IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

ELP111

Fonctions électroniques logiques et analogiques

Fiche séance

PC6 – CIRCUITS COMBINATOIRES

Sommaire

IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PC6 – Circuits combinatoires

1. LES OBJECTIFS D'APPRENTISSAGE

OA6 À partir d'une description d'une fonction à réaliser, synthétiser le circuit combinatoire répondant au problème à partir de fonctions logiques combinatoires élémentaires.

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

2.1 QU'EST-CE QU'UNE FONCTION COMBINATOIRE ET UN CIRCUIT COMBINATOIRE ?

On appelle **opérateur** ou **fonction combinatoire** un opérateur logique dont les sorties ne dépendent, à chaque instant, que de l'état de ses entrées. A chaque configuration des entrées correspond une configuration des sorties et une seule. Un circuit numérique réalisant une fonction combinatoire est un **circuit combinatoire**.

Outre les opérateurs élémentaires cités au chapitre 2, on distingue comme opérateurs combinatoires standard :

- les opérateurs de transcodage,
- les opérateurs d'aiguillage,
- les opérateurs de comparaison,
- les opérateurs arithmétiques.

Avertissement : L'étude de ces différents opérateurs est illustrée par des exemples de circuits standard issus de catalogues de fabricants de circuits intégrés. Les références complètes de ces circuits contiennent, outre un numéro caractéristique de la fonctionnalité du circuit, des informations complémentaires inutiles dans le cadre de ce chapitre (fabricant, technologie de fabrication, produit commercial ou militaire, etc.). Nous nous limiterons donc, ici, à fournir une référence générique n'indiquant que la fonctionnalité du circuit. A titre d'exemple, « xx85 » représente tous les circuits dont la référence se termine par 85, numéro désignant un comparateur de deux mots de 4 bits. Les fiches techniques (**data sheets**) des circuits cités dans ce chapitre peuvent être consultées dans des catalogues tels que [Mot], [TI], ou [NS], par exemple.

2.2 AVANT-PROPOS : CLASSIFICATION DES CODES BINAIRES

Le champ d'application des systèmes numériques est très étendu. Lorsque l'application ne nécessite pas de calculs arithmétiques, les codages présentés à la PC3 sont inutiles ou peu adaptés. On utilise alors des codages possédant d'autres propriétés. On emploie ainsi dans certains systèmes des codes permettant d'éviter des états transitoires parasites lors de la saisie de données, ou de visualiser facilement des chiffres ou des lettres, ou bien encore de détecter des erreurs et/ou de les corriger dans un résultat susceptible d'être erroné. Nous présentons ci-après quelques codes fréquemment utilisés.

L'ensemble des codes binaires peuvent être regroupés en deux classes : les **codes pondérés** et les **codes non pondérés**.

2.2.1 Codes pondérés

Un code est dit **pondéré** si la position de chaque symbole dans chaque mot correspond à un poids fixé : par exemple 1, 10, 100, 1000 ... pour la numération décimale, et 1, 2, 4, 8, ... pour la numération binaire. Les codes pondérés ont, en général, des propriétés intéressantes du point de vue arithmétique.

2.2.1.1. Le code binaire pur et ses dérivés (octal, hexadécimal)

Ce sont les codes utilisés en arithmétique binaire et qui ont été étudiés dans la PC3.

2.2.1.2. Le code DCB (Décimal Codé Binaire) ou BCD (Binary-Coded Decimal)

Ce code est utilisé dans de nombreux systèmes **d'affichage**, de **comptage** ou même les **calculatrices** de poche. Dans le code BCD chaque chiffre d'un nombre décimal (de $0_{(10)}$ à $9_{(10)}$) est codé à l'aide de 4 bits (de $0000_{(2)}$ à $1001_{(2)}$). Ainsi le code BCD n'utilise que 10 **mots de codes** de 4 bits. Par exemple la représentation du nombre $1995_{(10)}$ est : $(0001\ 1001\ 1001\ 0101)_{(BCD)}$. Il est possible d'effectuer des opérations arithmétiques en BCD, mais celles-ci sont plus complexes qu'en binaire classique. Ce code est pondéré avec les poids 1, 2, 4, 8, 10, 20, 40, 80, 100, ...

2.2.2 Codes non pondérés

Dans le cas des codes non pondérés, il n'y a pas de poids affecté à chaque position des symboles. On convient simplement d'un tableau de correspondance entre les objets à coder et une représentation binaire. De tels codes peuvent néanmoins parfois posséder des propriétés arithmétiques intéressantes, comme le code **excédent 3**.

2.2.2.1. Code binaire réfléchi ou code de Gray

Ce code numérique n'étant pas pondéré, il est peu employé pour les opérations arithmétiques. Il est, par contre, utilisé pour le codage des déplacements angulaires, linéaires ou pour la réalisation des tableaux de Karnaugh (cf. chapitre « Propriétés des variables et fonctions logiques »). La propriété principale de ce code est que **deux mots successifs du code ne diffèrent que par un élément binaire**. Ceci permet, d'une part d'éviter la génération d'aléas (états parasites) au passage de deux combinaisons successives, et d'autre part de tirer parti de cette adjacence du codage pour simplifier les fonctions.

L'appellation "binaire réfléchi" provient de sa technique de construction : on peut construire un code de Gray sur *n* bits à partir d'un code de Gray sur *n*-1 bits en procédant comme suit : on copie les mots du code de départ, précédés d'un 0, suivis des mots du même code, pris dans **l'ordre inverse** et précédés d'un 1. Ceci permet de construire un code de Gray de n'importe quel format (cf. **Error! Reference source not found.**).

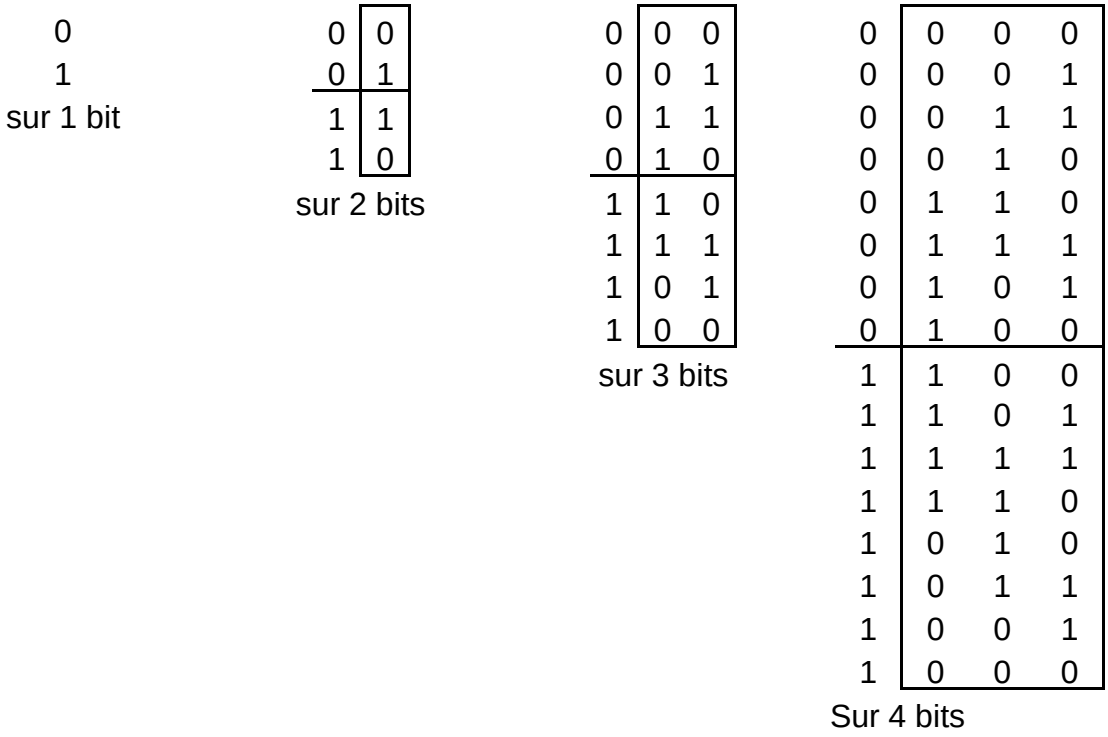


figure 1.1 : construction du code de Gray sur 1, 2, 3, et 4 bits

2.2.3 Codes alphanumériques

Certains codes peuvent avoir une signification non numérique. Le plus connu d'entre eux est le code ASCII (American Standard Code for Information Interchange), qui est utilisé pour représenter les caractères alphanumériques. Dans ce code, 128 combinaisons (lettres, chiffres, signes de ponctuation, caractères de contrôle, etc.) sont codées à l'aide de 7 bits. Les transmissions asynchrones entre machines s'effectuant souvent sur un format de 8 bits, le dernier bit est alors utilisé pour contrôler la parité du message.

2.3 LES OPERATEURS DE TRANSCODAGE

2.3.1 Définition

On désigne par opérateur de **transcodage** un opérateur qui traduit une information dans un code donné (**codeur** ou **encodeur**) ou bien qui, au contraire, permet d'extraire une ou des informations à partir d'un code donné (**décodeur**) ou bien encore réalise la conversion d'un code vers un autre (**convertisseur de code** ou **transcodeur**).

2.3.2 Les codeurs

Le principe de fonctionnement d'un codeur est le suivant : lorsqu'une entrée est activée, les sorties affichent la valeur correspondant au numéro de l'entrée dans le code binaire choisi. Un codeur peut être vu comme un convertisseur du code décimal vers un code binaire.

Une seule entrée du codeur doit normalement être activée à la fois. Dans le cas où le code en sortie est le code binaire pur, le circuit correspondant possède *N* entrées et *n* sorties, avec $2^{n-1} < N \leq 2^n$.

2.3.2.1. Exemples de codeurs

- **<u>Exemple 1</u> : codeur décimal vers binaire (10 entrées vers 4 sorties)**

Ce codeur reçoit un chiffre décimal sur une des dix entrées et génère l'équivalent binaire sur les sorties $A0$ à $A3$. Une seule entrée doit être active à la fois.

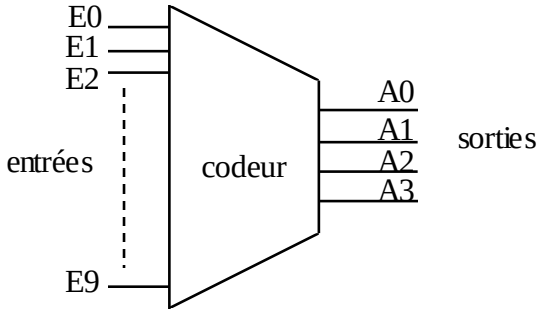


figure 4.2 : codeur décimal vers binaire

On trouvera ci-après (tableau 4.1) la table de vérité simplifiée et les équations de fonctionnement de ce type de codeur.

	Sorties			
Entrée activée (= 1)	$A3$	$A2$	$A1$	$A0$
$E0$	0	0	0	0
$E1$	0	0	0	1
$E2$	0	0	1	0
$E3$	0	0	1	1
$E4$	0	1	0	0
$E5$	0	1	0	1
$E6$	0	1	1	0
$E7$	0	1	1	1
$E8$	1	0	0	0
$E9$	1	0	0	1

tableau 4.1 : table de vérité réduite du codeur 10 vers 4

Equations logiques :

$$A0 = E1 + E3 + E5 + E7 + E9$$
$$A1 = E2 + E3 + E6 + E7$$
$$A2 = E4 + E5 + E6 + E7$$
$$A3 = E8 + E9$$

Cet opérateur, qui n'existe pas sous forme de circuit intégré standard, peut facilement être réalisé à partir de portes élémentaires.

- **<u>Exemple 2</u> : codeur binaire 8 vers 3**

Ce codeur reçoit une information codée sur une de ses huit entrées et génère l'équivalent binaire sur les sorties $A0$ à $A2$. Une seule entrée doit être active à la fois.

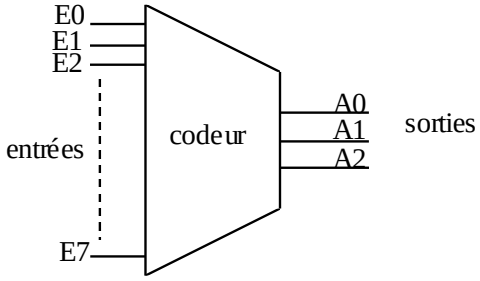


figure 4.3 : codeur 8 vers 3

	Sorties		
Entrée activée (= 1)	$A2$	$A1$	$A0$
$E0$	0	0	0
$E1$	0	0	1
$E2$	0	1	0
$E3$	0	1	1
$E4$	1	0	0
$E5$	1	0	1
$E6$	1	1	0
$E7$	1	1	1

tableau 4.2 : table de vérité réduite du codeur 8 vers 3

Equations logiques :

$$A0 = E1 + E3 + E5 + E7$$
$$A1 = E2 + E3 + E6 + E7$$
$$A2 = E4 + E5 + E6 + E7$$

Dans les deux exemples précédents il n'y a en théorie qu'une seule entrée active parmi *N*. Si, dans une application, plusieurs entrées peuvent être actives simultanément il faut utiliser un codeur **prioritaire**.

2.3.2.2. Codeur prioritaire

Ce type de codeur fixe un ordre de priorité entre les entrées. Dans le cas d'un encodage en binaire pur, le codeur prioritaire donne en général la priorité à l'entrée de poids le plus élevé. Par exemple, si les entrées 2, 8 et 9 sont activées simultanément, le codage de sortie correspondra à l'entrée 9. Ce circuit permet de détecter le rang du premier bit positionné à 1 dans un mot d'entrée.

- **<u>Exemple</u> : encodeur prioritaire 4 vers 2**

L'opérateur de la figure 4.4 est un encodeur prioritaire possédant 4 entrées ; l'entrée $E3$ correspond à l'entrée la plus prioritaire, et l'entrée $E0$ correspond à l'entrée la moins prioritaire

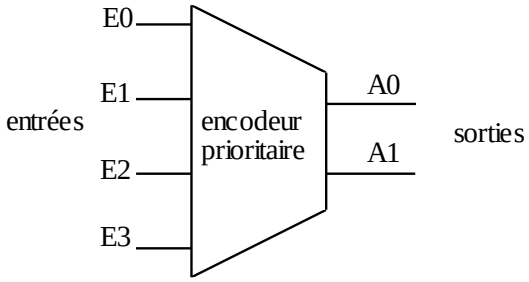


figure 4.4 : encodeur prioritaire 4 vers 2

Sa table de vérité et ses équations de fonctionnement sont présentées ci-après (tableau 4.3).

$E3$	$E2$	$E1$	$E0$	$A1$	$A0$
1	X	X	X	1	1
0	1	X	X	1	0
0	0	1	X	0	1
0	0	0	1	0	0

tableau 4.3: table de vérité de l'encodeur prioritaire 4 vers 2

Après simplification, nous obtenons les équations suivantes :

$$A0 = E3 + \overline{E2}\,E1$$
$$A1 = E3 + E2$$

2.3.3 Les décodeurs

Un décodeur est un opérateur à *n* entrées et *N* sorties avec $N \leq 2^n$. Une sortie du décodeur est activée lorsqu'une configuration particulière du code est affichée en entrée. Suivant le nombre de sorties, le décodeur peut décoder toutes les configurations possibles du code en entrée ou une partie seulement. Un décodeur à *n* entrées, permettant de décoder toutes les configurations du code binaire pur sur *n* bits, possède 2^n sorties. <u>Une seule sortie est activée à la fois</u>. En plus des *n* entrées, certains décodeurs possèdent une ou plusieurs entrées de validation. Par exemple, pour une validation active au niveau bas, si l'entrée de validation *V* vaut 0, alors le décodage est autorisé ; dans le cas contraire (*V* = 1), les sorties sont inhibées et

restent inactives. Ces entrées de validation permettent de grouper plusieurs décodeurs afin d'augmenter l'amplitude de décodage (voir section 4.2).

2.3.3.1. Les principaux types de décodeurs

a. Les décodeurs binaires

Ce type d'opérateur permet d'associer une sortie parmi 2^n avec une information d'entrée codée sur *n* bits.

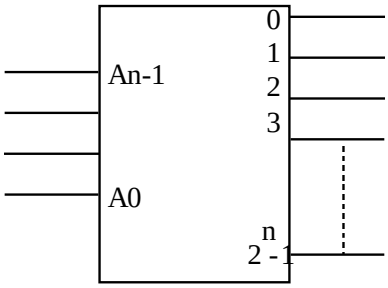


figure 4.5 : décodeur binaire « *n* vers 2^n » ou « 1 parmi 2^n »

- **Exemple 1 : décodeur 2 vers 4**

Le tableau 4.4 présente le fonctionnement d'un décodeur binaire 2 vers 4 ou 1 parmi 4.

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

tableau 4.4 : table de vérité d'un décodeur binaire 2 vers 4

Dans cet exemple, on a choisi d'associer la valeur "0" lorsque la sortie est décodée (sorties actives à 0). Les équations de fonctionnement de ce circuit sont les suivantes :

$$\overline{S0} = \overline{E1}.\overline{E0}$$
$$\overline{S1} = \overline{E1}.E0$$
$$\overline{S2} = E1.\overline{E0}$$
$$\overline{S3} = E1.E0$$

La réalisation du logigramme de ce composant à l'aide d'opérateurs logiques élémentaires ne présente pas de difficulté particulière.

- **Exemple 2 : décodeur 3 vers 8**

Contrairement à l'exemple précédent, on associe la valeur logique "1" lorsque le rang de la sortie active correspond à la valeur binaire présentée en entrée (sorties actives à 1). La table de vérité (tableau 4.5) comporte 8 sorties qui correspondent aux mintermes des variables d'entrée.

Entrées			Sorties							
C	*B*	*A*	*S*0	*S*1	*S*2	*S*3	*S*4	*S*5	*S*6	*S*7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

tableau 4.5 : table de vérité du décodeur 3 vers 8

Les sorties du décodeur sont données par les relations suivantes :

$$S0 = \overline{C}\ \overline{B}\ \overline{A}$$
$$S1 = \overline{C}\ \overline{B}\ A$$
$$\dots$$
$$S7 = C\ B\ A$$

Une réalisation possible du décodeur 3 vers 8 est immédiate :

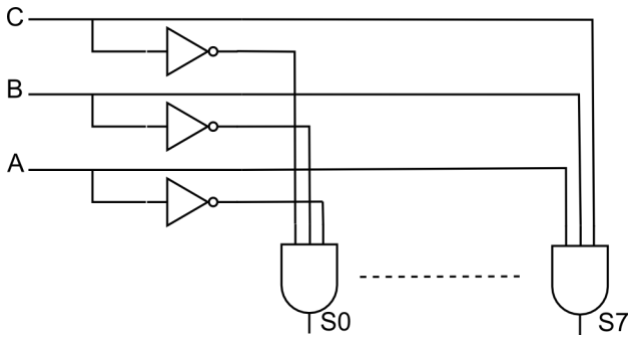


figure 4.6 : réalisation d'un décodeur 3 vers 8

b. Le décodeur BCD

Le code BCD (Binary-Coded Decimal, cf. chapitre 1, section 2.5.1.2) est utilisé pour coder les dix chiffres décimaux. Ce code à 4 bits laisse donc inutilisées 6 combinaisons sur les 16 possibles. Lorsqu'une combinaison, comprise entre 0 et 9, est appliquée sur les entrées *ABCD* (*A* est le bit de poids fort), la sortie correspondante est validée. Les sorties restent au repos (niveau 1 par exemple) dans le cas où une combinaison comprise entre 10 et 15 est appliquée sur les entrées.

Table de vérité :

A	B	C	D	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

tableau 4.6 : table de vérité du décodeur BCD

Représentation symbolique usuelle :

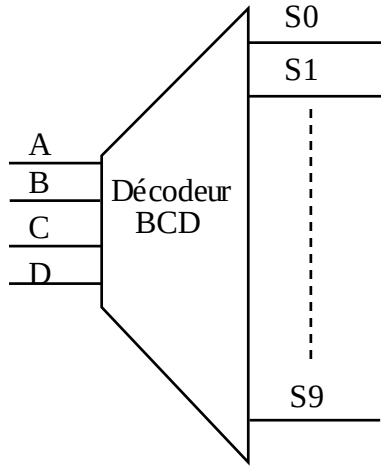


figure 4.7 : entrées et sorties du décodeur BCD

2.3.3.2. Applications des décodeurs

Les applications des décodeurs sont nombreuses. Nous avons choisi de les illustrer à travers trois exemples.

- **<u>Exemple 1</u> : décodage d'adresse d'une mémoire ou sélection de composants dans une carte microprocesseur**

Dans une carte mère, qui possède en général un microprocesseur et plusieurs composants annexes (RAM, ROM, REPROM, boîtiers d'entrées-sorties, etc.), la

fonction de décodage est essentielle afin de ne valider qu'un seul boîtier à la fois. Les signaux qui servent au décodage, et donc à la sélection des boîtiers sont issus du bus d'adresse et du bus de contrôle du microprocesseur. La figure 4.8 montre un synoptique simplifié d'une telle application des décodeurs.

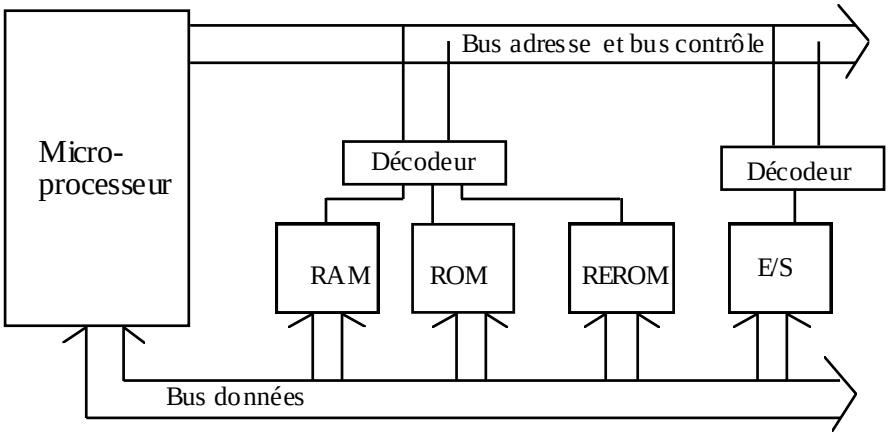


figure 4.8 : sélection d'un composant par décodeur

- **<u>Exemple 2</u> : générateur de fonction**

Nous avons vu dans la section a que les sorties Si d'un décodeur 3 vers 8 étaient régies par les équations suivantes :

$$S0 = \overline{C}\ \overline{B}\ \overline{A}$$
$$S1 = \overline{C}\ \overline{B}\ A$$
$$\ldots$$
$$S7 = C\ B\ A$$

On remarque que chaque sortie Si représente un des mintermes des variables *A*, *B*, et *C*. En réunissant ces mintermes on peut donc calculer n'importe quelle fonction de trois variables à l'aide d'un décodeur 3 vers 8 associé à des portes OU. Il est cependant à noter que l'utilisation de décodeurs en générateur de fonctions est en pratique peu fréquente.

- **<u>Exemple 3</u> : démultiplexage**

Le décodeur est parfois utilisé pour **démultiplexer** des informations. Dans ce type d'application, son rôle consiste à aiguiller une information présente sur une entrée vers une sortie choisie parmi les *N* disponibles.

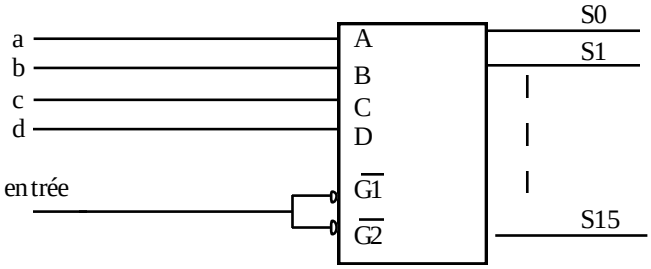


figure 4.9 : décodeur *XX*154 utilisé pour démultiplexer

Dans l'exemple de la figure 4.9, les informations multiplexées sont injectées dans les entrées de validation ($\overline{G1}$ et $\overline{G2}$) du décodeur 4 vers 16. Les entrées *A*, *B*, *C*, et *D* possèdent ici un rôle d'entrées d'adressage. Elles servent à aiguiller l'information présente sur $\overline{G1}$ et $\overline{G2}$ vers une des 16 sorties Si.

N. B. Les sorties du décodeur xx154 sont actives au niveau bas (à 0).

2.3.4 Les transcodeurs

Ces opérateurs permettent de convertir un nombre écrit dans un code C1 vers un code C2.

2.3.4.1. Exemple 1 : le transcodeur BCD/7 segments

Le transcodeur BCD/7 segments permet de commander un afficheur alphanumérique possédant 7 segments (des diodes électroluminescentes, par exemple). Cet afficheur permet la visualisation des chiffres 0 à 9 codés en binaire naturel sur 4 bits *D*, *C*, *B*, et *A*, où *A* représente le bit de poids le plus faible.

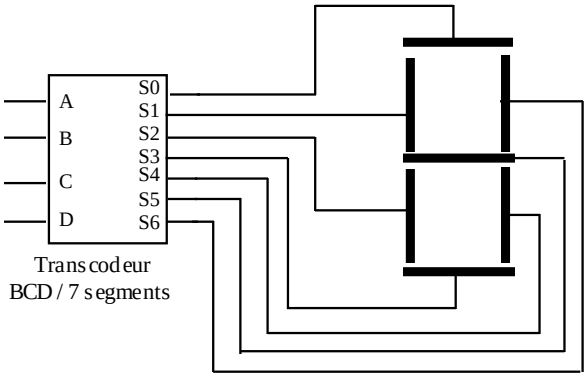


figure 4.10 : commande d'un afficheur par transcodeur

Les équations logiques du transcodeur de la figure 4.10 s'établissent sans difficultés à l'aide de 7 tableaux de Karnaugh :

$$S0 = \overline{A}.\overline{C} + A.C + B + D$$
$$S1 = \overline{A}.C + \overline{A}.\overline{B} + \overline{B}.C + D$$
$$S2 = \overline{A}.\overline{C} + \overline{A}.B$$
$$S3 = A.\overline{B}.C + \overline{A}.\overline{C} + \overline{A}.B + B.\overline{C} + D$$
$$S4 = A + \overline{B} + C$$
$$S5 = \overline{B}.C + \overline{A}.B + B.\overline{C} + D$$
$$S6 = A.B + \overline{C}$$

Ces transcodeurs existent sous la forme de circuits standard proposés par les fabricants sous les références xx46 à xx49. Ces circuits possèdent des entrées de contrôle supplémentaires : extinction de l'affichage des segments, allumage de tous les segments, etc. D'autres types de transcodeurs sont capables de commander des afficheurs alphanumériques. Les fonctions de trancodages sont parfois intégrées dans les afficheurs.

2.3.4.2. *Exemple 2 : les convertisseurs Gray/binaire et binaire/Gray*

Le code de Gray ou code binaire réfléchi (cf. chapitre 1, section 2.5.2.2) est largement utilisé dans les systèmes numériques, notamment dans les capteurs de position (pour coder des positions angulaires, par exemple). En effet, un seul bit change entre deux positions successives, et les risques d'informations parasites lors des transitions sont éliminés.

Les équations logiques d'un convertisseur de code Gray/binaire ou binaire/Gray s'établissent sans problème à l'aide de la méthode de Karnaugh à partir de la table de vérité suivante :

Décimal	Binaire				Gray			
	B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

tableau 4.7 : passage du code binaire au code Gray sur 4 bits

Pour des mots codés sur *n* bits, on obtient :

- **Convertisseur Gray/binaire**

$$B_n = G_n$$

et

$$B_{n-i} = G_n \oplus G_{n-1} \oplus \cdots \oplus G_{n-i} \text{ pour } i = 1,\ldots,n-1$$

- **Convertisseur binaire/Gray**

$$G_i = B_i \oplus B_{i+1} \text{ pour } i = 1,\ldots,n-1$$

et

$$G_n = B_n$$

2.4 LES OPERATEURS D'AIGUILLAGE

On distingue deux classes d'opérateurs d'aiguillage : les **multiplexeurs** et les **démultiplexeurs**.

2.4.1 Les multiplexeurs

Ce sont des opérateurs à $N = 2^n$ entrées d'information ou de données, $E_0, E_1, \cdots, E_{N-2}, E_{N-1}$, *n* entrées de sélection ou d'adressage $A_{n-1}, A_{n-2}, \cdots, A_0$, et une sortie *S*. L'affichage d'une adresse permet de sélectionner une entrée de données parmi *N*, pour l'aiguiller vers la sortie *S*.

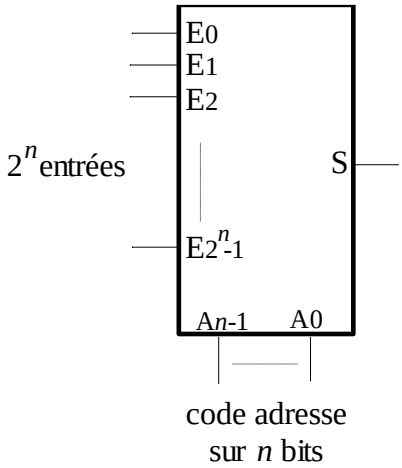


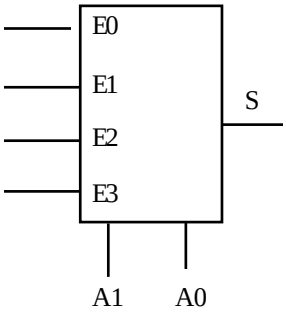
figure 4.11 : multiplexeur 2^n vers 1 ou $2^n:1$

La fonction logique réalisée par le multiplexeur est régie par l'équation suivante :

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

où m_i est le *i*ème minterme des variables logiques $A_{n-1}, A_{n-2}, \cdots, A_0$. Par exemple, $m_0 = \overline{A_{n-1}}.\overline{A_{n-2}} \cdots \overline{A_1}.\overline{A_0}$, $m_1 = \overline{A_{n-1}}.\overline{A_{n-2}} \cdots \overline{A_1}.A_0$, …

La représentation symbolique usuelle et l'équation de sortie d'un multiplexeur 4 vers 1 sont donnés par la figure 4.12.



$$S = \overline{A_1}.\overline{A_0}.E_0 + \overline{A_1}.A_0.E_1 + A_1.\overline{A_0}.E_2 + A_1.A_0.E_3$$

figure 4.12 : multiplexeur 4 vers 1

En général, les opérateurs de multiplexage disposent d'une entrée supplémentaire de validation *V* qui permet de valider ou d'inhiber la sortie *S*. Cette entrée est souvent utilisée afin d'augmenter les capacités de multiplexage à partir d'une fonctionnalité de base. Cet aspect est détaillé dans la section suivante.

2.4.1.1. Exemples de multiplexeurs

La figure 4.13 présente la réalisation d'un multiplexeur 2 vers 1 à l'aide de deux interrupteurs CMOS et d'un inverseur élémentaire. La structure inverseur CMOS a été présenté dans la partie Fonctions électroniques analogiques d'ELP111. Elle sera revue dans le cours C4.

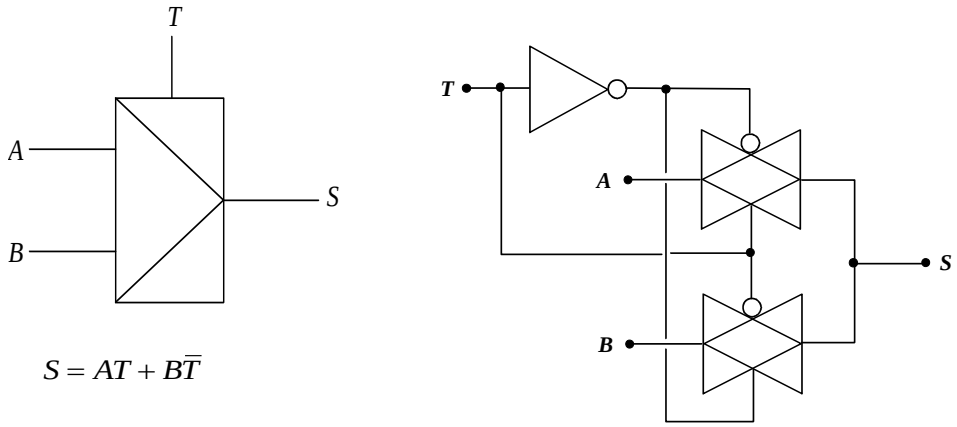


figure 4.13 : représentation usuelle et réalisation en logique CMOS d'un multiplexeur 2 vers 1

2.4.1.2. Multiplexage de mots

Certains types de multiplexeurs travaillent non pas sur des bits mais sur des mots de *x* bits. Les *x* bits qui forment le mot désigné par l'adresse sont traités simultanément et aiguillés vers les *x* bits de la sortie.

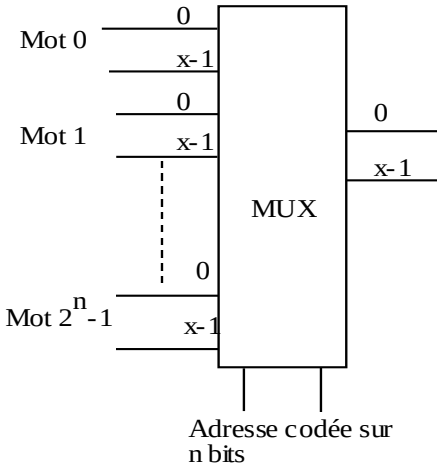


figure 4.14 : multiplexeur de 2^n mots de *x* bits

2.4.1.3. Applications des multiplexeurs

- **Conversion parallèle/série**

La fonction première du multiplexeur est d'aiguiller des informations provenant de *N* canaux vers un canal unique. Ce dispositif permet donc de convertir une information présente en parallèle sur les entrées d'information en une information de type série, si toutes les combinaisons d'adresses sont énumérées sur les entrées de sélection.

- **Génération de fonctions**

Les multiplexeurs peuvent également, tout comme les décodeurs, être utilisés pour la génération de fonctions logiques. En effet, toute fonction logique peut se décomposer sous la forme d'une somme de mintermes (cf. chapitre 2, section 3.1) :

$$\mathrm{f}(x_{n-1},\ldots x_1,x_0)=\sum_{i=0}^{2^n-1} d_i.m_i$$

où m_i est le *i*ème minterme des variables logiques $x_{n-1},\ldots,x_1,x_0$ et $d_i = 0$ ou 1.

Cette expression peut être rapprochée de l'équation de fonctionnement du multiplexeur. En effet, les mintermes m_i peuvent être identifiés avec les entrées d'adresse d'un multiplexeur. Les coefficients d_i sont identifiés avec les entrées d'information.

L'exemple du tableau 4.8 et de la figure 4.15 illustre la réalisation d'une fonction de deux variables à l'aide d'un multiplexeur 4 vers 1. Le passage de la table vérité au montage est immédiat.

A	*B*	F(*A*,*B*)
0	0	1
0	1	0
1	0	1
1	1	1

tableau 4.8 : table de vérité de la fonction F à réaliser

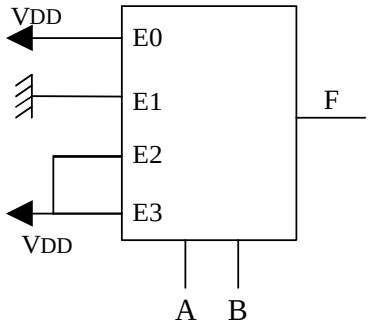


figure 4.15 : réalisation de la fonction F à l'aide d'un MUX 4 : 1

La réalisation de fonctions logiques à l'aide de multiplexeurs est en pratique utilisée dans certaines familles de circuits programmables (cf. ELP213 et SIT212). Par exemple, dans les FPGA (Field Programmable Gate Arrays) ACTEL, toutes les fonctions combinatoires sont réalisées à base de multiplexeurs.

2.4.2 Les démultiplexeurs

Ces opérateurs, comme leur nom l'indique, réalisent la fonction inverse des multiplexeurs. Ils possèdent une entrée d'information ou de données, *n* entrées de sélection (ou de commande), et $N = 2^n$ sorties. L'affichage d'une adresse sur les entrées de sélection permet de sélectionner la sortie vers laquelle l'entrée sera aiguillée. Le démultiplexeur peut, tout comme le multiplexeur, être doté d'une entrée de validation des sorties.

2.4.3 Exemples de démultiplexeurs

En pratique, les fabricants de circuits intégrés standard proposent dans leurs catalogues des circuits pouvant être utilisés en tant que décodeurs ou en tant que multiplexeurs (cf. références *XX* 137, *XX* 138, *XX* 139). A titre d'exemple, la table de vérité du tableau 4.9 et le schéma de la figure 4.16 illustrent le fonctionnement d'un décodeur/démultiplexeur 1 vers 4. Les circuits intégrés *XX* 139 comportent deux décodeurs/démultiplexeurs de ce type.

Entrées			Sorties			
Validation	Sélection					
$\overline{G}$	*S1*	*S0*	*Y*0	*Y*1	*Y*2	*Y*3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

tableau 4.9 : table de vérité d'un décodeur/démultiplexeur 1 vers 4 du circuit de référence *XX* 139

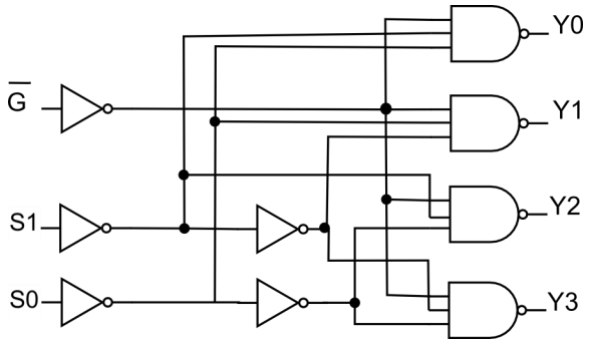


figure 4.16 : structure interne d'un décodeur/démultiplexeur du circuit *XX* 139

Lorsque ce circuit est utilisé **en décodage**, l'entrée $\overline{G}$, active à 0, est utilisée pour valider ou non les sorties. Dans ce mode de fonctionnement, les sorties sont également **actives à la valeur logique 0**.

En mode **démultiplexage**, *S1* et *S0* sont les entrées d'adresse et $\overline{G}$ joue le rôle d'entrée d'information. La valeur 0 de $\overline{G}$ est donc démultiplexée sur les sorties Y en fonction de la valeur de la sélection *S1S0*.

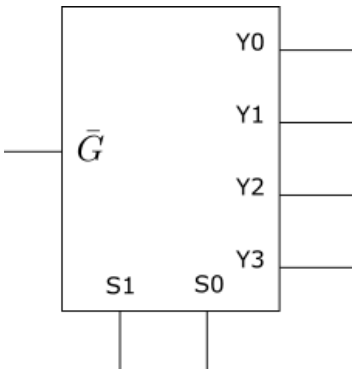


figure 4.17 : vue externe d'un décodeur/démultiplexeur du circuit *XX* 139

2.4.4 Applications des démultiplexeurs

Une des applications les plus classiques du démultiplexeur est la conversion d'une information présente en série en une information de type parallèle.

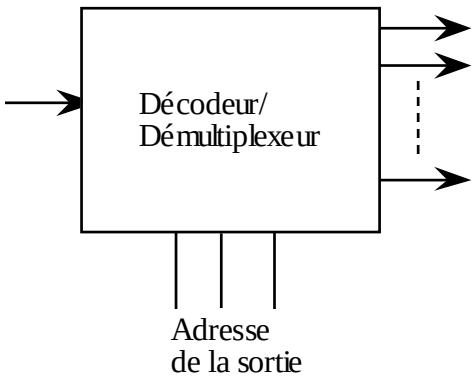


figure 4.18 : conversion série/parallèle

2.5 LES OPERATEURS DE COMPARAISON

2.5.1 Définition

On désigne par **opérateur de comparaison** ou **comparateur** un opérateur capable de détecter l'égalité ou l'inégalité de 2 nombres (comparateur simple) et éventuellement d'indiquer le plus grand ou le plus petit (comparateur complet). Le tableau 4.10 donne le résultat de la comparaison de 2 éléments binaires.

A	B	$A=B$	$A>B$	$A<B$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

tableau 4.10 : résultat de la comparaison de 2 bits A et B

Les équations logiques correspondantes sont les suivantes :

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

Dans le cas plus général de nombres binaires, deux nombres A et B, $A=A_{n-1}\cdots A_i\cdots A_0$ et $B=B_{n-1}\cdots B_i\cdots B_0$, sont égaux si tous les bits de même rang, A_i et B_i, sont égaux. L'équation de fonctionnement du circuit comparateur simple est alors la suivante :

$$S=\overline{A_0\oplus B_0}.\cdots.\overline{A_i\oplus B_i}.\cdots.\overline{A_{n-1}\oplus B_{n-1}}$$
$$S=\prod_{i=0}^{n-1}\overline{A_i\oplus B_i}$$

soit encore :

$$S=\overline{(A_0\oplus B_0)+(A_i\oplus B_i)+(A_{n-1}\oplus B_{n-1})}$$
$$S=\overline{\sum_{i=0}^{n-1}A_i\oplus B_i}$$

Cette détection peut être réalisée à l'aide d'opérateurs OU exclusif et d'une fonction NOR (figure 4.19).

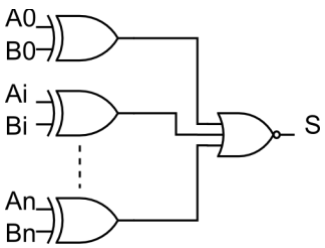


figure 4.19 : comparaison simple de 2 nombres *A* et *B*

Dans le cas d'une comparaison complète de deux nombres binaires *A* et *B*, les sorties $A < B$ et $A > B$ sont obtenues en effectuant une comparaison complète des bits de même rang, A_i et B_i, de façon prioritaire à partir des bits de poids forts (cf. tableau 4.11). Par exemple, si $A_n > B_n$ alors $A > B$, et si $A_n < B_n$ alors $A < B$. Mais si $A_n = B_n$, il est nécessaire de comparer les bits de rang $n-1$ pour décider, et ainsi de suite …

2.5.2 Exemple de comparateurs

Le circuit standard de référence *XX* 85 (figure 4.20) compare 2 mots de 4 bits *A* et *B*. Outre les entrées de données recevant les deux mots à comparer, il possède également trois entrées $A = B_{in}$, $A < B_{in}$, et $A > B_{in}$, permettant de cascader les comparateurs pour pouvoir comparer des nombres de plus de 4 bits. Si le comparateur est utilisé seul, les entrées $A = B_{in}$, $A < B_{in}$, et $A > B_{in}$ doivent être connectées respectivement à 1, 0, et 0. Le tableau 4.11 donne la table de vérité condensée de ce circuit.

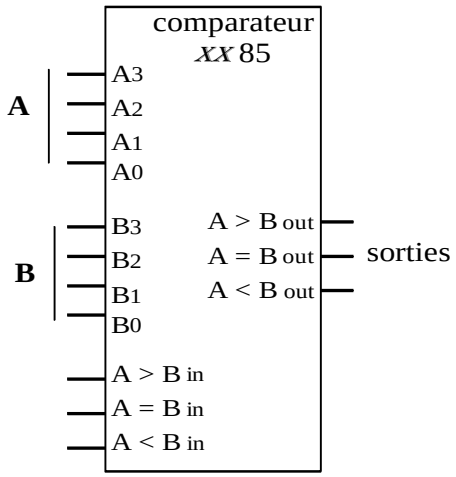


figure 4.20 : comparateur complet 4 bits *XX* 85

Entrées de données				Entrées de mise en cascade			Sorties		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$A > B_{in}$	$A = B_{in}$	$A < B_{in}$	$A > B_{out}$	$A = B_{out}$	$A < B_{out}$
$A_3 > B_3$	X	X	X	X	X	X	1	0	0
$A_3 < B_3$	X	X	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	1	0	0
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	0	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0	0	1	0

tableau 4.11 : table de vérité du comparateur 4 bits *XX* 85

A titre d'exemple, pour comparer deux nombres de 8 bits, il suffit de relier les sorties $A > B_{out}$, $A = B_{out}$, et $A < B_{out}$ du comparateur gérant les 4 bits de poids faibles aux entrées $A < B_{in}$, $A = B_{in}$, et $A > B_{in}$ du comparateur gérant les bits de poids forts. Dans ce cas, les valeurs logiques 010 sont appliquées sur les entrées $A < B_{in}$, $A = B_{in}$, et $A > B_{in}$ du premier comparateur.

3. LES EXERCICES D'APPLICATION

3.1 MUX A TOUT FAIRE

Soit un multiplexeur 8 vers 1. Nous souhaitons réaliser les 3 fonctions logiques combinatoires suivantes. Donner le schéma du circuit pour chacune d'entre elles, à partir du symbole du MUX 8 vers 1.

$f_1 = \bar{A}.\bar{B}.\bar{C} + \bar{A}.B.C$

$f_2 = A.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + A.B.C$ (à la maison)

$f_3 = \bar{A}.\bar{B}.\bar{C} + + \bar{A}.\bar{B}.C + A.B.\bar{C} + A.B.C$ (à la maison)

3.2 DECODEUR BINAIRE / DEMULTIPLEXEUR

Soit un **décodeur binaire** à 2 entrées A_0 et A_1 et à 4 sorties S_0, S_1, S_2, et S_3 tel qu'une seule sortie soit active à la fois et que le rang de la sortie active correspond à la valeur binaire présentée en entrée (ex : A_0=A_1=0 => S_0 =1).

1. Etablir la table de vérité de ce décodeur.
2. Réaliser ce décodeur à l'aide d'un circuit démultiplexeur (actif à 1), dont le symbole est donné ci-dessous.

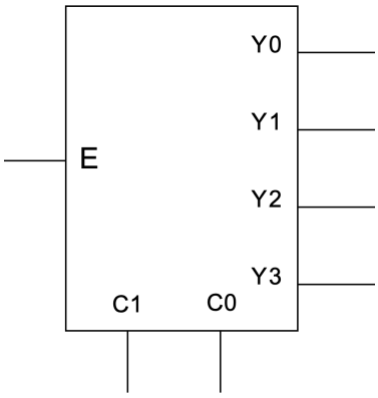


Figure. Symbole du démultiplexeur (actif à 1). 2 entrées de commande C1 et C0 et 4 sorties de Y0 à Y3. 1 entrée de donnée E.

3.3 COMPARATEUR DE MOTS BINAIRES

On souhaite réaliser un **comparateur** de deux mots de *n* bits (figure 1)

$A = (A_n \cdots A_i \cdots A_1)$ et $B = (B_n \cdots B_i \cdots B_1)$

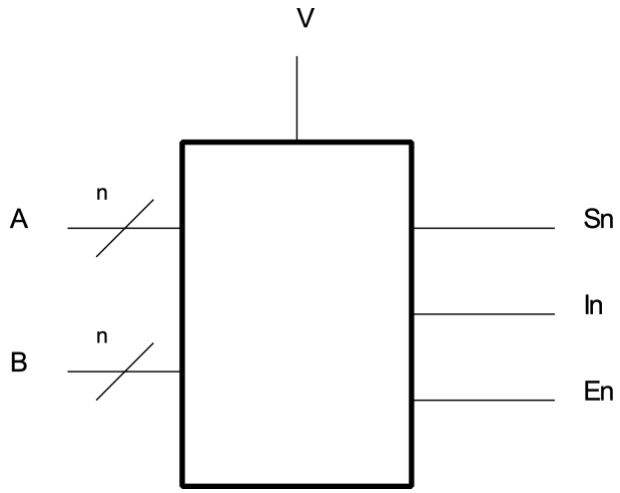


Figure. Symbole du comparateur.

Fonctionnement du comparateur :

- Si l'entrée de validation (*V*) vaut 0, alors les trois sorties (Egal : E_n), (Supérieur : S_n) et (Inférieur : I_n) sont égales à 0.
- Si l'entrée de validation (*V*) vaut 1, alors :
  - $S_n = 1$ ssi $A > B$
  - $I_n = 1$ ssi $A < B$
  - $E_n = 1$ ssi $A = B$

3.3.1 Comparateur élémentaire

Dans un premier temps, on souhaite réaliser un comparateur élémentaire de deux mots de 1 bit ($n = 1$).

- Etablir les équations des sorties S_1, I_1, et E_1 en fonction des entrées A_1, B_1, et V.
- Dessiner le schéma de ce comparateur à partir d'opérateurs élémentaires (INV, NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

3.3.2 Extension à 2 mots de 2 bits

On souhaite maintenant étendre l'amplitude du comparateur à deux mots de 2 bits.

1. Etablir les équations de S_2, I_2, et E_2 en fonction des bits A_2, A_1, B_2, B_1, et de l'entrée de validation *V*
2. Concevoir le schéma de ce comparateur en associant des comparateurs élémentaires et un minimum d'opérateurs (NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

3.3.3 Généralisation (à faire à la maison)

A partir des équations trouvées précédemment, établir les relations de récurrence ci-dessous :

$$S_n = \mathrm{f}(S_{n-1}, A_n, B_n, V)$$
$$I_n = \mathrm{g}(I_{n-1}, A_n, B_n, V)$$
$$E_n = \mathrm{h}(E_{n-1}, A_n, B_n, V)$$

3.4 FONCTION LOGIQUE COMBINATOIRE ET ASSURANCE (A FAIRE A LA MAISON)

Les conditions de délivrance de la police d'assurance n° 15 sont les suivantes :

- avoir souscrit à la police n° 10, être du sexe féminin et mariée,

ou

- n'avoir pas souscrit à la police n° 10, être du sexe masculin et marié,

ou

- avoir souscrit à la police n° 10, être marié(e) et âgé(e) de moins de 25 ans,

ou

- être marié(e) et avoir plus de 25 ans,

ou

- être du sexe féminin et âgée de moins de 25 ans.

1. Donner la table de vérité de la fonction logique f telle que f=1 si la police d'assurance n°15 est délivrée.
2. Donner la fonction logique en utilisant la méthode de simplification de Karnaugh.
3. Donner le circuit à base d'opérateurs logiques combinatoires (OR, AND)

4. POUR ALLER PLUS LOIN

4.1 AUTRES CODES BINAIRES

4.1.1 Code excédent 3 ou excess 3

Le code excédent 3 est un **code non pondéré** qui utilise, tout comme le code BCD, 10 mots de codes, auxquels on fait correspondre les 10 chiffres décimaux.

$N_{(10)}$	$N_{(XS3)}$
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

tableau 12 : code excédent 3

Il est obtenu en décalant le code binaire de trois lignes vers le haut. Ce code peut être intéressant pour effectuer des soustractions car le complément à 1 de la représentation binaire d'un chiffre correspond au complément à 9 de ce chiffre. Ainsi, toute opération de soustraction se ramène à une addition.

Par exemple $5_{(XS3)} = 1000$, son complément à 1 est obtenu par inversion des bits, $\overline{5}_{(XS3)} = 0111 = 4_{(XS3)}$, le résultat en excédent 3 est le nombre 4, qui est le complément à 9 de 5 en décimal. Ainsi, pour faire une soustraction, il suffit d'ajouter le complément à 1 du nombre à retrancher, puis 1. Par exemple, $(7-5)_{(XS3)} = 7_{(XS3)} + \overline{5}_{(XS3)} + 1_{(XS3)} = 1010 + 0111 + 0100 = 0101 = 2_{(XS3)}$.

Le code excédent 3 ne présente pas d'intérêt en addition.

4.1.2 Codes redondants

Il existe un ensemble de codes conçus pour pouvoir **détecter**, voire **corriger des erreurs dans des messages binaires**. Leur principe repose sur l'insertion de données redondantes dans l'information initiale. Leur étude approfondie relève du **domaine des communications numériques**, et ne sera pas traité dans ce cours. Nous citerons cependant quelques exemples simples de codes redondants, les codes *p* parmi *n* et les codes de contrôle de parité.

4.1.2.1. Code p parmi n

Ce code est constitué de $\mathbf{C}_n^p = \frac{n!}{p!(n-p)!}$ mots de code. Chaque mot de code est codé sur *n* bits et contient exactement *p* "1" et (*n* - *p*) "0". Par exemple, le code 2 parmi 5

(**Error! Reference source not found.**) est constitué de 10 mots de codes et permet de coder les chiffres décimaux.

$N_{(10)}$	$N_{(2\ parmi\ 5)}$				
0	0	0	0	1	1
1	1	1	0	0	0
2	1	0	1	0	0
3	0	1	1	0	0
4	1	0	0	1	0
5	0	1	0	1	0
6	0	0	1	1	0
7	1	0	0	0	1
8	0	1	0	0	1
9	0	0	1	0	1

tableau 13 : code 2 parmi 5

L'utilisation de ce code permet, à la réception d'une information, de vérifier par comptage du nombre de 1 si une erreur s'est introduite dans l'information transmise. Dans le cas où plus d'une erreur s'est glissée dans un mot de code, la détection n'est pas assurée dans tous les cas. Ce code ne permet pas non plus de trouver la place de l'erreur, donc de la corriger. D'autre part, le décodage des combinaisons est particulièrement simple, car il ne porte que sur 2 bits par combinaison.

4.1.2.2. Code contrôle de parité

Le codage d'un mot de *n* bits par contrôle de parité consiste à y adjoindre un $(n+1)^{ème}$ bit dont le rôle est de rendre systématiquement pair ou impair le nombre total de 1 contenus dans l'information codée. Si une erreur se glisse dans l'information, le nombre de 1 devient impair et l'erreur est détectée. Ce code ne permet pas non plus de corriger les erreurs.

4.2 DECODEURS DE GRANDE CAPACITE (NB D'ENTREES SUPERIEUR A 3)

4.2.1 Matrice de décodage

Dès que l'on souhaite réaliser un décodeur de plus de 3 bits en entrée, il est conseillé d'utiliser une structure en matrice (figure 4.21) pour obtenir un opérateur de complexité matérielle minimale.

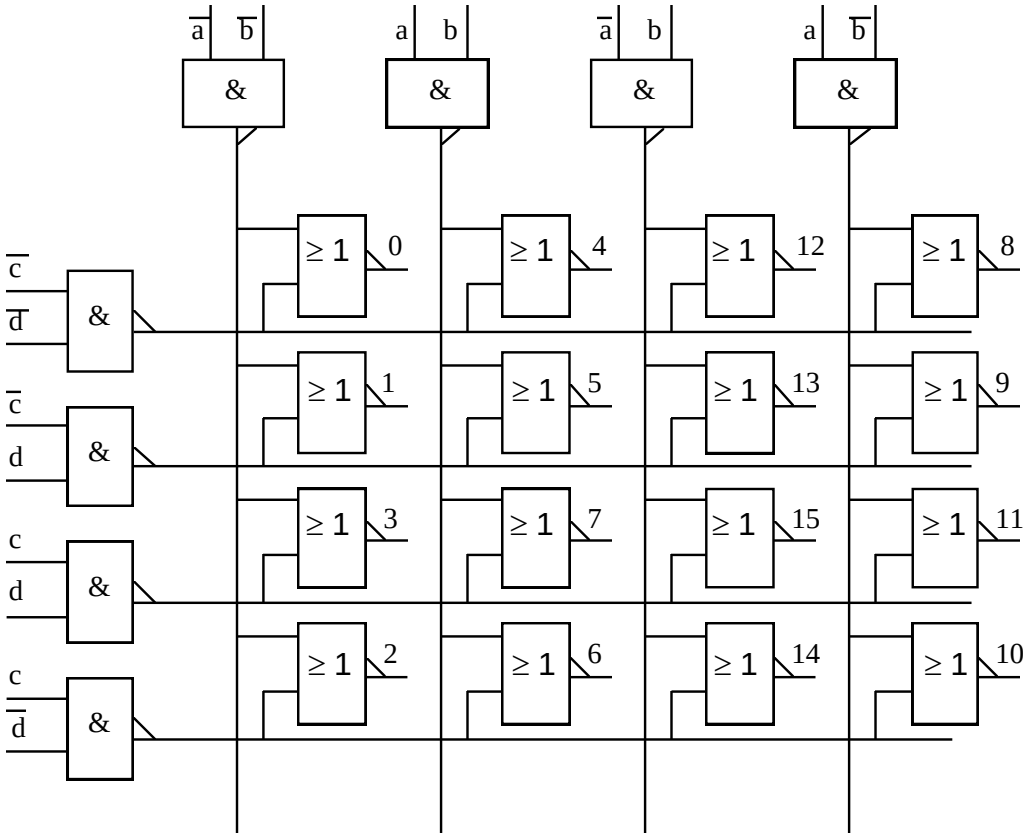


figure 4.21 : matrice de décodage de 4 bits

Dans la matrice de la figure 4.21, chaque case du tableau de Karnaugh ou chaque ligne de la table de vérité correspond à une structure matérielle composée de 2 portes NAND et d'une porte NOR. L'utilisation multiple des mintermes des variables *c* et *d*, ainsi que des variables *a* et *b*, permet réduire la complexité matérielle par rapport à un circuit décodant les seize sorties indépendamment les unes des autres. On notera également que tous les chemins entre entrées et sorties traversent des couches logiques identiques.

4.2.2 Association de décodeurs

Il est possible d'associer plusieurs décodeurs afin d'augmenter les capacités d'un système. La figure 4.22 montre comment deux décodeurs 1 parmi 16 (référence *XX* 154 des catalogues de circuits standard) sont associés pour former un décodeur de 1 parmi 32 (fonctionnalité qui n'existe pas chez les fabricants de circuits standard, mais qui peut être directement réalisée dans un circuit spécifique). Outre les entrées *A*, *B, C,* et *D*, le circuit *XX* 154 possède deux entrées de validation $\overline{G1}$ et $\overline{G2}$. La notation barrée ainsi que les ronds sur ces entrées signifient qu'elles sont actives au niveau logique 0 : les sorties du décodeur ne sont validées que lorsque $\overline{G1} = \overline{G2} = 0$.

Si l'entrée *a* vaut 0, les entrées $\overline{G1}$ et $\overline{G2}$ du décodeur **1** sont actives, celles du décodeur **2** inactives. Le décodeur **1** est validé, et le décodeur **2** inhibé. Les sorties $S0$ à $S15$ sont alors représentatives des entrées *b c d e*. Si l'entrée *a* vaut 1, seul le décodeur **2** est validé et les entrées *b*, *c*, *d*, et *e* sont décodées sur les sorties $S16$ à $S31$.

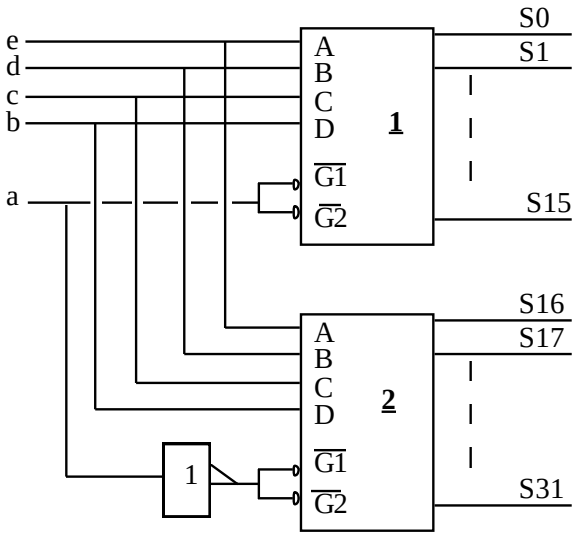


figure 4.22 : réalisation d'un décodeur 1 parmi 32 à partir de 2 décodeurs 1 parmi 16

1.1.1. Décodage à plusieurs niveaux

Afin de réaliser des circuits décodeurs de grande capacité, il est possible d'associer des décodeurs de base sur plusieurs niveaux.

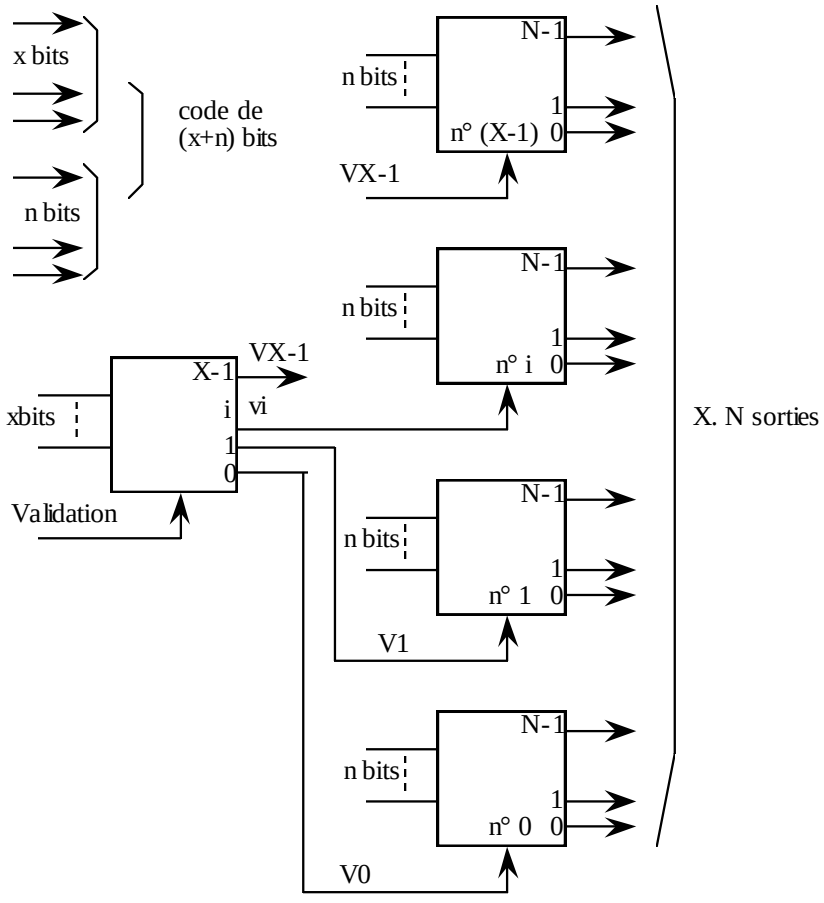


figure 4.23 : décodage de 1 parmi $X.N$ sur deux niveaux

Dans l'exemple de la figure 4.23, le dispositif, qui possède deux niveaux de décodeurs, accepte $x+n$ entrées et délivre $X.N$ sorties, avec $X=2^x$ et $N=2^n$. Le premier niveau

est constitué d'un décodeur de type 1 parmi *X*. Celui-ci reçoit *x* bits en entrées et fournit sur une des sorties $V0$ à $VX-1$ un signal de sélection qui est utilisé pour valider un des *X* décodeurs du second niveau. Ces *X* décodeurs reçoivent en entrée les *n* bits restants et possèdent chacun *N* sorties. Le signal *Validation* appliqué sur le décodeur de premier niveau permet d'autoriser ou non le décodage.

4.3 ASSOCIATION DE MULTIPLEXEURS

Dans l'exemple de la figure 4.24, deux multiplexeurs élémentaires 8 vers 1 avec entrées de validation sont associés afin d'obtenir un dispositif capable d'aiguiller 1 entrée parmi 16 vers la sortie. On remarque que le fil d'adresse A_3 joue un rôle particulier. Si $A_3 = 0$, le multiplexeur **1** est validé, et le multiplexeur **2**, qui correspond aux 8 bits de poids forts, est inhibé (sa sortie est forcée à 0). Les adresses A_0 à A_2 permettent donc de choisir une des 8 entrées du premier multiplexeur. Lorsque le signal A_3 vaut 1, le premier multiplexeur est inhibé (sortie à 0) et le second validé. Une porte OU permet de recueillir la sortie du multiplexeur validé.

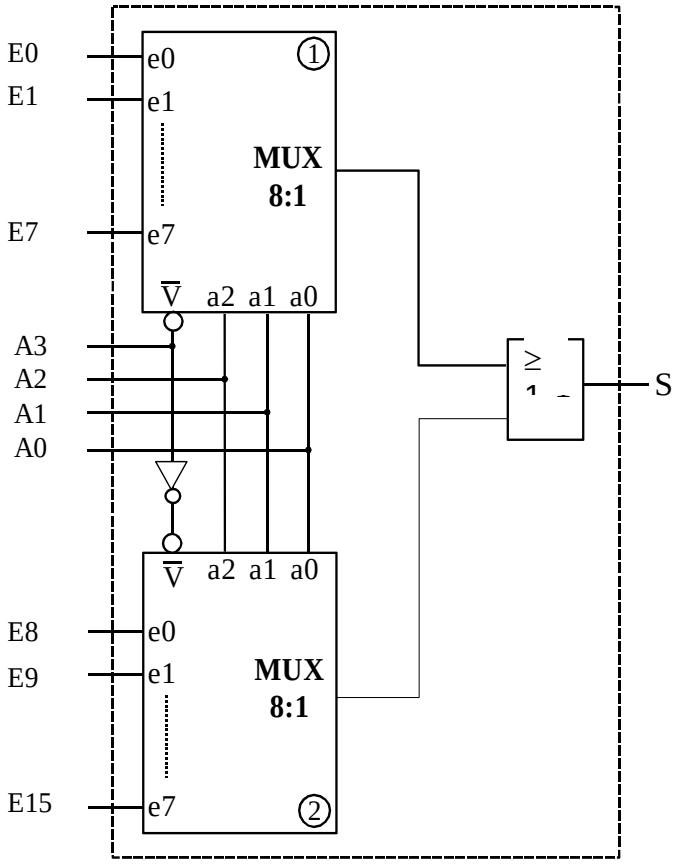


figure 4.24 : réalisation d'un multiplexeur 16 vers 1 à partir de deux multiplexeurs 8 vers 1