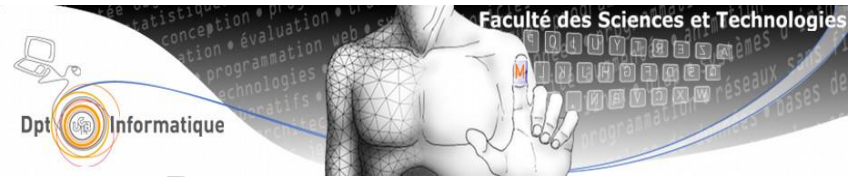


Université Claude Bernard Lyon 1

Circuits et Architecture des Ordinateurs

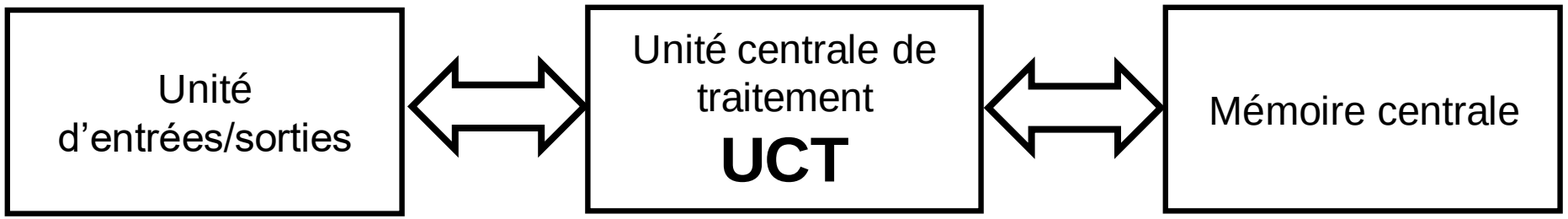
hamid.ladjal@univ-lyon1.fr
hamid.ladjal@liris.cnrs.fr

Vue d'ensemble de l'ordinateur.

- **Architecture de Von Neumann**
- **Fonctions logiques**
- **Circuits combinatoires**
- **Circuits séquentiels**

Architecture de Von Neumann

Dans le modèle de **Von Neumann**, l'ordinateur se compose d'une:



- **Mémoire centrale**, qui contient le **programme** et les **données**;
- **Unité centrale de traitement (UCT),** qui exécute un programme contenu en mémoire centrale;
- **Unité(s) d'entrée-sortie** permettant l'échange d'informations avec l'environnement de **l'UCT**.

Mémoire centrale

RAM (Random Access Memory) :

- la mémoire -→ des cases (lue/écrite) identifiée par une adresse unique.

- Vue comme un tableau de 2^m cases mémoires :

Unité centrale de traitement (UCT)

L'UCT contient une petite quantité de registres :

- Registre est une cellule mémoire, physiquement présente dans **l'UCT**, permettant de stocker un mot de longueur fixée (typiquement, 8, 16, 32 ou 64 bits).

- Registres stockent temporairement les données et les résultats intermédiaires des calculs, ou des informations de contrôle.

- Ils ne sont pas référencés par une adresse en mémoire ; il existe un mode d'adressage dédié pour eux.

Stockage des programmes en mémoire centrale

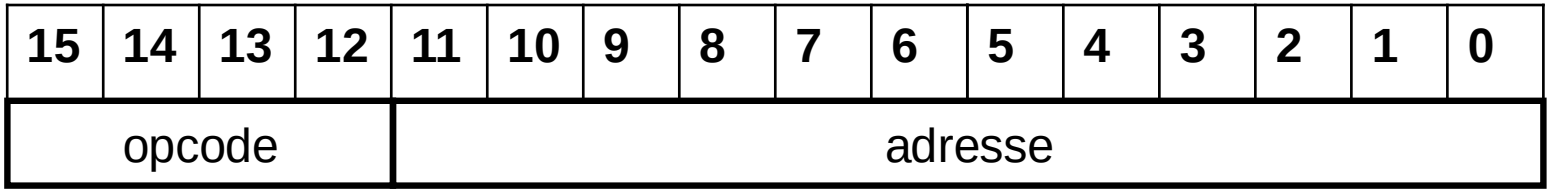
Chaque instruction est stockée en mémoire sous la forme d'un ou plusieurs **mots binaire**.

Le code d'une instruction se décompose en :

- **Code-opération ou opcode**, qui spécifié l'opération qui doit être réalisée lors de l'exécution de l'instruction;

- **opérandes** qui définissent les emplacements où l'instruction doit lire ses sources et écrire son résultat.

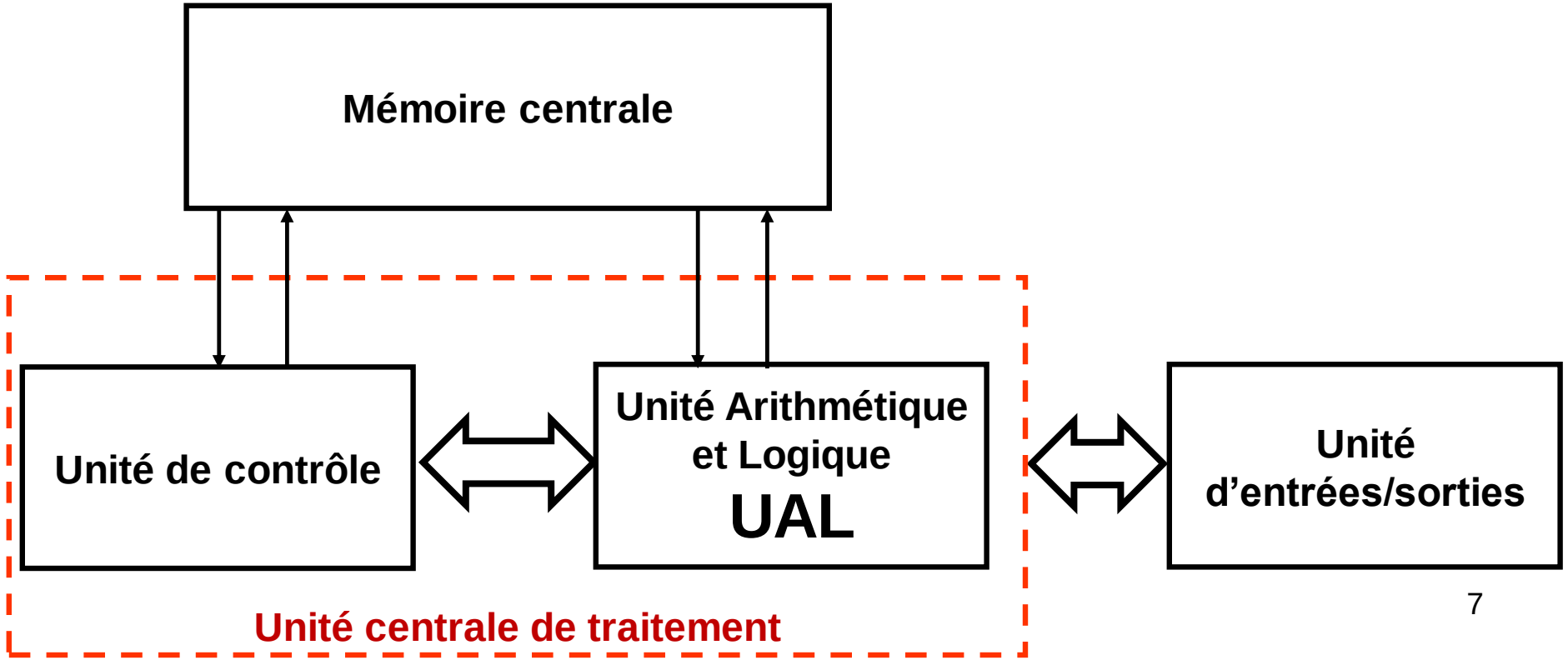
Ex : une machine, les instructions sont codées sur 16 bits :



Unité centrale de traitement

L'UCT contient (au moins) deux unités fonctionnelles :

L'unité de contrôle (UC), pour gérer l'exécution des instructions;
L'unité arithmétique et logique (UAL), pour exécuter les opérations sur les données.



L'unité de contrôle (UC)

L'unité de contrôle (UC) active dans un ordre précis les circuits nécessaires à l'exécution des instructions d'un programme.

Elle utilise deux registres :

Registre d'instruction IR, qui contient l'instruction en cours d'exécution ;

Compteur de programme PC, qui contient l'adresse en mémoire centrale de la prochaine instruction a exécuter.

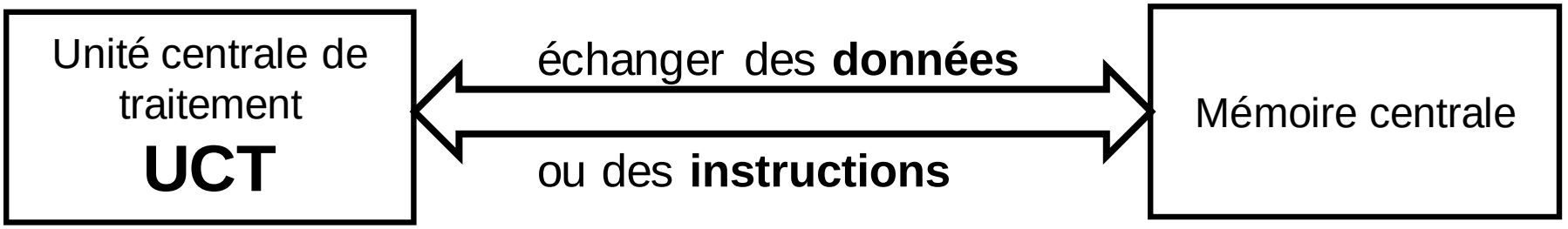
L'unité de contrôle (UC)

L'UC contient un circuit pour le décodage des instructions, qui détermine quelle opération doit être exécutée en fonction de l'opcode de l'instruction courante.

L'UC est synchronisée sur une **horloge**, dont la fréquence détermine celle à laquelle sont exécutées les instructions.

UAL contient les **circuits nécessaires** pour exécuter des instructions du langage machine : addition, soustraction, multiplication, division, opérations logiques (or, and, not. . .).

Unité centrale de traitement



L'UCT utilise deux registres internes :

Un registre d'adresse AR, qui spécifie l'adresse de la mémoire centrale où devra être réalisée la prochaine lecture ou d'écriture.

Un registre de données mémoire MDR est utilisé pour les accès en écriture et les accès en lecture.

Instructions et modes d'adressage

On distingue essentiellement trois classes d'instructions :

Opération arithmétique et logique : addition, soustraction, ou, et, ou-exclusif. . .

Accès mémoire : transfert entre un registre de **l'UCT** et la mémoire;

LOAD : chargement du contenu d'une case mémoire dans un registre,
STORE : rangement du contenu d'un registre dans une case mémoire.

Contrôle du flot d'instructions :
Le **PC** est incrémenté à chaque cycle d'instruction…

Merci pour les cours de **N. Louvet** de LIF6: http://perso.ens-lyon.fr/nicolas.louvet/LIF6/

Exemple LC-3

Le **LC-3** est un processeur développé dans un but pédagogique par Yale N. Patt et J. Patel dans http://highered.mcgraw-hill.com/sites/0072467509/

Le LC3 dispose de 8 registres, notés R0,. . . ,R7.
L'instruction d'addition se décline de deux façons :

ADD DR, SR1, SR2, qui effectue **DR <- SR1 + SR2.** ,
Tous les opérandes sont des registres : **adressage par registre**
→ **Exemple : ADD R1, R2, R3** effectue **R1 <- R2 + R3**.

ADD DR SR1, imm5, qui effectue **DR <- SR1 + imm5**. ,
Le dernier opérande est un **immédiat**
→ **Exemple : ADD R1, R2, 5** effectue **R1 <- R2 + 5**.

Suite:
Le LC3 présente deux instructions d'accès à la mémoire :

LD DR, add, qui effectue **DR <- mem[add].**
ST SR, add, qui effectue **mem[add] <- SR**.

Dans les deux cas, **add** désigne une adresse mémoire, et **mem[add]** la case mémoire d'adresse **add**.

On parle **d'adressage direct** pour le second opérande de ces instructions, car c'est une adresse en mémoire centrale qui est désignée.

Merci pour les cours de **N. Louvet** de LIF6: http://perso.ens-lyon.fr/nicolas.louvet/LIF6/

Instructions et modes d'adressage

La méthode de localisation des opérandes, dans la mémoire ou parmi les registres, est appelé **mode d'adressage** ; on parle d'adressage :

- **par registre** lorsque l'emplacement désigné est simplement un registre.

- **immédiat** lorsque l'opérande est une valeur codée dans l'instruction.

- **direct** lorsque l'opérande est une adresse en mémoire centrale.

Instructions et modes d'adressage

Exemple2: réaliser une petite calculatrice

Voici un exemple d'un programme qui évalue : (10*2)*(9+6)

Language machine:

```
SET R1, 10          // effectue R1 <- 10
SET R2, 2           // effectue R2 <- 2
MUL R3, R1, R2      // effectue R3 <- R1 * R2
SET R1, 9           // effectue R1 <- 9
SET R2, 6           // effectue R2 <- 6
ADD R4, R1, R2      // effectue R4 <- R1 + R2
MUL R5, R3, R4      // effectue R5 <- R3 * R4
```

Architecture de Von Neumann

- Nous avons présenté brièvement le modèle dit de **Von Neumann**, qui permet d'aborder le fonctionnement général des ordinateurs.

- Les outils nécessaires à l'implantation d'une architecture de **Von Neumann simple**, **celle du LC3** : codage et représentation de l'information, **circuit logiques combinatoires et séquentiels**. . .

- Pour la partie LC3, je vous recommande les cours de **N. Louvet** de LIF6: http://perso.ens-lyon.fr/nicolas.louvet/LIF6/

Logique combinatoire

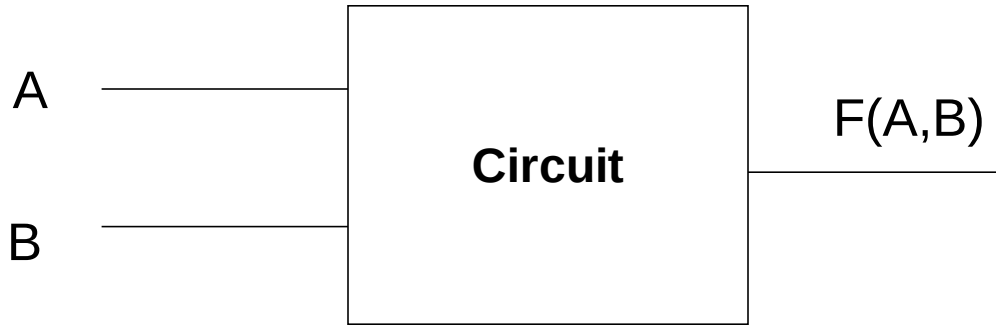
Logique combinatoire

- L’algèbre de Boole
- Opérateurs de base
- Propriétés et les fonctions combinatoires
- Circuits combinatoires:
  - Multiplexeur et démultiplexeur
  - Codeur, décodeur et transcodeur
  - Additionneur et comparateur….
- Circuits séquentiels…

Introduction

- Les machines numériques (ordinateur, tablette, téléphone…) sont constituées d'un ensemble de circuits électroniques.

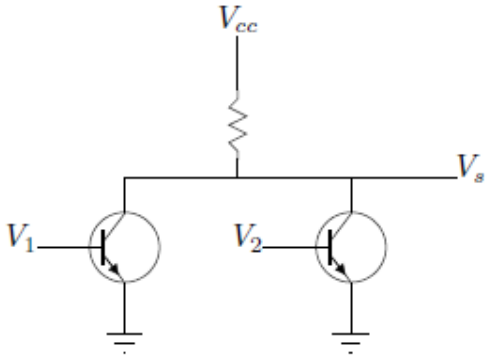
- Chaque circuit fournit une fonction logique bien déterminée; opérations logiques ou arithmétiques (addition, soustraction, comparaison ,….).

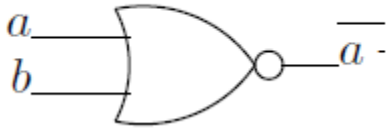


- Une fonction logique de base est réalisée à l'aide des portes logiques qui permettent d'effectuer des opérations élémentaires.

Introduction

- Ces portes logiques sont aujourd'hui réalisées à l'aide de transistors.





- 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.

Algèbre de Boole

1854 : Georges Boole propose une algèbre

Propositions vraies ou fausses
et opérateurs possibles ⟶ Algèbre de Boole

↓

Étude des systèmes binaires :
Possédant deux états s'excluant mutuellement

C'est le cas des systèmes numériques
(des sous ensembles : les circuits logiques)

Algèbre binaire

On se limite : Base de l'algèbre de Boole
Propriétés indispensables aux systèmes logiques

Définitions :

- États logiques : 0 et 1, Vrai et Faux, H et L
  (purement symbolique)

- Variable logique : Symbole pouvant prendre
  comme valeur des états logiques (A,b,c, Out ...)

- Fonction logique : Expression de variables et d'opérateurs
  (f = not(a)^ (c OR r.t))

Calcul propositionnel

Algèbre de Boole sur [0,1] = algèbre binaire
Structure d'algèbre de Boole

- 2 lois de composition interne (LCI)
- 1 application unaire

2 LCI : ET, OU

- Somme (OU, Réunion, Disjonction)

  $s = a + b = a \vee b$
- Produit (ET, intersection, Conjonction)

  $s = a . b = ab = a \wedge b$

Application unaire :

- Not (complémentation, inversion, négation, non) $s = \overline{a}$ = not(a) = $\neg a$

Fonctions logiques

Fonction logique à n variables f(a,b,c,d,...,n)

$$[0,1]^n \longrightarrow [0,1]$$

- Une fonction logique ne peut prendre que deux valeurs
- Les cas possibles forment un ensemble fini (2^n)

- Descriptions, preuves possibles par énumération
  comparer f(a,b,c,..n) et g(a,b,c,..,n)
  = comparer les tables représentant f et g

La table de fonction logique = table de vérité

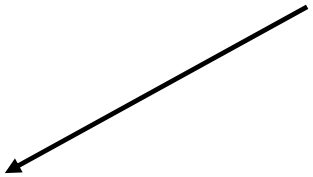
Opérateurs logiques de base

OU (OR)

- Le OU est un opérateur binaire (deux variables) , à pour rôle de réaliser la somme logique entre deux variables logiques.
- Le OU fait la disjonction entre deux variables.
- Le OU est défini par F(A,B)= A + B (il ne faut pas confondre avec la somme arithmétique)

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

table de vérité



ET (AND)

- Le ET est un opérateur binaire (deux variables) , à pour rôle de réaliser le Produit logique entre deux variables booléennes.
- Le ET fait la conjonction entre deux variables.

- Le ET est défini par : F(A,B)= A . B

A	B	A . B
0	0	0
0	1	0
1	0	0
1	1	1

NON (négation)

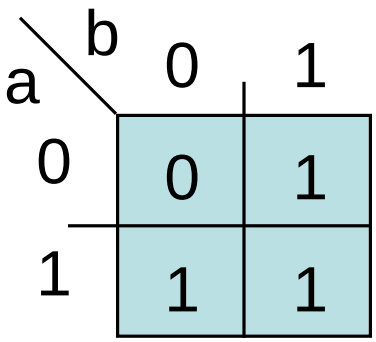
- NON : est un opérateur unaire (une seule variable) qui à pour rôle d'inverser la valeur d'une variable .

F(A)= Non A = $\overline{A}$

(lire : A barre)

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

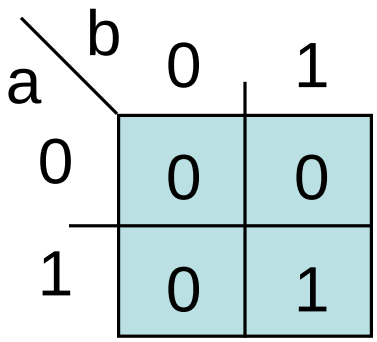
Tables de vérité de ET, OU, NON



$s = a + b$

S est vrai si a OU b est vrai.

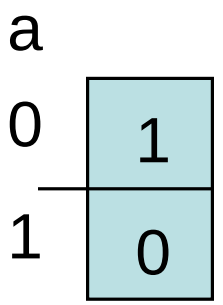
a	b	s
0	0	0
0	1	1
1	0	1
1	1	1



$s = a \cdot b$

S est vrai si a ET b sont vrais.

a	b	s
0	0	0
0	1	0
1	0	0
1	1	1



$s = \overline{a}$

S est vrai si a est faux

a	s
0	1
1	0

Deux autres opérateurs : NAND,NOR

a \ b	0	1
0	1	1
1	1	0

$s = a \uparrow b = \overline{a.b}$

S est vrai si a OU b est faux.

NAND (Not-AND)

a \ b	0	1
0	1	0
1	0	0

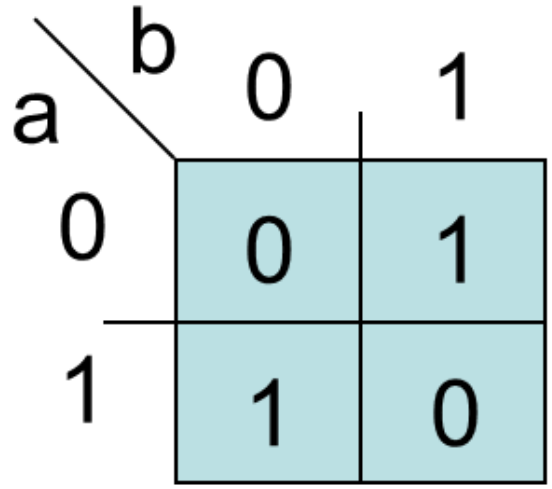
$s = a \downarrow b = \overline{a+b}$

S est vrai si ni a, ni b ne sont vrais.

NOR (Not-OR)

NAND et NOR ne sont pas associatifs

Encore un opérateur : XOR



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

S est vrai si a OU b est vrai mais pas les deux.

XOR (Ou-Exclusif) vaut 1 si a est différent de b
Opérateur de différence (disjonction)

Encore un opérateur : XOR

XOR est associatif $s = a \oplus b \oplus c \quad \ldots\ldots \oplus n$

vaut 1 si le nombre de variables à 1 est impair.

$$s = \overline{a \oplus b} = \overline{a} \oplus b = a \oplus \overline{b} = a \text{ XNOR } b$$

XNOR = $\overline{\text{XOR}}$ vaut 1 si $a = b$

Inverseur programmable : (le programme vaut 0 ou 1)

$a \oplus 1 = \overline{a} \qquad a \oplus 0 = a$

Simplification des fonctions logiques

Simplification /optimisation ?

Méthodes «classiques» de simplifications :

- pas de solution unique
- indépendant de la technologie
- le temps n'est pas pris en compte

La simplification «mathématique» n'est pas toujours optimale en regard des critères d'optimisation technologiques.

Simplification des fonctions logiques

- L'objectif de la simplification des fonctions logiques est de :
  - réduire le nombre de termes dans une fonction
  - et de réduire le nombre de variables dans un terme

- Cela afin de réduire le nombre de portes logiques utilisées → réduire le coût du circuit

- Plusieurs méthodes existent pour la simplification :

1) **Les méthodes algébriques**
2) **Les méthodes graphiques : (ex : tableaux de karnaugh)**

Propriétés de ET,OU,NON

1) Les méthodes algébriques

- **Commutativité**

$a+b = b+a$

$a.b = b.a$

- **Associativité**

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

$a.(b.c) = (a.b).c$

- **Distributivité**

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

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

- **Idempotence**

$a+a = a$

$a.a = a$

- **Absorption**

$a+a.b = a$

$a.(a+b) = a$

- **Involution**

$\overline{\overline{a}} = a$

Propriétés de ET,OU,NON

Les méthodes algébriques

- **Elément neutre**

$a+0 = a$

$a.1 = a$

- **Elément absorbant**

$a+1 = 1$

$a.0 = 0$

- **Inverse**

$a+\overline{a} = 1$

$a.\overline{a} = 0$

- **Théorème de "De Morgan"**

$$\overline{a+b} = \overline{a} \,.\, \overline{b}$$

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

$$\overline{\sum_i x_i} = \prod_i \overline{x_i}$$

$$\overline{\prod_i x_i} = \sum_i \overline{x_i}$$

- **Théorème du Consensus**

$$a.x+b.\overline{x}+a.b = a.x+b.\overline{x}$$

$$(a+x)(\overline{b}+x)(a+b)=(a+x)(\overline{b}+x)$$

Propriétés de ET,OU,NON

Exercice 1:

Démontrez la proposition suivante :

$$ABC + AB\overline{C} + A\overline{B}CD = AB + ACD$$

$$A\,B\,C + \overline{A}BC + A\overline{B}C + AB\overline{C} = BC + AC + AB$$

Propriétés de ET,OU,NON

Correction

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

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

$$B.C + A.C + A.B$$

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

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

$$= A(B + \overline{B}(CD))$$

$$= A(B + CD)$$

$$= AB + ACD$$

Simplification par la table de Karnaugh

Description de la table de karnaugh

•La méthode consiste à mettre en évidence par une méthode graphique (un tableau) tous les termes qui sont adjacents (qui ne différent que par l'état d'une seule variable).

•Un tableau de Karnaugh = table de vérité de 2^n cases avec un changement unique entre 2 cases voisines d'où des codes cycliques (Gray ou binaire réfléchi).

•La méthode peut s'appliquer aux fonctions logiques de 2,3,4,5 et 6 variables.

•Les tableaux de Karnaugh comportent 2^n cases (n: est le nombre de variables).

Description de la table de karnaugh

Règles de regroupement :

- groupe de 2^n cases : 1,2,4 ou 8
- en ligne, colonne, rectangle, carré, mais pas diagonale
- tous les 1, mais pas les 0 au moins une fois dans les groupements

Règles de minimisation de la fonction :

- rechercher les groupements en commençant par les cases qui n'ont qu'une seule façon de se grouper
- rechercher les groupements les plus grands
- les groupements doivent contenir au moins un 1 non utilisé par les autres groupements
- L'expression logique finale est la réunion (la somme) des groupements après simplification et élimination des variables qui changent d'état.

Description de la table de karnaugh

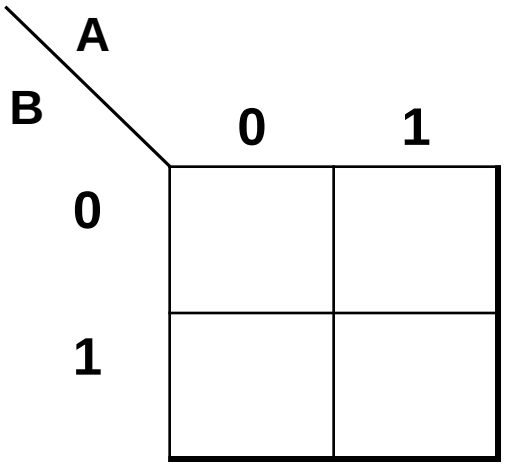
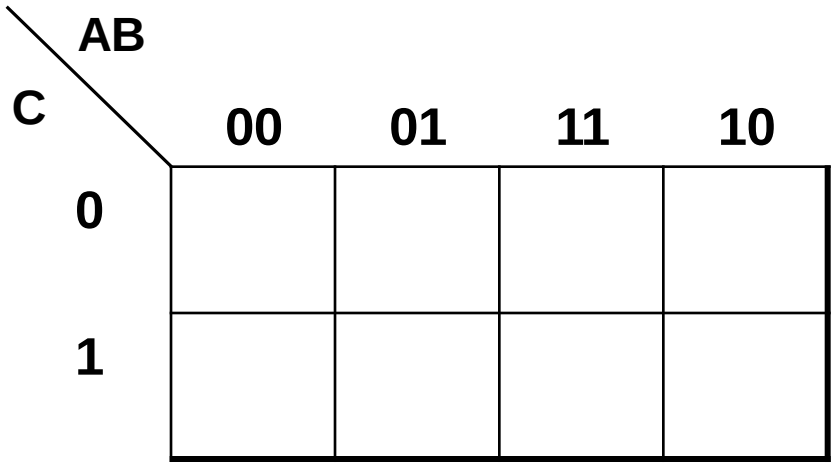


Tableau à 2 variables

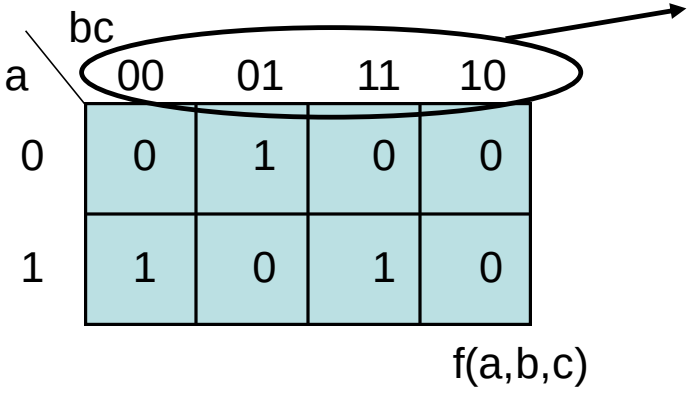


Tableaux à 3 variables

Tableaux de Karnaugh

f (a,c,d, ..,n) fonction logique à N entrées sera représentée par
une table à 2^N lignes
un tableau à 2^N cases

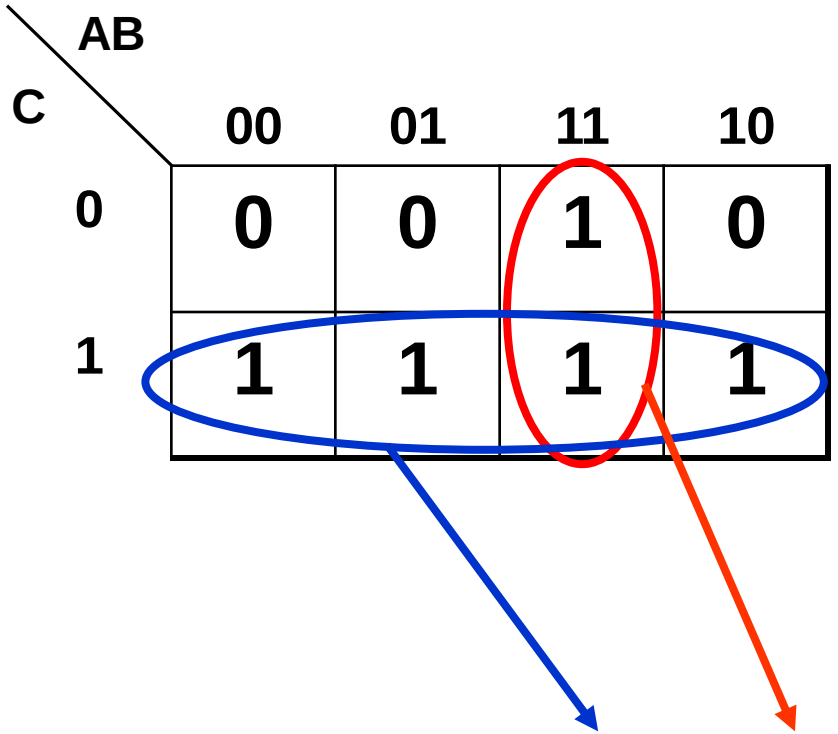
a b c	f(a,b,c)
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1



Code Gray ou
binaire réfléchi
=
1 seul changement
entre 2 codes
successifs

Tableaux de Karnaugh

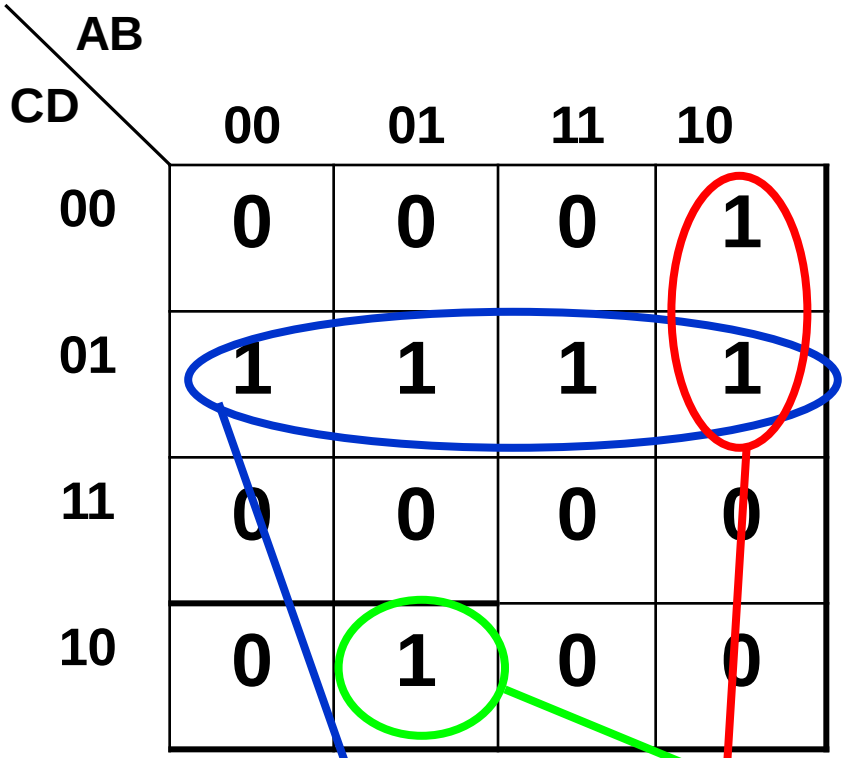
Exemple 1 : 3 variables



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

Tableaux de Karnaugh

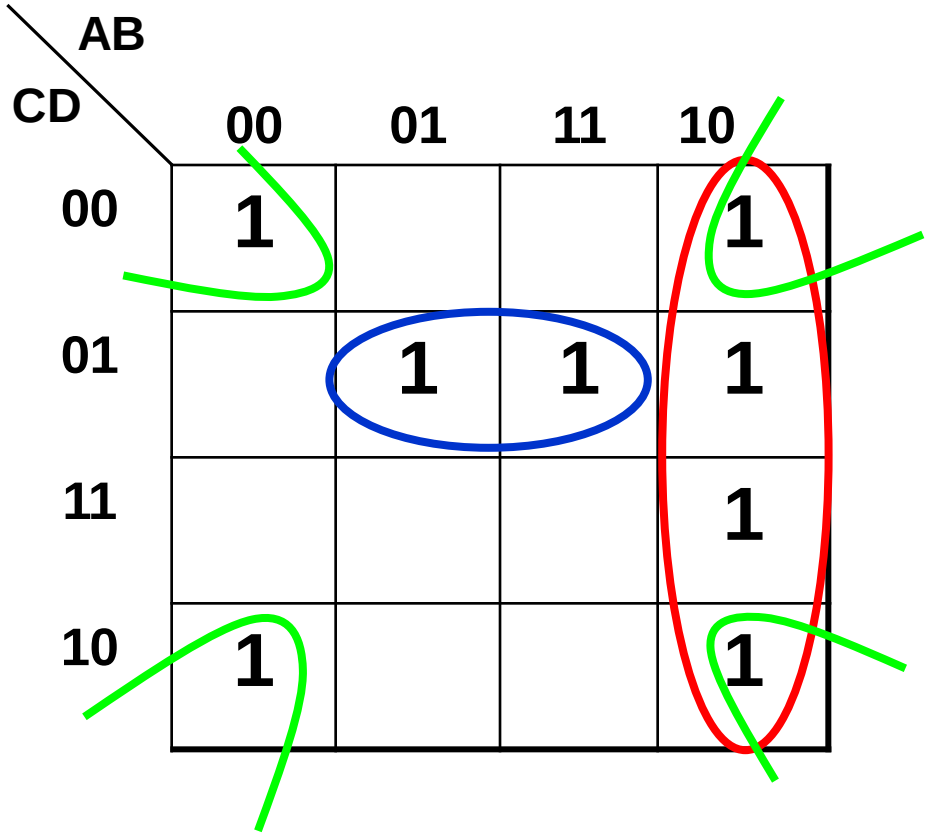
Exemple 2 : 4 variables



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

Tableaux de Karnaugh

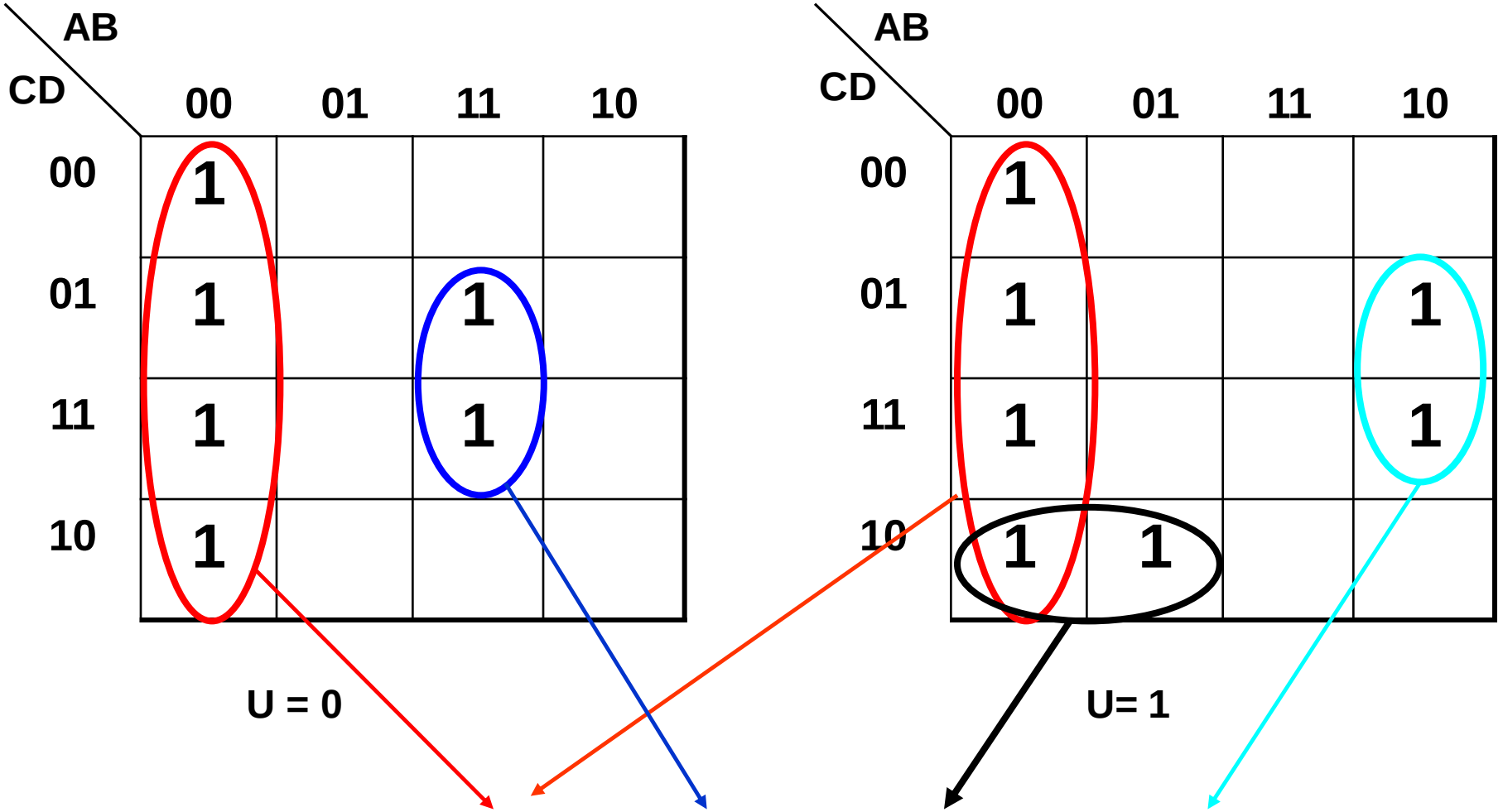
Exemple 3 : 4 variables



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

Tableaux de Karnaugh

Exemple 4 : 5 variables



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

Exercice

Trouver la forme simplifiée des fonctions à partir des deux tableaux ?

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

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

Logique multi-niveaux

On peut généraliser l'algèbre binaire à plus de 2 niveaux

a \ b	0	1	Z	X
0	0	X	0	X
1	X	1	1	X
Z	0	1	Z	X
X	X	X	X	X

0 logique

1 logique

Z déconnecté

X inconnu

Logique multi-niveaux

•Pour les cas impossibles ou interdites

il faut mettre un X dans la T.V .

•Les cas impossibles sont représentées

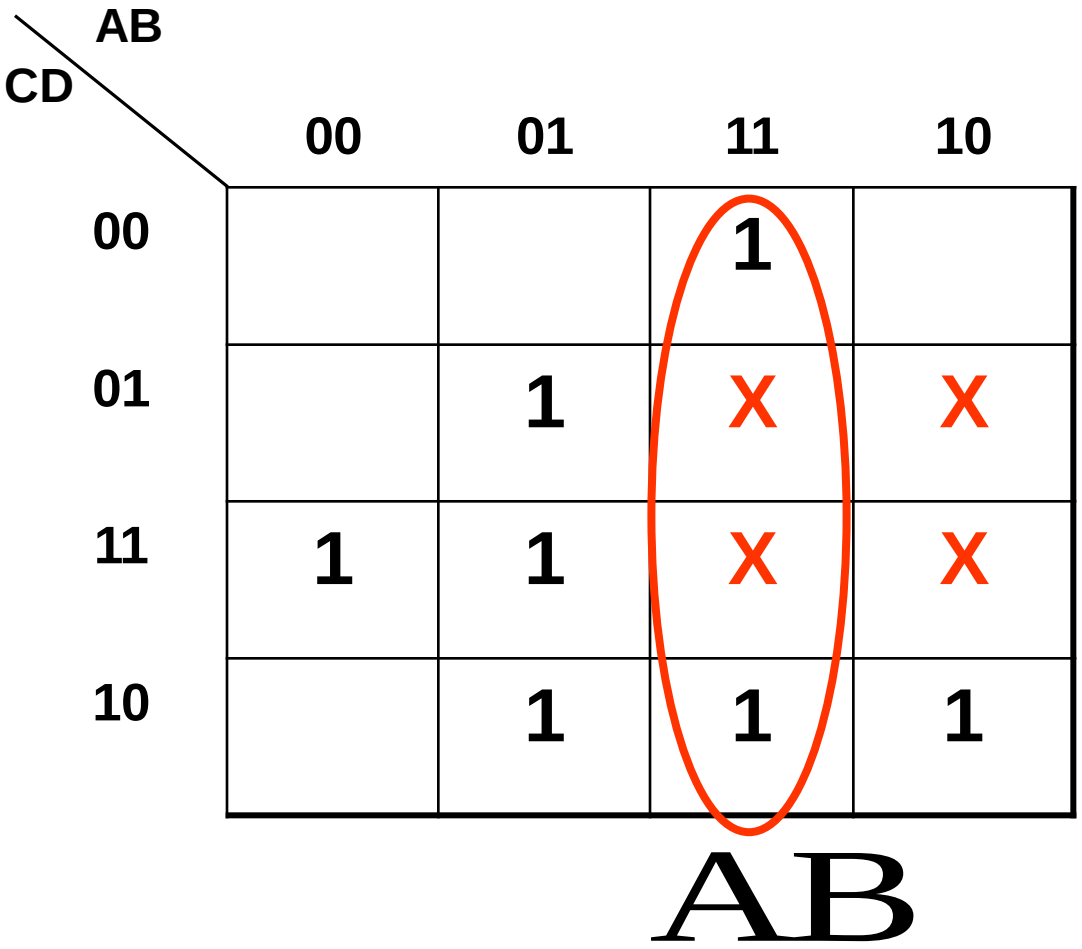
aussi par des X dans la table de karnaugh

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

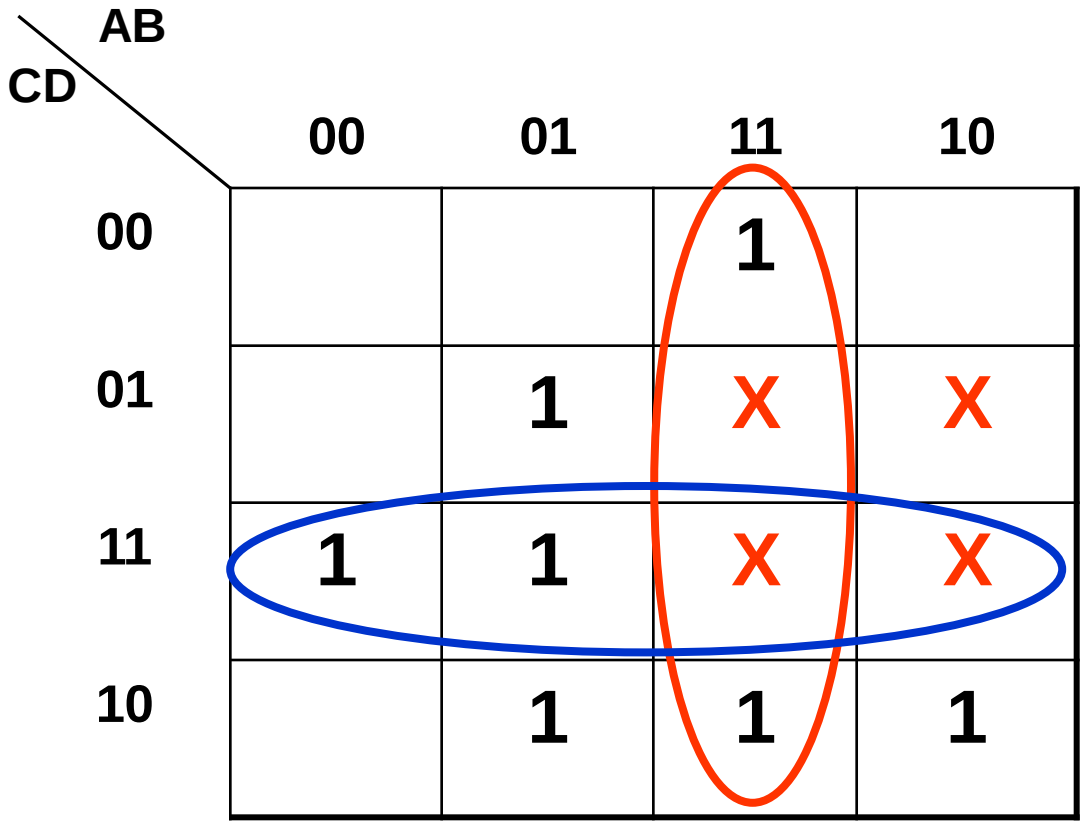
A	B	C	D		S
0	0	0	0		0
0	0	0	1		0
0	0	1	0		0
0	0	1	1		1
0	1	0	0		0
0	1	0	1		1
0	1	1	0		1
0	1	1	1		1
1	0	0	0		0
1	0	0	1		X
1	0	1	0		1
1	0	1	1		X
1	1	0	0		1
1	1	0	1		X
1	1	1	0		1
1	1	1	1		X

Tableaux de Karnaugh

- 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

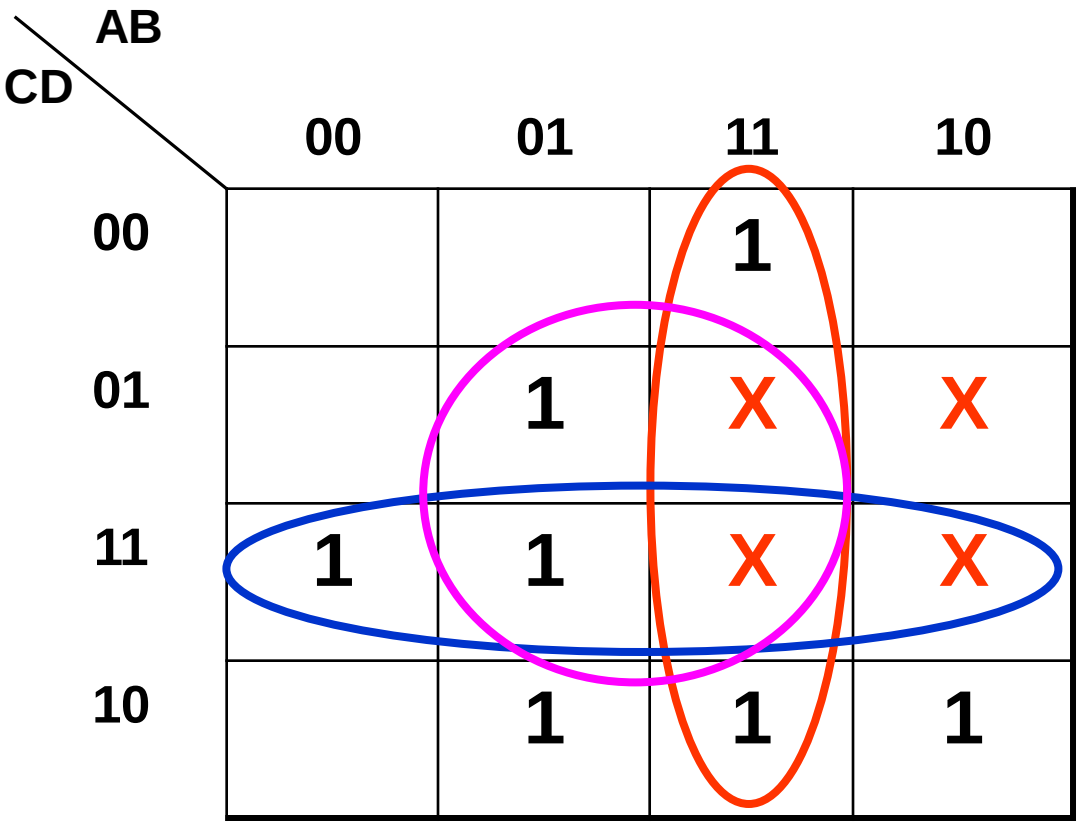


Tableaux de Karnaugh



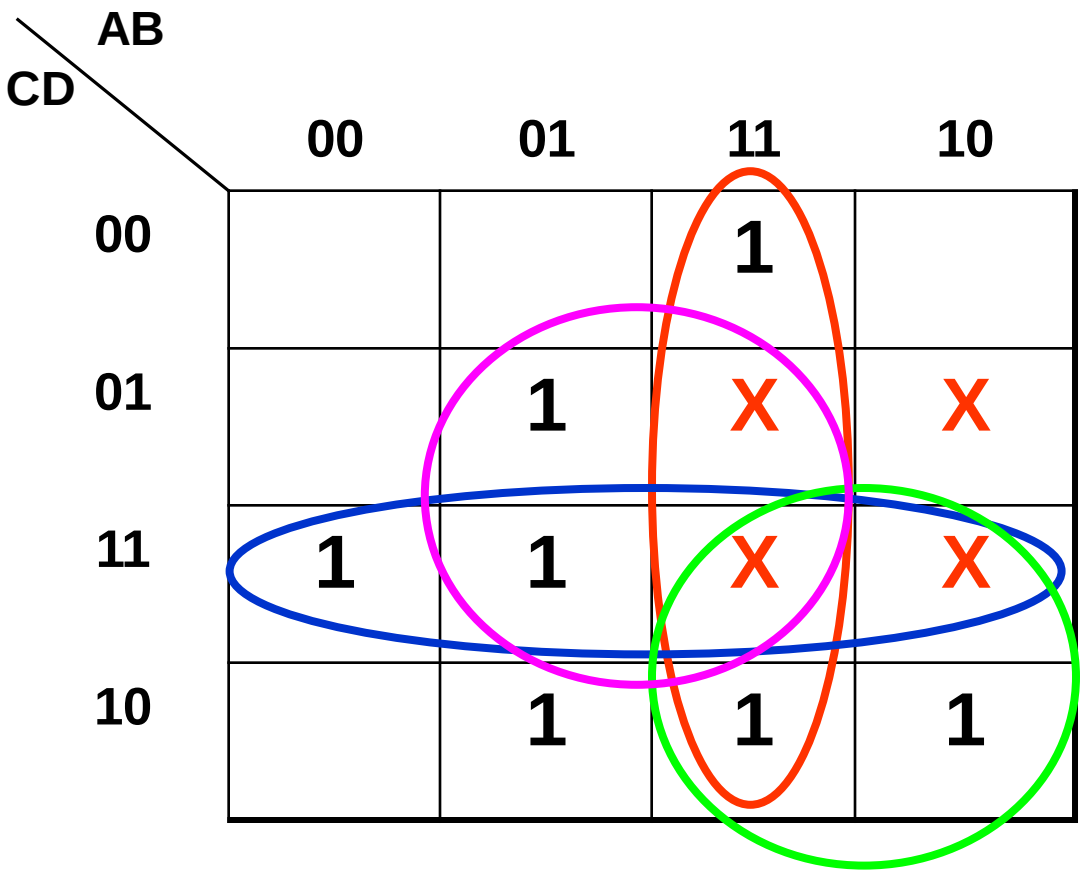
$$AB + CD$$

Tableaux de Karnaugh



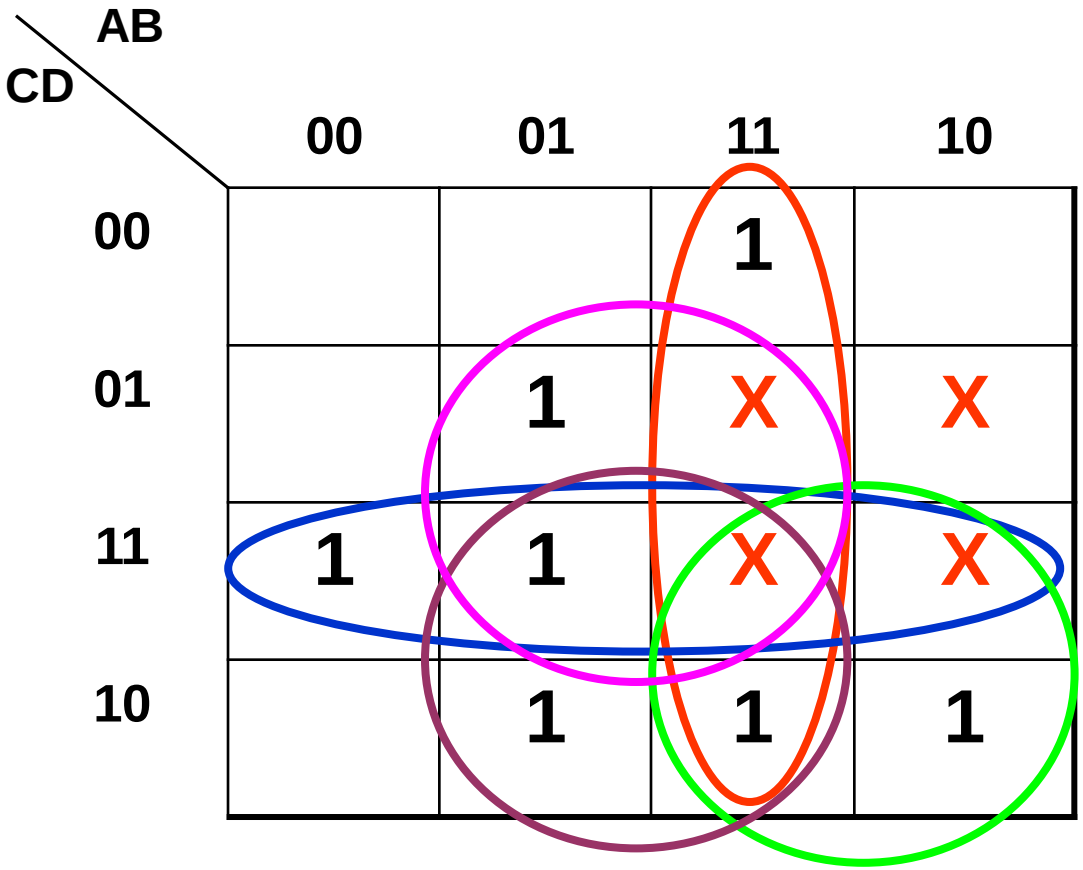
$$AB + CD + BD$$

Tableaux de Karnaugh



$$AB + CD + BD + AC$$

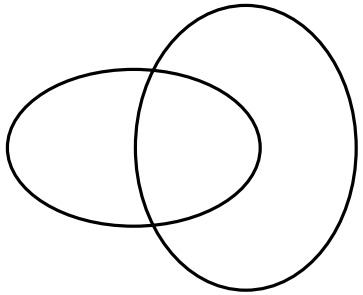
Tableaux de Karnaugh



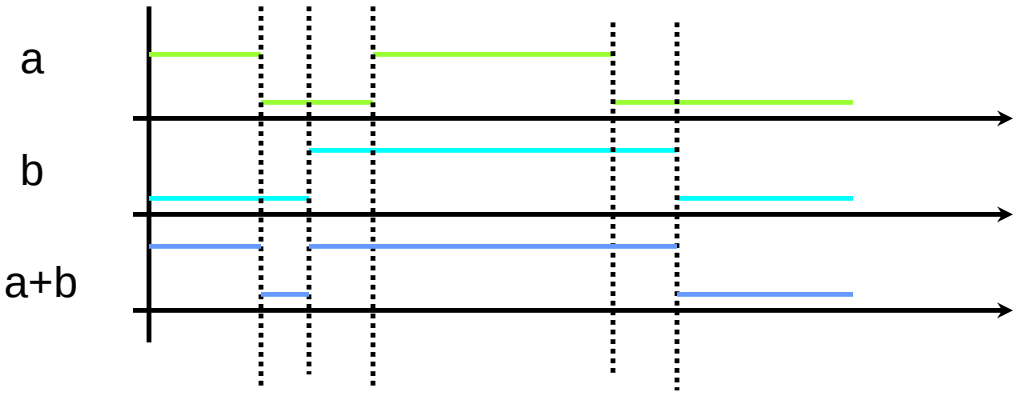
$$AB + CD + BD + AC + BC$$

Représentation des fonctions

- Diagramme de Venn ou d'Euler
  vue ensembliste



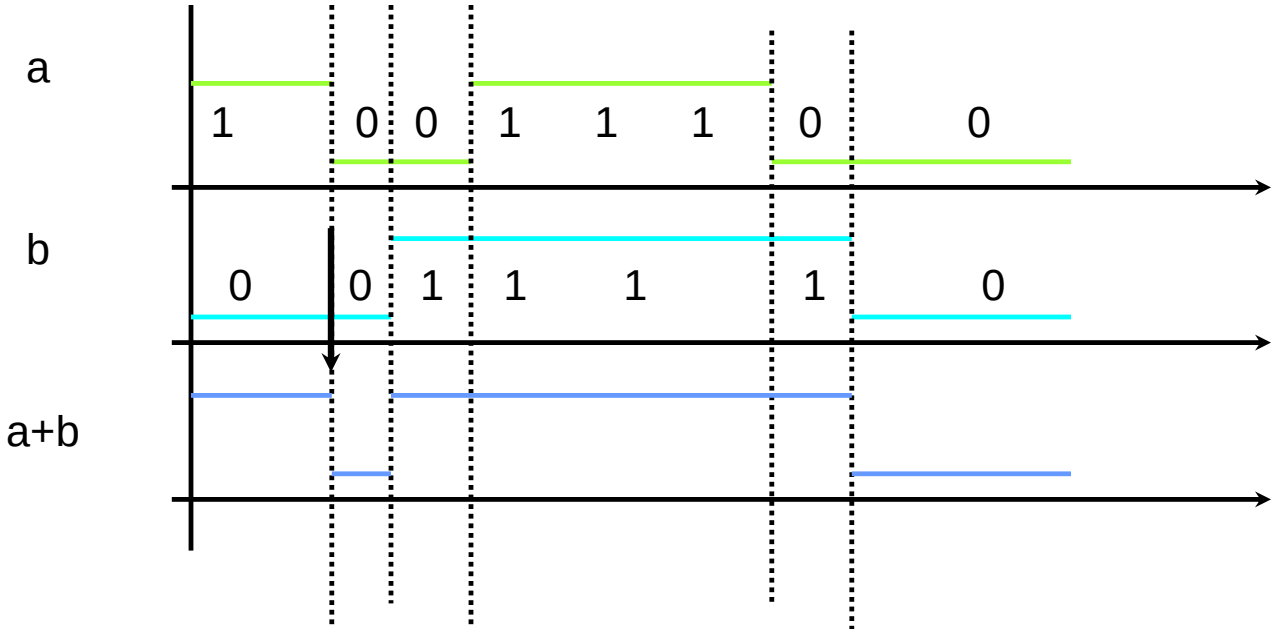
- Table de vérité
- Tableau de Karnaugh
- Équation logique ex: **f(a,b)=a+b**
- Chronogramme : Graphe d'évolution temporelle



Chronogrammes

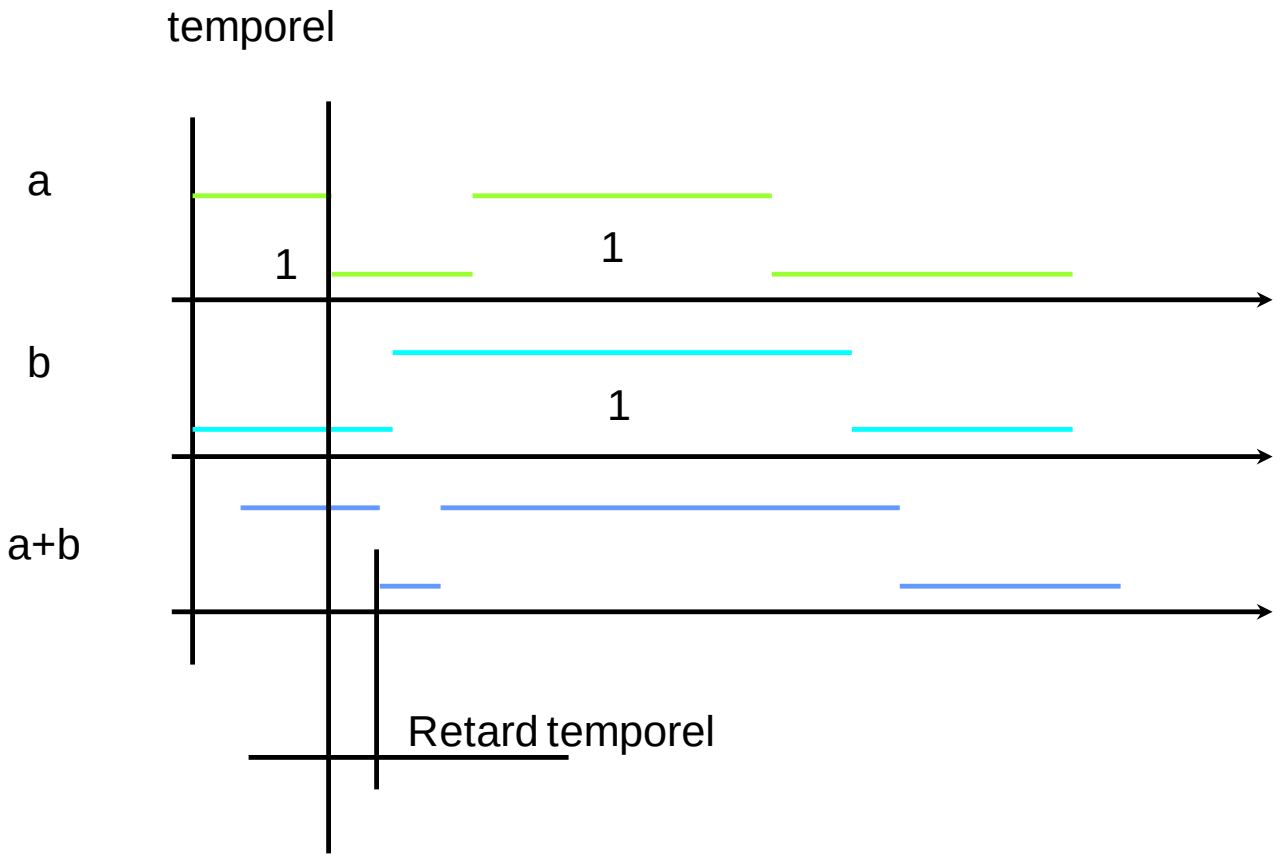
Plusieurs niveaux d'abstraction :

fonctionnel,



Chronogrammes

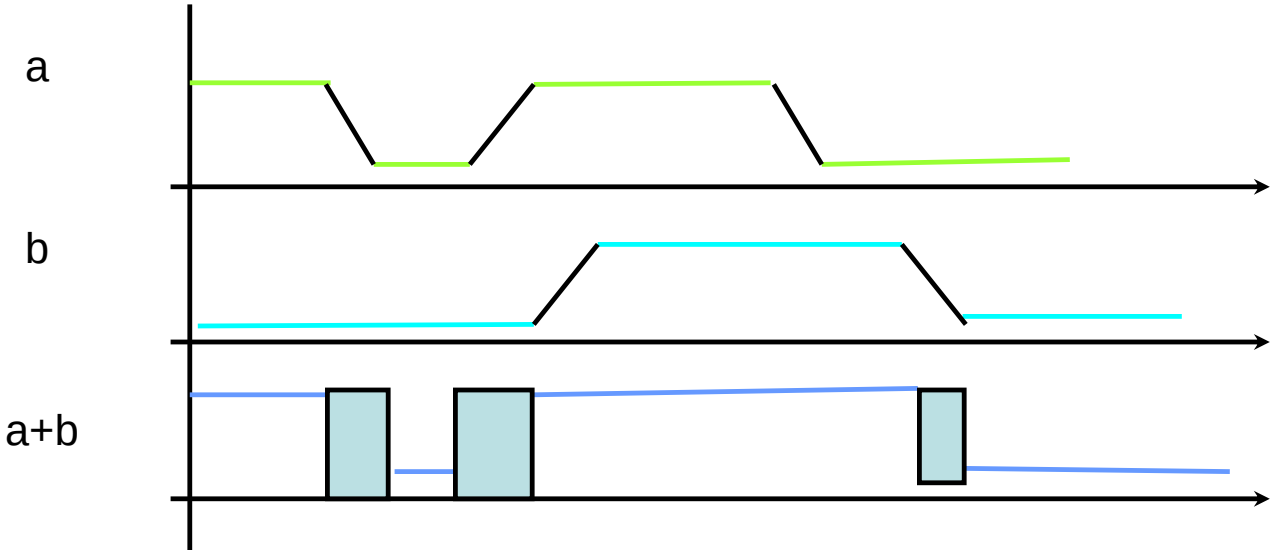
Plusieurs niveaux d'abstraction :



Chronogrammes

Plusieurs niveaux d'abstraction :

analogique symbolique



Réalisation en électronique

0/1 représentés par des tensions, courants, charges, fréquences, ...

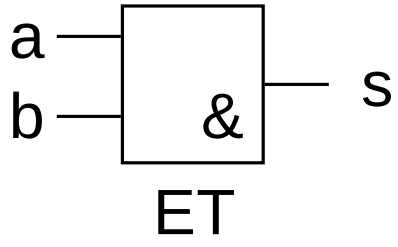
Classiquement TENSIONS : Niveau haut = H (le plus positif)
Niveau bas = L (B) (le plus négatif)

Association d'une information binaire à un niveau :

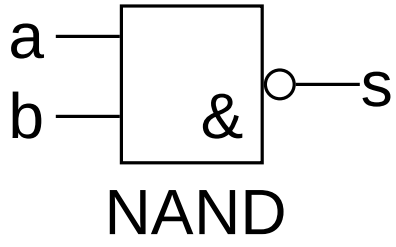
Convention positive	H ⟶ 1
(ou logique positive)	L ⟶ 0
Convention négative	H ⟶ 0
(ou logique négative)	L ⟶ 1

Représentation graphique : Norme française

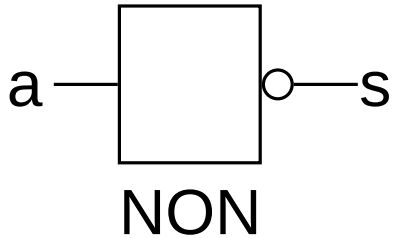
Les portes logiques:



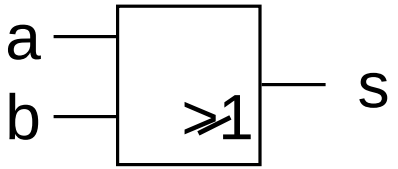
ET



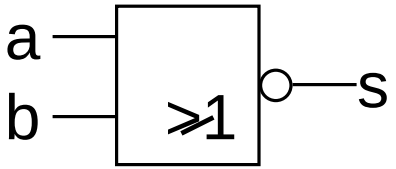
NAND



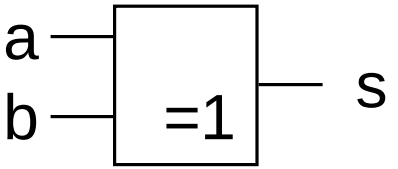
NON



OU



NOR



XOR

Représentation graphique : Norme américaine

Les portes logiques:

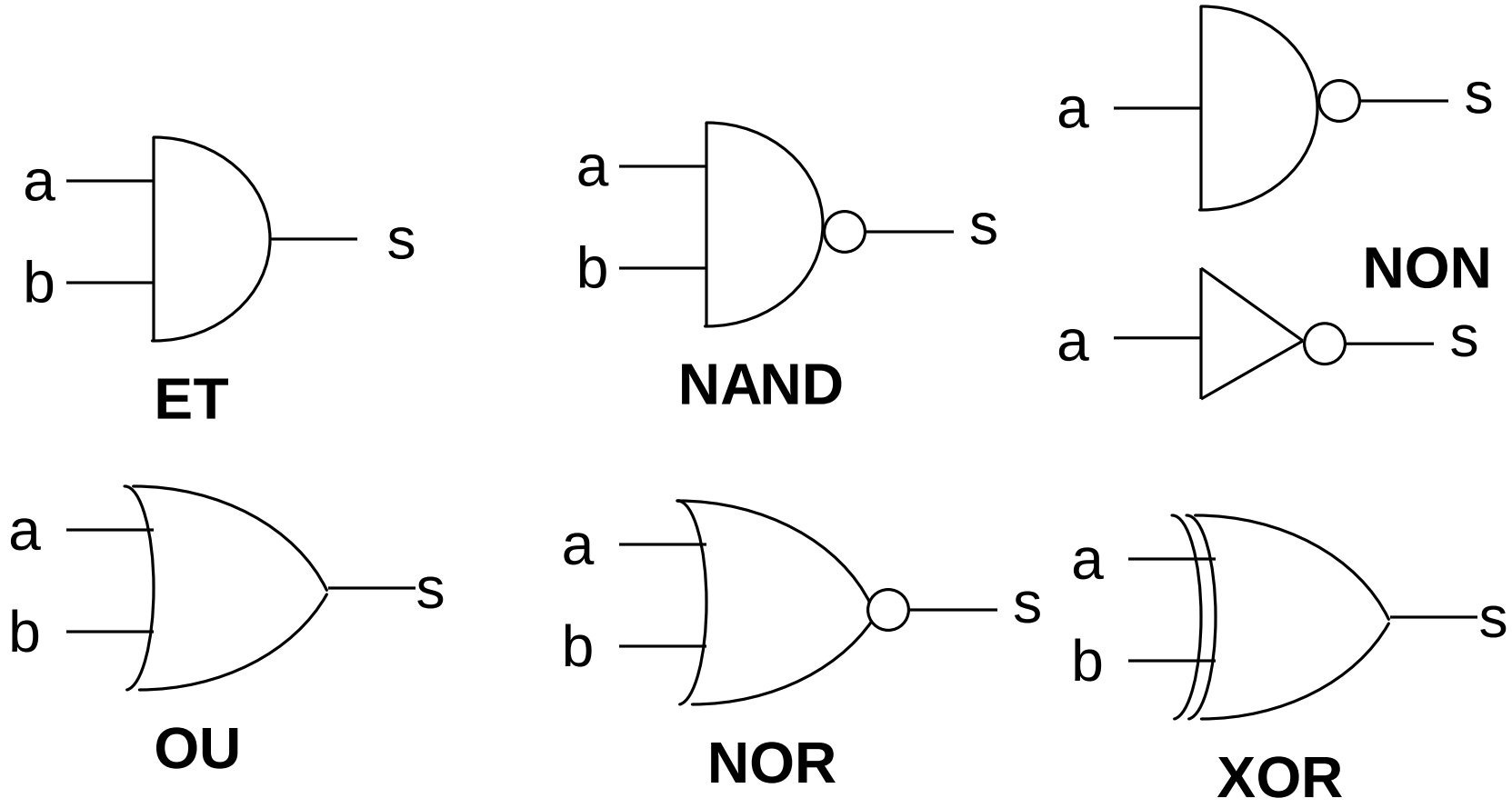


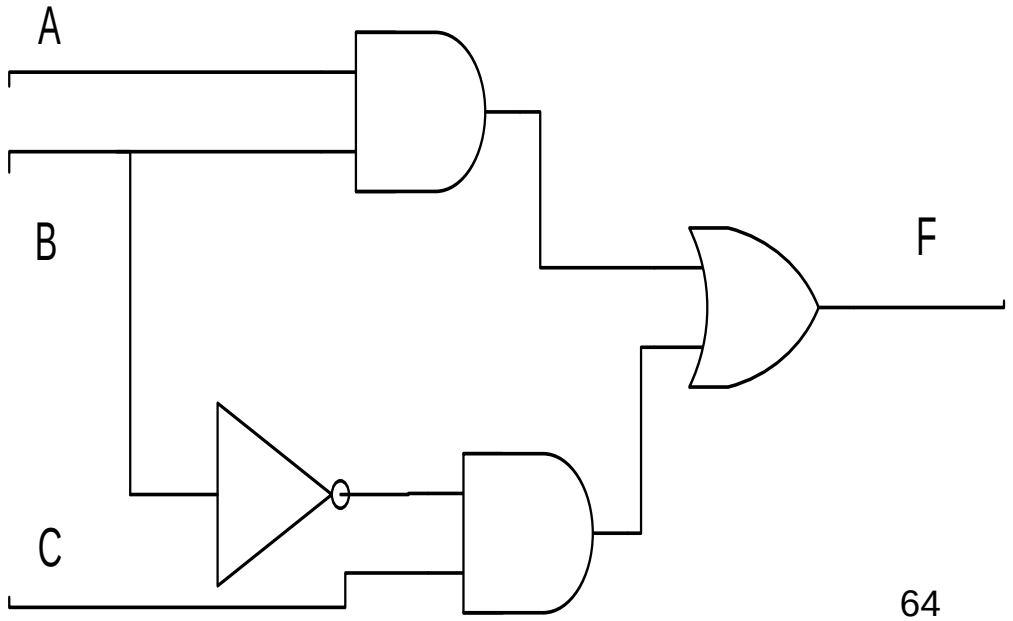
Schéma d'un circuit logique (Logigramme)

•Un logigramme est la traduction de la fonction logique en un schéma électronique.

•Le principe consiste à remplacer chaque opérateur logique par la porte logique qui lui correspond.

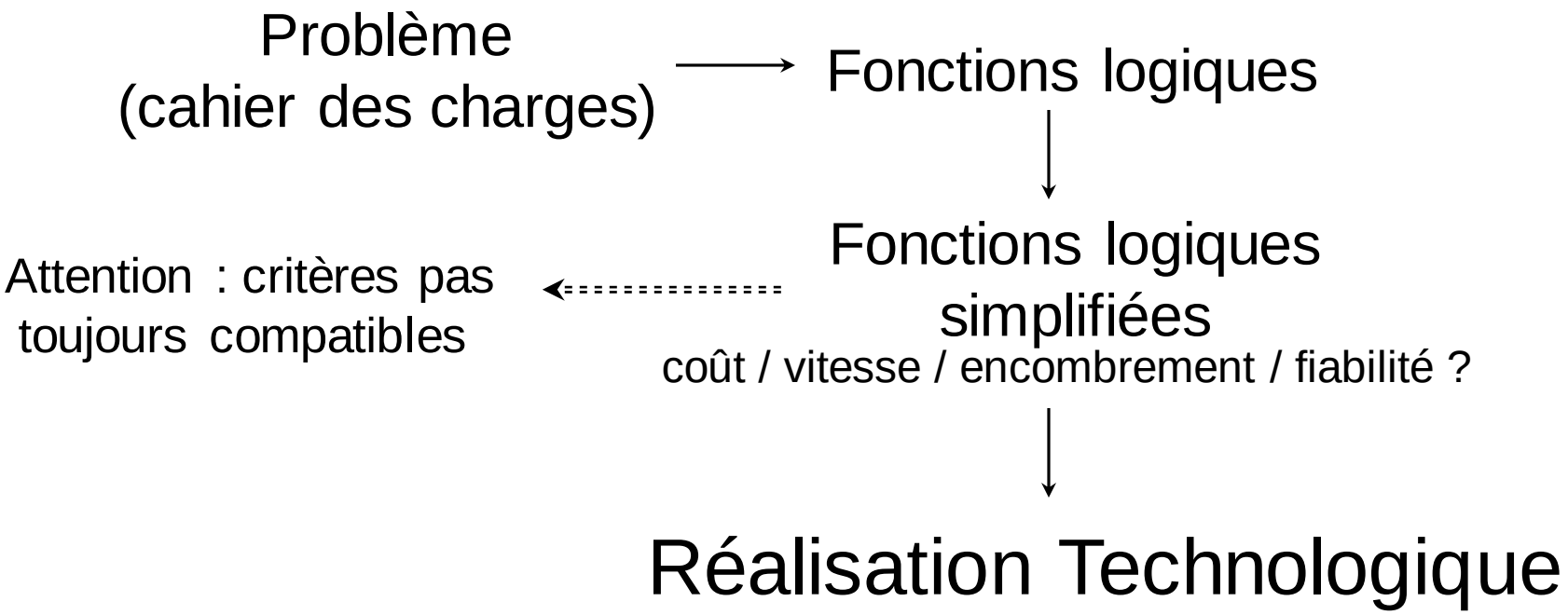
Exemple 1

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



Les circuits combinatoires

Moyens physiques de réalisation des fonctions logiques



Les circuits combinatoires

Objectifs

- Apprendre la structure de quelques circuits combinatoires souvent utilisés (multiplexeur, codeur et decodeur, demi additionneur , additionneur complet,……..).

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

Circuits combinatoires

- Un circuit combinatoire est un circuit numérique dont les sorties dépendent uniquement des entrées.
- $S_i=F(E_i)$
- $S_i=F(E_1,E_2,\ldots,E_n)$

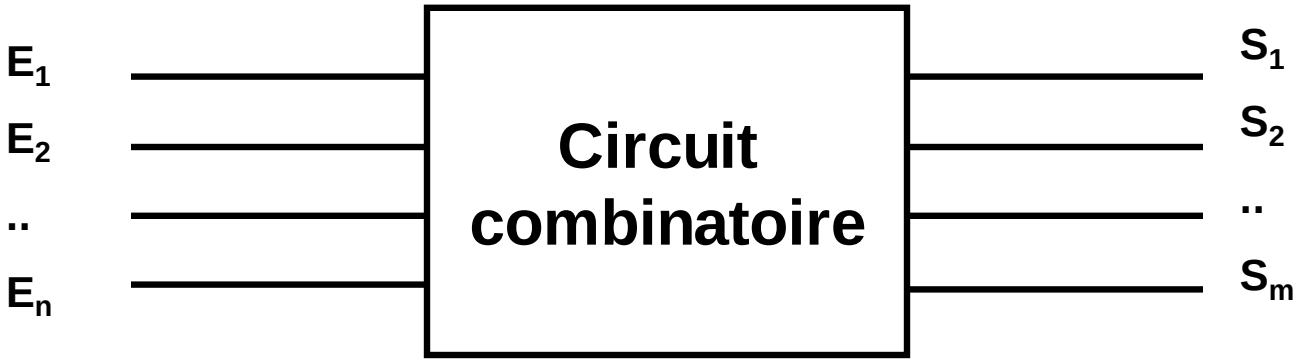


Schéma Bloc

- C'est possible d'utiliser des circuits combinatoires pour réaliser d'autres circuits plus complexes.

Composants combinatoires

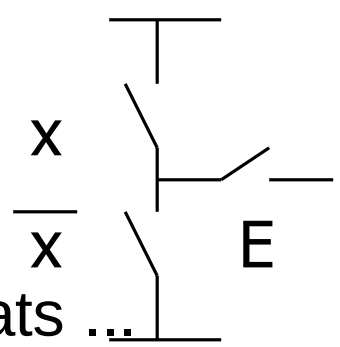
- Inverseurs
- Multiplexeur / démultiplexeur
- Codeurs / Décodeurs
- Transcodeurs
- Additionneur, comparateurs
- Unité arithmétique et logique UAL

Portes intégrées

Options technologiques : familles logiques
(TTL,CMOS, BiCMOS, ECL ..)

Entrées : classique, triggée

Sorties : collecteur (drain) ouvert, sortie 3 états ...



Remarque 1 :

10 entrées = 2^{10} fonctions possibles

⟶ Choix des meilleures fonctions

Portes intégrées

Remarque 2:

Problème du nombre de boîtiers pour réaliser une fonction logique ⟶ INTEGRATION

SSI (*small scale integration*) petite : inférieur à 12 portes

MSI (*medium*) moyenne : 12 à 99

LSI (*large*) grande : 100 à 9999

VLSI (*very large*) très grande : 10 000 à 99 999

ULSI (*ultra large*) ultra grande : 100 000 et plus

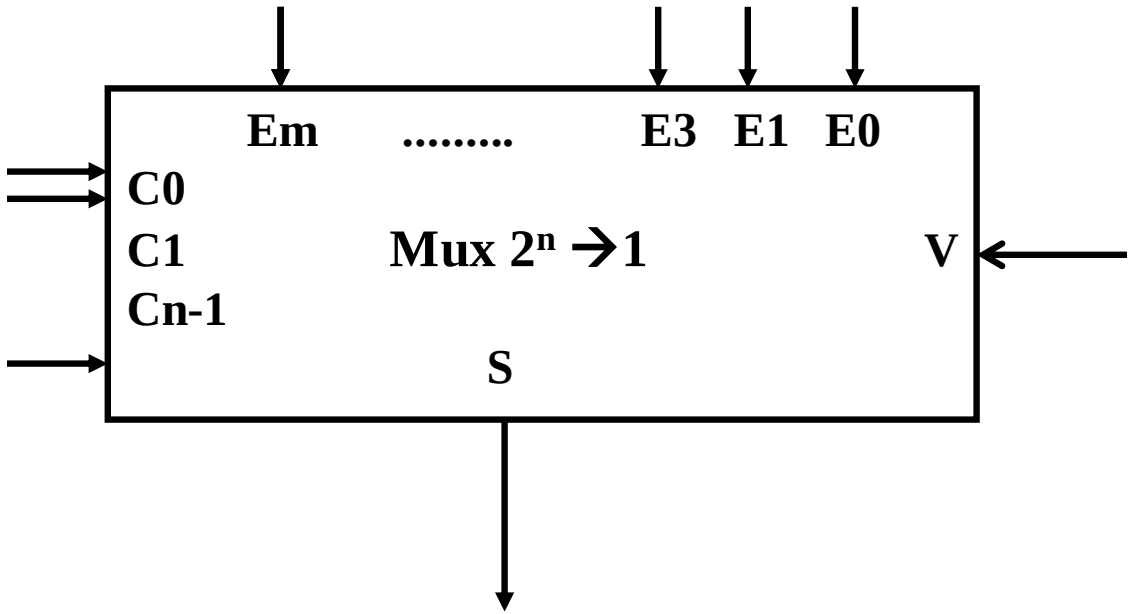
Remarque 3:

Une manière d'augmenter la puissance de traitement est de construire des CI dédiés à une application

(**ASIC** pour *Application Specific Integrated Circuit*)

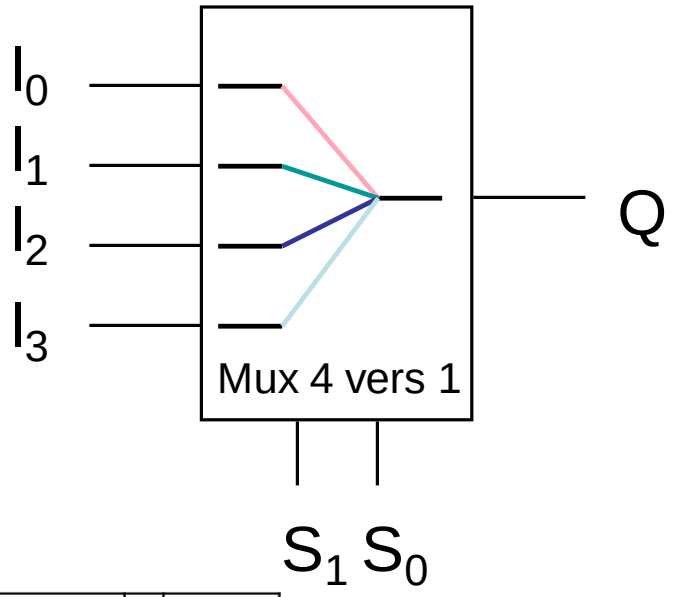
Multiplexeur

- Un multiplexeur est un circuit combinatoire qui permet de sélectionner une information (1 bit) parmi 2^n valeurs en entrée.
- Il possède :
  - 2^n entrées d'information
  - Une seule sortie
  - N entrées de sélection (commandes)



Multiplexeur 4 → 1

Sélection d'une voie parmi 2^N par N bits de commande



Si $(S_1S_0)_2 = (0)_{10}$ alors $Q = I_0$

$$Q = \overline{S_1}.\overline{S_0}.I_O$$

Si $(S_1S_0)_2 = (1)_{10}$ alors $Q = I_1$

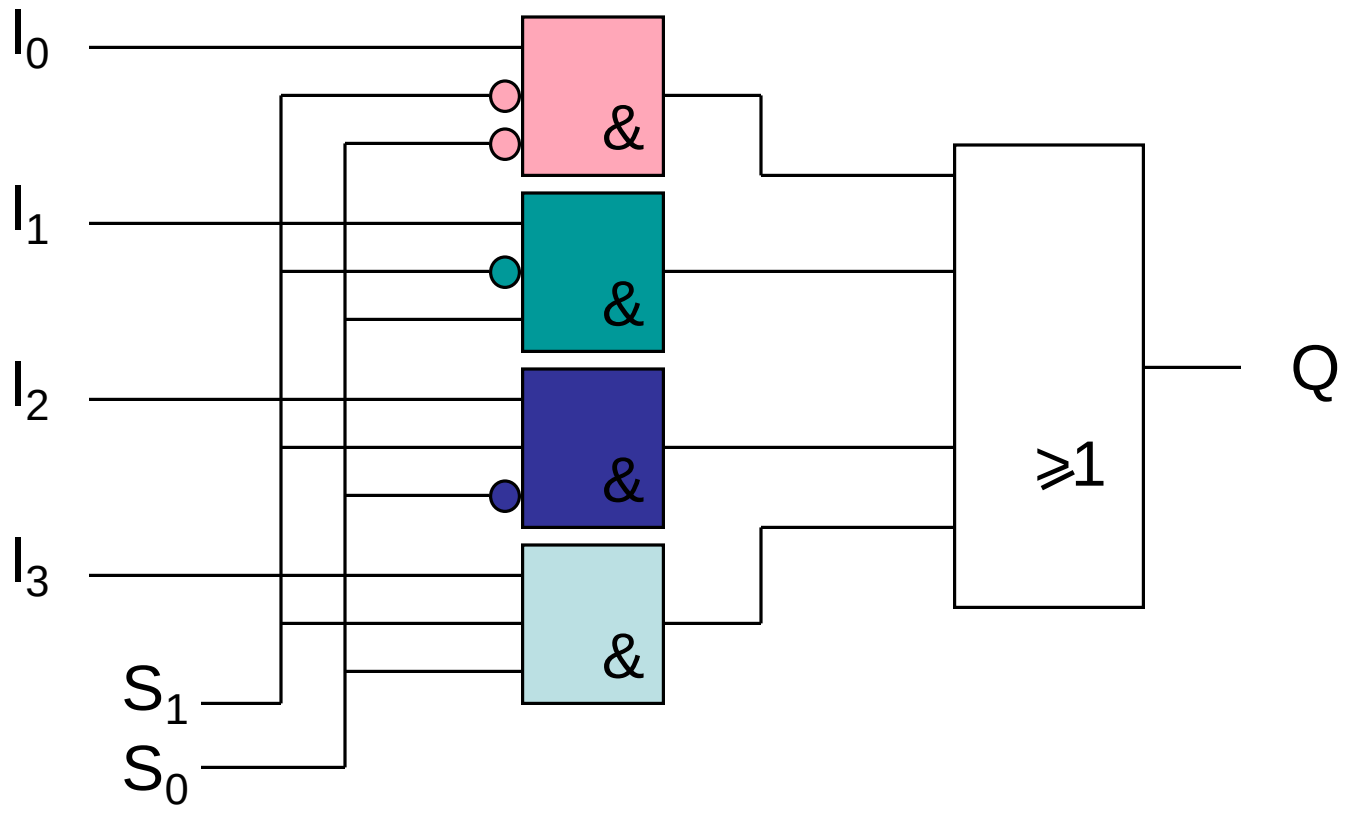
$$Q = \overline{S_1}.S_0.I_1$$

S1	S0		Q
0	0		I_0
0	1		I_1
1	0		I_2
1	1		I_3

$$Q = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

Multiplexeur (logigramme)

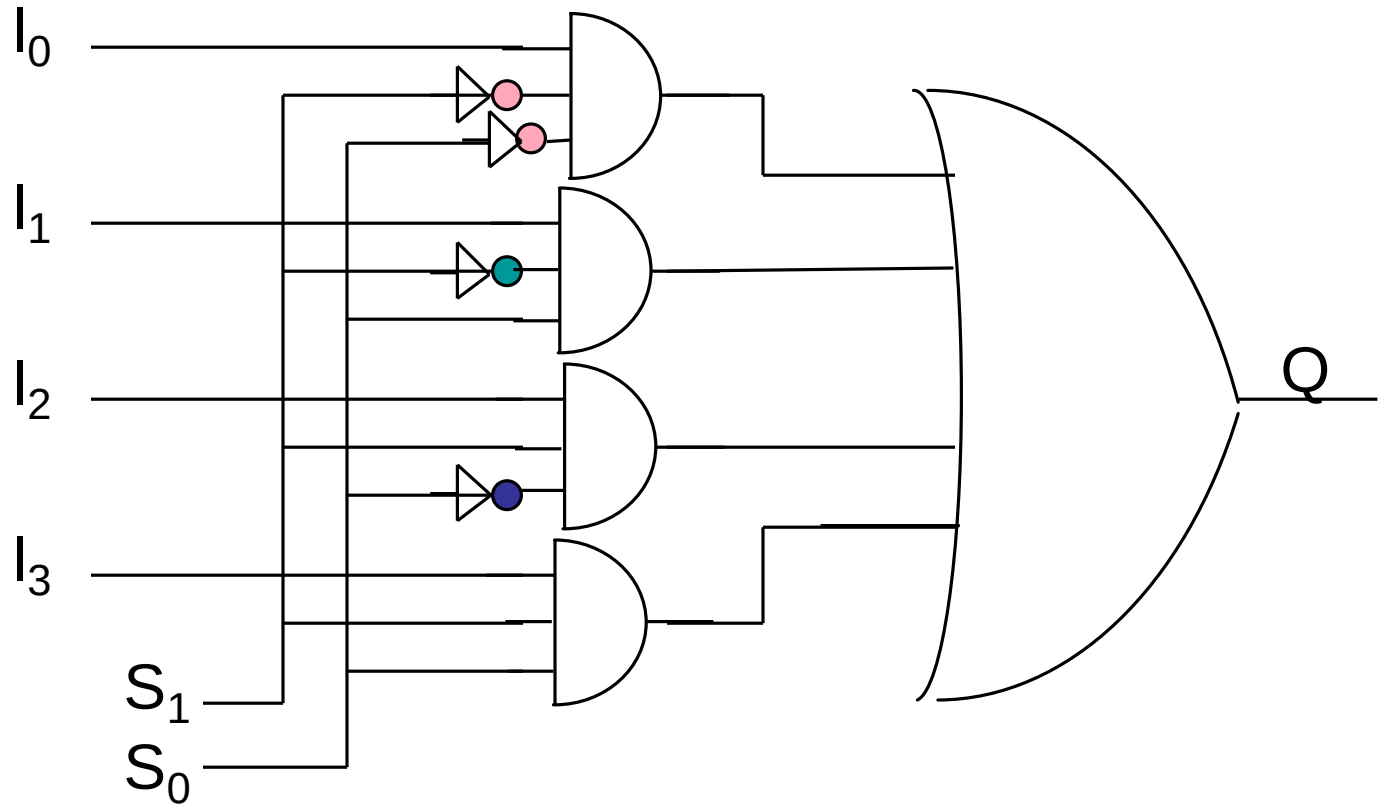
$$Q = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$



Applications : La conversion parallèle / série d'informations

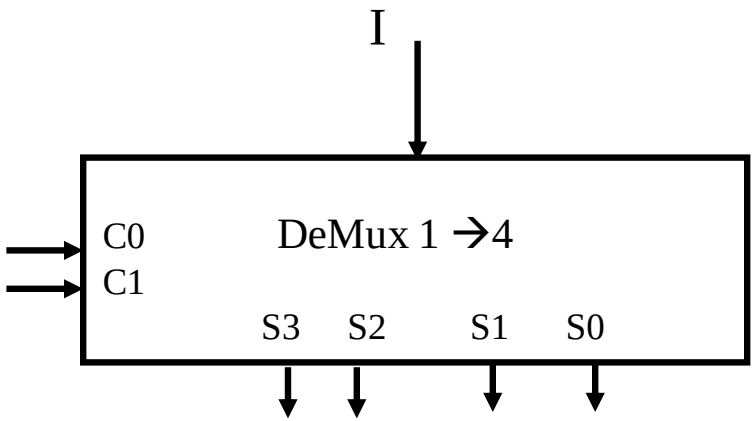
Multiplexeur (logigramme)

$$Q = \overline{S_1}.\overline{S_0}.I_0 + \overline{S_1}.S_0.I_1 + S_1.\overline{S_0}.I_2 + S_1.S_0.I_3$$

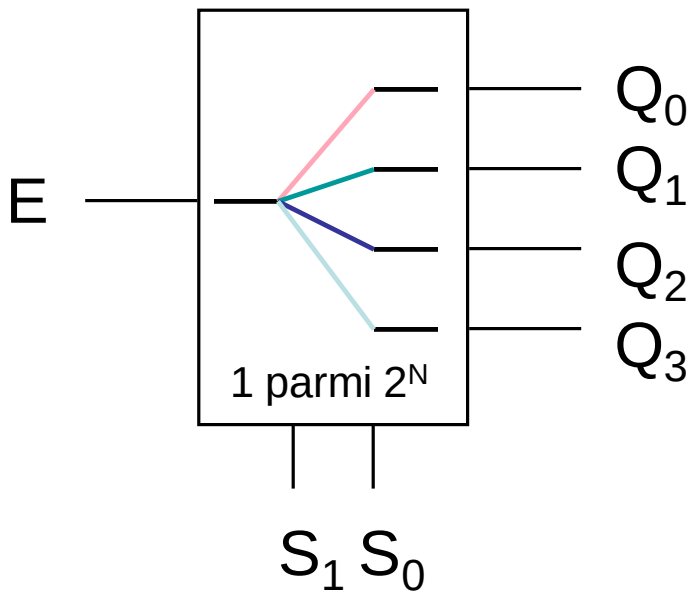


Démultiplexeur

- Il joue le rôle inverse d'un multiplexeurs, il permet de faire passer une information dans l'une des sorties selon les valeurs des entrées de commandes.
- Il possède :
  - une seule entrée
  - 2^n sorties
  - N entrées de sélection (commandes)



Démultiplexeur : 1 parmi 2^n

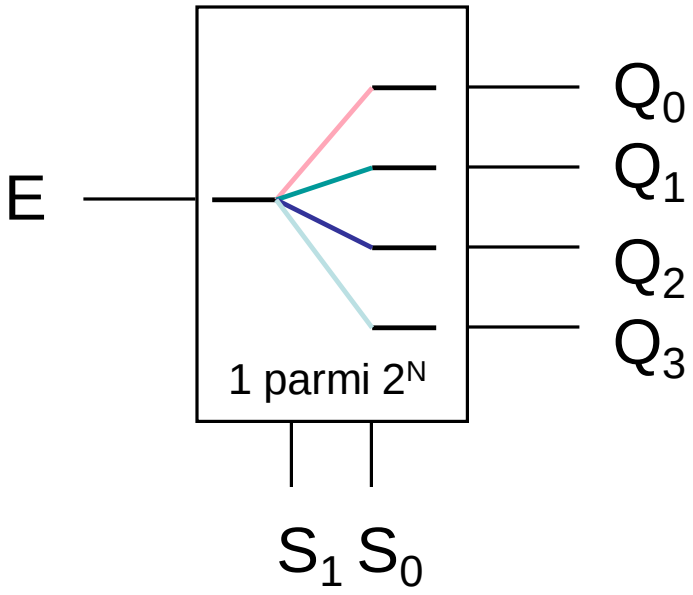


$Q_0 = E$ si $(S_1S_0)_2=0$
0 sinon

$Q_1 = E$ si $(S_1S_0)_2=1$
0 sinon

Remarque : E peut ne pas être «disponible»
Sortie sélectionnée = 1 les autres 0
ou Sortie sélectionnée = 0 les autres 1

Démultiplexeur : 1 →4



$$Q0 = \overline{S1}.\overline{S0}.(E)$$

$$Q1 = \overline{S1}.S0.(E)$$

$$Q2 = S1.\overline{S0}.(E)$$

$$Q3 = S1.S0.(E)$$

S_1	S_0	Q3	Q2	Q1	Q0
0	0	0	0	0	E
0	1	0	0	E	0
1	0	0	E	0	0
1	1	E	0	0	0

Table de vérité

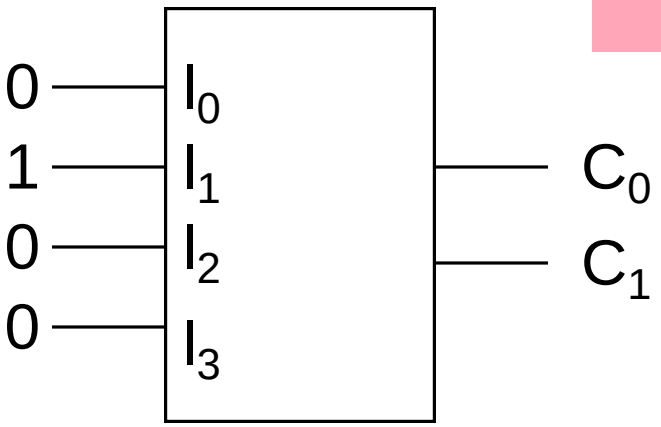
Codeur (ou Encodeur)

Faire correspondre un mot code à un symbole

2^N entrées

N sorties

1 entrée parmi 2^N



Le code de l'entrée

Mot

Code

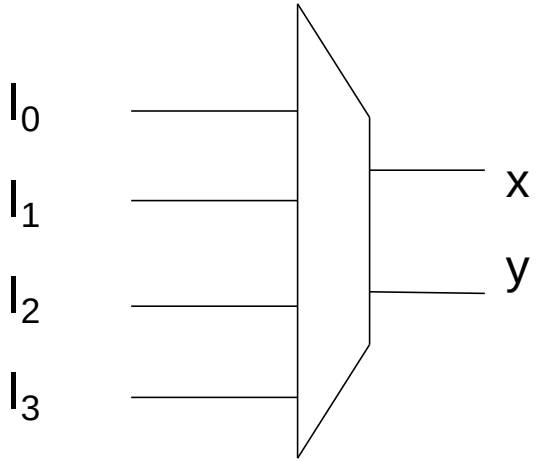
Traduit le rang de l'entrée active en un code binaire

Exemple : Clavier / Scan code
Caractère / Code ASCII

L'encodeur binaire (4→2)

Table de vérité

I_0	I_1	I_2	I_3		x	y
0	0	0	0		0	0
1	x	x	x		0	0
0	1	x	x		0	1
0	0	1	x		1	0
0	0	0	1		1	1



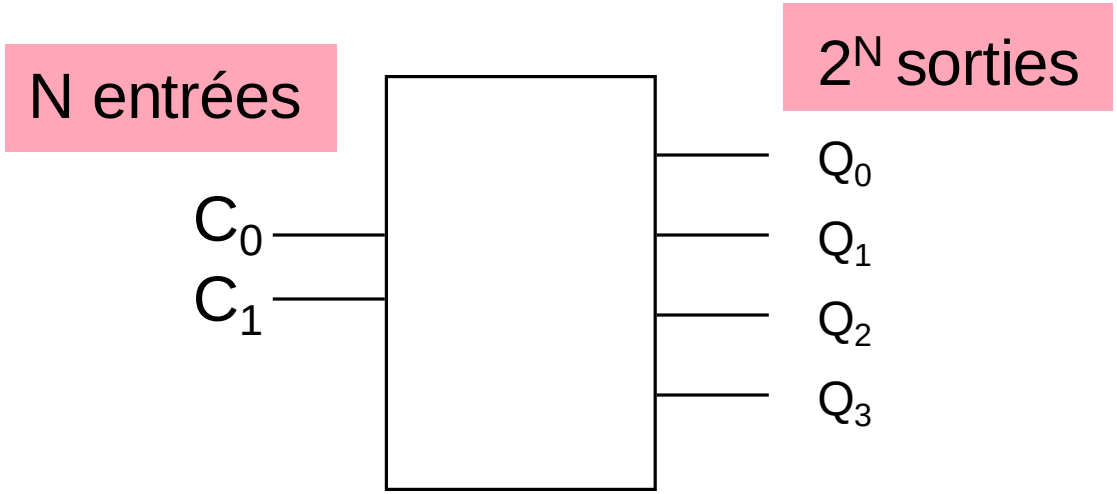
Equations

$$X = \overline{I0}.\overline{I1}.(I2 + I3)$$

$$Y = \overline{I0}.(I1 + .\overline{I2}.I3)$$

Le décodeur binaire

- C'est un circuit combinatoire qui est constitué de :
  - N : entrées de données
  - 2^n sorties
  - Pour chaque combinaison en entrée une seule sortie est active à la fois

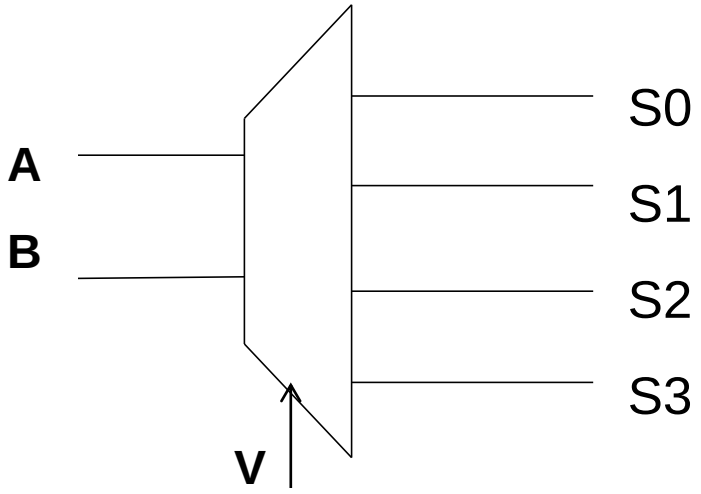


Active la ligne de sortie correspondant au code binaire présent en entrée

Décodeur 2→4

Table de vérité

V	A	B		S0	S1	S2	S3
0	X	X		0	0	0	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	0	1



$$S_0 = (\overline{A}.\overline{B}).V$$

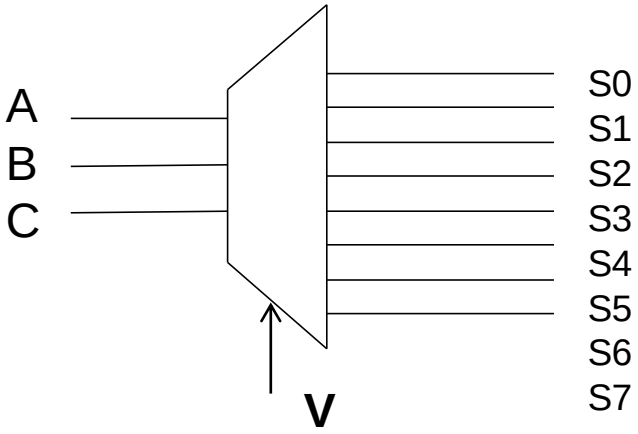
$$S_1 = (\overline{A}.B).V$$

$$S_2 = (A.\overline{B}).V$$

$$S_3 = (A.B).V$$

Décodeur 3→8

A	B	C		S0	S1	S2	S3	S4	S5	S6	S7
0	0	0		1	0	0	0	0	0	0	0
0	0	1		0	1	0	0	0	0	0	0
0	1	0		0	0	1	0	0	0	0	0
0	1	1		0	0	0	1	0	0	0	0
1	0	0		0	0	0	0	1	0	0	0
1	0	1		0	0	0	0	0	1	0	0
1	1	0		0	0	0	0	0	0	1	0
1	1	1		0	0	0	0	0	0	0	1



$$S_0 = \overline{A}.\overline{B}.\overline{C}$$
$$S_1 = \overline{A}.\overline{B}.C$$
$$S_2 = \overline{A}.B.\overline{C}$$
$$S_3 = \overline{A}.B.C$$
$$S_4 = A.\overline{B}.\overline{C}$$
$$S_5 = A.\overline{B}.C$$
$$S_6 = A.B.\overline{C}$$
$$S_7 = A.B.C$$

Remarque :

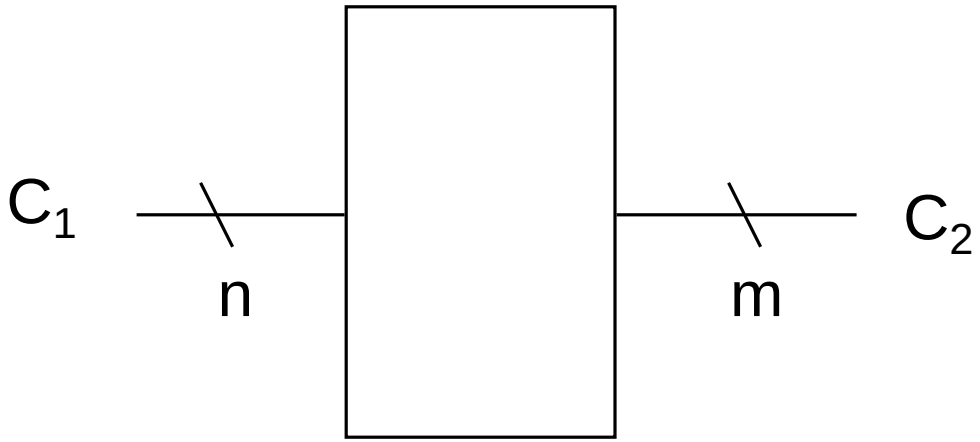
Multiplexeur ⟺ Démultiplexeur

Codeur ⟺ Décodeur

Transcodeur

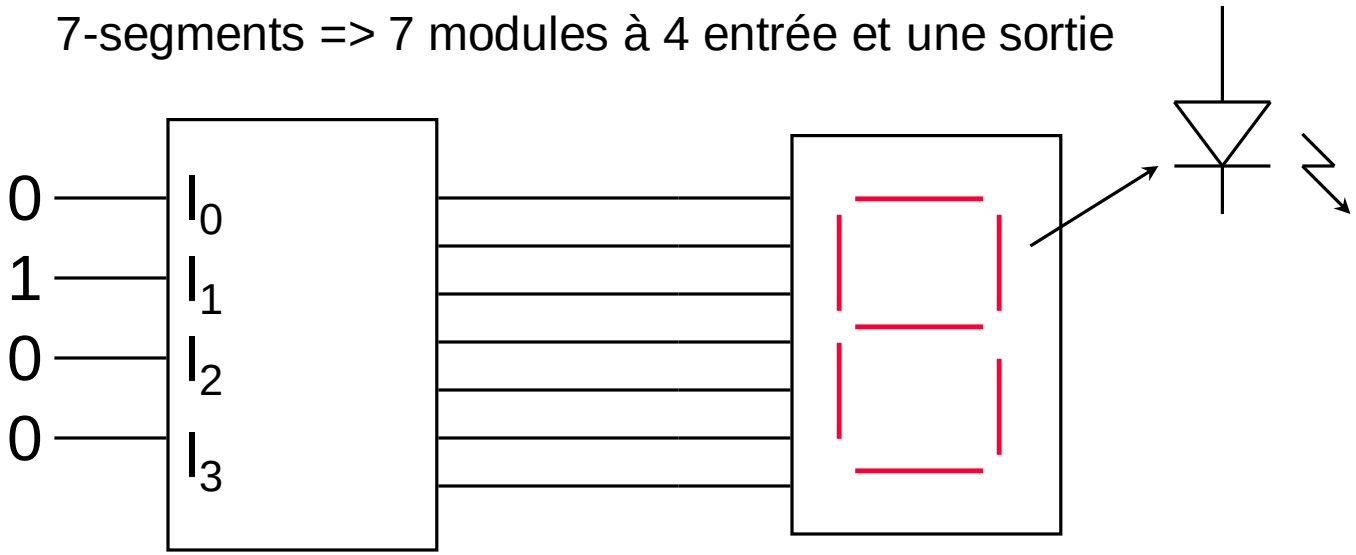
C’est un circuit combinatoire qui permet de transformer un code X (sur n bits) en entrée en un code Y (sur m bits) en sortie.

Passage d’un code C_1 à un code C_2



Transcodeur : exemple

7-segments => 7 modules à 4 entrée et une sortie



Code binaire 0 à 9

Configuration alimentation des diodes (ou LCD)

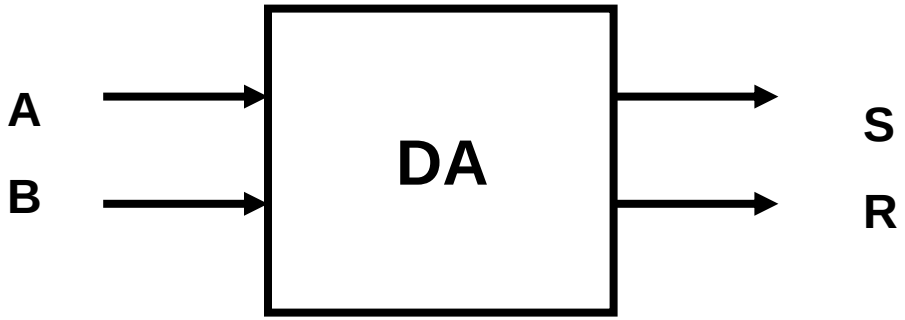
Exemples de code :

Binaire, binaire réfléchi, 7-segments, BCD, …

Additionneur

- Le demi additionneur est un circuit combinatoire qui permet de réaliser la somme arithmétique de deux nombres A et B chacun sur un bit.

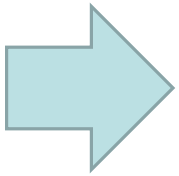
- A la sotie on va avoir la somme S et la retenu R (Carry).



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

Demi Additionneur

- 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

De la table de vérité on trouve :

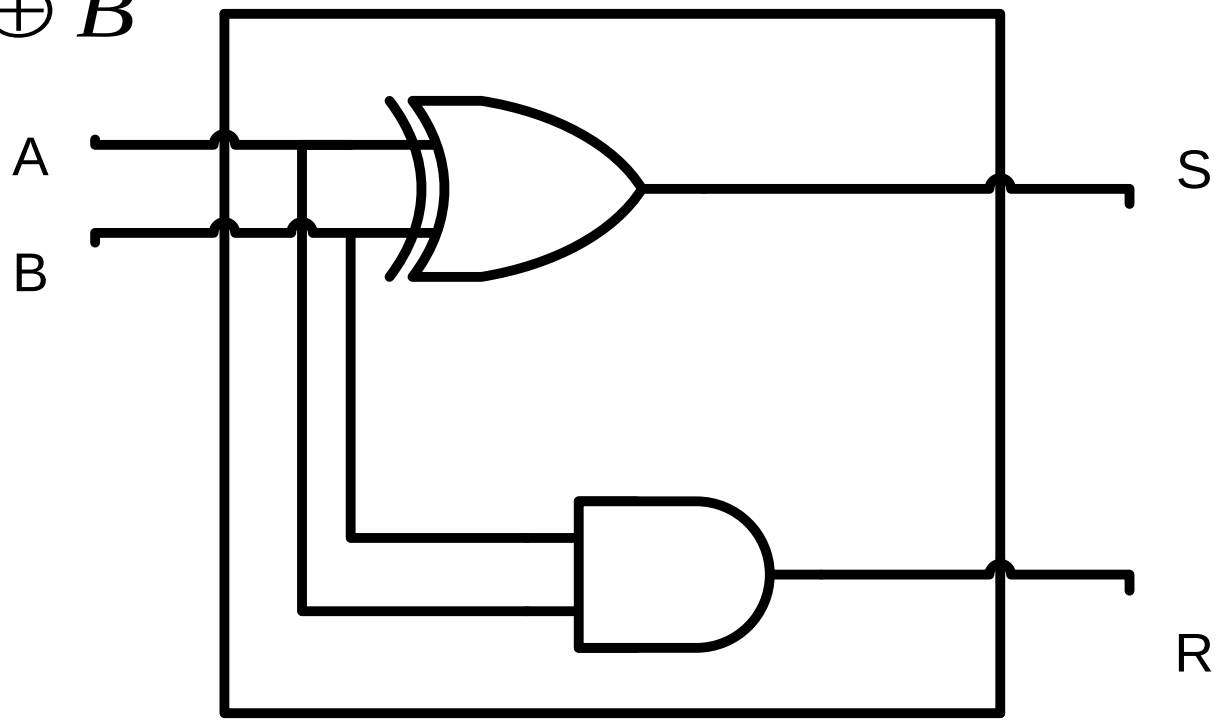
$$R = A.B$$

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

Demi Additionneur

$$R = A.B$$

$$S = A \oplus B$$



Logigramme Demi-Additionneur

Additionneur complet

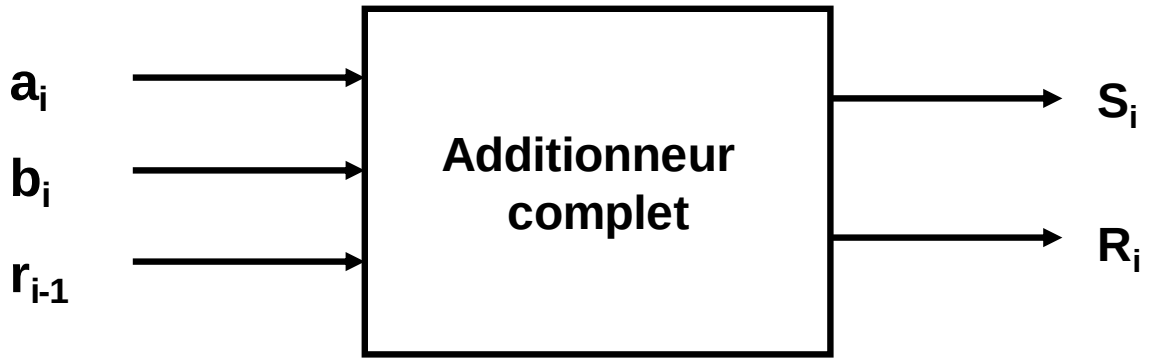
- Lorsque on fait une addition (binaire) il faut tenir en compte de la retenue entrante.

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
r_4	s_4	s_3	s_2	s_1

	r_{i-1}	
	a_i	
+	b_i	
r_i	s_i	

Additionneur complet 1 bit

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



Additionneur complet 1 bit

Table de vérité d'un additionneur complet sur 1 bit

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

Equations

$$S_i = \overline{A}_i.\overline{B}_i.R_{i-1} + \overline{A}_i.B_i.\overline{R}_{i-1} + A_i.\overline{B}_i.\overline{R}_{i-1} + A_i.B_i.R_{i-1}$$

$$R_i = \overline{A}_i B_i R_{i-1} + A_i \overline{B}_i R_{i-1} + A_i B_i \overline{R}_{i-1} + A_i B_i R_{i-1}$$

Additionneur complet 1 bit

Si on veut simplifier les équations on obtient :

$$S_i = \overline{A_i}.\overline{B_i}.R_{i-1} + \overline{A_i}.B_i.\overline{R_{i-1}} + A_i.\overline{B_i}.\overline{R_{i-1}} + A_i.B_i.R_{i-1}$$

$$S_i = \overline{A_i}.(\overline{B_i}.R_{i-1} + B_i.\overline{R_{i-1}}) + A_i.(\overline{B_i}.\overline{R_{i-1}} + B_i.R_{i-1})$$

$$S_i = \overline{A_i}(B_i \oplus R_{i-1}) + A_i.(\overline{B_i \oplus R_{i-1}})$$

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

$$R_i = \overline{A_i}B_iR_{i-1} + A_i\overline{B_i}R_{i-1} + A_iB_i\overline{R_{i-1}} + A_iB_iR_{i-1}$$

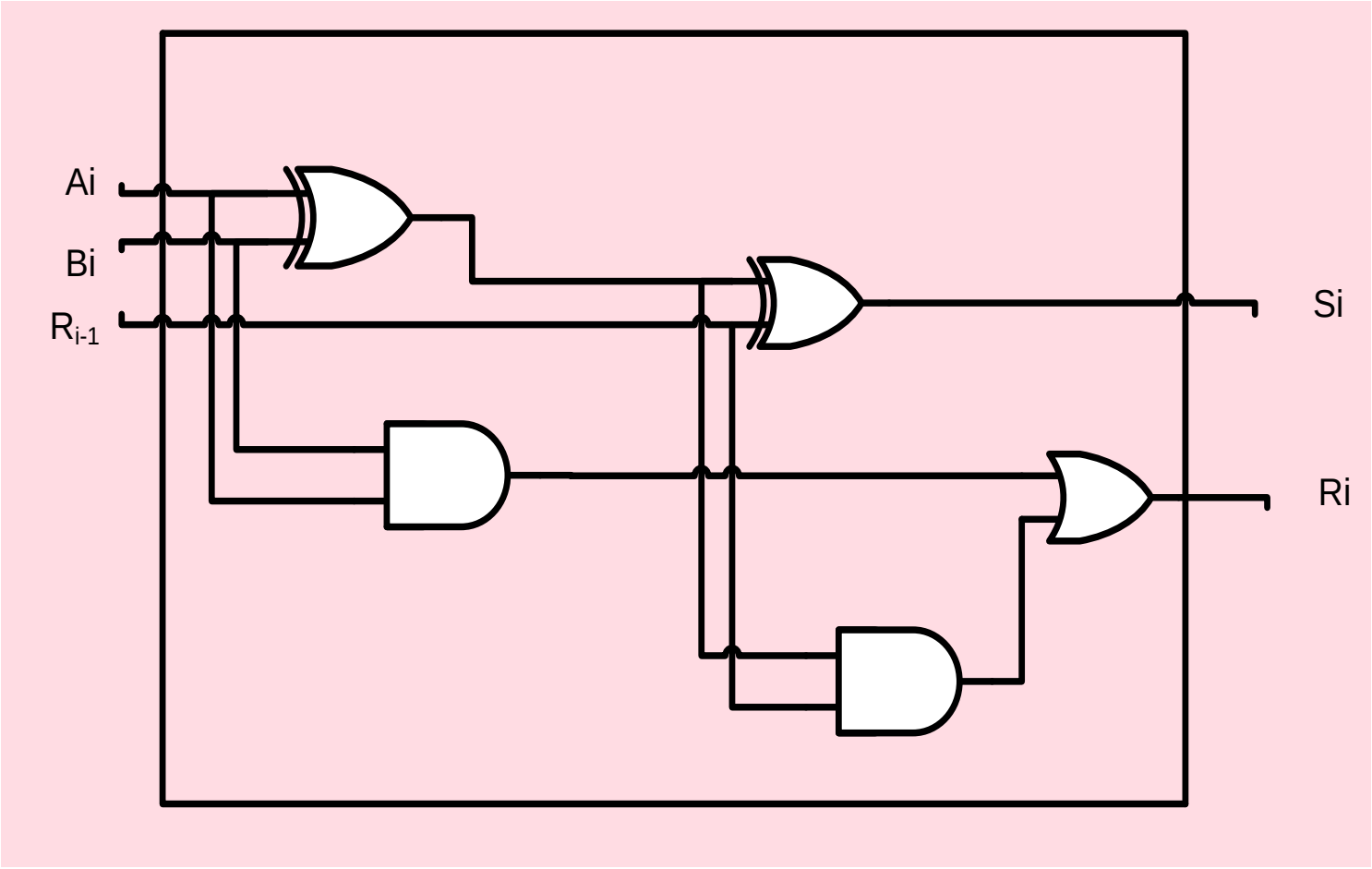
$$R_i = R_{i-1}.(\overline{A_i}.B_i + A_i.\overline{B_i}) + A_iB_i(\overline{R_{i-1}} +_i R_{i-1})$$

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

Schéma d'un additionneur complet

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

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



Additionneur sur 4 bits

- Un additionneur sur 4 bits est un circuit qui permet de faire l'addition de deux nombres A et B de 4 bits chacun
  - $A(a_3a_2a_1a_0)$
  - $B(b_3b_2b_1b_0)$

  ⇨ En plus il prend en compte de la retenu entrante

- En sortie on va avoir le résultat sur **4 bits ainsi que la retenu (5 bits en sortie)**

- Donc au total le circuit possède 9 entrées et 5 sorties.

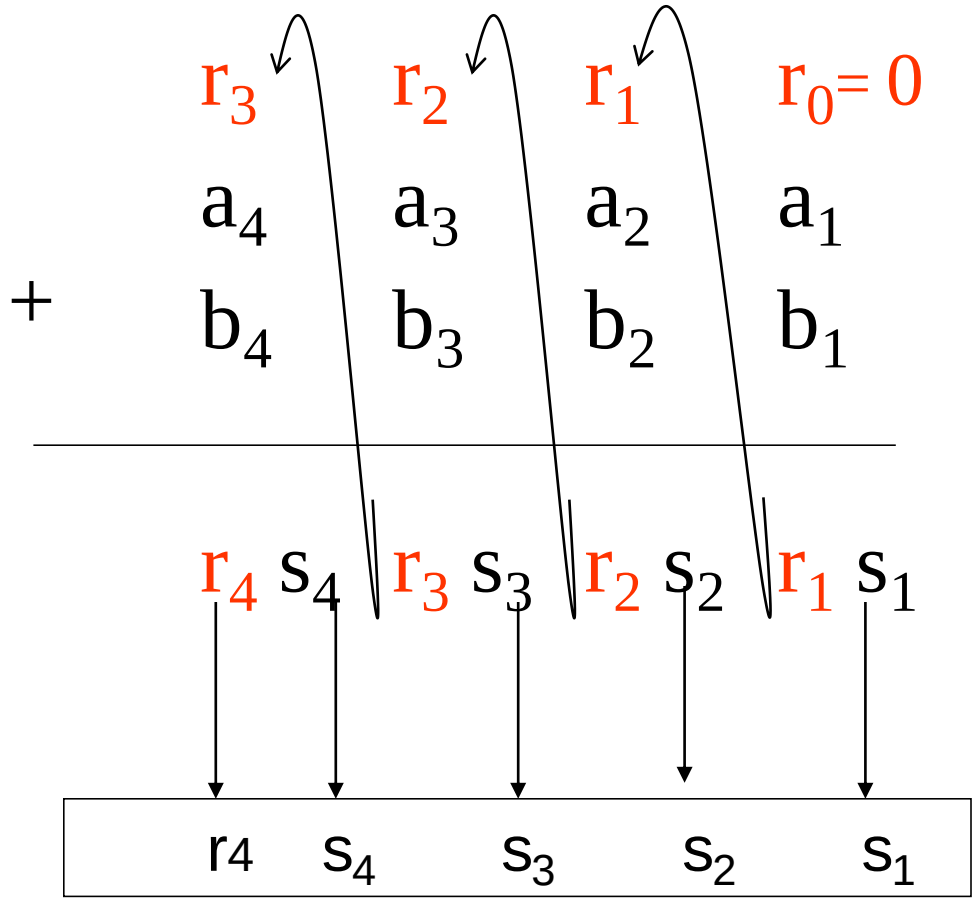
- Avec 9 entrées on a 2^9=**512 combinaisons** !!!!!! Comment faire pour représenter la table de vérité ?????

- Il faut trouver une solution plus facile et plus efficace pour concevoir ce circuit ?

Additionneur sur 4 bits

•Lorsque on fait l'addition en binaire , on additionne bit par bit en commençant à partir du poids fiable et à chaque fois on propage la retenue sortante au bit du rang supérieur.

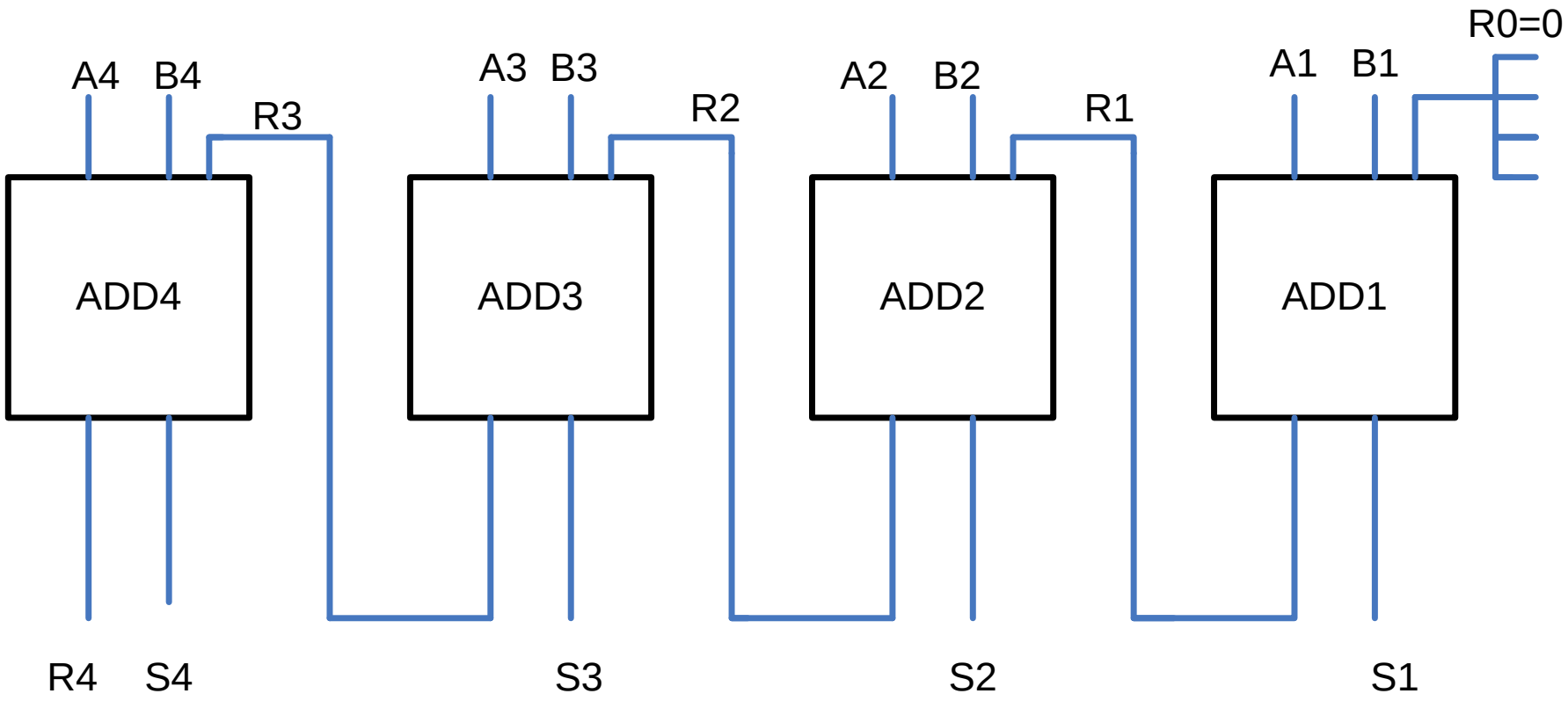
L'addition sur un bit peut se faire par un additionneur complet sur 1 bits.



Résultat final

Additionneur 4 bits (schéma)

Le premier mot $A(a_3a_2a_1a_0)$
Le deuxième mot $B(b_3b_2b_1b_0)$



Comparateur

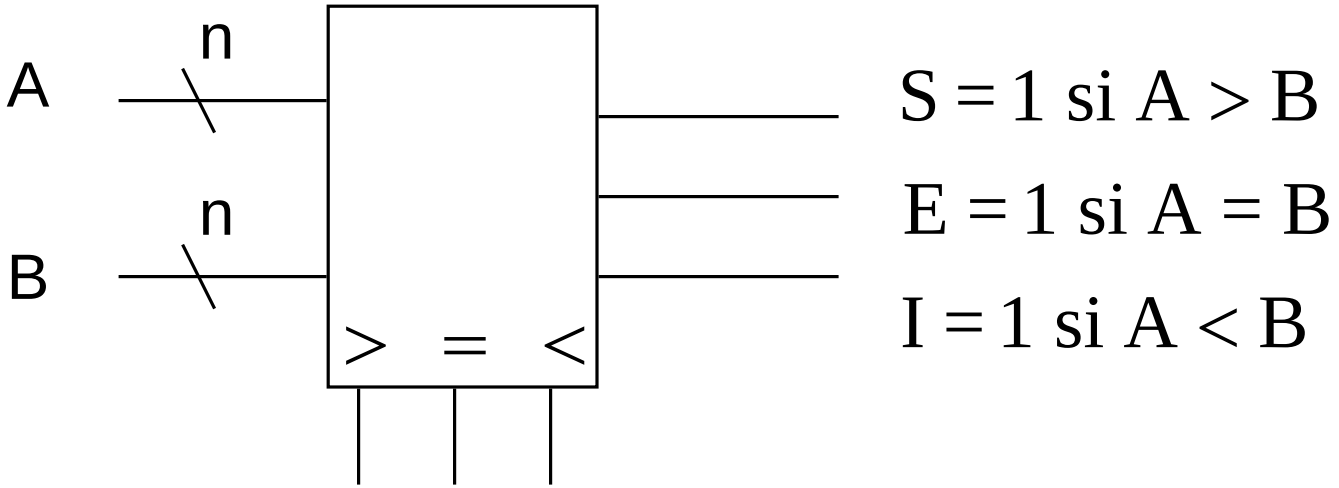
- C'est un circuit combinatoire qui permet de comparer entre deux nombres binaire A et B.
- Il possède 2 entrées :
  - A : sur n bit
  - B : sur n bit

Il possède 3 sorties

E : égalité (A=B)

I : inférieur (A < B)

S : supérieur (A > B)



Entrées de cascadage

Pour une comparaison à n autres bits

Comparateur sur un bit

Il possède 2 entrées :

A : sur un bit

B : sur un bit

Il possède 3 sorties

fe : égalité (A=B)

fi : inférieur (A < B)

fs : supérieur (A > B)

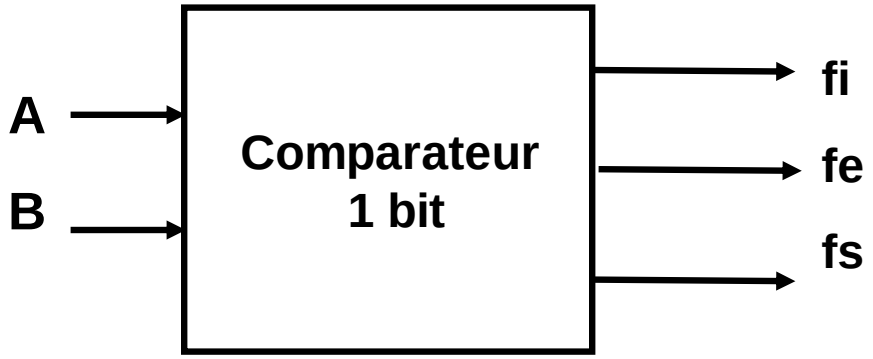


Table de vérité

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

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

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

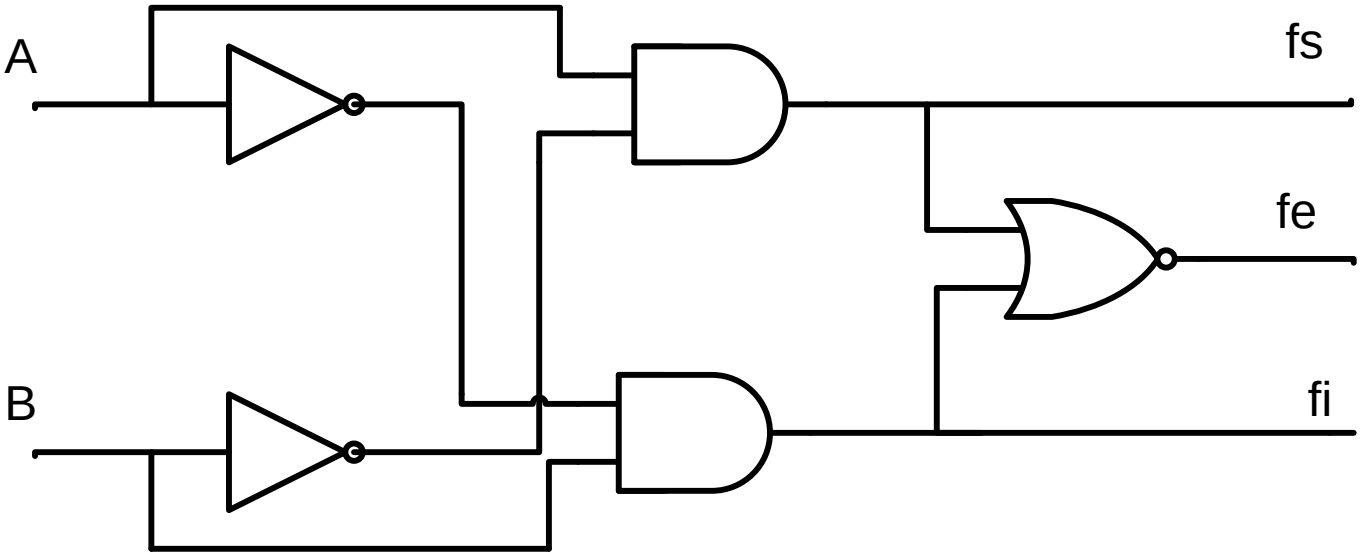
$$fe = \overline{A}\overline{B} + AB = \overline{A \oplus B} = \overline{fs + fi}$$

Logigramme comparateur sur un bit

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

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

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



Logigramme comparateur sur 1 bit

Exemple 2 : Comparateur sur 2 bits

Il possède 2 entrées :

A : sur 2 bits (A_2A_1)

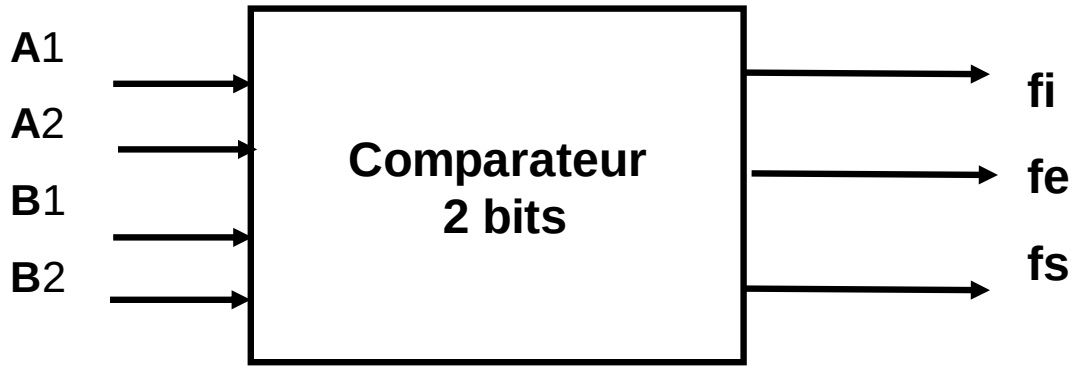
B : sur 2 bits (B_2B_1)

Il possède 3 sorties

fe : égalité

fi : inférieur

fs : supérieur



Comparateur 2 bits

A=B si **A2=B2** et **A1=B1**

$$fe = \overline{(A2 \oplus B2)}.\overline{(A1 \oplus B1)}$$

A>B si

A2 > B2 ou (A2=B2 et A1>B1)

$$fs = A2.\overline{B2} + \overline{(A2 \oplus B2)}.(A1.\overline{B1})$$

A<B si

A2 < B2 ou (A2=B2 et A1<B1)

$$fi = \overline{A2}.B2 + \overline{(A2 \oplus B2)}.(\overline{A1}.B1)$$

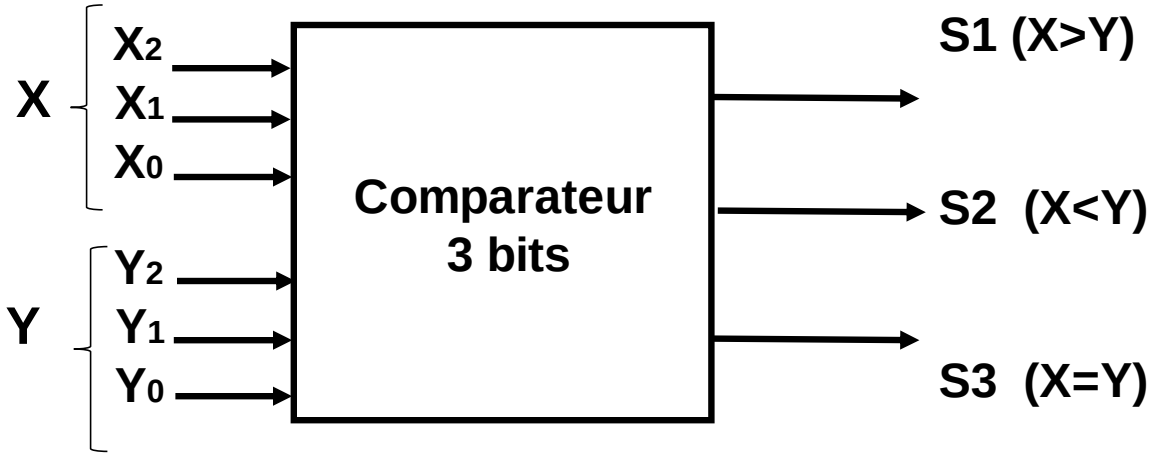
A2	A1	B2	B1		fs	fe	fi
0	0	0	0		0	1	0
0	0	0	1		0	0	1
0	0	1	0		0	0	1
0	0	1	1		0	0	1
0	1	0	0		1	0	0
0	1	0	1		0	1	0
0	1	1	0		0	0	1
0	1	1	1		0	0	1
1	0	0	0		1	0	0
1	0	0	1		1	0	0
1	0	1	0		0	1	0
1	0	1	1		0	0	1
1	1	0	0		1	0	0
1	1	0	1		1	0	0
1	1	1	0		1	0	0
1	1	1	1		0	1	0

Comparateur 3 bits

- Un circuit combinatoire qui permet de comparer entre deux nombres binaire X et Y.

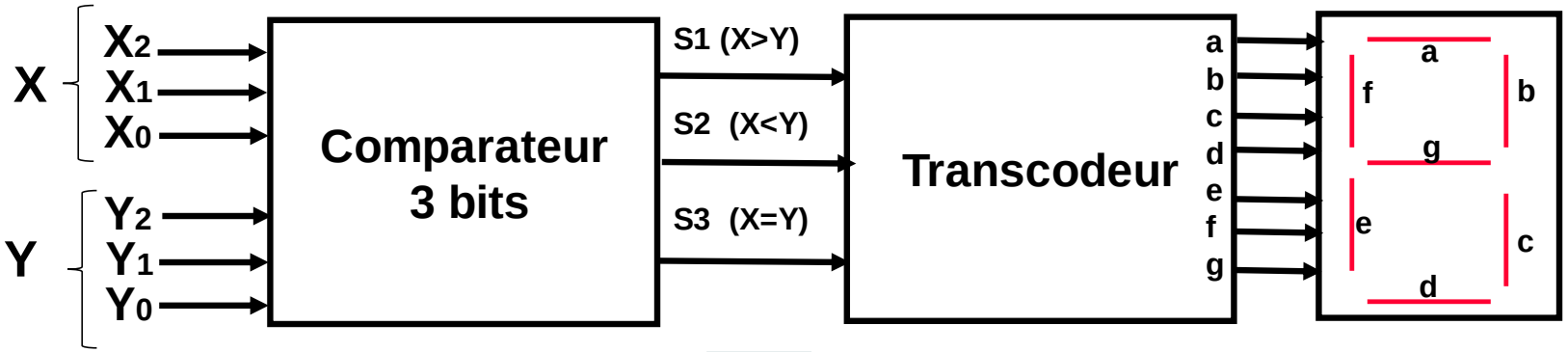
- Il possède 2 entrées :
  - A : sur 3 bits
  - B : sur 3 bits

Il possède 3 sorties
fe : égalité (X=Y)
fi : inférieur (X < Y)
fs : supérieur (X> Y)

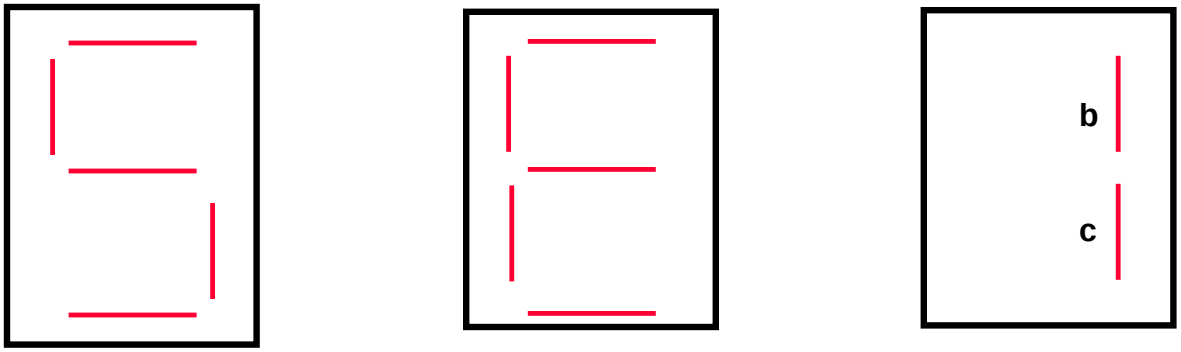


Deux circuits combinatoires

Exemple: Circuit plus complexe = Comparateur + Transcodeur

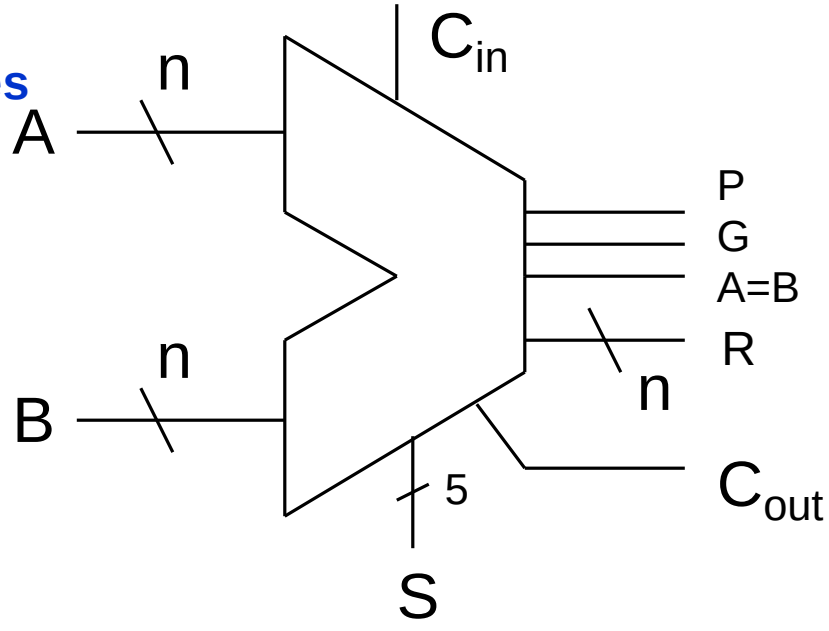


Par exemple



(Si X>Y) (Si X=Y) (Si X<Y)

ALU (ou UAL)
Unité Arithmétique et Logique



Exemple :

Résultat

$R = A + \overline{B}$

$R = A + B$

$R = A + B + 1$

...

$R = A$ ou B

$R = A$ nand B

...

Circuits séquentiels

Plan

- Introduction
- Définition d’une bascule
- Présentation de quelques bascules (RS, D, JK)

Applications :

Les registres; les registres à décalage

Les compteurs modulo n

Définition

- Rappel : **Circuit combinatoire** = la valeur des sorties **St** dépendent de la valeur des entrées **(Ei)**

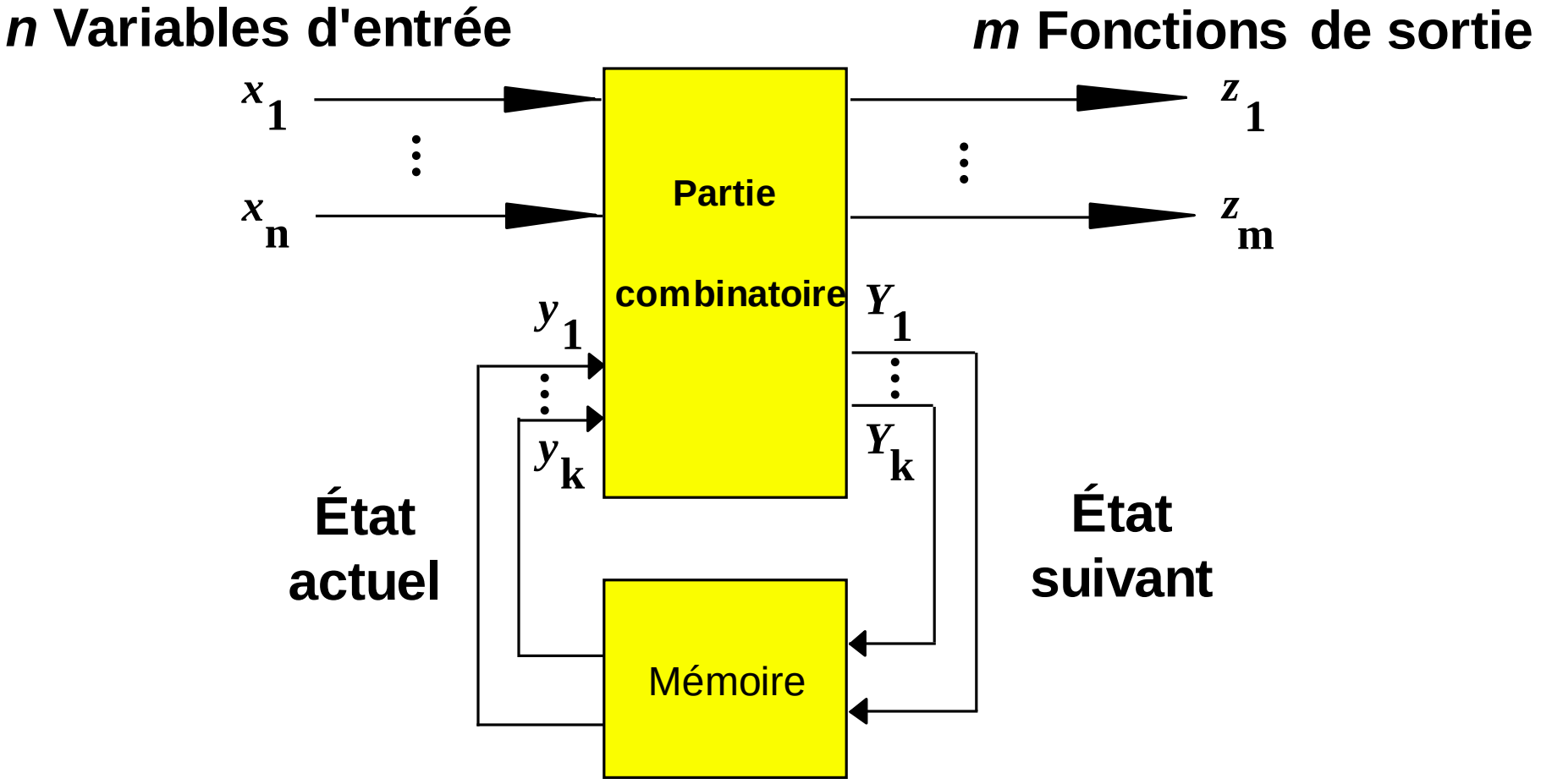
  St = f(E0, E1, E2…) **sans mémoire**

- Un **circuit séquentiel** : faculté de **mémorisation**

- La valeur des sorties à l'instant **t** dépendent de la valeur des entrées **e(t)** de la valeur des sorties à l'instant **t-1**

St = f(E0, E1, E2…, St-1)

Circuit séquentiel



Circuit séquentiel : Etats Stables

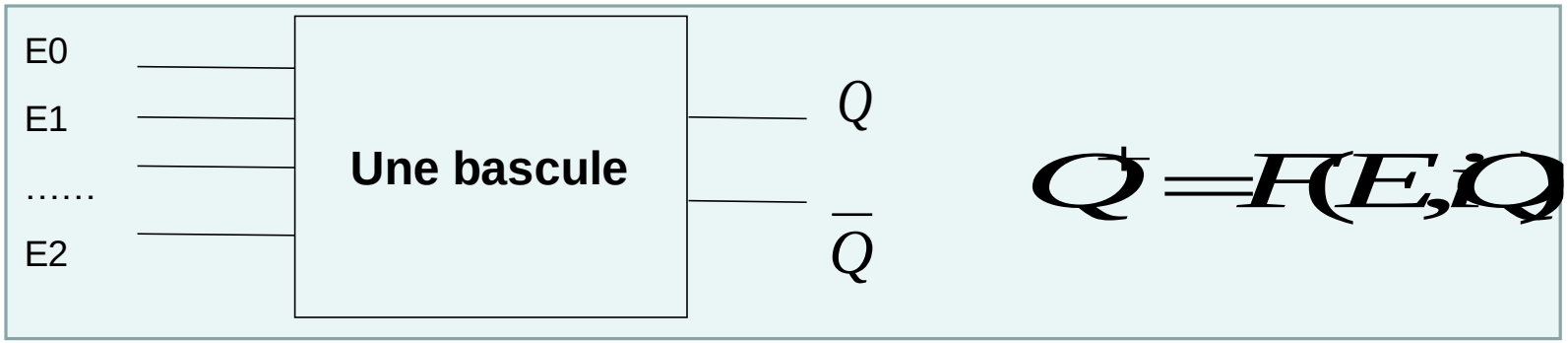
- **Les circuits séquentiels** de base sont les **bascules** (flip-flops)
- Une bascule à deux états stables (bistables)
- Les bascules : **capables de conserver l'état de leur sortie** même si la combinaison des signaux d'entrée ayant provoqué cet état de sortie disparaît.

Les bascules (flip-flops)

Bascules —(Éléments de base)→ **Logique sequentielle**

- Bascules Synchrones ou des bascules Asynchrone .
- Chaque bascule **possède des entrées** et **deux sorties** Q et $\overline{Q}$

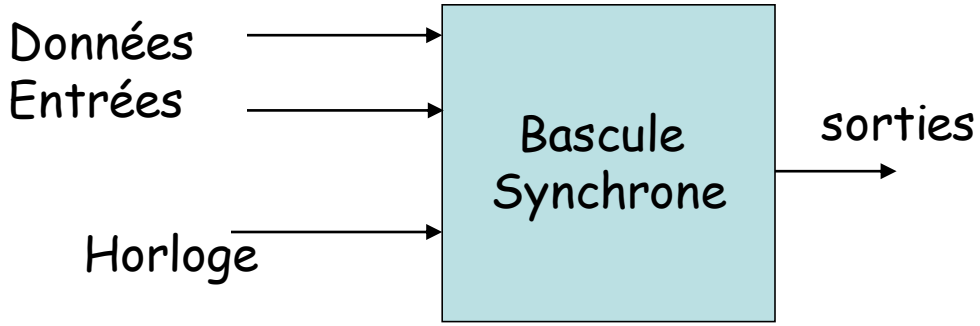
- Une bascule possède la fonction de **mémoration** et de **basculement**.



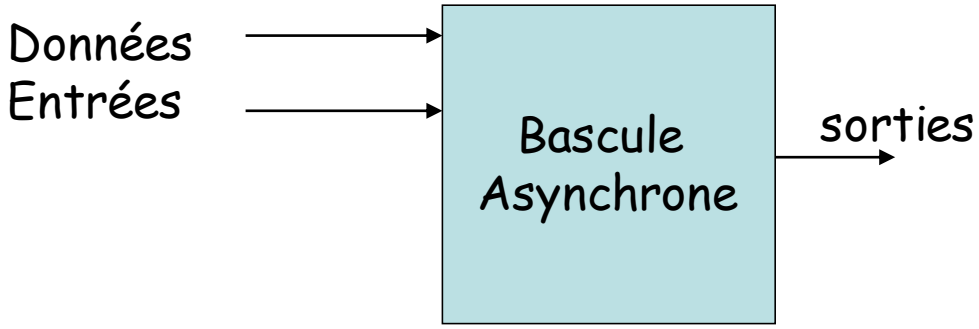
Types de bascules : RS, RST ,D ,JK , T

Bascules Synchrones/Asynchrones

- Les bascules synchrones : asservies à des impulsions d'horloge et donc insensibles aux bruits entre deux tops

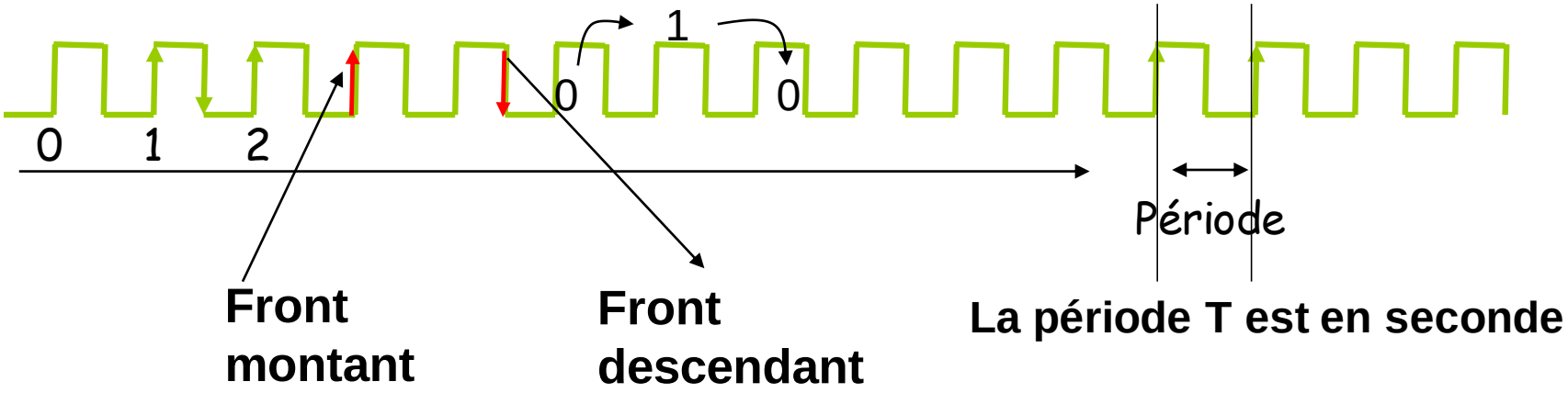


- Les bascules asynchrones, non asservies à une horloge et prenant en compte leurs entrées à tout moment.



Horloge (Clock)

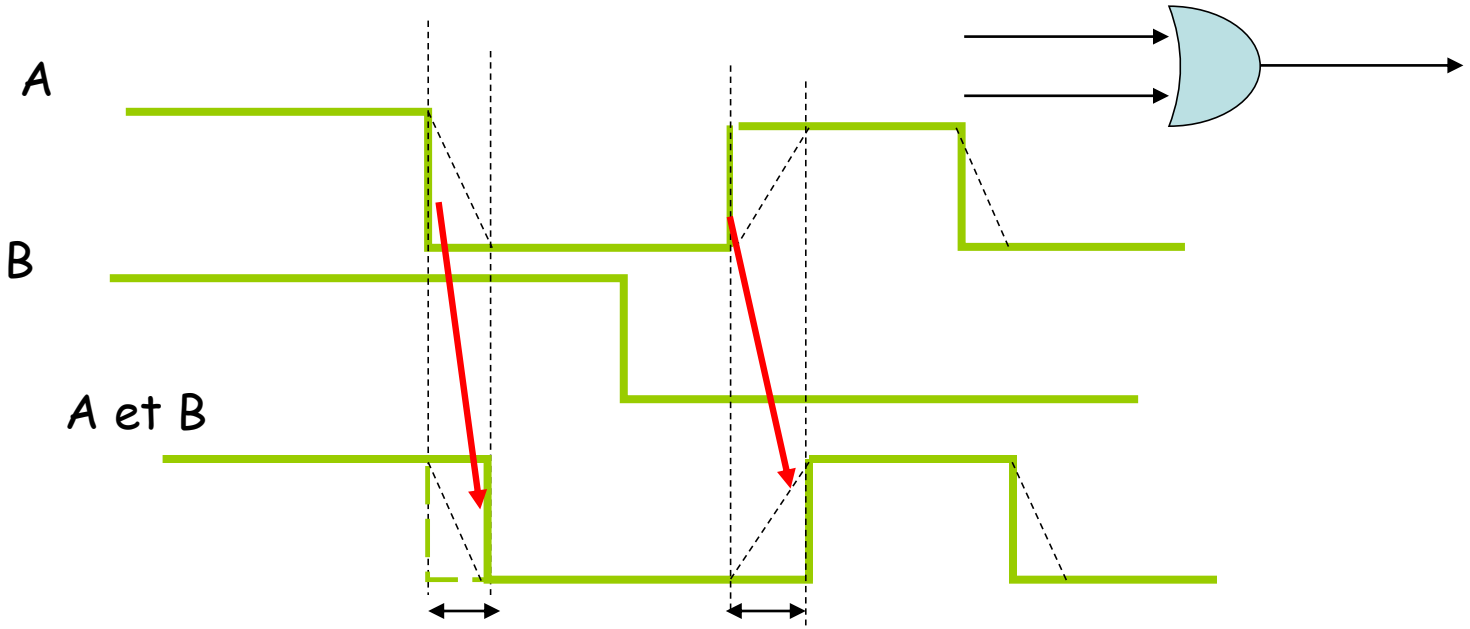
- **Horloge :** composant passant indéfiniment et régulièrement d'un niveau haut à un niveau bas (succession de 1 et de 0), chaque transition s'appelle un top.



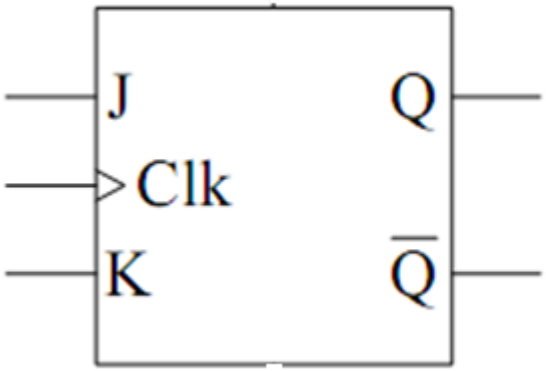
Fréquence = nombre de changement par seconde en hertz (Hz)
Fréquence = 1/période
Une horloge de 1 hertz a une période de 1 seconde
……1 megahertz……1 microseconde
……1 gigaHz……1 nanoseconde

Temps de réponse des portes logiques

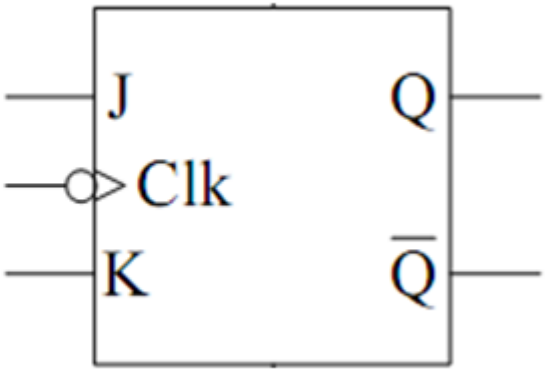
- L'instant séparant l'instant où les données sont appliquées de l'instant où les sorties sont positionnées n'est pas nul.



Horloge (Clock)



Front montant

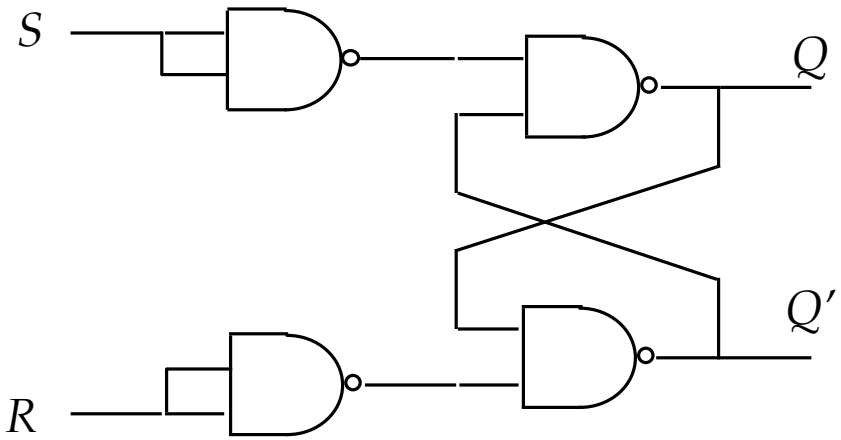


Front descendant

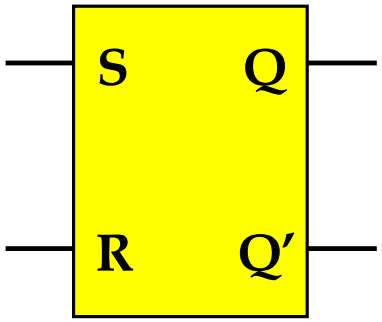
Bascules
RS D JK

Bascule RS

- Diagramme, symbole et table de transition :

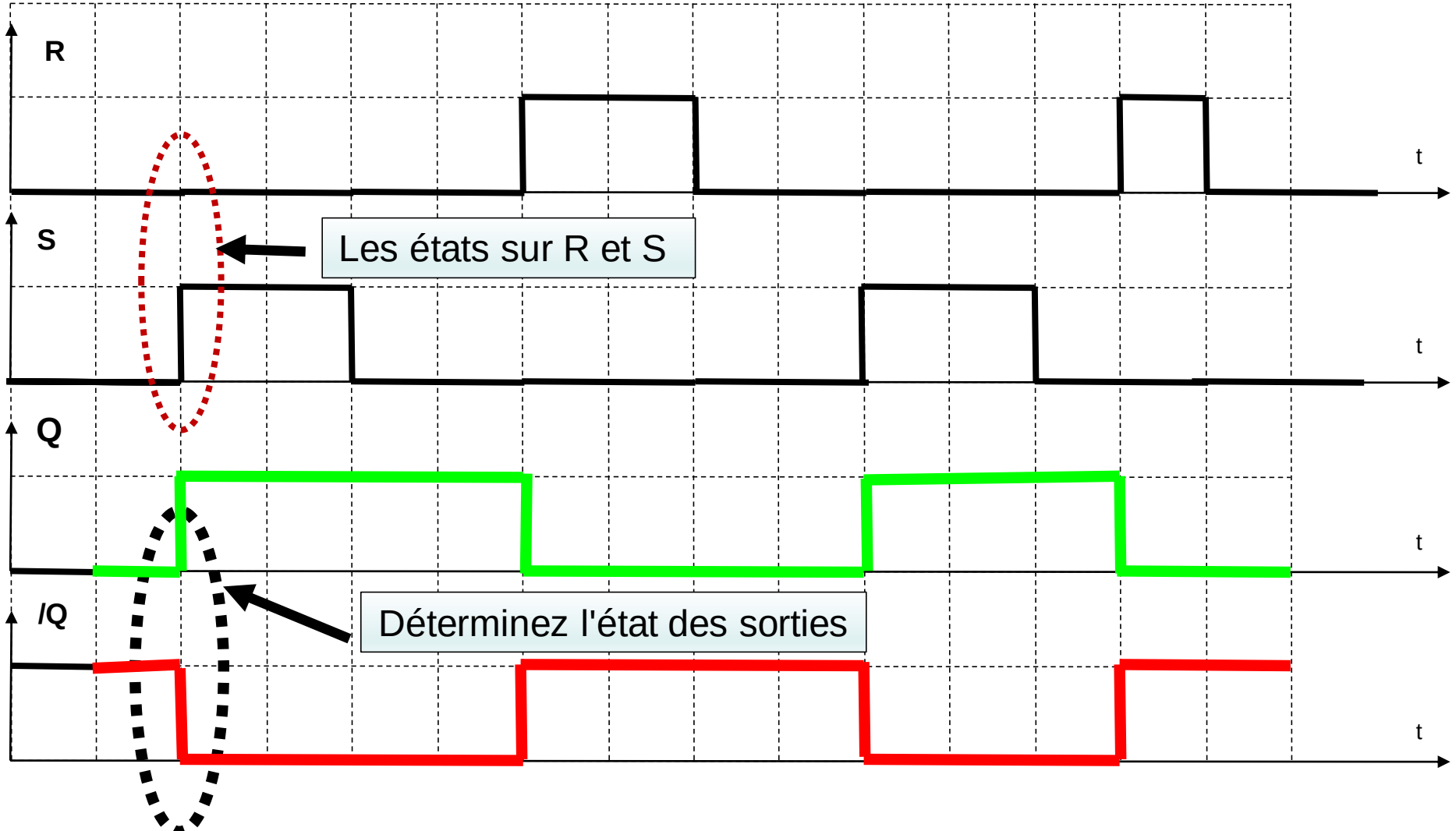


R = Reset (Mise à 0)
S = Set (Mise à 1)



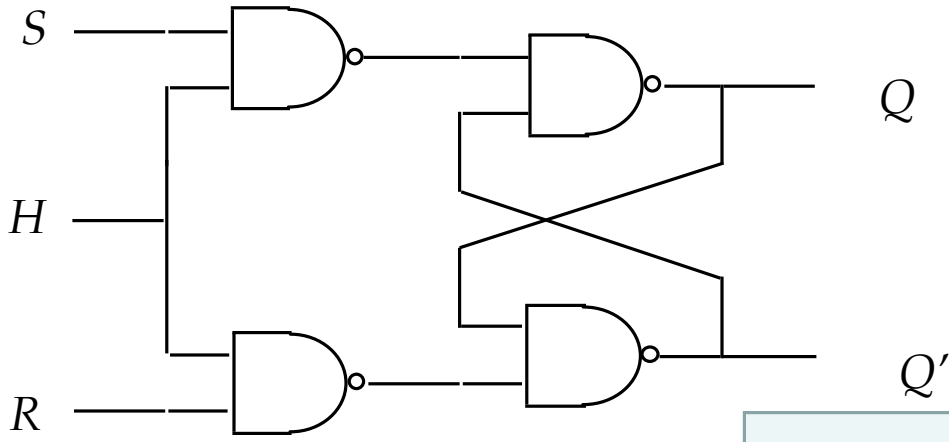
R	S	Q_{t+1}	
0	0	Q_t	Ne change pas d'état
0	1	1	Mise à 1
1	0	0	Mise à 0
1	1	?	Interdit

Bascule RS



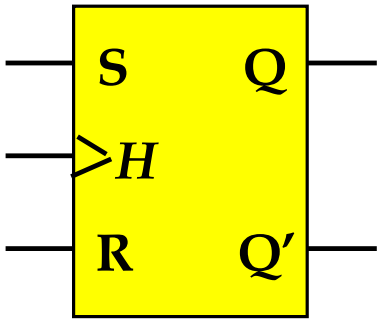
Bascule RSH

- Diagramme, symbole et table de transition :



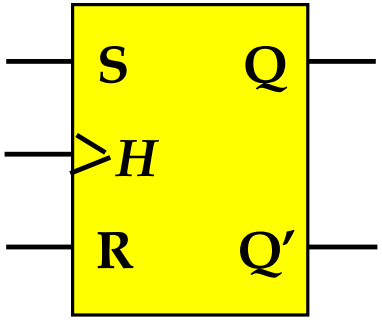
Condition supplémentaire :
H actif

- **Si H=1 mémoire classique**
- **Si H=0 mémoire figée**



R	S	Q_{t+1}	
0	0	Q_t	Ne change pas d'état
0	1	1	Mise à 1
1	0	0	Mise à 0
1	1	?	Interdit

Bascule RSH



R	S	Q_{t+1}	
0	0	Q_t	Ne change pas d'état
0	1	1	Mise à 1
1	0	0	Mise à 0
1	1	?	Interdit

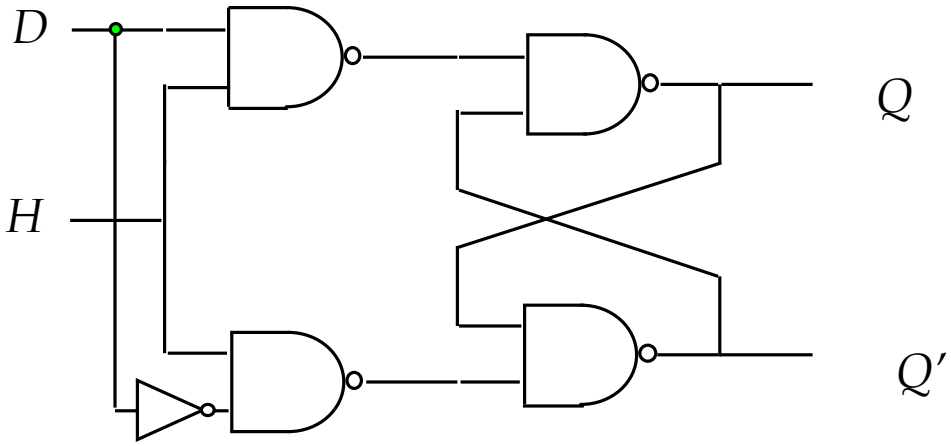
Table de vérité

Qt	Q(t+1)	R	S
0	0	*X*	*0*
0	1	*0*	*1*
1	0	*1*	*0*
1	1	*0*	*X*

Table de transition

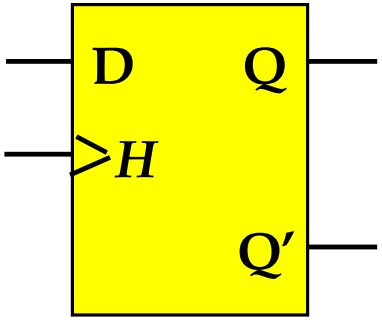
Bascule D

- Diagramme, symbole et table de transition :



D	Q_{t+1}
0	0
1	1

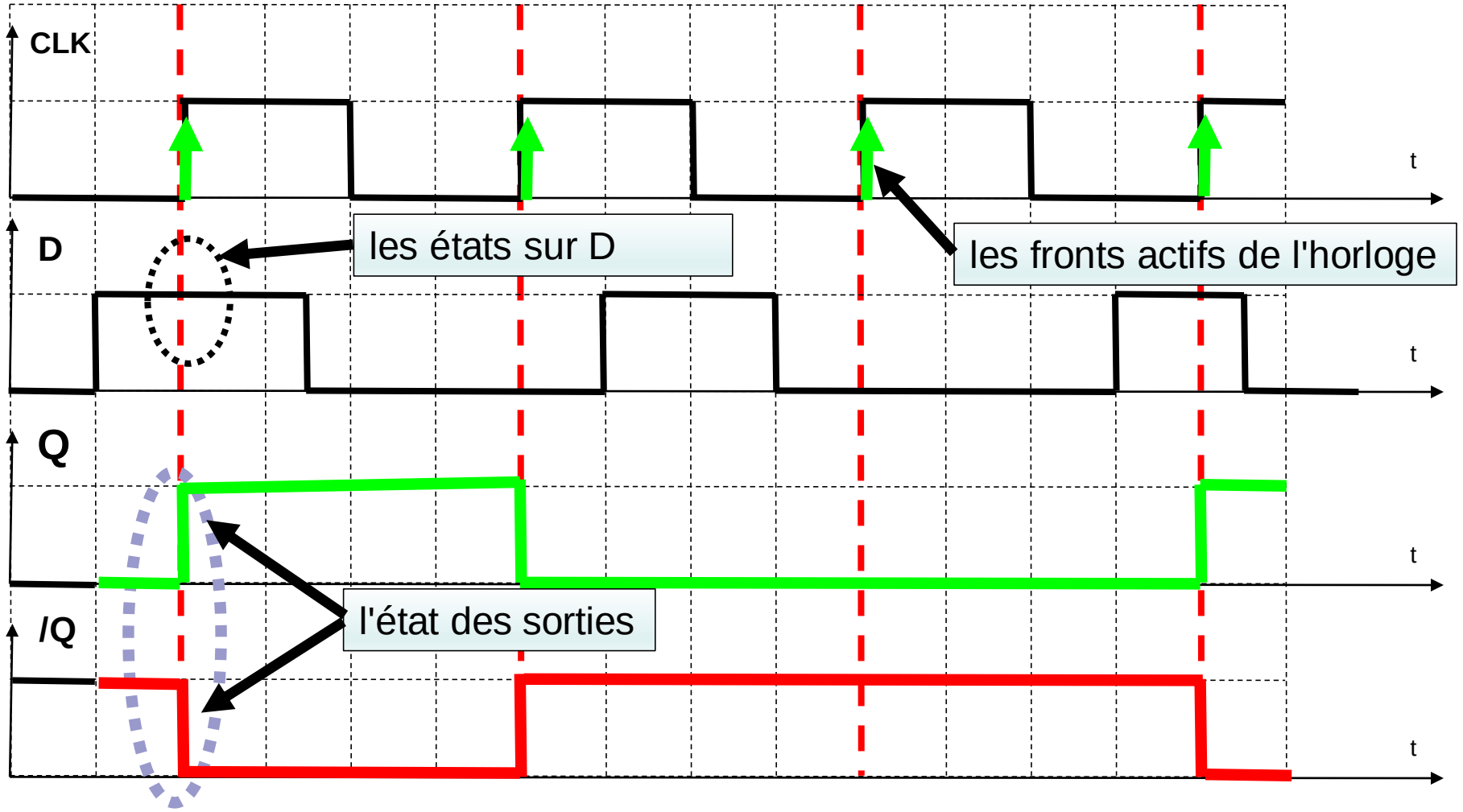
Table de vérité



Qt	Q(t+1)	D	C	(exemple Front montant)
0	0	*0*	*X*	
0	1	*1*	*Front montant*	
1	0	*0*	*Front montant*	
1	1	*1*	*X*	

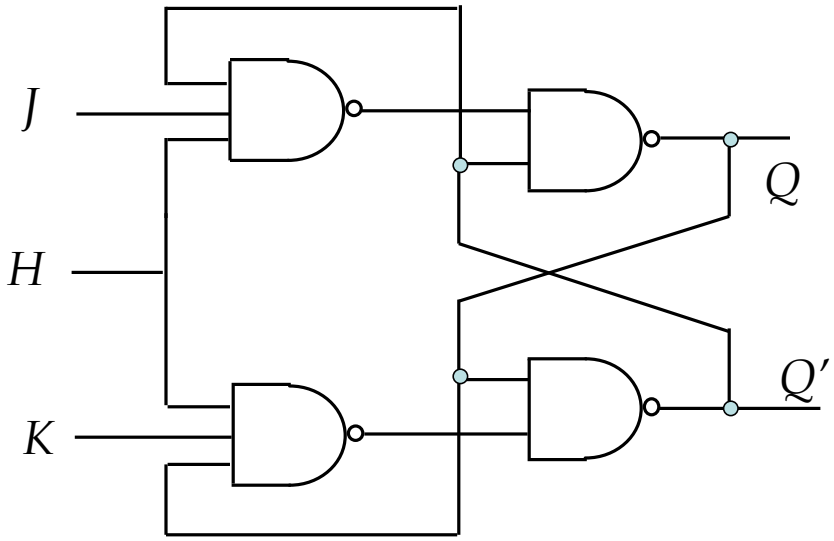
Table de transition

Bascule D


CLK
t
D
les états sur D
les fronts actifs de l'horloge
t
Q
t
/Q
l'état des sorties
t

Bascule JK

- Diagramme, symbole et table de transition :



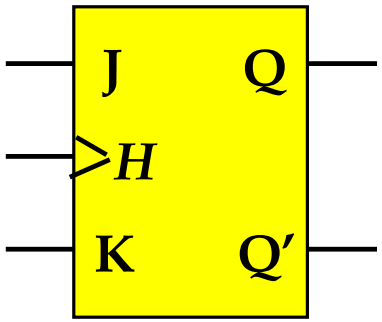


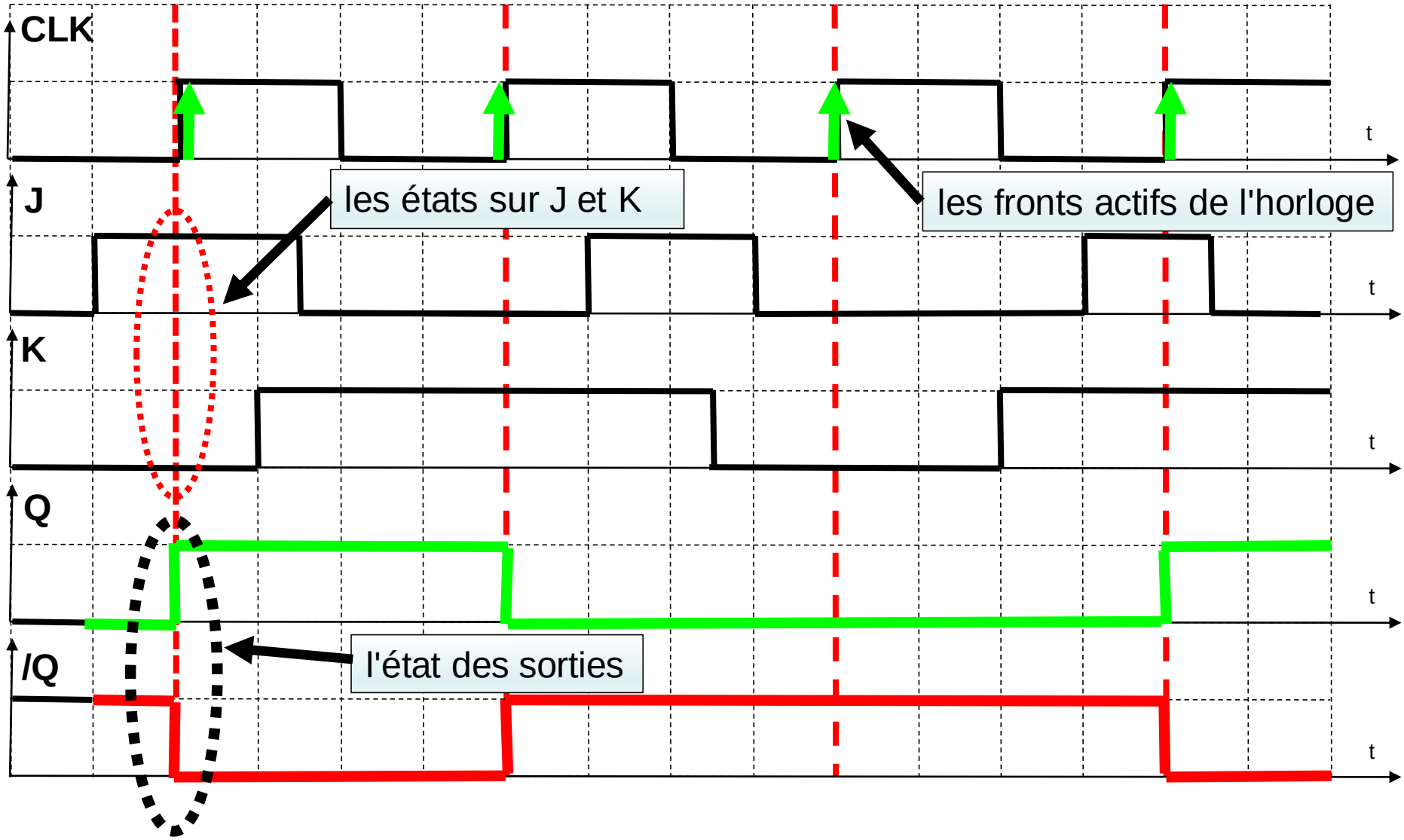
Table de vérité

J	K	Q_{t+1}	
0	0	Q_t	Ne change pas d'état
0	1	0	Mise à 0
1	0	1	Mise à 1
1	1	$\overline{Q_t}$	Change d'état

Qt	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table de transition

Bascule JK


CLK
J
K
Q
/Q
t
les états sur J et K
les fronts actifs de l'horloge
l'état des sorties

Applications des circuits séquentiels

- Les registres
- Les compteurs

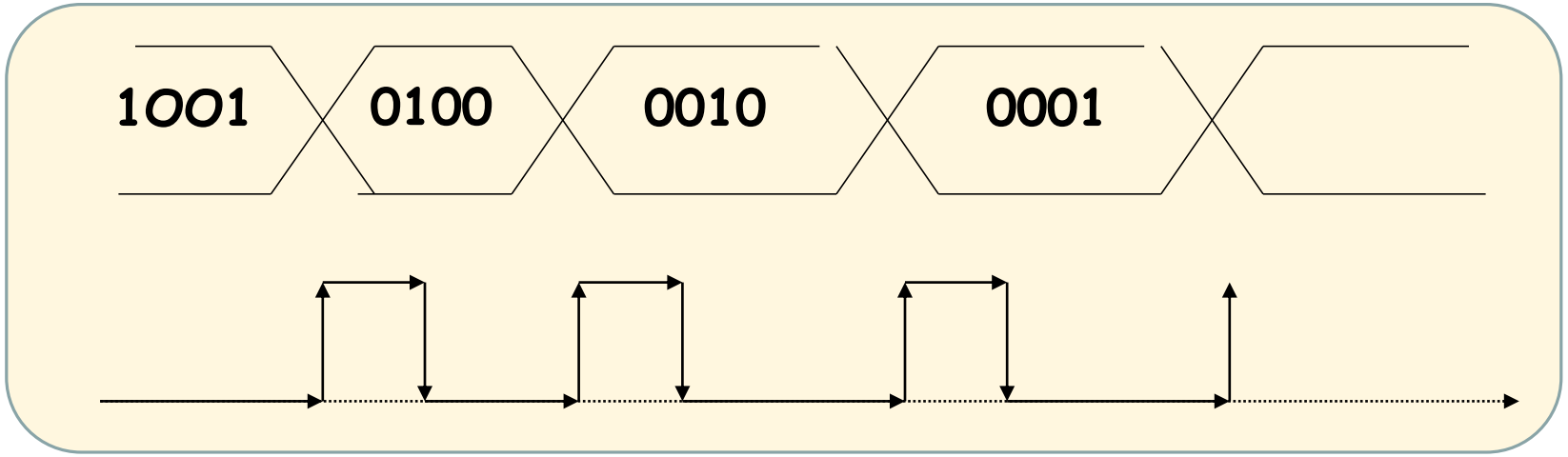
Applications des circuits séquentiels

- **Les registres à décalage :**

Dans un registre à décalage droite (resp. gauche) :

<n-1, ……i+1, i, i-1, …,1, 0>

La sortie de la bascule **i** à l'instant **t** correspond à la sortie de la bascule **i+1** (resp. **i-1**) à l'instant **t-1**.

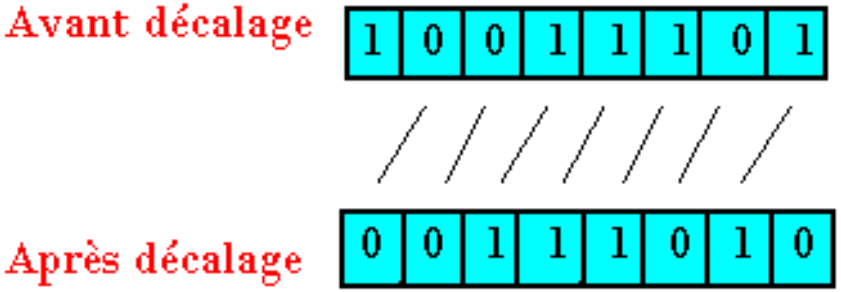


Applications des circuits séquentiels

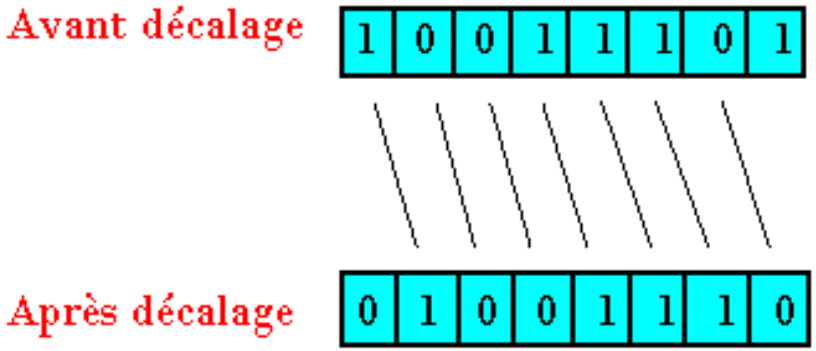
Les registres à décalage :

Le décalage à droite consiste à faire avancer l'information vers la droite:

Exemples:



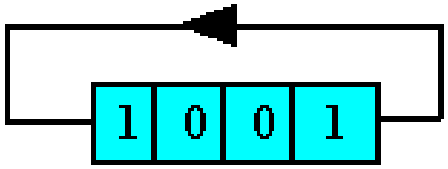
décalage à droite



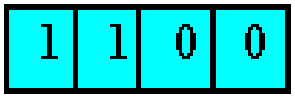
décalage à gouche

Applications des circuits séquentiels

Les registres à décalage :

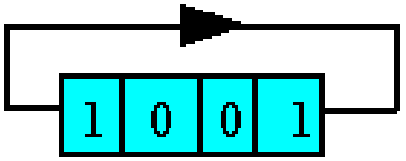


Avant décalage

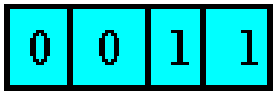


Après décalage

Décalage circulaire à droite



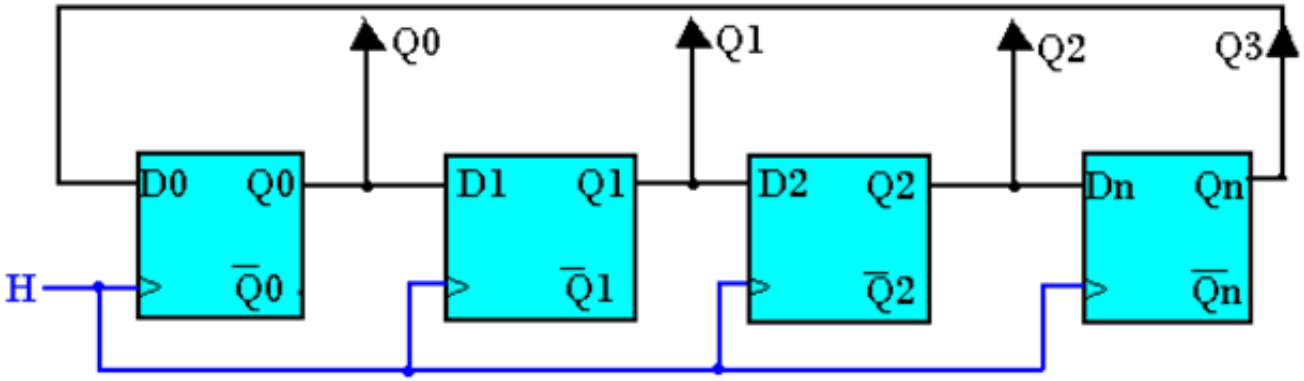
Avant décalage



Après décalage

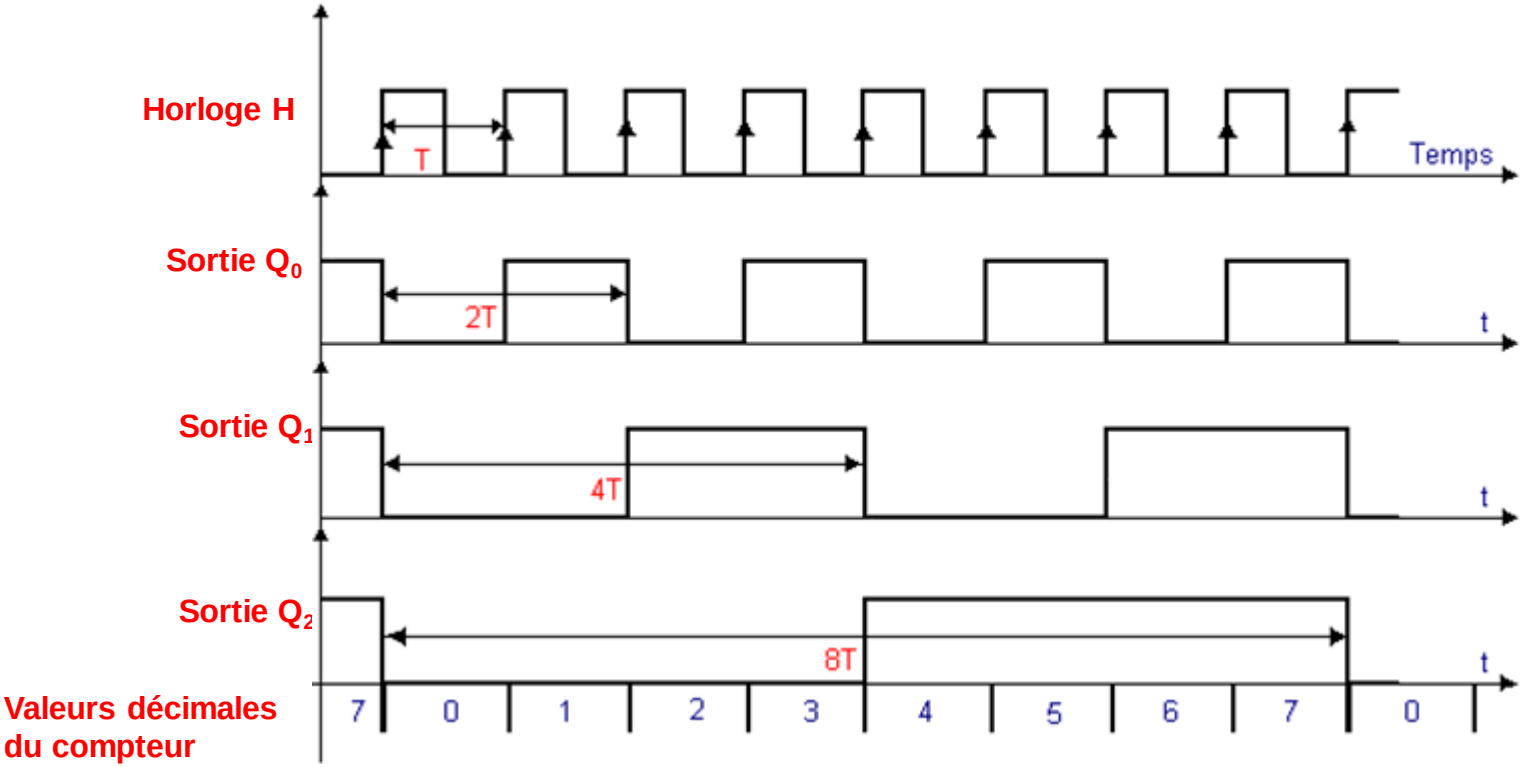
Décalage circulaire à gauche

Exemple: registre à décalage circulaire 4 bits à bascule D



Les compteurs :

Exercice: on désire réaliser un compteur modulo 8 : 0, 1, 2, …7, 0, 1….
En utilisant les bascules JK



Nous avons trois bits : donc trois bascules 0,1, 2

Exercice : réaliser ce compteur avec des bascules JK.

Etat Actuel Avant			Etat Suivant après			Ce qu'il faut appliquer aux entrées		
Q2	Q1	Q0	Q2	Q1	Q0	J2/K2	J1/K1	J0/K0
0	0	0	0	0	1	?	?	?
0	0	1	0	1	0	?	?	?
0	1	0	0	1	1	?	?	?
0	1	1	1	0	0	?	?	?
1	0	0	1	0	1	?	?	?
1	0	1	1	1	0	?	?	?
1	1	0	1	1	1	?	?	?
1	1	1	0	0	0	?	?	?
						?	?	?

Trouvez les équations de J2, K2, J1, K1, J0, K0 en fonction des Qi avant (à l'instant t)

Table de transition de la bascule JK

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\overline{Q_t}$

Table de vérité

Q_{avant}	$Q_{après}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table de transition

Tables de transition

Q_{avant}	$Q_{après}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table de transition JK

Trouvez les équations de J2, K2, J1, K1, J0, K0 en fonction des Qi avant

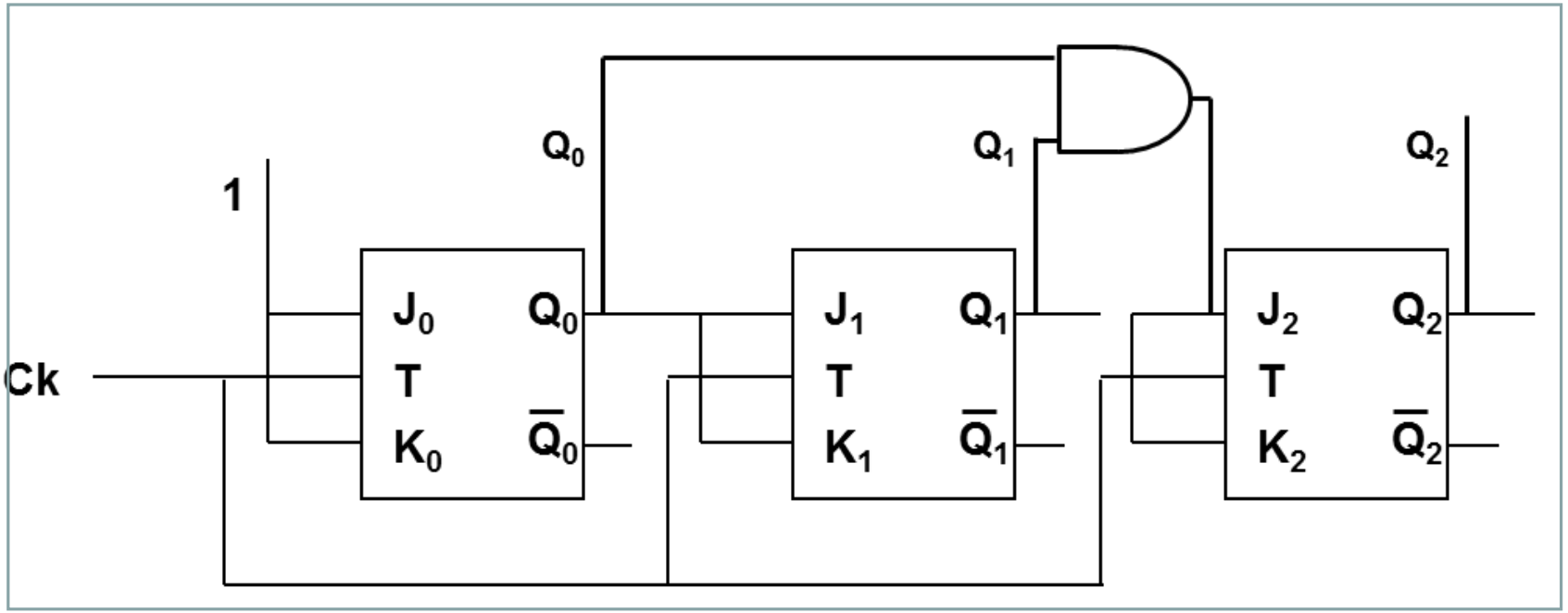
Etat Actuel Avant			Etat Suivant après			Ce qu'il faut appliquer aux entrées		
Q2	Q1	Q0	Q2	Q1	Q0	J2/K2	J1/K1	J0/K0
0	0	0	0	0	1	0 X	0 X	1 X
0	0	1	0	1	0	0 X	….	
0	1	0	0	1	1			
0	1	1	1	0	0			
1	0	0	1	0	1			
1	0	1	1	1	0			
1	1	0	1	1	1			
1	1	1	0	0	0			

Applications des circuits séquentiels

- Resultats

$J_0=K_0=1$ $J_1=K_1=Q_0$ $J_2=K_2=Q_0.Q_1$

ATTENTION : Poids fort Q_2, Poids Faible Q_0



Compteur synchrone modulo 8 à l'aide des bascules JK

Merci pour votre attention