Chapitre 3

Les Circuits Combinatoires

I. Introduction 31
II. Additionneurs 31
III. Soustracteur 33
IV. Comparateur 34
V. Multiplexeur (Mux) / Démultiplexeur (DMux) 37
VI. Décodeurs / Codeurs / Transcodeur 39

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.

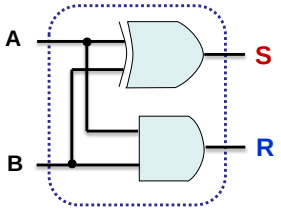
Table de vérité :

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 :

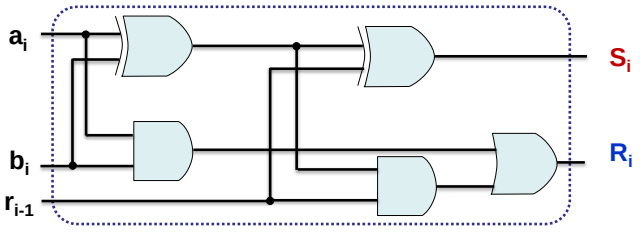
a_i	b_i	r_{i-1}	R_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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 :***

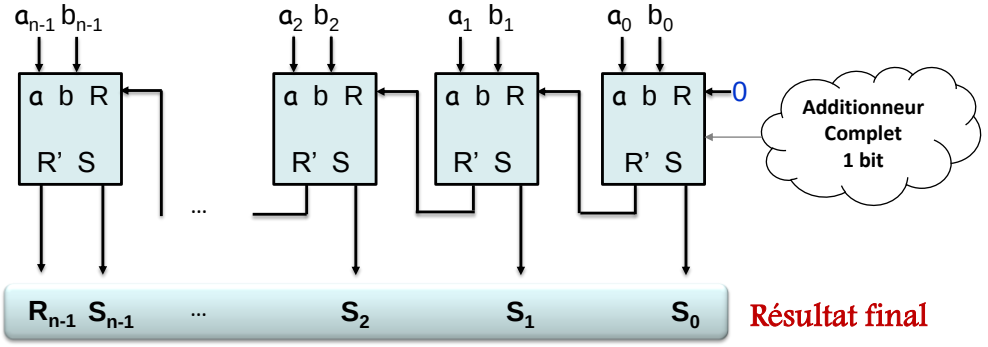
Cette structure montre la possibilité de réaliser un additionneur complet à partir de deux demi-additionneurs et d'une porte "**OR**".

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 :

$$A - B = A + (-B)$$
$$= A + CA_2(B)$$

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** :

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

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 :

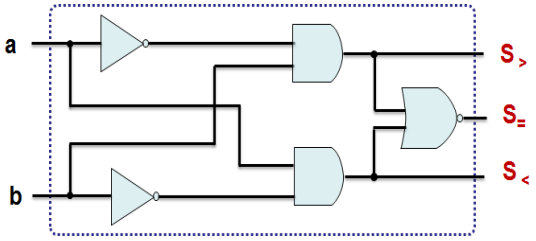
Table de vérité :

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 :

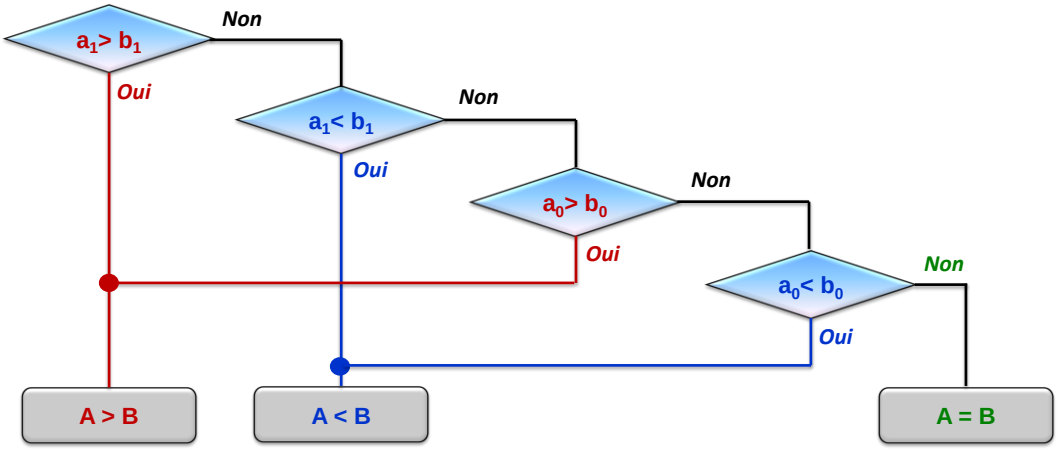


IV.2. Comparateur de deux nombres de n bits

❖ ***Principe et organigramme :***

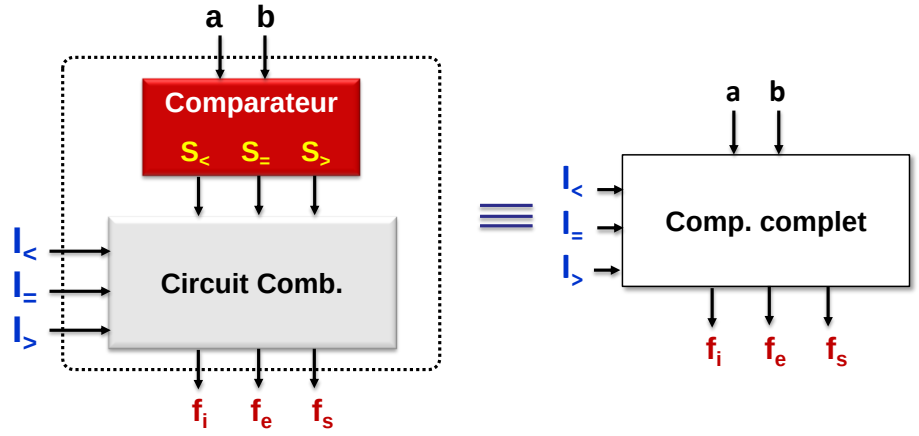
Prenant l'exemple de deux nombres ***A*** et ***B*** de **deux** bits : $\boldsymbol{A = a_1 a_0}$; $\boldsymbol{B = b_1 b_0}$

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**.



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

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**

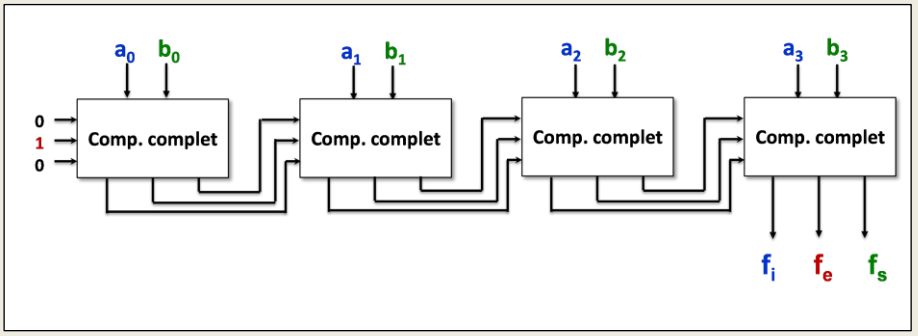
- A partir de la table de vérité, on déduit les **équations de sorties** : $\mathbf{f_s}$; $\mathbf{f_e}$ et $\mathbf{f_i}$.

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

EXEMPLE 3.1 :

Comparaison de deux nombres de 4 bits

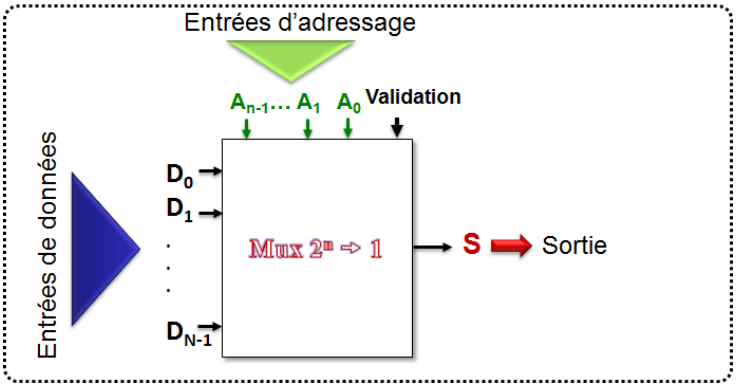
Le comparateur **4 bits** sera réalisé par la mise **en cascade** de **4 comparateurs de 1 bit**. Le résultat de la comparaison est recueilli sur la sortie du dernier comparateur :



V. Multiplexeur/ Démultiplexeur

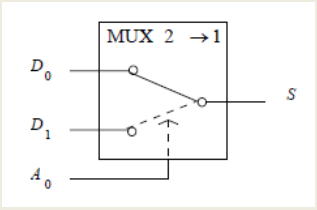
V.1. Multiplexeur

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 :

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

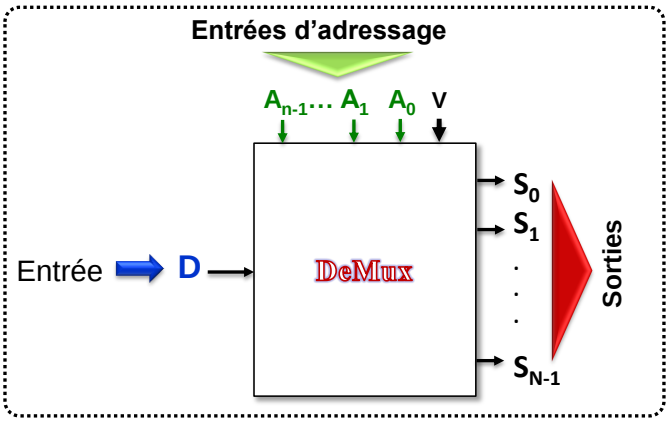
V.2. Démultiplexeur

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.

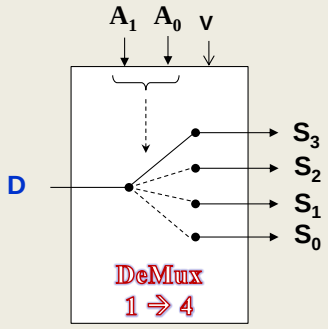
Un **DeMux** possède :

- 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) :**

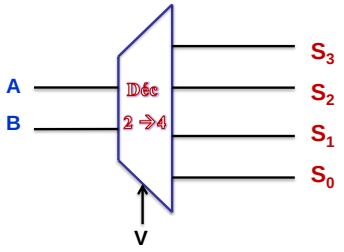


Table de vérité :

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) :**

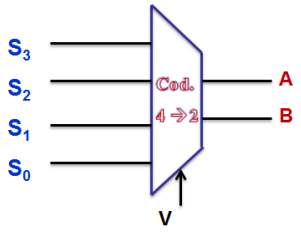


Table de vérité :

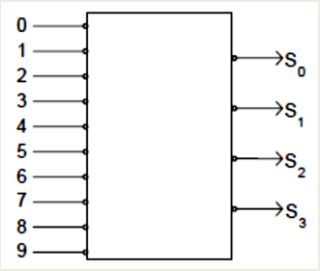
v	S_3	S_2	S_1	S_0		A	B
0	x	x	x	X		0	0
1	0	0	0	1		0	0
1	0	0	1	0		0	1
1	0	1	0	0		1	0
1	1	0	0	0		1	1

Equations de sorties :

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

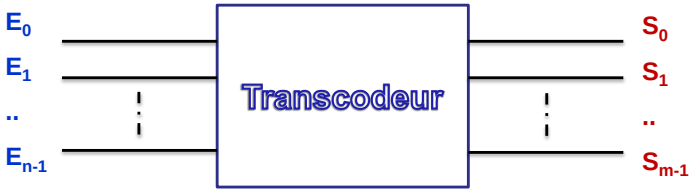
EXEMPLE 3. 4 :

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.

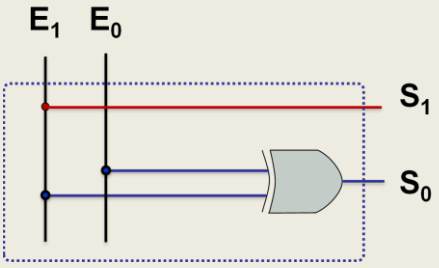
EXEMPLE 3. 5 :

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 :