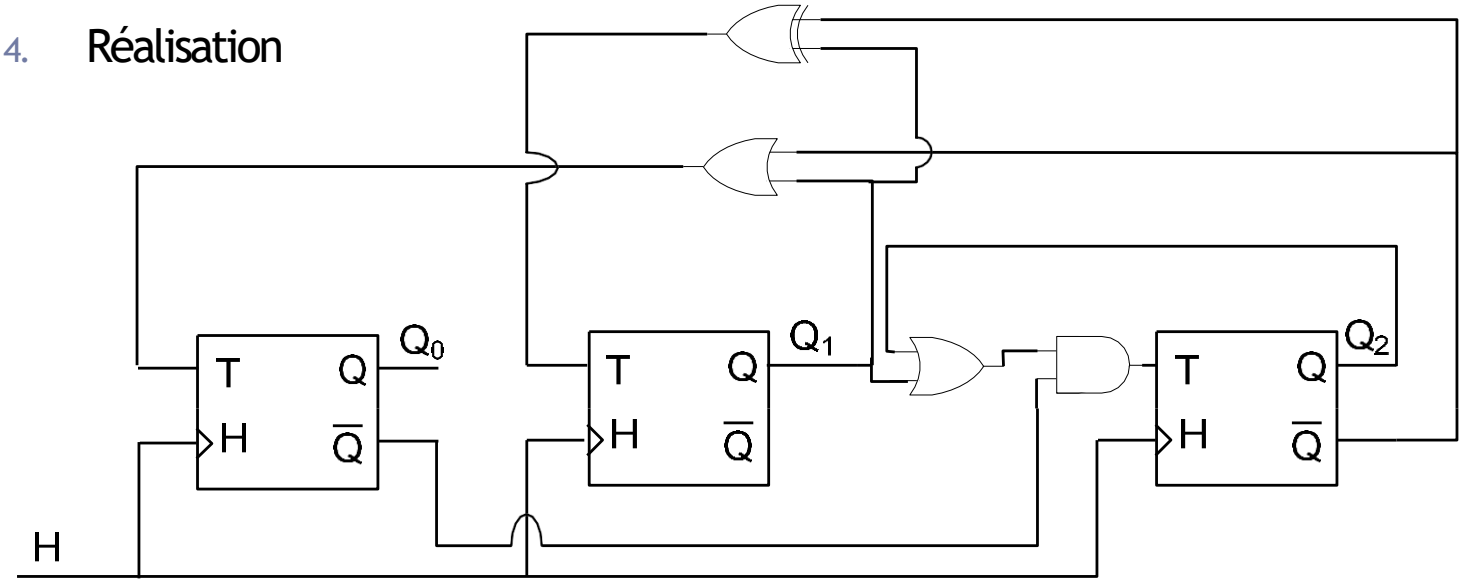
Expression de T2

Q2 \ Q1Q0	00	01	11	10
0	0	X	0	1
1	1	X	0	X

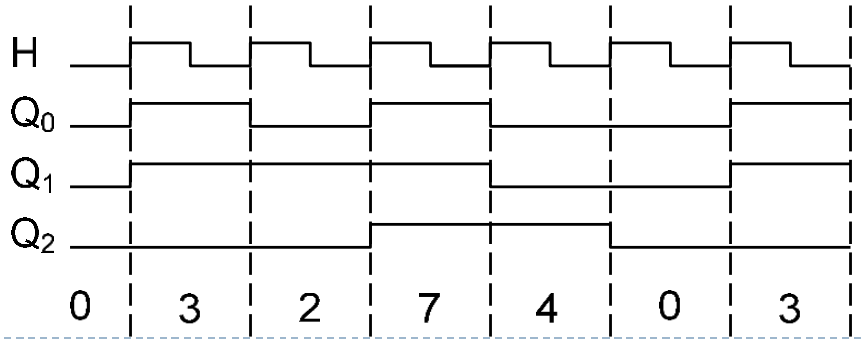
$$T_2 = Q_1\overline{Q_0} + Q_2\overline{Q_0} = \overline{Q_0}(Q_1 + Q_2)$$

Réponses à la démarche (3/3)

4. **Réalisation**



5. **Chronogramme**



Compteur synchrone modulo 6 à base de bascules JK (1/4)

Table de transition de la bascule JK

Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

1. 3 bascules JK
2. Table de transition du compteur

Q2	Q1	Q0	Q2+	Q1+	Q0+	J2	K2	J1	K1	J0	K0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1

Compteur synchrone modulo 6 à base de bascules JK (2/4)

3. Expression de J2

Q2 \ Q1Q0	00	01	11	10
0	0	0	1	0
1	x	x	x	x

$$J_2 = Q_1 Q_0$$

Expression de K2

Q2 \ Q1Q0	00	01	11	10
0	x	x	x	x
1	0	1	x	x

$$K_2 = Q_0$$

Expression de J1

Q2 \ Q1Q0	00	01	11	10
0	0	1	x	x
1	0	0	x	x

$$J_1 = \overline{Q_2} Q_0$$

Compteur synchrone modulo 6 à base de bascules JK (3/4)

Expression de K1

Q2 \ Q1Q0	00	01	11	10
0	x	x	1	0
1	x	x	x	x

$K_1 = Q_0$

Expression de J0

Q2 \ Q1Q0	00	01	11	10
0	1	x	x	1
1	1	x	x	x

$J_0 = 1$

Expression de K0

Q2 \ Q1Q0	00	01	11	10
0	x	1	1	x
1	x	1	x	x

$K_0 = 1$

Compteur synchrone modulo 6 à base de bascules JK (4/4)