Université Mohammed V
Faculté des Sciences de
Rabat

Support de cours du module :

Electronique Numérique

Préparé par : Pr. Aziz AMARI

Pour les étudiants de la Licence d'Excellence :

Electronique, Informatique et Robotique

Année universitaire : 2019-2020

Table des matières

Chapitres du cours

Ch. 1 : Fonctions et Opérateurs Logiques

Ch. 2 : Les Circuits Combinatoires

Ch. 3 : Les Circuits Séquentiels

Chapitre 1

Fonctions et Opérateurs Logiques

George Boole (1815-1864)

I. Introduction

Les systèmes logiques fonctionnent en mode binaire ; les variables d'entrée et de sortie ne prennent que deux valeurs : « **0** » ou « **1** ». Ces deux valeurs (états) correspondent à des plages définies à l'avance.

Concrètement, lors de la réalisation de circuits électroniques numériques, les 2 niveaux logiques sont constitués par **2 tensions différentes**. La tension correspondant au niveau 0 est en généra l0 V. La tension correspondant au niveau 1 dépend de la technologie utilisée. Une norme couramment répandue est la norme TTL :

- niveau logique **0** ↔ **0** à **0,8** V
- niveau logique **1** ↔ **2,4** à **5** V

Dans le domaine de la logique numérique, on utilise d'autres expressions qui sont synonymes de 0 et de 1 :

Niveau logique 0	Niveau logique 1
Non	Oui
Faux	Vrai
Ouvert	Fermé
Arrêt	Marche
Bas	Haut
Éteint	Allumé

La logique binaire basée sur l'algèbre de Boole permet de décrire dans un modèle mathématique les manipulations et traitement des informations binaires, et d'analyser les systèmes numériques.

En général, la logique utilisée est la logique **positive**, dans laquelle le niveau dit actif est le **niveau 1**. Ça sera le cas dans la totalité de ce cours. Mais il existe également la **logique négative**, dans laquelle il s'agit du **0**.

II. Variables et fonctions logiques

Cette partie abordera la représentation des fonctions sous forme algébrique, sous forme de table de vérité puis sous forme de schémas, et leur simplification au moyen des règles de l'algèbre de Boole et des tableaux de Karnaugh.

II.1. Algèbre de Boole

George Boole, philosophe et mathématicien irlandais du 19ème siècle est l'auteur d'une théorie sur l'art de construire un raisonnement logique au moyen de propositions qui ont une seule réponse **OUI** (**VRAI**) ou **NON** (**FAUX**). Il est le premier, avec de Morgan, à essayer de fonder la logique mathématique indépendamment de la philosophie. Pour cela, il crée une algèbre binaire n'acceptant que deux valeurs numériques, **0** ou **1**. '**1**' pour une proposition toujours vraie, et '**0**' une proposition fausse. Il définit des lois (**ET** et **OU**) sur cette algèbre satisfaisant un certain nombre de propriétés (distributivité, commutativité,...). L'ensemble des opérations formelles appliquées à ces propositions forme une structure mathématique appelée

algèbre de Boole. A son époque, il s'agissait de développement purement théorique car on ignorait l'importance qu'allait prendre cette algèbre avec l'informatique.

Les concepts de la logique booléenne ont été ensuite appliqués aux circuits électroniques par ***Claude Shannon*** (**1916-2001**). Cette algèbre est applicable à l'étude des systèmes possédant deux états s'excluant mutuellement. Dans la logique positive (la plus couramment utilisée), on associe **OUI** à **1** et **NON** à **0**. Dans la logique négative, c'est l'inverse. L'algèbre booléenne binaire est à la base de la mise en œuvre de tous les systèmes numériques : ordinateurs, systèmes numériques portables, systèmes de communication numériques, etc. Elle permet entre autres de simplifier les fonctions logiques, et donc les circuits électroniques associés.

II.2. Variables logiques

Ce sont des variables ne pouvant prendre que deux valeurs distinctes : « **0** » ou « **1** ». Une variable binaire peut représenter n'importe quel dispositif binaire (contact, Interrupteur, lampe, électrovanne,...).

L'association de ces variables dites booléennes ou logiques munie d'un certain nombre d'opérateurs donne naissance à des fonctions logiques.

II.3. Fonctions logiques

Une fonction logique est une fonction d'une ou plusieurs variables logiques, combinées entre elles par **3** fonctions **élémentaires** simples : **NON**, **OU** et **ET**.

Il existe également des fonctions élémentaires composées de fonctions élémentaires simples : NON-ET, NON-OU, OU-EXCLUSIF, NON-OU-EXCLUSIF.

Tout circuit logique peut être décrit par des fonctions logiques et/ou une table de vérité, et être réalisé à partir des opérateurs logiques élémentaires.

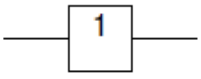
III. Opérateurs logiques de base

Les fonctions logiques peuvent être représentées schématiquement par des opérateurs logiques, encore appelés portes logiques. Chaque **opérateur** est représenté par un **symbole** et sa **fonction** est définie par une **table de vérité**.

III.1. Opérateur identité 'Oui'

Cette fonction fait intervenir une seule variable d'entrée (***A***). Le niveau logique de la sortie est égal au niveau logique de l'entrée : ***S = A***. On peut exprimer cette propriété sous forme d'un tableau entrée/sortie, appelé table de vérité.

Symbole européen



Symbole américain

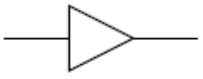
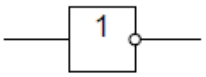
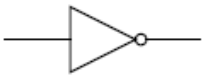


Table de vérité

A	$S = A$
0	0
1	1

III.2. Opérateur inverseur 'Non' ou 'Not'

Elle fait également intervenir une seule variable d'entrée. Le niveau logique de sortie est l'inverse de celui présent à l'entrée.

Symbole européen	*Symbole américain*	*Table de vérité*
		A | $S=\overline{A}$ 0 | 1 1 | 0

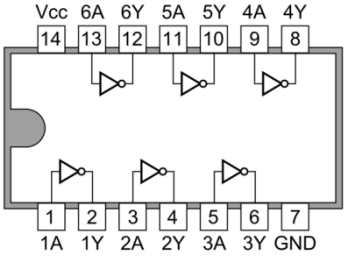
Soit $\boldsymbol{A}$ une variable quelconque. La fonction complément de la variable $\boldsymbol{A}$ est notée : $\boldsymbol{S = f(A) = \overline{A}}$ (Prononcer "**A barre**", "**non A**", ou encore "**complément de A**").

❖ ***Remarque 2.1 :***

Le cercle est en général utilisé pour indiquer une complémentation. On appelle souvent cet opérateur "inverseur".

❖ ***Exemple de Portes logiques NOT***

Le circuit intégré de référence : **74LS04**

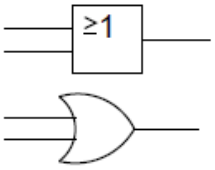


- Ce circuit comporte **14 broches** (**pin** ou **pattes**).
- L'alimentation en tension est faite entre les pattes **7** et **14**; respectivement pour la masse (**GND**) et **5V** (**VCC**).
- Il contient **6 portes** logiques NOT indépendantes ayant **2 entrées chacune**. Les entrées de chacune de ces portes NOT sont identifiées par la broche **A** et **B**, tandis que la sortie est représenté par **Y**.

III.3. Fonction OU (ou somme logique)

La fonction logique OU est également appelée "somme logique", ou "union logique". Sa notation algébrique utilise le symbole de la somme arithmétique. Pour 2 variables A et B, on a :

$$\boldsymbol{f(A,B) = A + B}$$



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

Propriétés du OU :

$x+1=1$ $\quad x+\overline{x}=1$
$x+0=x$ $\quad x+x=x$

Élément neutre : 0

Élément absorbant : 1

Le terme anglais est "OR".

III.3.1 Représentation d'Euler ou de Venn de la fonction OR

Si l'ensemble $\boldsymbol{A}$ est l'ensemble pour lequel la variable $\boldsymbol{A = 1}$, et l'ensemble $\boldsymbol{B}$ celui pour lequel la variable $\boldsymbol{B = 1}$, $\boldsymbol{f}$ est **l'union** de $\boldsymbol{A}$ et $\boldsymbol{B}$, c'est-à-dire l'ensemble des valeurs de $\boldsymbol{f}$ égales à $\boldsymbol{1}$ **(**Figure 2. 1**)**. Donc L'ensemble dans lequel les variables **A** ou **B** sont à **1**, ou **somme logique**, sera la surface formée de la réunion des deux régions précédentes.

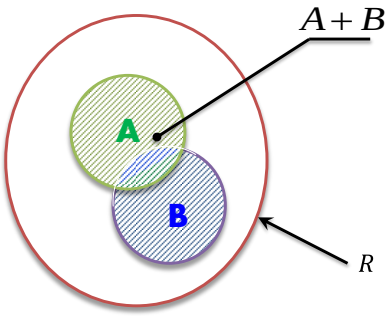
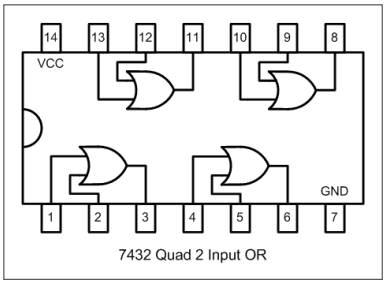


Figure 2. 1. Représentation d'Euler de la fonction OR.

III.3.2 Exemple de Portes logiques OR

Le circuit intégré de Référence : **74LS32**

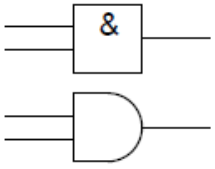
- Ce circuit comporte **14 broches** (**pin** ou **pattes**).
- L'alimentation en tension est faite entre les pattes **7** et **14**; respectivement pour la masse (**GND**) et **5V** (**VCC**).
- Il contient **4 portes** logiques OR indépendantes ayant **2 entrées chacune** et une sortie.



III.4. Fonction ET (ou produit logique)

La fonction ***ET*** est également appelée "produit logique", ou "intersection logique". Sa notation algébrique utilise le symbole de la multiplication arithmétique :

$$f(A,B) = A \cdot B$$



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

Propriétés du ET :

$x.1 = x$ $\quad x.\overline{x} = 0$

$x.0 = 0$ $\quad x.x = x$

Élément neutre : 1

Élément absorbant : 0

Le terme anglais est "***AND***".

III.4.1 Représentation d'Euler ou de Venn de la fonction ET

Si l'ensemble ***A*** est l'ensemble pour lequel la variable ***A = 1***, et l'ensemble ***B*** celui pour lequel la variable ***B = 1***, ***f*** est **l'intersection** de ***A*** et ***B***, c'est-à-dire l'ensemble des valeurs de ***f*** égales à ***1*** **(**Figure 2. 2**)**. Donc L'ensemble dans lequel les variables **A** ou **B** sont à **1**, ou **produit logique**, sera la surface formée de l'intersection des deux régions précédentes.

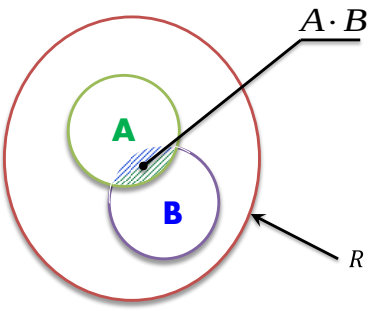


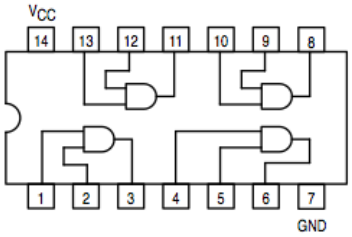
Figure 2. 2. Représentation d'Euler de la fonction AND.

❖ ***Remarque 2.2 :***

Avec la fonction ET à plus de 2 entrées, le seul cas où la sortie serait à 1 serait le cas où toutes les entrées sont à 1.

III.4.2 Exemple de Portes logiques AND

Le circuit intégré de Référence : **74LS08**



- Ce circuit comporte **14 broches** (**pin** ou **pattes**).
- L'alimentation en tension est faite entre les pattes **7** et **14**; respectivement pour la masse (**GND**) et **5V** (**VCC**).
- Il contient **4 portes** logiques **AND** indépendantes ayant **2 entrées chacune** et une sortie.

❖ ***Remarque 2.3 :***

Les opérateurs {ET, OU, NON} permettent à eux trois de réaliser n'importe quelle fonction logique : on dit qu'ils forment un groupe complet.

Le théorème de De Morgan permet de dire que les groupes {ET, NON} et {OU, NON} sont également des groupes complets.

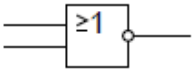
IV. Opérateurs composés

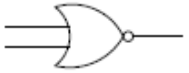
Les fonctions élémentaires composées (ou combinées, ou induits) sont obtenues en combinant entre eux les fonctions élémentaires simples **NON**, **ET** et **OU**. L'ensemble des fonctions élémentaires simples et des fonctions élémentaires combinées **NON-ET**, **NON-OU**, **OU-EXCLUSIF**, **NON-OU-EXCLUSIF** définissent un ensemble complet d'opérateurs.

IV.1. Opérateur Non-Ou (NOR)

Les deux opérateurs OU et NON peuvent être combinés en un seul opérateur NON-OU : NON-OU est donc un opérateur complet.

Symbole européen *Symbole américain* *Table de vérité*





A	B	$S=\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Si variable ***A*** est représentée par l'ensemble ***A***, et la variable ***B*** est représentée par l'ensemble ***B***, ***f*** est **le complément de l'union** de ***A*** et ***B***, c'est-à-dire l'ensemble des valeurs de ***f*** égales à ***1*** **(**Figure 2. 3**).**

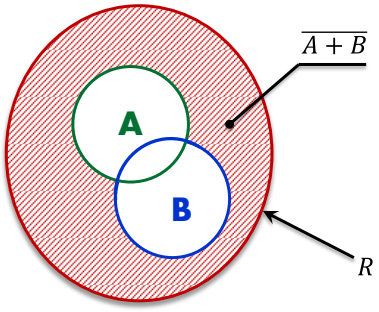
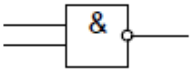


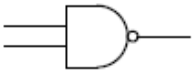
Figure 2. 3. Représentation d'Euler de la fonction NOR.

IV.2. Opérateur Non-Et (NAND)

Les deux opérateurs **ET** et **NON** peuvent être combinés en un seul opérateur NON-ET : NON-ET est donc un opérateur complet.

Symbole européen *Symbole américain* *Table de vérité*





A	B	$S=\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

Si variable ***A*** est représentée par l'ensemble ***A***, et la variable ***B*** est représentée par l'ensemble ***B***, ***f*** est **le complément de l'intersection** de ***A*** et ***B***, c'est-à-dire l'ensemble des valeurs de ***f*** égales à ***1*** **(**Figure 2. 3**).**

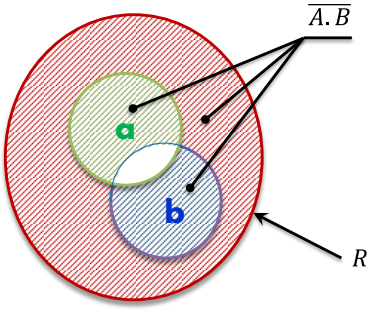


Figure 2. 4. Représentation d'Euler de la fonction NAND.

IV.3. Opérateur Ou exclusif (XOR)

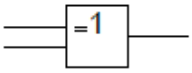
Pour **2** variables **A** et **B**, La sortie d'une porte OU-exclusif est au niveau haut seulement lorsque les deux entrées sont à des niveaux logiques différents, la fonction OU EXCLUSIF est définie par :

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

On la note également :

$$f(A,B) = A \oplus B$$

Symbole européen



Symbole américain

Table de vérité

A	B	$S = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

La représentation d'Euler de la fonction XOR est la suivante (Figure 2. 5) :

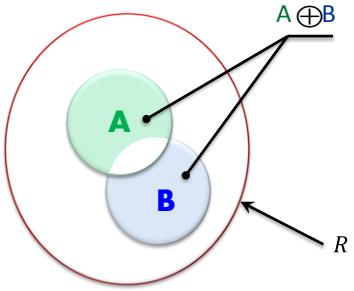
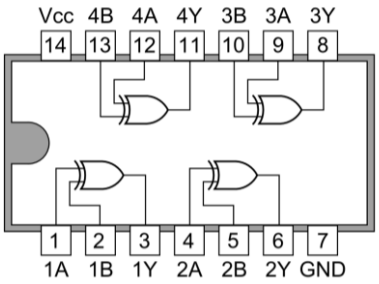


Figure 2. 5. Représentation d'Euler de la fonction XOR.

❖ ***Exemple de Portes logiques XOR***

Le circuit intégré de référence : **74LS86**



- Ce circuit comporte **14 broches** (**pin** ou **pattes**).
- L'alimentation en tension est faite entre les pattes **7** et **14**; respectivement pour la masse (**GND**) et **5V** (**VCC**).
- Il contient **4 portes** logiques XOR indépendantes ayant **2 entrées chacune**. Les entrées de chacune de ces portes XOR sont identifiées par la broche **A** et **B**, tandis que la sortie est représenté par **Y**.

IV.4. Fonction NON-OU EXCLUSIF (XNOR)

La fonction NON-OU EXCLUSIF est obtenue en complémentant la sortie d'un OU EXCLUSIF, c'est à dire en appliquant la sortie de la fonction OU EXCLUSIF à la fonction NON. Pour 2 variables A et B, elle est notée :

$$f(A,B) = \overline{A \oplus B}$$

On la note également :

$$f(A,B) = A \odot B$$

La représentation d'Euler de la fonction XNOR est la suivante (Figure 2. 6) :

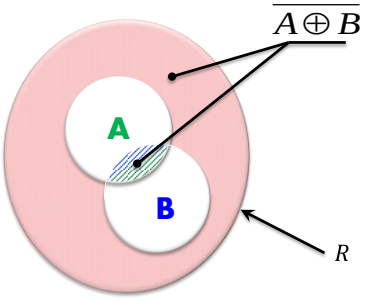
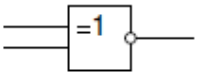


Figure 2. 6. Représentation d'Euler de la fonction XNOR.

❖ ***Remarque 2.4 :***

Pour obtenir la table de vérité du NON-OU EXCLUSIF, il suffit d'inverser les valeurs de sortie dans la table de vérité du OU EXCLUSIF.

Symbole européen



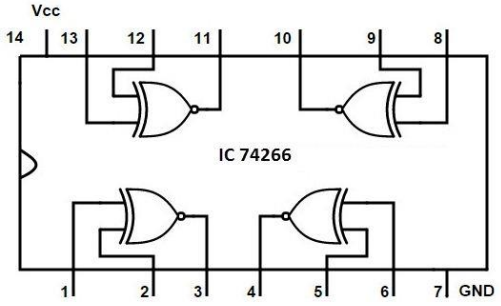
Symbole américain

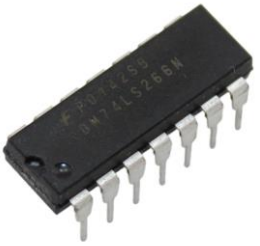
Table de vérité

A	B	$S = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

❖ ***Exemple de Portes logiques XNOR***

Le circuit intégré de référence : **74LS266**





- Ce circuit comporte **14 broches** (**pin** ou **pattes**).
- L'alimentation en tension est faite entre les pattes **7** et **14**; respectivement pour la masse (**GND**) et **5V** (**VCC**).
- Il contient **4 portes** logiques XNOR indépendantes ayant **2 entrées chacune** et une sortie.

V. Propriétés et théorèmes

V.1. Propriétés monovariables

V.1.1 Elément neutre

À chaque opérateur correspond un élément neutre qui, lorsqu'il est opéré avec une variable quelconque ***A***, donne un résultat identique à cette variable.

$$\boldsymbol{A + 0 = A} \qquad \boldsymbol{A \cdot 1 = A}$$

V.1.2 Elément nul

À chaque opérateur correspond un élément nul (appelé aussi élément absorbant) qui, lorsqu'il est opéré avec une variable quelconque ***A***, donne un résultat identique à cet élément nul.

$$\boldsymbol{A + 1 = 1} \qquad \boldsymbol{A \cdot 0 = 0}$$

V.1.3 Idempotence

Le résultat d'une opération entre une variable ***A*** et elle-même est égal à cette variable.

$$\boldsymbol{A + A = A} \qquad \boldsymbol{A \cdot A = A}$$

V.1.4 Complémentation

$$\boldsymbol{A + \overline{A} = 1} \qquad \boldsymbol{A \cdot \overline{A} = 0}$$

V.1.5 Involution

Le complément du complément d'une variable ***A*** est égal à cette variable.

$$\boldsymbol{\overline{\overline{A}} = A}$$

V.2. Propriétés multivariables

V.2.1 Associativité

Les opérations $+$, $\cdot$, et $\oplus$ sont associatives :

$$A + B + C = (A + B) + C = A + (B + C)$$
$$A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$$
$$A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C)$$

V.2.2 Commutativité

Les opérations $+$, $\cdot$, et $\oplus$ sont commutatives :

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$
$$A \oplus B = B \oplus A$$

V.2.3 Distributivité

Chacune des opérations + et $\cdot$ est distributive sur l'autre :

$$A \cdot (B + C) = A \cdot B + A \cdot C$$
$$A + B \cdot C = (A + B) \cdot (A + C)$$

❖ ***Remarque 2.5 :***

On peut remarquer que cette propriété est particulière dans l'algèbre booléenne puisqu'ici les deux expressions sont vraies, alors que seule la première l'est dans l'algèbre ordinaire.

V.2.4 Absorption

$$A + A \cdot B = A$$
$$A \cdot (A + B) = A$$

V.2.5 Dualité

Deux expressions sont dites **duales** si l'on obtient l'une en **changeant** dans l'autre, les **ET** par des **OU**, les **OU** par des **ET**, les '**1**' par des '**0**' et les '**0**' par des '**1**'.

Conséquences de Principe de dualité :

*Toute expression logique reste **vrais** si on remplace le **ET** par le **OU**, le **OU** par le **ET**, les **1** par des **0** et les **0** par des **1**.*

V.2.6 Théorème de Consensus

Le **théorème du consensus** ou la **règle du consensus** sont les identités :

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

La 2ème égalité est donc le dual de la 1ère ; on peut donc l'obtenir par application du principe de dualité.

Les termes $\boldsymbol{B \cdot C}$ et $(\boldsymbol{B + C})$ sont appelés les consensus des termes $A \cdot B$ et $\bar{A} \cdot C$ pour la 1ère expression et $(A + B)$ et $(\bar{A} + C)$ pour la 2ème.

V.2.7 Théorème de De Morgan

❶ Le complément d'un produit est égal à la somme des termes complémentés :

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

❷ Le complément d'une somme est égal au produit des termes complémentés :

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

Généralisation de théorème de De Morgane :

$$\overline{f(a, b, c, \dots, \cdot, +)} = f(\bar{a}, \bar{b}, \bar{c}, \dots, +, \cdot)$$

VI. Représentation des fonctions logiques

VI.1. Formes canonique d'une fonction

Les expressions booléennes peuvent être manipulées sous différentes formes.

Une expression est sous sa **forme canonique** si **chaque terme** de la fonction comporte **toutes les variables**. Lorsqu'une équation est écrite à partir de sa table de vérité, elle est dans sa forme canonique.

Ils existent plusieurs formes canoniques : les plus utilisées sont la **première** et la **deuxième forme**.

VI.1.1 1ère Forme Canonique ou Forme disjonctive (Sommes de Produits)

On appelle «**minterme**» de **n** variables, l'un des **produits** de ces variables ou de leurs complémentaires.

EXEMPLE 2.1 :

La table suivante donne les **mintermes** d'une fonction de deux variables :

		m_0	m_1	m_2	m_3
A	B	$\bar{A} \cdot \bar{B}$	$\bar{A} \cdot B$	$A \cdot \bar{B}$	$A \cdot B$
0	**0**	**1**	0	0	0
0	**1**	0	**1**	0	0
1	**0**	0	0	**1**	0
1	**1**	0	0	0	**1**

Une fonction booléenne peut être représentée sous forme d'une **somme de produits** (forme disjonctive) utilisant les mintermes. Ces mintermes sont représentés par des '**1**' dans une table de vérité.

EXEMPLE 2.2 :

$$f(A,B,C) = \underbrace{\bar{A} \cdot \bar{B} \cdot C}_{Minterme} + \underbrace{\bar{A} \cdot B \cdot C}_{\dots} + \underbrace{A \cdot B \cdot C}_{\dots}$$

$$f(A,B,C) = \sum m(1,3,7)$$

VI.1.2 2ème Forme Canonique ou Forme conjonctive (Produits de Sommes)

On appelle «**maxterme**» de **n** variables, l'une des **sommes** de ces variables ou de leurs complémentaires.

Une fonction booléenne peut être représentée sous forme d'un **produit de sommes** (forme **conjonctive**) utilisant les maxtermes. Ces maxtermes sont représentés par des '**0**' dans une table de vérité.

EXEMPLE 2.3 :

La table suivante donne les **maxtermes** d'une fonction de deux variables :

		M_0	M_1	M_2	M_3
A	B	$A + B$	$A + \bar{B}$	$\bar{A} + B$	$\bar{A} + \bar{B}$
0	**0**	**0**	1	1	1
0	**1**	1	**0**	1	1
1	**0**	1	1	**0**	1
1	**1**	1	1	1	**0**

EXEMPLE 2.4 :

$$f(A,B,C) = \underbrace{(A+B+C)}_{Maxterme} \cdot \underbrace{(\bar{A}+B+C)}_{\dots} \cdot \underbrace{(\bar{A}+\bar{B}+C)}_{\dots}$$

$$f(A,B,C) = \prod M(0,4,6)$$

VI.1.3 Représentation d'une fonction sous forme de mintermes et maxtermes

Une fonction logique peut être représentée sous :

❶ Sa 1ère forme canonique (Σ de mintermes) : on développe la fonction sous la forme d'une somme de produits (SDP) puis on prend chaque terme avec pour variable manquante **X** et on applique un ET logique avec $\boldsymbol{X + \bar{X}}$;

❷ Sa 2ème forme canonique (Π de maxtermes) : on développe la fonction sous la forme d'un produit de sommes (PDS) puis on prend chaque terme avec pour variable manquante **X** et on applique un OU logique avec $\boldsymbol{X.\bar{X}}$.

EXEMPLE 2.5 :

1. $f(A,B) = A + B$

$$f(A,B) = A \cdot (B+\bar{B}) + B \cdot (A+\bar{A})$$
$$= A \cdot B + A \cdot \bar{B} + B \cdot A + B \cdot \bar{A})$$
$$= \bar{A} \cdot B + A \cdot \bar{B} + A \cdot B$$
$$f(A,B) = \sum m(1,2,3)$$

2. $f(A,B,C) = A \cdot B + C$

$$f(A,B,C) = (A+C) \cdot (B+C)$$
$$= (A+C+B \cdot \bar{B}) \cdot (B+C+A \cdot \bar{A})$$
$$= (A+C+B) \cdot (A+C+\bar{B}) \cdot (B+C+A) \cdot (B+C+\bar{A})$$
$$f(A,B,C) = (A+B+C) \cdot (A+\bar{B}+C) \cdot (\bar{A}+B+C)$$
$$f(A,B,C) = \prod M(0,2,4)$$

VI.1.4 Décomposition de Shannon

Lorsque ***Shannon*** introduisit l'algèbre en vue d'une utilisation dans le cadre des circuits à relais, il ajouta une notion importante, connue aujourd'hui sous le nom de la ***décomposition de Shannon***. La décomposition de Shannon est très utile pour *la simplification des fonctions logiques*, et elle nous servira pour mieux comprendre le fonctionnement de certains circuits usuels vus plus loin dans ce cours.

La décomposition de Shannon s'énonce comme suit :

Soient $\boldsymbol{f}$ une fonction logique de $\boldsymbol{n}$ paramètres $\boldsymbol{x_0, x_1, \dots x_{n-1}}$ alors :

$$f(\boldsymbol{x_0, x_1, \dots x_{n-1}}) = \overline{\boldsymbol{x_0}}.f(\boldsymbol{0, x_1, \dots x_{n-1}}) + \boldsymbol{x_0}.f(\boldsymbol{1, x_1, \dots x_{n-1}})$$

Notons ici que la variable « x_0 » peut être remplacée par n'importe quelle autres variable parmi les n variables restantes. La décomposition peut également être appliquée récursivement sur l'ensemble des variables. De plus, en appliquant le principe de ***dualité***, on peut trouver que, pour toute fonction logique f de de n paramètres $x_0, x_1, \ldots x_{n-1}$, il est possible d'écrire :

$$f(x_0, x_1, \ldots x_{n-1}) = (\overline{x_0} + f(1, x_1, \ldots x_{n-1})).(x_0 + f(0, x_1, \ldots x_{n-1}))$$

VI.2. Représentations tabulaires

VI.2.1 Table de vérité

Une table de vérité est l'écriture des valeurs d'une fonction logique pour toutes les combinaisons possibles de ses variables. Chaque ligne présente la combinaison des variables d'entrée ainsi que la ou les sorties correspondante(s).

EXEMPLE 2.6 :

On définit la fonction logique $f(A, B, C) = 1$ si $(A\,B\,C)_2 > 4$. La table de vérité correspondante est :

A	B	C	$f(A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Le principal inconvénient de la table de vérité est qu'elle devient rapidement très encombrante lorsque le nombre de variables d'entrée augmente.

VI.2.2 Diagramme de Karnaugh et termes adjacents

Quand une équation logique est établie, il faut la simplifier si possible car ceci diminue le nombre de circuits électroniques à utiliser. Cette simplification s'effectue à l'aide de l'algèbre de Boole ou d'un diagramme de Karnaugh. Ce diagramme représente l'état des variables à l'intérieur de cases, pour n variables il y a 2^n cases, donc pour 2, 3 et 4 variables il faut 4, 8 et 16 cases, pour 5 variables il faut deux diagrammes de 16 cases.

Deux termes sont adjacents quand ils ne diffèrent l'un de l'autre que par une seule variable (ABC et $AB\bar{C}$ sont adjacents). Un diagramme – ou tableau – de Karnaugh est une table d'implication logique disposée de telle manière que deux termes logiquement adjacents soient également adjacents géométriquement. Afin d'exploiter la notion ***d'adjacence*** ente les termes, les cases doivent être ordonnées selon le ***code binaire réfléchi***, au lieu du code binaire naturel.

❖ ***Remarque 2.6 :***

Les tableaux de Karnaugh se présentent comme des cylindres fermés dans les deux sens (Figure 2. 7).

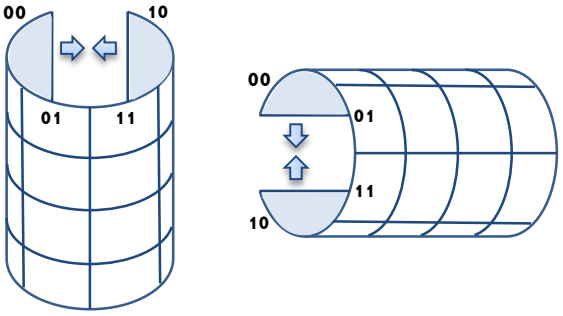


Figure 2. 7. Représentation cylindrique d'un tableau de Karnaugh à 4 variables.

EXEMPLE 2.7 :

Tableau de Karnaugh d'une fonction à **3 variables** :

A	B	C	$f(A,B,C)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

→

f	C		B	
	0	1	0	1
A	1	0	1	0

VI.3. Représentations graphiques

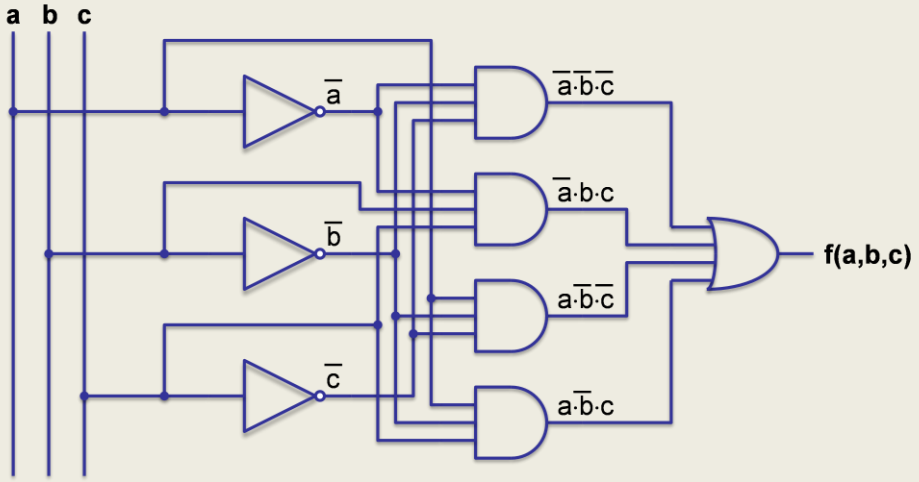
VI.3.1 Logigramme

Un logigramme est un schéma illustrant l'expression d'une fonction logique sans tenir compte des constituants technologiques. Le principe consiste à remplacer chaque opérateur logique par la porte logique qui lui correspond.

EXEMPLE 2.8 :

Soit la fonction logique $f(a,b,c) = \bar{a}\bar{b}\bar{c} + \bar{a}bc + a\bar{b}\bar{c} + a\bar{b}c$

Le Logigramme ***f*** est :

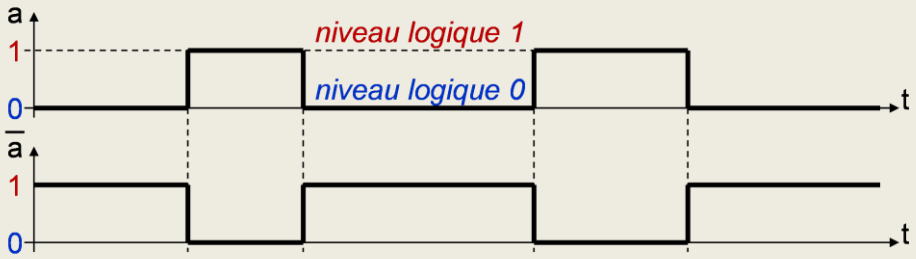


VI.3.2 Chronogramme

C'est le graphe d'évolution temporelle des variables et des fonctions logiques.

EXEMPLE 2.9 :

Le chronogramme de la fonction **NON**



VII. Simplification des fonctions logiques

La simplification d'une fonction logique est son écriture sous forme d'une expression contenant le minimum de lettres et de termes.

VII.1. Simplification algébrique

On réalise cette simplification en utilisant l'ensemble des propriétés et théorème de l'algèbre de Boole et en particulier l'absorption et le théorème de consensus.

VII.1.1 Simplification par mise en facteur commun

On peut faire cette simplification si on a une variable dans un terme et son inverse dans l'autre et si le reste des variables est **identique**.

EXEMPLE 2.10 :

$$\begin{aligned} f &= \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}\bar{d} + \bar{a}bcd + ab\bar{c}d + abcd + ab\bar{c}\bar{d} \\ &= \bar{a}\bar{c}\bar{d}(\bar{b}+b) + \bar{a}bd(c+\bar{c}) + abd(c+\bar{c}) + ab\bar{c}\bar{d} \\ &= \bar{a}\bar{c}\bar{d} + \bar{a}bd + abd + ab\bar{c}\bar{d} \\ &= \bar{a}\bar{c}\bar{d} + bd + ab\bar{c}\bar{d} \end{aligned}$$

VII.1.2 Simplification par Consensus et absorption

On peut faire cette simplification si les deux termes n'ont pas le même nombre de variables et s'il y a une variable dans un terme et sont inverse dans l'autre.

EXEMPLE 2.11 :

$$f = \bar{a}\bar{b} + \bar{a}bc = \bar{a}\bar{b} + \bar{a}bc + \underbrace{\bar{a}c}_{\text{Terme Consensus}}$$

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

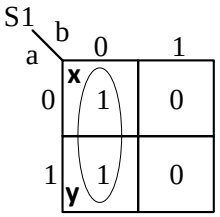
VII.2. Simplification par tableau de Karnaugh

VII.2.1 Principe

Le diagramme de Karnaugh est un outil graphique qui permet de simplifier une équation. Soit une fonction définie par un tableau de Karnaugh, on peut simplifier la fonction en effectuant de groupement des cases adjacentes contenant la valeur **1** (ou encore la valeur **0**).

VII.2.2 Groupement de 2 cases adjacentes

Prenons l'exemple de la fonction S1 :



Les cases **x** et **y** sont adjacentes :

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

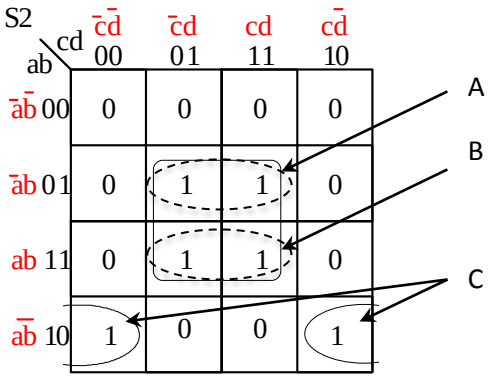
$y = a\,\bar{b}$

⇨ $x + y = \bar{a}\,\bar{b} + a\bar{b} = (\bar{a} + a)\bar{b} = \bar{b}$ d'où $\boldsymbol{S_1 = \bar{b}}$

Le groupement de **deux cases** adjacentes contenant la valeur 1 correspond à deux termes qui différent d'une variable complémentée dans un terme et non complémentée dans l'autre. Le terme résultant du groupement ne comporte pas cette variable qui change d'état.

VII.2.3 Groupement de 4 cases adjacentes

Prenons l'exemple de la fonction $\mathbf{S_2}$:



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

$B = ab\,(\bar{c}d + cd) = abd$

$C = a\bar{b}\bar{d}$

⇒ $A + B = bd(\bar{a} + a) = bd$

d'où $\boldsymbol{S_2 = A + B + C = bd + a\bar{b}\bar{d}}$

Le groupement de **4 cases** adjacentes contenant la valeur 1 conduit à un terme réduit dans lequel deux variables disparaissent.

D'une manière générale, le **groupement** de $\mathbf{2^n}$ (2, 4, 8, 16, …) cases adjacentes conduit à un terme réduit dans lequel **n variables disparaissent**.

❖ ***Principe de la simplification graphique :***

La simplification graphique consiste à faire apparaître sur le tableau de Karnaugh des groupements en **puissance de 2**, aussi ***importants*** que possible, de cases adjacentes contenant la valeur '**1**'. Une même case peut faire partie de plusieurs groupements. L'écriture simplifiée de la fonction est la somme des termes engendrés par chaque groupement.

❖ ***Remarque 2.7 :***

- L'adjacence existe aussi sur les extrémités de tableau :

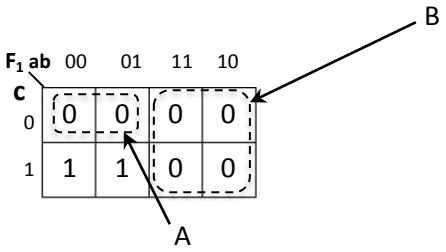
S ab \ cd	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

$S = \bar{b}\bar{c}$

S ab \ cd	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$S = \bar{b}\bar{d}$

- Pour représenter la fonction sous forme de produits de sommes, on procède par groupement des '**0**' :



$$A = (a+b+c)(a+\bar{b}+c) = (a+c) \qquad B = (\bar{a}+c)(\bar{a}+\bar{c}) = \bar{a}$$

$$\Rightarrow \quad AB = \bar{a}(a+c)$$

d'où $\boldsymbol{F_1 = AB = \bar{a}c}$

VII.2.4 Fonctions incomplètement définies

Dans des cas pratiques, certains combinaisons de variables n'ont aucun sens physique et n'apparaissent jamais dans la réalité. Il est donc inutile de spécifier la valeur de la fonction pour de telles combinaisons.

Dans ce cas, le concepteur peut à sa convenance attribuer à ces cases la valeur **0** ou **1** de manière à obtenir le maximum de groupements.

EXEMPLE 2.12 : *F* est définie par ce tableau de Karnaugh :

ab \ cd	00	01	11	10	
00	1	0	0	0	$\bar{a}\bar{c}\bar{d}$
01	1	1	Φ	0	bd
11	0	Φ	1	0	$a\bar{b}\bar{d}$
10	Φ	Φ	0	1	

$$F = \bar{a}\bar{c}\bar{d} + a\bar{b}\bar{d} + bd$$

Chapitre 2

Les Circuits Combinatoires

I. Introduction

Un circuit combinatoire possède un certain nombre d'entrées et un certain nombre de sorties. Les sorties sont reliées aux entrées par des fonctions logiques. L'aspect temporel n'intervient pas, contrairement aux circuits logiques séquentiels.

Les circuits combinatoires sont établis à partir d'une opération appelée synthèse combinatoire. Cette synthèse est définie comme étant la traduction d'une fonction logique, à partir d'un cahier des charges, en un schéma. Diverses méthodes de synthèse sont possibles ; elles diffèrent sur la forme de la fonction utilisée (canonique ou simplifiée), sur le type des opérateurs ou des circuits intégrés choisis, et sur la technique de découpage fonctionnel employée.

Dans cette partie, nous allons étudier quelques grandes circuits combinatoires couramment utilisées.

II. Additionneur

Nous allons dans cette section voir comment construire un circuit pour l'addition de 2 nombres en binaire. Ce circuit étant assez complexe, nous allons le réaliser en plusieurs étapes :

- Le demi-additionneur fera une simple addition de deux bits.
- L'additionneur complet devra ajouter à cette addition celle d'un report précédent.
- Enfin nous assemblerons n additionneurs pour faire l'addition de nombres de n bits.

II.1. Demi-Additionneur (Half Adder)

Le demi-additionneur effectue la somme de deux bits. ***S*** est la **somme** et ***R*** le **report** (carry). Le demi-additionneur ne tient pas compte d'une retenue antérieure.

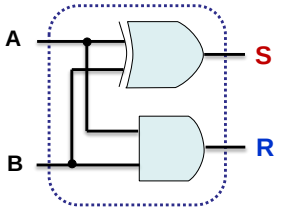
Table de vérité :

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

Equations de sortie :

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

Logigramme :



Ce schéma n'est cependant pas suffisant pour réaliser la somme de nombres de plusieurs bits. Car il ne prend pas en compte une éventuelle retenue provenant du résultat de l'addition des 2 bits de rang directement inférieur.

On voit bien que l'addition arithmétique sur **1 bit** s'apparente au **OU Exclusif**.

II.2. Additionneur complet (Full Adder)

II.2.1 Addition complète sur 1 bit

Pour tenir compte du report précédent, il faut prévoir un circuit avec **3** entrées et **2** sorties.

Un additionneur complet comporte donc **3** entrées : les deux bits à additionner a_i et b_i, et la retenue issue de l'addition de deux bits de rang inferieurs (dite entrante), r_{i-1} .

Il possède **2** sorties : la somme S et la retenue sortante R_i .

Table de vérité :

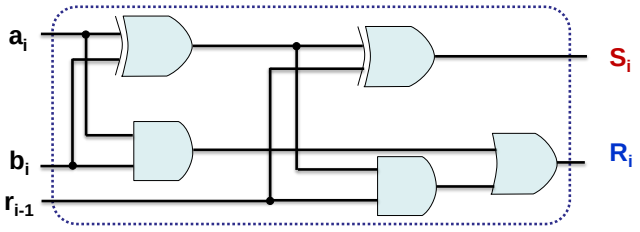
a_i	b_i	r_{i-1}	R_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Equations de sortie :

$$S_i = a_i \oplus b_i \oplus r_{i-1}$$

$$R_i = a_i \cdot b_i + r_{i-1}(a_i \oplus b_i)$$

Logigramme :



❖ ***Remarque 3.1 :***

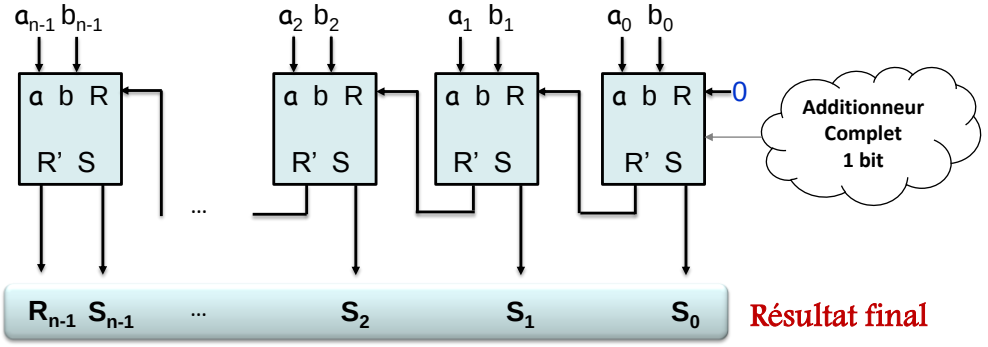
Cette structure montre la possibilité de réaliser un additionneur complet à partir de deux demi-additionneurs et d'une porte "**OR**".

L'additionneur complet est le circuit de base pour effectuer la somme de nombres de plusieurs bits.

On peut représenter ce circuit sous la forme d'une boite noire :

II.2.2 Addition de deux nombre binaires de n bits

L'addition de deux mots de **n bits** nécessite **n additionneurs**. La retenue se propage des éléments binaires de poids le plus faible vers les éléments binaires de poids le plus fort. Le schéma suivant présente un exemple d'un additionneur de mots de **4 bits** :



L'entrée de retenue du premier additionneur (R_{-1}) est mise à **0**. La sortie de retenue du dernier additionneur (R_3).

Cette architecture est intéressante d'un point de vue matériel car elle est répétitive. Par contre, le résultat obtenu dépend du nombre d'additionneurs donc de la taille des mots à additionner. La retenue R_0 est délivrée après la première addition et ainsi de suite.

III. *Soustracteur*

III.1. Principe

Jusqu'à maintenant, nous n'avons traité que les nombres positifs. On pourrait imaginer de traiter les nombres négatifs en ajoutant un signe moins devant ; comme ce qu'on fait d'habitude pour les décimaux mais malheureusement l'informatique, qui ne connaît que les **1** et les **0**, traite les nombres négatifs différemment.

On peut en revanche penser à transformer l'opération de soustraction en une simple opération d'addition binaire, en utilisant un codage en ***complément à 2*** pour les nombres négatifs.

III.2. Complémentation à 1 (CA_1)

En décimal, on peut former **le complément à 9** d'un nombre quelconque en remplaçant chaque chiffre de ce nombre par sa différence avec 9.

Exemple. *Le complément à **9** de **16** est **83**.*

En binaire, on forme **le complément à 1** d'un nombre en remplaçant chaque chiffre de ce nombre par sa différence avec **1** en remplaçant les **1** par des **0** et réciproquement.

Exemple. *Le complément à **1** de **10100** est **01011**.*

III.3. Complémentation à 2 (CA_2)

Pour former le ***complément à 10*** d'un nombre décimal, on remplace le chiffre des unités par sa différence avec ***10*** et les autres chiffres de ce nombre par leur différence avec **9**. On peut déduire donc que le ***complément à 10*** d'un nombre s'obtient en ajoutant ***1*** au complément à ***9***.

Exemple. *Le complément à **10** de **16** est **84**.*

On va utiliser la règle ci-dessus pour former le ***complément à 2*** d'un nombre **binaire** en ajoutant 1 au complément à 1.

Exemple. *Le complément à **2** de **10100** est **01011+1=01100**.*

III.4. Soustraction par complémentation à 2

Pour faire la soustraction, on fait appel au ***complément à 2*** pour les nombres négatifs afin de pouvoir transformer l'opération de soustraction en une addition :

$$\boldsymbol{A - B = A + (-B)}$$
$$\boldsymbol{= A + CA_2(B)}$$

III.4.1 Soustraction décimale

Soit à faire la **soustraction décimale** suivante **574 – 391** (= **183**)

On a **391 = 1000 – 609**, donc **609** est le **complément à 10** de **391,** on peut écrire ainsi **574 - 391** = **574 -** (1000 **- 609**) = **574** + **609** – 1000 = **183.**

L'opération qu'on a effectuée s'écrit donc comme suit :

A – B = A + CA_{10}(B) – (puissance de 10 immédiatement supérieure à A)

Remarque importante :

Au lieu de soustraire 1000, il aurait suffi de négliger la dernière retenue dans le résultat.

En effet **574** + **609** = **1183**= **183** si on néglige le **1 de gauche**

III.4.2 Soustraction binaire

Pour maintenant effectuer l'opération de soustraction en binaire, on suit la même procédure décrite ci-dessus. On aura donc :

A – B = A + CA_2(B) – (puissance de 2 immédiatement supérieure à A)

Exemple. **1110 - 1001 = 1110 + 0111 = 10101 = 0101**

On oublie ce 1 en trop

IV. Comparateur

Les comparateurs logiques dits aussi circuits d'identification permettent de comparer deux nombres ***A*** et ***B*** de **n** bits. En général, le résultat de la comparaison est fourni sur **3 sorties** :

$$\boldsymbol{A = B \Rightarrow S_{=} = 1},$$
$$\boldsymbol{A > B \Rightarrow S_{>} = 1},$$
$$\boldsymbol{A < B \Rightarrow S_{<} = 1}.$$

Deux nombres $\boldsymbol{A} = \boldsymbol{a_{n-1}} \ldots \boldsymbol{a_1 a_0}$ et $\boldsymbol{B} = \boldsymbol{b_{n-1}} \ldots \boldsymbol{b_1 b_0}$ sont égaux si tous les bits du même poids sont égaux.

IV.1. Comparateur élémentaire de deux nombres de 1 bits

Etudions un circuit de comparaison entre deux bits :

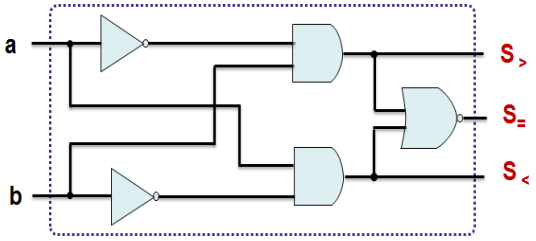
Table de vérité :

a	b	$S_>$	$S_=$	$S_<$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Equations de sortie :

$$S_> = a.\bar{b}$$
$$S_< = \bar{a}b$$
$$S_= = \bar{a}\bar{b} + ab = \overline{a \oplus b} = \overline{S_> + S_<}$$

Logigramme :

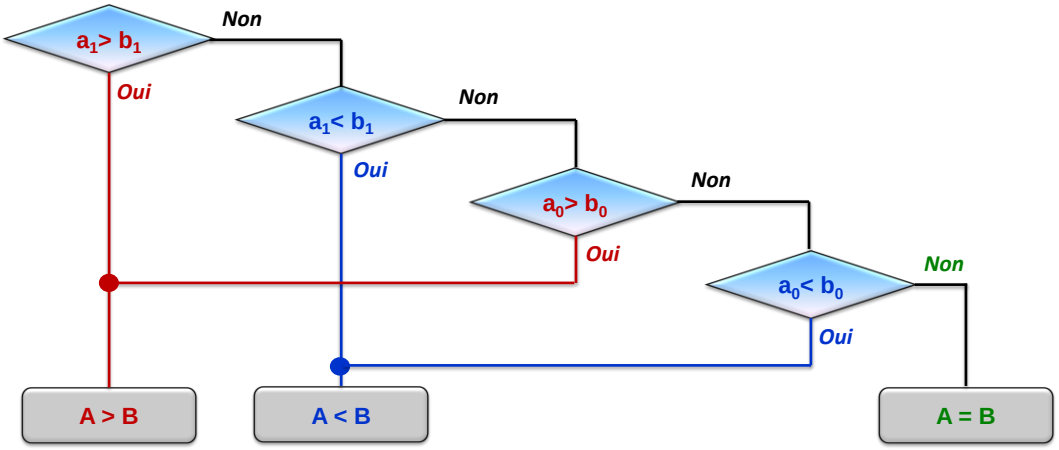


IV.2. Comparateur de deux nombres de n bits

❖ ***Principe et organigramme :***

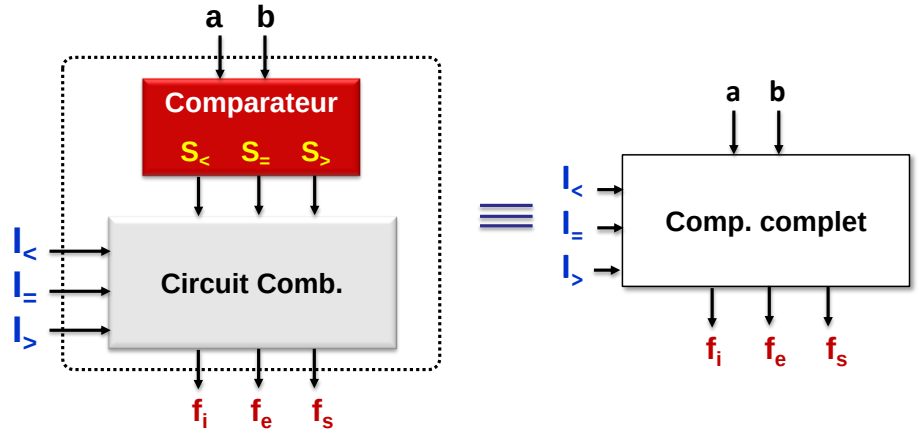
Prenant l'exemple de deux nombres ***A*** et ***B*** de **deux** bits : $\boldsymbol{A = a_1 a_0}$; $\boldsymbol{B = b_1 b_0}$

La démarche de comparaison est la suivante :



On commence par comparer les bits de **poids forts** et on ne passe aux bits de poids Inférieur qu'en cas d'égalité.

La **cellule de base** de comparaison doit donc disposer d'entrées permettant la prise en compte du résultat de la comparaison des bits de **poids inférieur**.



- **$I_<$; $I_=$** et **$I_>$** : **Entrées** recevant le résultat de la comparaison des bits de **poids inférieur**.
- D'après **l'organigramme**, les entrées **$I_<$; $I_=$** et **$I_>$** ne sont prises en compte qu'on cas d'égalité des bits de poids supérieur (**$S_= = 1$).** Dans ce cas leur état est directement transmis vers les sorties **f_i ; f_e** et **f_s**.

Table de vérité :

a	b	$S_>$	$S_=$	$S_<$	$I_>$	$I_=$	$I_<$	f_S	f_e	f_i
1	0	**1**	0	0	X	X	X	**1**	0	0
0	1	0	0	**1**	X	X	X	0	0	**1**
0	0				**1**	0	0	**1**	0	0
		0	**1**	0	0	**1**	0	0	**1**	0
1	**1**				0	0	**1**	0	0	**1**

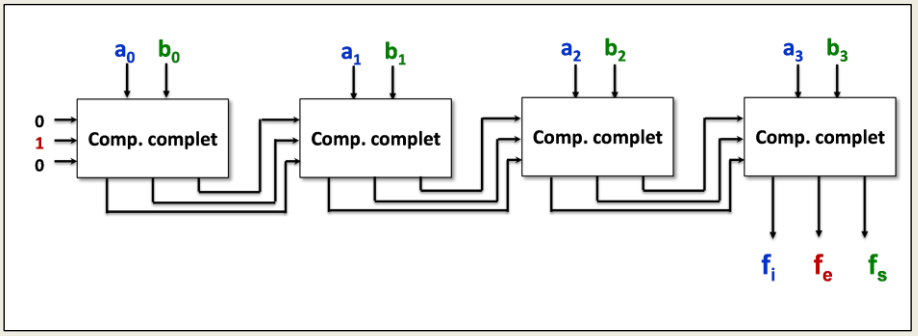
- A partir de la table de vérité, on déduit les **équations de sorties** : **f_s ; f_e** et **f_i**.

$$\begin{cases} f_s = S_\rangle + S_= I_\rangle \\ f_e = S_= I_= \\ f_i = S_\langle + S_= I_\langle \end{cases}$$

EXEMPLE 3.1 :

Comparaison de deux nombres de 4 bits

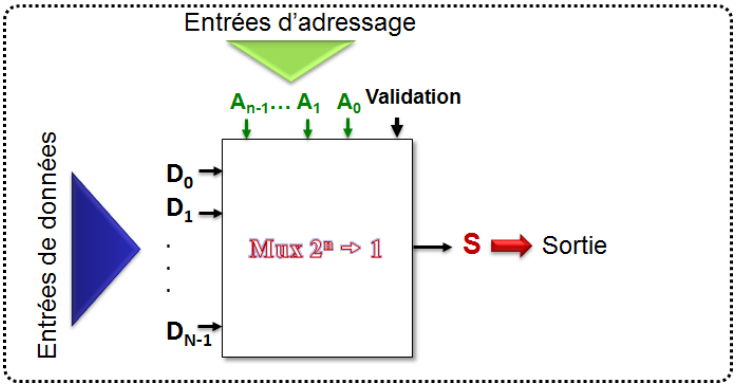
Le comparateur **4 bits** sera réalisé par la mise **en cascade** de **4 comparateurs de 1 bit**. Le résultat de la comparaison est recueilli sur la sortie du dernier comparateur :



V. Multiplexeur/ Démultiplexeur

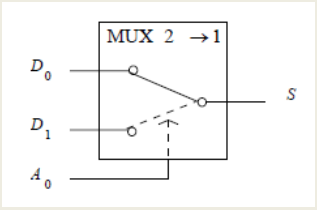
V.1. Multiplexeur

Un multiplexeur (Mux) est un circuit à 2^n **entrées** d'informations, n entrées de sélection, et **une sortie unique**. Il permet l'aiguillage (par la commande de **n entrées d'adresse**) de l'une de ces entrées vers la sortie.



- La relation entre le nombre des entrées **de données** et des entrées **d'adressage** est : **N=2^n**

EXEMPLE 3.2 : Multiplexeur 2 → 1 : Table de vérité : Equation de S :



A_0	S
0	D_0
1	D_1

$$S = D_0 \quad si \quad A_0 = 0$$
$$S = D_1 \quad si \quad A_0 = 1$$

De façon générale, la sortie d'un multiplexeur à ***n*** entrées d'adresses s'exprime en fonction des entrées de données D_i et des mintermes m_i sur les entrées d'adresses :

$$S = \sum_{i=0}^{2^n-1} D_i m_i$$

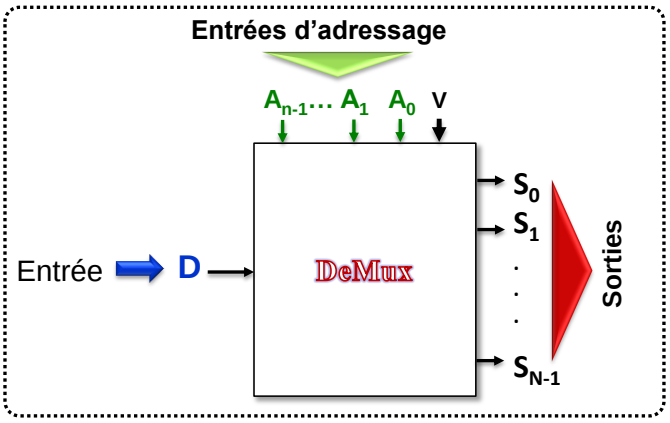
V.2. Démultiplexeur

Il joue le rôle inverse d'un multiplexeur, il permet de faire passer **une donnée** dans **l'une** des **2^n** sorties selon les valeurs des entrées de commandes ou d'adresses (**n** entrées d'adresses).

Le module sélection ou adressage joue presque le même rôle que dans le **Mux**. Il permet de **sélectionner la sortie** qui doit **recevoir l'information** de l'entrée.

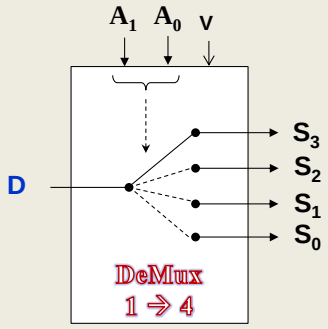
Un **DeMux** possède :

- une **seule** entrée
- **N=2^n** sorties
- **n** entrées de sélection (commandes)



EXEMPLE 3. 3 :

Exemple d'un DeMux (1 → 4)



V	A_1	A_0	S_3	S_2	S_1	S_0
0	X	X	0	0	0	0
1	0	0	0	0	0	D
1	0	1	0	0	D	0
1	1	0	0	D	0	0
1	1	1	D	0	0	0

$$S_0 = V.\overline{A_1}.\overline{A_0}.D \quad S_2 = V.A_1.\overline{A_0}.D$$
$$S_1 = V.\overline{A_1}.A_0.D \quad S_3 = V.A_1.A_0.D$$

V.3. Applications des multiplexeurs

V.3.1 Générateur de fonctions

Toute fonction logique peut être réalisée à partir des **MUX**. Les entrées de sélection (**commande**) sont alors les **variables de la fonction**.

V.3.2 Conversion parallèle ↔ série

Considérons un mot de **n bits**, il peut être transmis soit sur un fil unique, bit après bit (transmission série), soit sur plusieurs fils à la fois, un fil par bit (transmission parallèle).

Conversion parallèle → série : elle est effectuée à l'aide d'un multiplexeur : on envoie en entrée les **n bits** du mot à transmettre, et en même temps, on fait varier les bits d'adresse en les incrémentant. En sortie on obtient **la série** des **n bits** du mot.

Conversion série → parallèle : elle est effectuée à l'aide d'un démultiplexeur. On envoie en entrée successivement les **n bits** du mot, et en même temps, on fait varier les bits d'adresse en les incrémentant. En sortie, les fils doivent être reliés à une mémoire, qui stocke l'un après l'autre les bits du mot.

VI. Décodeur, Codeur, transcodeur

VI.1. Décodeur

Un **décodeur** est un circuit **logique combinatoire** qui a une **entrée binaire** de **n bits** et **2^n sorties**.

Pour chaque **combinaison d'entrée**, une **seule ligne de sortie est activée à la fois**.

❖ **Principe d'un décodeur (2 → 4) :**

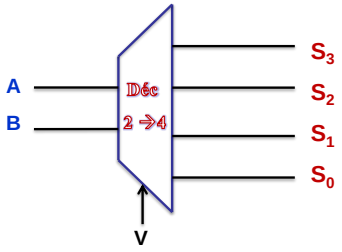


Table de vérité :

V	A	B		S_3	S_2	S_1	S_0
0	X	X		0	0	0	0
1	0	0		0	0	0	1
1	0	1		0	0	1	0
1	1	0		0	1	0	0
1	1	1		1	0	0	0

Equations de sorties :

$$S_0 = (\overline{A}.\overline{B}).V$$
$$S_1 = (\overline{A}.B).V$$
$$S_2 = (A.\overline{B}).V$$
$$S_3 = (A.B).V$$

❖ ***Remarque 3.2 :***

La plupart des décodeurs sont dotés d'une ou plusieurs entrées de **validation** (**V**) qui commandent son fonctionnement.

VI.2. Codeur

Un codeur est un circuit à **2^n entrées** et **n sorties** qui code en binaire le rang de la **seule entrée active**.

❖ **Principe d'un codeur (4 → 2) :**

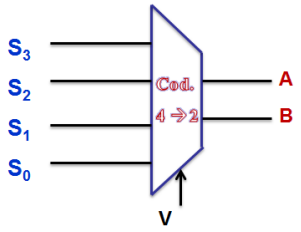


Table de vérité :

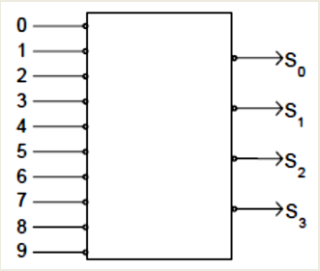
v	S_3	S_2	S_1	S_0		A	B
0	x	x	x	X		0	0
1	0	0	0	1		0	0
1	0	0	1	0		0	1
1	0	1	0	0		1	0
1	1	0	0	0		1	1

Equations de sorties :

$$A = \left(\overline{S_1}.\overline{S_0}(S_3 \oplus S_2)\right).V$$
$$B = \left(\overline{S_2}.\overline{S_0}(S_3 \oplus S_1)\right).V$$

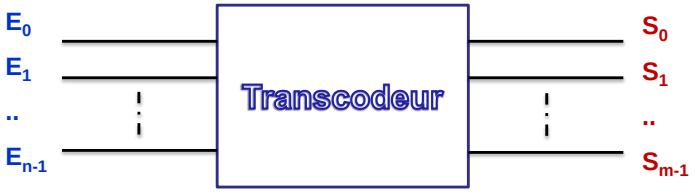
EXEMPLE 3. 4 :

Comme exemple, on peut penser au codeur **décimal → BCD**



VI.3. Transcodeur

Le **transcodeur** désigne l'ensemble des ***codeurs***, ***décodeurs*** ou encore ***convertisseur*** de codes. Ces circuits combinatoires permettent de **transformer** une **information** présentée à **l'entrée** sous forme d'un **code X** (**sur n bit**) en la même information sous un **code Y** (sur **m bit**) en **sortie**.



- Un **codeur** est un **transcodeur** avec **2^n entrée et n sorties**.
- Un **décodeur** est un **transcodeur** avec **n entrée et 2^n sorties**.
- Un **transcodeur** est un circuit de transcodage de **n entrées** vers **m sorties**.

❖ Etapes de réalisation d'un transcodeurs :

A partir d'un cahier des charges on établit :

❶ **TV** pour extraire les relations entre les **sorties** et les **entrées (définition des fonctions) ;**
❷ **Simplification** des fonctions obtenues ;
❹ Réalisation des **logigrammes** ;
❹ La **conception** des circuits à l'aide des techniques disponibles.

EXEMPLE 3. 5 :

Code binaire pur ➔ code Gray (2 bits)

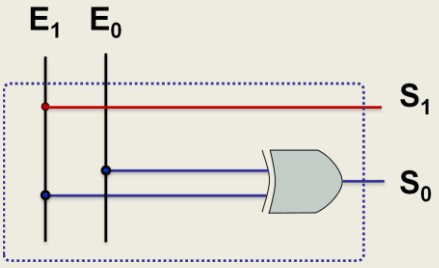
Table de vérité :

E_1	E_0		S_1	S_0
0	0		0	0
0	1		0	1
1	0		1	1
1	1		1	0

Equations de sortie :

$$S_1 = E_1$$
$$S_0 = E_1 \oplus E_0$$

Logigramme :



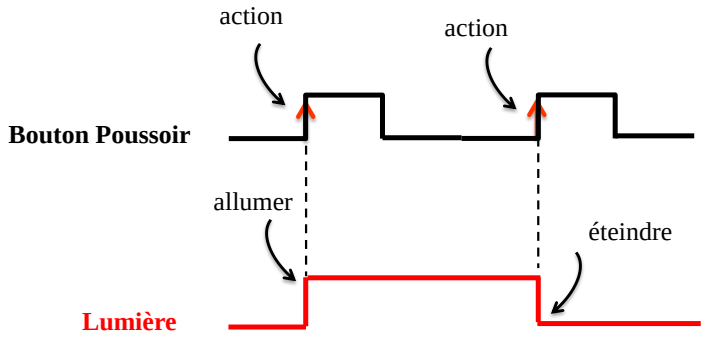
Chapitre 3

Les Circuits Séquentiels

I. Introduction

Un circuit combinatoire est un circuit numérique dont les sorties dépendent uniquement des entrées. La différence essentielle entre les systèmes combinatoires que nous avons étudié dans le chapitre précédent et les systèmes séquentiels que nous allons aborder dans ce chapitre, réside dans le fait que la fonction de sortie de ces derniers systèmes dépend à la fois des variables d'entrée et de l'état antérieur des sorties. Or ce dernier dépendait de l'état des entrées précédentes, par conséquence de l'ordre dans lequel se succèdent les états d'entrée, d'où la terminologie « séquentielle » attribuée à ces circuits.

A titre d'illustration, prenons l'exemple très simple d'une lampe électrique commandée par un **bouton poussoir** de telle façon qu'une action sur le bouton allume la lampe, et qu'une action successive l'éteigne, selon l'exemple du chronogramme suivant :



On note bien que l'évolution de ce système dépend non seulement de la position du bouton poussoir à un instant donné, mais aussi du fait que la lampe soit allumée ou non. Le système dépend donc de l'état précédent, car il conserve la mémoire de l'action précédente ; un dispositif de mémoire à maintien la lampe allumée. C'est la caractéristique essentielle d'un système séquentiel.

Les fonctions séquentielles de base sont :

La mémorisation ; le comptage ; le décalage.

Les circuits séquentiels fondamentaux sont :

Les bascules (3 types) ; les compteurs ; les registres ; les RAM (Random AccessMemory).

Ces circuits peuvent travailler soit en mode synchrone, soit en mode asynchrone.

II. Système asynchrones / synchrones

II.1. Systèmes asynchrones

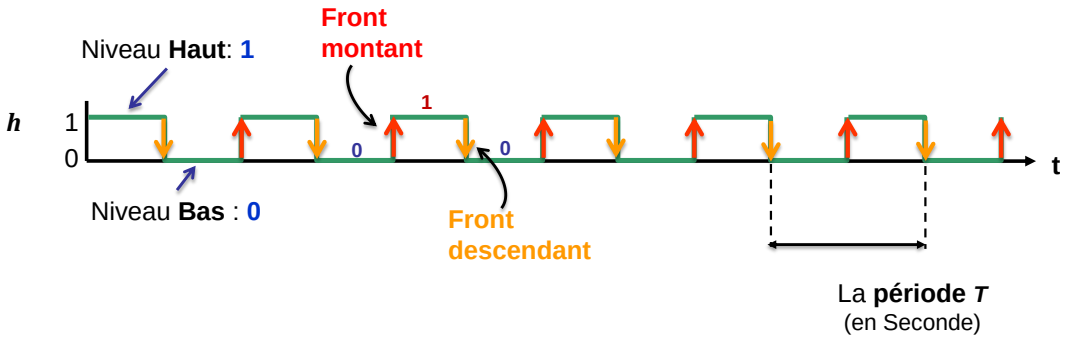
Les signaux d'entrée de ces systèmes peuvent provoquer à tout moment le changement d'état des sorties (après un certain retard qu'on appelle « temps de réponse ». Ces systèmes sont difficiles à concevoir et à dépanner.

II.2. Systèmes synchrones

Le changement d'état sur les sorties se produit pendant une transition appelé « front » (montant ou descendant) d'un signal maître, l'horloge. Les entrées servent à préparer le changement d'état, mais ne provoquent pas de changement des sorties. Tout changement d'état interne du montage est synchronisé sur le front actif de l'horloge.

La majorité des systèmes numériques séquentiels sont synchrones même si certaines parties peuvent être asynchrone (**ex.** : reset).

Une **horloge** est 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 synchrone. L'horloge est notée par **h** ou **ck** (**clock**).



III. Les Bascules

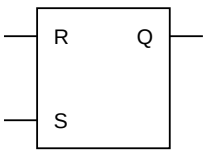
III.1. Bascule RS

La bascule RS **asynchrone** possède une entrée **R** (**Reset**) de mise à zéro, une entrée **S** (**Set**) de mise à 1 et une sortie **Q**.

Table de fonctionnement :

R	S	Q^+	
0	0	Q	Mémoire
0	1	1	Mise à 1
1	0	0	Mise à 0
1	1	Φ	Interdit

Symbole :



L'état $\boldsymbol{R = S = 0}$ maintient l'état de la sortie (mode **mémoire**).

L'état $\boldsymbol{R = S = 1}$ est interdit (mode **interdit**), car il conduit à mettre simultanément la sortie à **1** et à **0**.

III.1.1 Réalisation à base des portes NAND

Table de vérité :

R	S	Q	Q^+	
0	0	0	0	Mémoire
0	0	1	1	
0	1	0	1	Mise à 1
0	1	1	1	
1	0	0	0	Mise à 0
1	0	1	0	
1	1	0	Φ	Interdit
1	1	1	Φ	

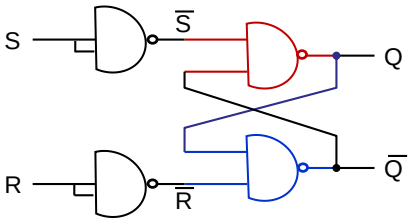
Tableau de Karnaugh :

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

Equation logique de sortie :

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

Logigramme :



Cette bascule **RS** est **prioritaire au 1** car, pour la combinaison $\boldsymbol{R = S = 1}$, la sortie Q est mise à 1 (les Φ ayant été fixés à 1 pour la simplification de Q).

❖ ***Remarque 4.1 :***

Le logigramme fait apparaître une sortie supplémentaire égale au **complément** de la sortie Q uniquement si la combinaison $\boldsymbol{R = S = 1}$ n'apparaît pas.

III.1.2 Réalisation à base des portes NOR

Table de vérité :

R	S	Q	Q^+	
0	0	0	0	Mémoire
0	0	1	1	
0	1	0	1	Mise à 1
0	1	1	1	
1	0	0	0	Mise à 0
1	0	1	0	
1	1	0	Φ	Interdit
1	1	1	Φ	

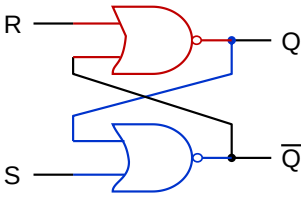
Tableau de Karnaugh :

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

Equation logique de sortie :

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

Logigramme :



Cette bascule **RS** est **prioritaire au 0** car, pour la combinaison $\boldsymbol{R = S = 1}$, la sortie Q est mise à 0 (les Φ ayant été fixés à 0 pour la simplification de Q).

❖ ***Remarque 4.2 :***

Le logigramme fait apparaître une sortie supplémentaire égale au **complément** de la sortie Q uniquement si la combinaison $\boldsymbol{R = S = 1}$ n'apparaît pas.

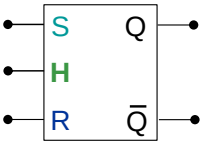
III.2. Bascule RSH

La bascule **RSH** est une bascule **synchronisée** par un **signal d'horloge H.** Les entées ***R*** et ***S*** de la bascule ne sont prises en compte que lorsque l'horloge ***H*** est **active** :

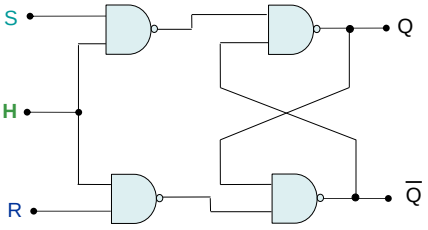
Table de vérité :

R	S	H	Q^+	
0	0	**0**	Q	Mémorisation
0	1	**0**	Q	
1	0	**0**	Q	
1	1	**0**	Q	
0	0	**1**	Q	Valeurs prises par Q^+ quand H passe de 0 à 1.
0	1	**1**	1	
1	0	**1**	0	
1	1	**1**	X	

Symbole :



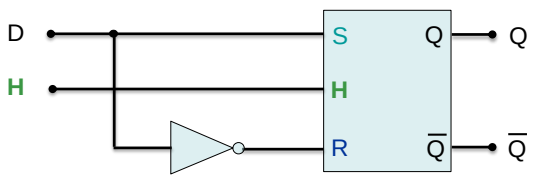
Logigramme avec des NAND :



III.3. Bascule à verrouillage (D-latch)

La **D-latch** est une **bascule RSH** pour laquelle on n'a conservé que les deux combinaisons $RS = (0,1)$ et $RS = (1,0)$. La D-latch a une seule entrée, nommée **D**.

La donnée **D** est **copiée** dans la bascule lorsque l'horloge passe de **1** à **0**. Elle y est **mémorisée** jusqu'à ce que l'horloge redevienne active.



Symbole :

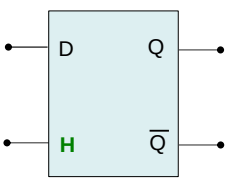


Table de vérité :

D	H	Q_{n+1}	Mode
X	0	Q_n	Verrouillée
0	1	0	Transparent
1	1	1	Transparent

III.4. Bascules J-K

Les bascules **JK** sont plus polyvalentes que les basculent RS, puisque l'état indéterminé est remplacé par un état complémenté, elles n'ont pas donc d'état ambigu $\boldsymbol{R = S = 1 \rightarrow Q_{n+1} = \overline{Q_n}}$.

La bascule **JK** synchrone est obtenue à partir d'une bascule RSH dont les sorties sont rebouclées sur les entrées. Ceci permet d'éliminer l'état indéterminé.

Logigramme avec des NAND :

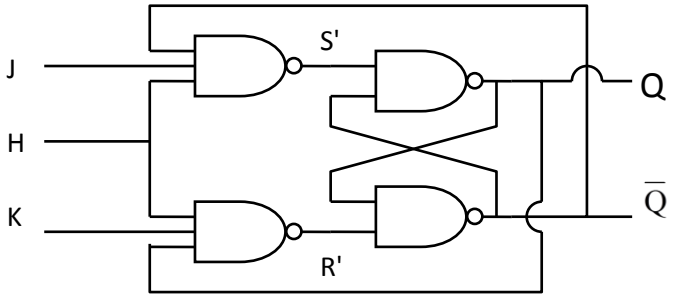


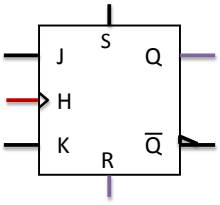
Table de vérité :

J	K	Q_{n+1}	fonctionnement
0	0	Q_n	état mémoire
0	1	0	mise à 0
1	0	1	mise à 1
1	**1**	$\overline{Q_n}$	**basculement**

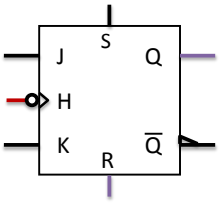
D'après cette table de vérité, nous remarquons que les 3 premières lignes correspondent bien à une bascule **RS** mais la dernière ligne introduit un mode **supplémentaire** qui sera très utilisé pour réaliser les compteurs synchrones que nous verrons plus loin.

Symbole :

bascule JK synchronisée sur front montant



bascule JK synchronisée sur front descend

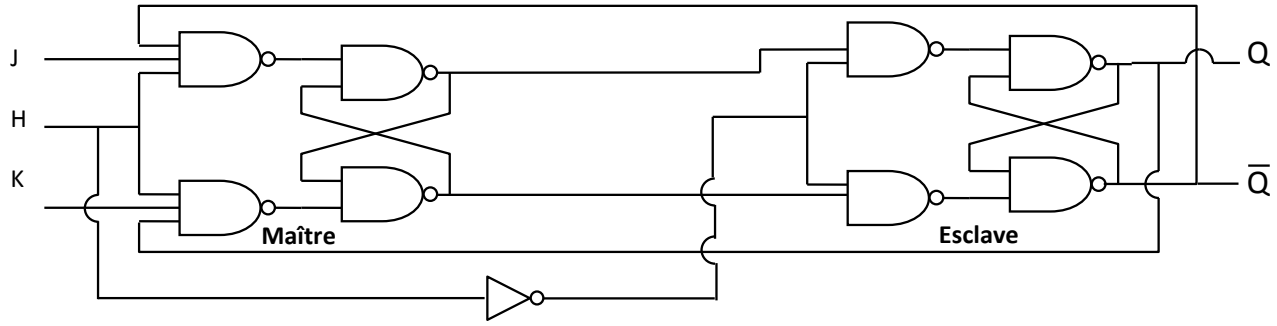


III.5. Bascules J-K Maitre Esclave

Dans une bascule JK, pour ***H=J=K=1*** on a la sortie ***Q*** qui ***oscille*** entre **'*0*'** et **'*1*'** pendant toute la durée de l'état haut du signal d'horloge.

La bascule JK maitre-esclave **remédie** à ce problème. Dans ce cas le transfert de la donnée s'effectue en deux temps. Sur le **front montant** de l'horloge, on **mémorise** la donnée dans le **Maître**, puis celle-ci est **transférée** à la sortie de l'**Esclave** sur le **front descendant**.

Logigramme de bascule JK Maître-Esclave:



IV. Les registres

Un registre est un groupe de bascules qui partagent une horloge commune et qui peut stocker un bit. Un registre à ***n bits*** est un groupe de ***n bascules*** qui peuvent stocker n bits. Un registre peut aussi avoir des portes combinatoires qui permettent de mieux traiter les bits. Les bascules dont le stockage de l'information, et les portes déterminent comment l'information est transférée dans le registre.

Les informations peuvent être écrites/lues en même temps (parallèle) ou une après l'autre (série).

Le nombre de bits du registre correspond au nombre de cellules mémoire (nombre de bascule D ou JK) du registre. On note que toutes les entrées d'horloge (H) des cellules sont reliées (ligne d'écriture).

Les registres sont classées par Les registres de mémorisation peuvent être classés selon la méthode d'écriture de données ou de lecture :

- Des registres à entrées parallèles et sorties parallèles : PIPO (**P**arallel **I**N-**P**arallel **O**UT).
- Des registres à entrées parallèles et sorties séries : PISO (**P**arallel **I**N-Serial **O**UT).
- Des registres à entrées séries et sorties parallèles : SIPO (**S**erial **I**N- **P**arallel **O**UT).
- Des registres à entrées séries et sorties séries : SISO (**S**erial **I**N- **S**erial **O**UT).

IV.1. Registres de mémorisation (Registre parallèle)

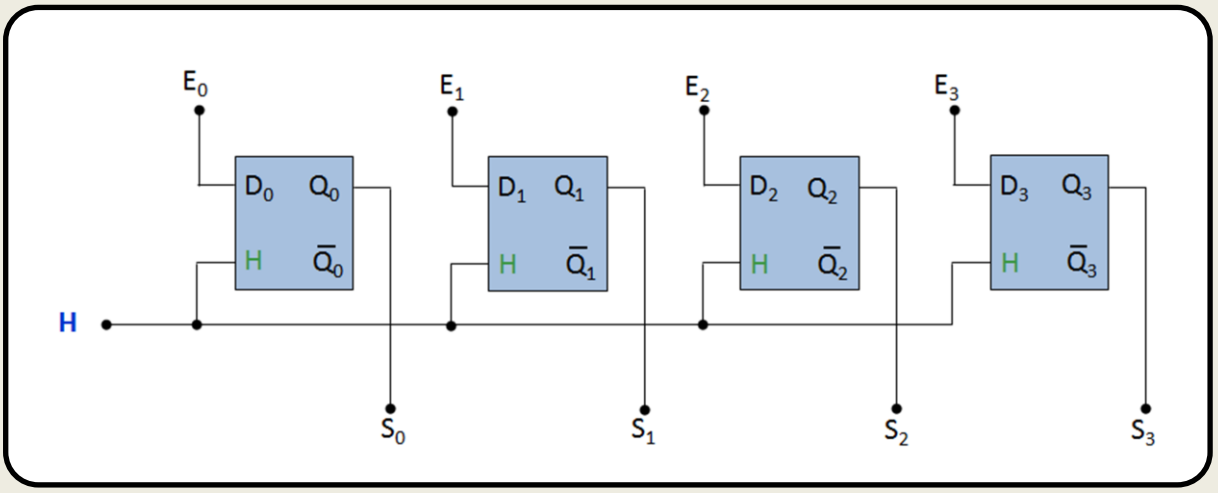
Un registre de mémorisation (ou registre de données) est un registre dans lequel les différents étages sont indépendants les uns des autres, cependant certains signaux agissent sur l'ensemble des étages ; tel que remise à **0** et remise à **1**.

IV.1.1 Registre de mémorisation 4 bits

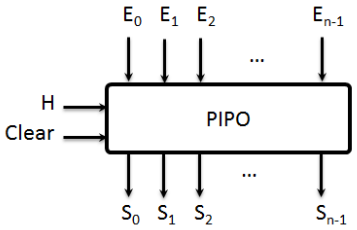
Dans l'exemple ci-dessous, les 4 bascules sont chargées en parallèle et lues en parallèle en synchrone avec le signal d'écriture H. Ce type de registre est appelé aussi registre **PIPO**.

EXEMPLE 4. 1 :

Registre de mémorisation 4 bits



IV.1.2 Schéma fonctionnel d'un registre PIPO.

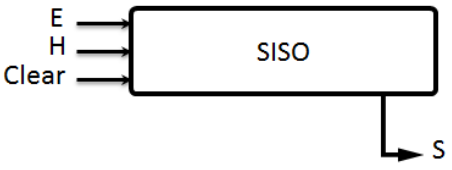


IV.2. Registres à décalage (Registre série)

Ce type de registre est principalement utilisé comme mémoire d'information dynamique ; la fonction de décalage consiste de faire glisser l'information de chaque cellule élémentaire dans une autre cellule élémentaire adjacente.

Ce type de registre est appelé aussi registre **SISO**.

IV.2.1 Schéma fonctionnel



IV.2.2 Décalage à droite

La bascule du rang i doit recopier la sortie de la bascule du rang (i-1) ainsi son entrée doit être connectée à la sortie (i-1).

IV.2.3 Décalage à gauche

L'entrée de la bascule du rang i doit recopier la sortie de la bascule du rang (i+1).

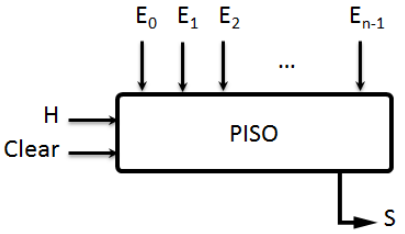
IV.2.4 Décalage réversible

Il existe des registres à décalage réversibles, c'est à dire des registres ou le décalage s'effectue vers la droite et vers la gauche en fonction du niveau logique applique à l'entrée S : « sens de décalage ».

IV.3. Registre mixte

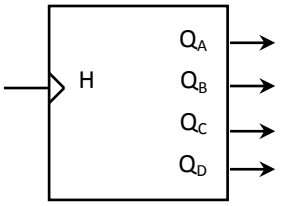
On peut trouver des registres mixtes, donc on peut écrire en parallèle et lire en série (PISO), ou vice versa (PISO), ou qui offrent les deux possibilités « au choix ».

Schéma fonctionnel d'un PISO :



V. Les compteurs / décompteurs

Un compteur est un circuit qui compte des impulsions et qui affiche sur ses sorties le nombre d'impulsions qu'il a reçues depuis le début du comptage (en binaire évidemment).
On le représente par le schéma suivant (exemple d'un compteur sur 4 bits synchronisé sur front montant) :



Q_A est bien sur la sortie de poids faible et Q_D celle de poids fort.

Il existe 2 types de compteurs : les compteurs asynchrones et les compteurs synchrones (les termes "synchrone" ou "asynchrone" n'ont pas la même signification que pour les bascules et les 2 types de compteurs sont réalisés avec des bascules synchrones).

La capacité d'un compteur encore appelée MODULO est le nombre maximum d'états différents que peuvent prendre l'ensemble de ses sorties.

V.1. Les compteurs asynchrones

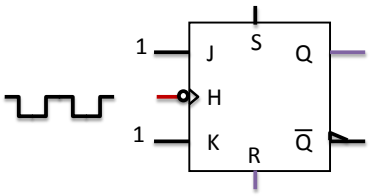
Rappel sur les entrées de forçage des bascules :

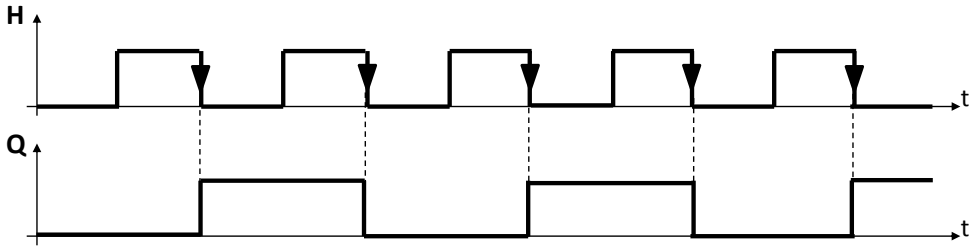
En plus des entrées 'normales', **RS**, **D** ou **JK** et de l'entrée d'horloge **H**, les bascules possèdent des entrées de mise à 0 (**Reset**) ou de mise à 1 (**Set** ou **Preset**) en général actives à l'état bas et qui sont prioritaires par rapport à toutes les autres entrées.

Dès que l'entrée **Reset = 0**, **Q = 0** ; dès que l'entrée **Preset = 0**, **Q = 1**. Evidemment, il faut éviter de les mettre toutes les **2** ensemble à **0** !

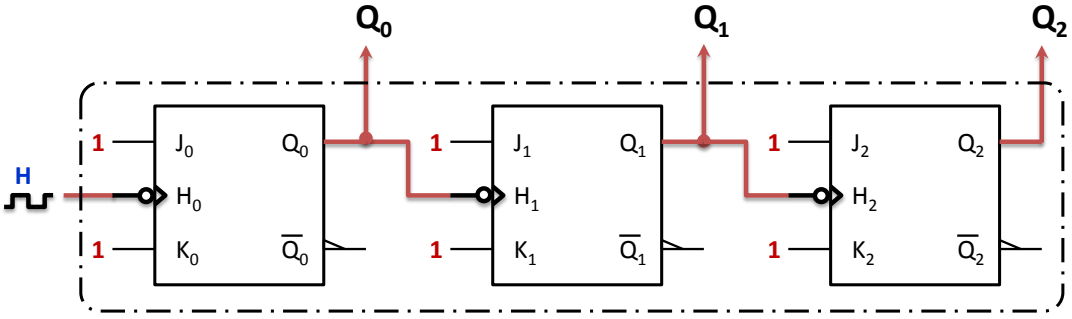
Les compteurs asynchrones sont réalisés avec des bascules montées en **diviseurs de fréquence** par 2 (avec des bascules **JK**, il suffit de mettre **J** et **K** à **1**).

Lorsque les entrées **J** et **K** de la bascule **J-K** sont à **1,** la sortie **Q** au front d'horloge suivant est complémentée. La sortie change d'état sur un *front descendant d'horloge*.





V.1.1 Compteurs modulo 2^n (exemple de 3 bits)



Pour compter en binaire naturel il faut que le bit de rang i + 1 change d'état quand le bit de rang i repasse à 0 : si l'on utilise des bascules synchronisées sur front descendant, il suffit de relier les sorties Q_i aux entrées d'horloge des bascules de rang i + 1, ce qui est réalisé sur le schéma ci-dessus.

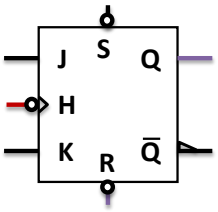
Le compteur représenté ici comptera donc de ***0*** (***000***) à ***7*** (***111***), le retour à 0 se faisant automatiquement à la $8^{ème}$ impulsion d'horloge.

En généralisant, on voit qu'avec n bascules, on obtient des compteurs modulo $\mathbf{2^n}$.

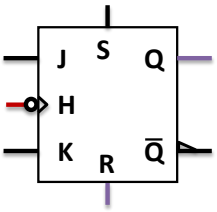
V.1.2 Compteurs modulo ≠ 2^n (cycle incomplet)

Pour revenir à 0 sur une autre valeur qu'une puissance de 2, il faut détecter cette valeur et remettre immédiatement le compteur à 0 : On utilisera l'entrée de remise à 0 asynchrone que possèdent toutes les bascules. Elle fonctionne de la façon suivante : dès que l'on applique l'état actif sur cette entrée, la sortie Q de la bascule passe à 0, indépendamment de l'état de l'horloge et des autres entrées de commande (D ou J, K). Elle est notée **R** (Reset) ou **Cl** (Clear) et elle est soit active à l'état bas ou à l'état haut.

Ci-après les schémas de deux bascules JK, synchronisées sur front descendant. La première est munie d'une entrée de remise à 0 active à l'état bas alors que la deuxième est munie d'une entrée de remise à 0 active à l'état haut.

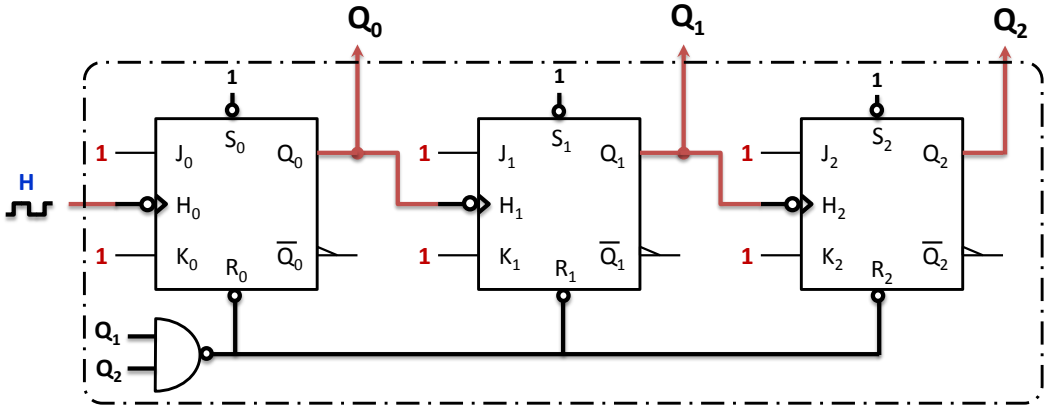


Reset active à l'état ***bas***



Reset active à l'état ***haut***

Par exemple pour réaliser un compteur asynchrone qui compte de **0** à **5** et qui revient à **0** à la 6ème impulsion, il faut détecter la valeur $(6)_{10}$ soit $(110)_2$ et activer les entrées Clear (**R**) des bascules. La remise à 0 doit donc être activée dès que $Q_1 = Q_2 = 1$ et $Q_0 = 0$ mais comme on compte par ordre croissant, c'est la 1ère fois que Q_1 et Q_2 sont tous les deux à **1** en même temps : il suffit donc de détecter le passage à **1** de ces **2** sorties et d'envoyer un **0** sur les entrées Clear (**R**) des 3 bascules actives à l'état bas, ce qui se fait avec une simple porte NAND selon le schéma suivant :



Par mesure de précaution, on remet à **0** toutes les bascules, même la bascule **0** qui est déjà.

V.2. Les décompteurs asynchrones

V.2.1 Décompteurs modulo 2^n (exemple de 3 bits)

Pour décompter en binaire naturel il faut que le bit de rang i + 1 change d'état quand le bit de rang i passe à **1** : si l'on utilise des bascules synchronisées sur ***front descendant***, il suffit de relier les sorties $\overline{Q_i}$ aux entrées d'horloge des bascules de rang **i + 1**. Le schéma est donc analogue à celui des compteurs modulo $\mathbf{2^n}$ en tenant compte de la remarque ci-dessus.

V.2.1 Décompteurs modulo $\neq 2^n$

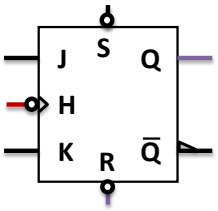
Le principe est analogue à celui des compteurs à quelques détails près : pour réaliser un décompteur modulo **n**, on doit obtenir la séquence suivante : **n-1**, **n-2**, …, **3**, **2**, **1**, **0**, **n-1**, …

La valeur 0 doit bien exister et rester stable tant que la nième impulsion n'est pas arrivée : si le modulo **n** était une puissance de **2**, la valeur suivant **0** serait **n-1** c'est à dire que toutes les sorties des bascules seraient à **1**.

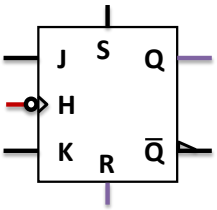
Par exemple pour réaliser un décompteur asynchrone qui décompte de **5** à **0** et qui revient à **5** à la **6ème** impulsion, il faut détecter la valeur $(7)_{10}$ soit $(111)_2$ et la remplacer par $(5)_{10}$ soit $(101)_2$.

La méthode la plus simple serait de détecter la valeur $(7)_{10}$ et de remettre la sortie Q_1 à **0** en utilisant un simple NAND à **3** entrées.

Pour plus de sécurité, on préfère repositionner toutes les sorties dans l'état correct c'est à dire **101**.On doit donc utiliser des bascules disposant, en plus de l'entrée de remise à **0** (R) d'une entrée de remise à **1** fonctionnant de la même façon que l'entrée Clear : dès que l'on applique ***l'état actif*** sur cette entrée, la sortie Q de la bascule passe à **1**, indépendamment de l'état de l'horloge et des autres entrées de commande (**D** ou J, K). Elle est notée **Pr** (Preset) ou **S** (Set) et elle est soit active à l'état bas ou à l'état haut.

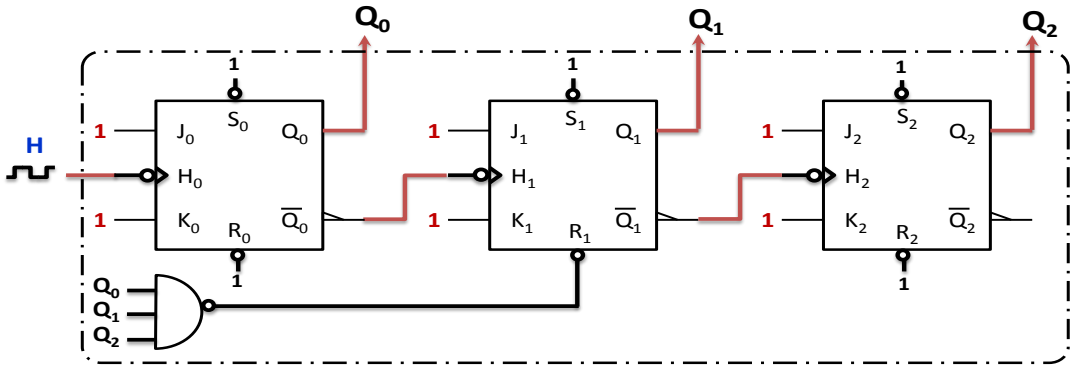


Set active à l'état ***bas***



Set active à l'état ***haut***

Le schéma de ce décompteur asynchrone est le suivant :



V.3. Les compteurs synchrones

Les compteurs asynchrones ont un certain nombre d'inconvénients :

- On ne peut compter ou décompter que dans l'ordre binaire pur : impossible par exemple de construire un compteur en *code Gray*.
- La fréquence maximum de comptage est **limitée** par les temps de retard des bascules qui **s'ajoutent** : si chaque bascule a un **temps de retard** de **50 ns**, le passage de **0111** à **1000** ne sera effectif qu'au bout de **200** ns, le temps que la dernière bascule ait réagi.

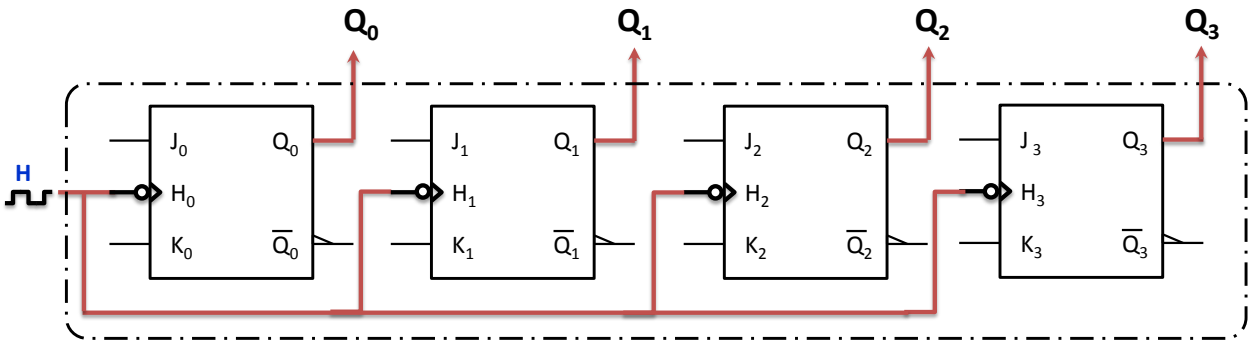
On préfère donc utiliser des compteurs dits **synchrones** : c'est quand les entrées d'horloge de toutes les bascules du compteur sont reliées au **même signal d'horloge**, contrairement aux compteurs asynchrones dans lesquels la sortie d'une bascule commandait l'entrée d'horloge de la suivante.

V.3.1 Définition

Un compteur synchrone est constitué de ***n bascules*** qui reçoivent ***en parallèle*** le ***même signal d'horloge***. L'entrée horloge est donc commune à toutes les bascules dont les sorties changent d'états simultanément

ensemble contrairement aux compteurs asynchrones dans lesquels la sortie d'une bascule commandait l'entrée d'horloge de la suivante.

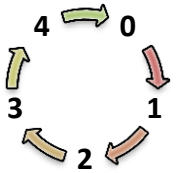
Pour un compteur de modulo ≤ **16**, cela donne le schéma suivant :



Le problème va donc consister à relier correctement les entrées ***J*** et ***K*** de chaque bascule aux sorties ***Q*** des autres bascules pour obtenir le **cycle voulu**.

On va utiliser l'équation d'une bascule **JK** qui donne la valeur de la sortie Q_{n+1} en fonction de son état antérieur noté Q_n et des états de ***J*** et de ***K*** au moment du front d'horloge, soit :

Le plus simple est de partir d'un exemple concret, celui d'un compteur ***modulo 5*** qui nécessitera ***3 bascules : 2***, ***1*** et ***0***. On doit donc obtenir le cycle ***0***, ***1***, ***2***, ***3***, ***4***, ***0***, ***1*** … dont le diagramme des états est le suivant :



V.3.2 Table et fonctions de transition

Les fonctions de commutation ou de transition φ_i sont définies de telle sorte à ce que $\varphi_i = 1$ lorsque la **bascule *i*** change d'état d'une combinaison à une autre (de ***l'état présent*** à ***l'état future***), soit la fonction de transition :

$$\varphi_n = J_n.\overline{Q_n} + K_n.Q_n$$

On commence donc par remplir la ***table de transition*** correspondant au cycle voulu.

	N_{10}	Q_2	Q_1	Q_0	φ_2	φ_1	φ_0
état initial	**0**	**0**	**0**	**0**	0	0	1
après la 1ère impulsion	1	0	0	1	0	1	1
après la 2ème impulsion	2	0	1	0	0	0	1
après la 3ème impulsion	3	0	1	1	1	1	1
après la 4ème impulsion	4	1	0	0	1	0	0
après la 5ème impulsion	**0**	**0**	**0**	**0**			

Puis il faut remplir les tableaux de Karnaugh de chaque fonction φ_i en fonction de Q_i.

φ_0	$Q_1\,Q_0$				
		00	01	11	10
Q_2	0	1	1	1	1
	1	0	X	X	X

φ_1	$Q_1\,Q_0$				
		00	01	11	10
Q_2	0	0	1	1	0
	1	0	X	X	X

φ_2	$Q_1\,Q_0$				
		00	01	11	10
Q_2	0	0	0	1	0
	1	1	X	X	X

On en tire les équations des sorties φ_i en utilisant **éventuellement** les valeurs indéterminées X :

$$\varphi_0 = \overline{Q_2} \qquad \varphi_1 = Q_0 \qquad \varphi_2 = Q_0.Q_1 + Q_2$$

On identifie chaque équation à l'équation type d'une bascule JK :

$$\varphi_i = J_i.\overline{Q_i} + K_i.Q_i$$

le coefficient de $\overline{Q_i}$ doit être égal à J_i et celui de Q_i doit être égal à K_i.

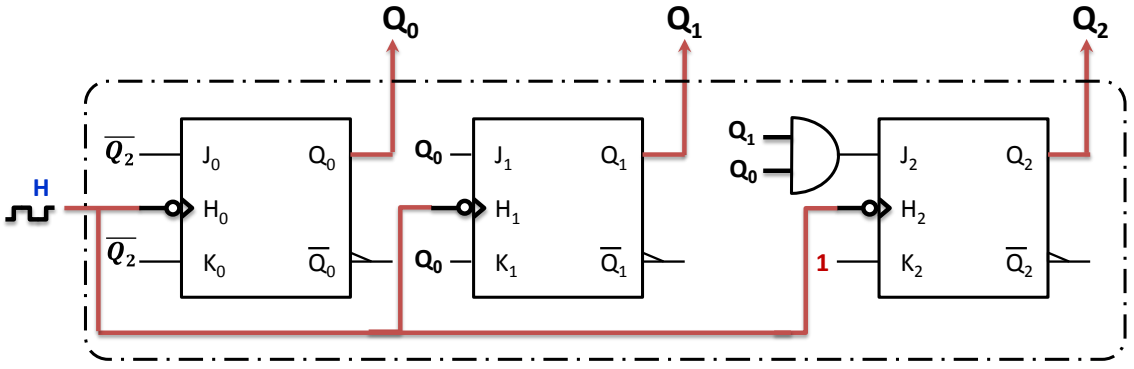
Donc on déduit les égalités suivantes :

$$J_0 = K_0 = \overline{Q_2}\ ;$$

$$J_1 = k_1 = Q_0\ ;$$

$$J_2 = Q_0.Q_1 \text{ et } K_2 = 1.$$

ce qui donne le schéma suivant :



V.4. Compteurs spéciaux

V.4.1 Compteurs réversibles

Ce sont des compteurs/décompteurs qui peuvent compter comme ils peuvent décompter. Le choix de la fonction comptage ou de la fonction décomptage s'effectue à l'aide d'une entrée de sélection.

V.4.2 Compteurs programmables ou pré-positionnés (presettables)

Ce sont des compteurs possédant des entrées de pré-chargement qui contiennent la valeur de départ du comptage ou de décomptage. Le chargement du compteur se fait grâce à une entrée de commande.

V.5. Compteurs synchrones à circuits intégrés

On cite à titre d'exemple :

- Les compteurs synchrones pré-positionnés modulo **10** : ***74LS160*** et ***74LS162.***
- Les compteurs synchrones pré-positionnés modulo **16** : ***74LS161*** et ***74LS163***.
- Les compteurs/décompteurs synchrones pré-positionnés modulo **10** : ***74LS190*** et ***74LS192***.
- Les compteurs/décompteurs synchrones pré-positionnés modulo **16** : ***74LS191*** et ***74LS193***.

Bibliographie

- Thomas Floyd, Martin Villeneuve, André Lebel, **Systèmes Numériques**, **2018**, 11e édition, Reynald Goulet.
- Ronald J. Tocci, **Circuits numériques (Théorie et applications)**, 2ème édition, Dunod.
- Mesnard Emmanuel - **Du binaire au processeur**, **2004**, Ellipse.
- Noël Richard, **Electronique numérique et séquentielle** (**Pratique des langages de description de haut niveau**), **2002**, Dunod.
- Mouloud Sbai, **Logique Combinatoire & Composants Numériques** (**Cours & Exercices Corrigés**), 2013, Ellipse.
- Robert Strandh, Irène Durand, **Architecture de l'ordinateur**, 2011, Dunod.
- Théron Alain**, Science de l'ingénieur - Automatique : logique (Cours et exercices corrigés), 1re année MPSI-PCSI-PTSI,** 2005, Ellipse.