Les systèmes combinatoires

I. Définition

On appelle **système combinatoire** tout système numérique dont les sorties sont exclusivement définies à partir des variables d'entrée (***Figure 1***).

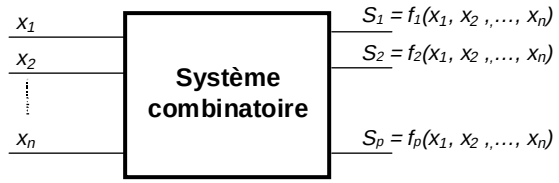


Figure 1 : système combinatoire générant p sorties S_i à partir des n variables d'entrée x_i.

II. Les fonctions combinatoires

II.1. Les opérateurs logiques élémentaires et leurs déclinaisons

Les opérateurs logiques élémentaires sont disponibles sous forme de circuits intégrés de taille et de nombre de broches variable suivant la quantité ou le nombre d'entrée des opérateurs.

II.1.1. Opérateurs NON

Les opérateurs NON ne possèdent évidemment qu'une seule entrée. Les sorties de ces portes peuvent être **amplifiées** (*buffer*) ou à **collecteur ouvert** (CO). Des seuils de déclenchement peuvent caractériser ces entrées : on dit alors que ce sont des entrées **trigger de Schmitt**.

II.1.2. Opérateurs ET

Sur la base des opérateurs ET à deux entrées, on en construit d'autres à 3, 4 et 8 entrées. Des portes dérivées font appel à des modes particuliers des entrées et des sorties.

II.1.3. Opérateurs OU

Le principe reste identique pour les opérateurs OU à 2, 3 ou 4 entrées.

II.1.4. Opérateurs NAND

Plus nombreuses, les déclinaisons de ces opérateurs offrent des portes à 2, 3, 4, 8 ou 13 entrées.

II.1.5. Opérateurs NOR

Les déclinaisons de ces opérateurs offrent des portes à 2, 3, 4 ou 5 entrées.

II.1.6. Opérateurs XOR

Différentes possibilités, mais toujours avec deux entrées.

II.2. Décodeurs et transcodeurs

II.2.1. Définitions

Un transcodeur permet de convertir une combinaison *n* de bits en une autre de *p* bits. La différence entre un transcodeur et un décodeur réside dans l'application de l'objet défini plutôt que dans la fonction réalisée. Par exemple, on parle plutôt de transcodeur Gray/binaire mais on utilise un décodeur pour un système d'affichage sept segments.

II.2.2. Décodeurs et transcodeurs usuels

Parmi les composants les plus courants retenons ceux-ci : binaire vers décimal (4 vers 10), Gray excès 3 vers décimal, BCD vers décimal ou binaire vers hexadécimal. (4 vers 16)

Un décodeur 7 segments permet, à partir d'une combinaison sur 4 bits, de piloter un afficheur sept segments en décimal (digits de 0 à 9) ou en hexadécimal (digits de 0 à F).

II.3. Multiplexeurs et démultiplexeurs

II.3.1. Définition

Un **multiplexeur** (*multiplexer*) (***Figure 2***) permet de transmettre en sortie un bit de donnée dont le rang est indiqué par un mot de sélection de *n* bits (le sélecteur), parmi les 2^n bits d'entrée. C'est une sorte d'aiguillage de 2^n vers 1 voie.

Un **démultiplexeur** (*demultiplexer*) (***Figure 3***) effectue l'opération inverse en assignant à la sortie de rang indiqué par le sélecteur, le bit de donnée en entrée. C'est une sorte d'aiguillage 1 voie vers 2^n voies.

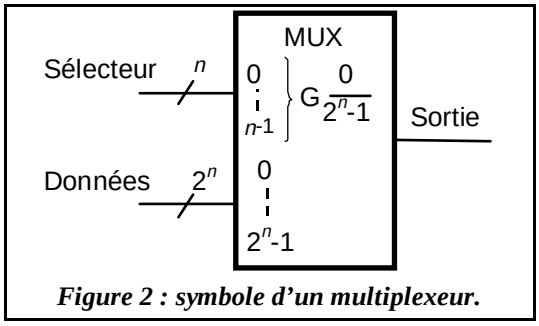


Figure 2 : symbole d'un multiplexeur.

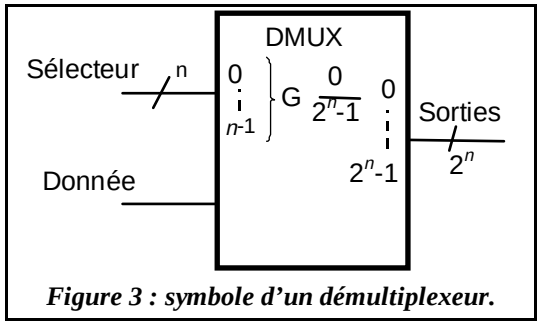


Figure 3 : symbole d'un démultiplexeur.

II.3.2. Applications

Un multiplexeur permet de synthétiser aisément une fonction logique ou d'aiguiller des informations au sein d'un système numérique.

II.3.3. Multiplexeurs et démultiplexeurs usuels

Multiplexeurs : 2 , 4, 8 ou 16 vers 1.

Démultiplexeurs :1 vers 2, 4, 8 ou 16.

II.4. Les opérations logiques sur mots binaires

II.4.1. Définitions

On appelle opérateur logique, une fonction combinatoire réalisant une opération logique entre deux mots binaires codés sur *n* bits.

Les opérations logiques sont :

- **la complémentation**

Exemple : NON(`1011001110`) = NON(`0000001011001110`)= NON(`$02CE`) = `1111110100110001` = `$FD31`.

Remarque : On ajoute des zéros en tête pour atteindre un codage sur un poids binaire de bits.

- **La somme logique (ou)**

Exemple : `101110010+1010010001` (lire ... ou ...) = `1111110011`
soit `$172 + $291=$3F3`

- **Le produit logique (et)**

Exemple : `101110010.1010010001` (lire ... et ...) = `10000`
soit `$172 . $291=$10`

- **La somme exclusive logique (xor)**

Exemple : `101110010` ⊕ `1010010001` (lire ... xor ...) = `1111100011`
soit `$172` ⊕ `$291=$3E3`

II.5. Opérateurs arithmétiques

II.5.1. Définitions

On appelle **opérateur arithmétique**, une fonction combinatoire réalisant une opération arithmétique entre deux mots binaires de *n* bits (codage pondéré). Les opérations arithmétiques de base sont l'addition et la multiplication. La soustraction est obtenue par addition de l'opérande codée en complément à 2.

II.5.2. L'additionneur

L'**additionneur** (*adder*) numérique réalise la somme de deux nombres de *n* bits en tenant compte éventuellement d'une retenue extérieure. Le résultat est fourni sur *n* + 1 bits, le dernier bit est la retenue.

II.5.3. Le multiplicateur

Le **multiplicateur** (*multiplier*) numérique réalise le produit de deux nombres de *n* bits en tenant compte d'une retenue extérieure. Le résultat est fourni sur 2*n* bits.

II.6. Unité arithmétique et logique

Une **unité arithmétique et logique** (UAL ou ALU, *Arithmetic and Logic Unit* en anglo-américain) permet tous les types d'opérations logiques et arithmétiques. A partir de 2 mots binaires de *n* bits, le mot binaire de sortie représente une opération logique ou arithmétique spécifiée dans une table de fonctionnement. Le résultat apparaît en sortie sur 2*n* bits.

Exemple : ALU 4 bits et générateur de fonction 74181.

II.7. Autres fonctions

II.7.1. Comparateur d'égalité

Un **comparateur d'égalité** fourni la valeur 1 si deux mots binaires de *n* bits en entrée sont identiques (au bit près). Dans le cas contraire la valeur 0 est fournie.

II.7.2. Comparateur arithmétique

Un **comparateur** (*magnitude comparator*) est une généralisation du comparateur d'égalité mais indiquant, grâce à deux sorties supplémentaires, la position relative d'un mot d'entrée (*A*) par rapport à l'autre (*B*). Cet opérateur possède donc trois sorties ($A < B$), ($A = B$) et ($A > B$).

III. Synthèse des systèmes logiques combinatoires

A partir du cahier des charges, on construit la table de vérité. Une mise en équation directe permet d'exprimer la sortie à partir des différentes variables d'entrée, mais la recherche de l'expression logique peut être optimisée à l'aide des **tables de Karnaugh**.

Disposant de la fonction, la synthèse devient possible de manière directe à l'aide d'opérateurs logiques, de multiplexeurs ou à l'aide de circuits spécialisés, les **réseaux logiques programmables**.

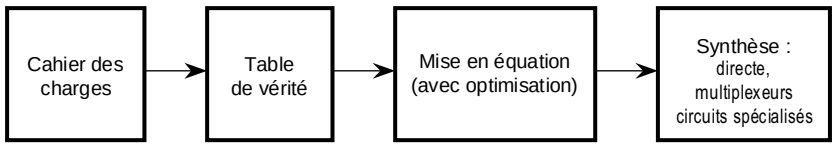


Figure 4 : descriptif de la synthèse d'une fonction logique combinatoire.

III.1. Synthèse d'une expression booléenne par table de Karnaugh

III.1.1. Introduction

III.1.1.1. Insuffisance et inadaptation du code binaire naturel

Un processus réel ne voit jamais une simultanéité parfaite des événements : deux variables logiques — ou davantage— ne peuvent changer d'état simultanément. On observe pourtant ce phénomène lors du passage entre les valeurs 2 et 3 codées en binaire naturel (***Figure 5***). Entre le début du changement et la prise d'effet du nouvel état, plusieurs combinaisons apparaissent de manière aléatoire suivant la rapidité d'évolution des variables (ici 111, puis 110 et enfin 100).

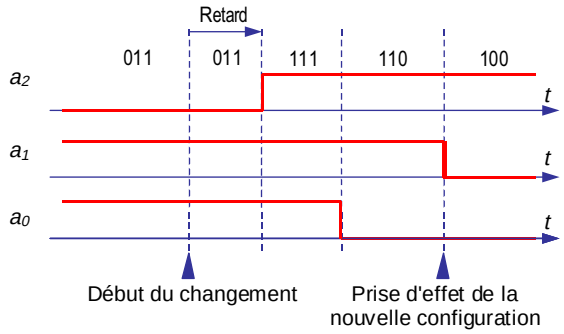


Figure 5 : chronogrammes d'évolution pour le passage de 011 à 100.

Il résulte de cette observation que le codage en binaire naturel n'est pas adapté à la description des phénomènes logiques réels. Il est nécessaire de mettre en place un code qui ne fait évoluer qu'un seul bit à la fois : le **code binaire réfléchi** ou **code Gray**.

III.1.1.2. Le code binaire réfléchi

Le code binaire réfléchi est construit par réflexions partielles (d'où son nom). Le ***Tableau 1*** le décrit sur 4 bits.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MSB r_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
r_2	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
r_1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
LSB r_0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

Tableau 1 : code binaire réfléchi.

III.1.2. Les tables de Karnaugh

III.1.2.1. Mise en place

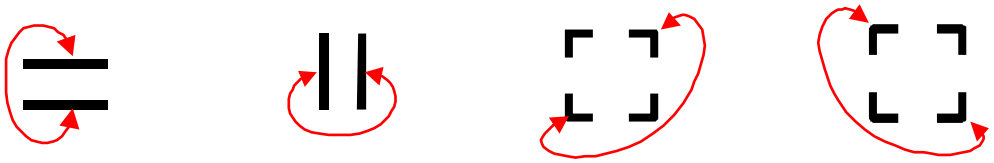
Par des moyens « semi-graphiques », la **méthode de Karnaugh** permet d'optimiser le nombre de termes en opérant des simplifications directes sans écrire tous les mintermes.

- table à deux entrées (lignes-colonnes) ;
- on équilibre les variables sur les lignes et les colonnes pour s'approcher d'un tableau carré ;
- les lignes et les colonnes sont codées en code Gray ;
- chaque case contient l'état de la sortie (0 ou 1) pour les entrées correspondantes.

III.1.2.2. Règles de simplification

Constituer des groupes de « 1 »

- ces groupes de taille maximale, doivent contenir un nombre de cases égal à un poids binaire (1, 2, 4, 8, 16,...) et doivent respecter les symétries de la table ;
- les bords de la table sont adjacents, ce qui permet d'élargir les possibilités de regroupement grace à des repliements (***Figure 6***).
- Un regroupement constitue un ensemble de mintermes (cases) liés par l'opération OU. Du fait de la symétrie, des variables se simplifient deux à deux. Un regroupement constitue donc une expression logique simplifiée.
- Pour extraire cette expression, on ne retient que les variables dont l'état logique n'est pas modifié par déplacement de case en case à l'intérieur du groupement.



Les bords haut et bas sont adjacents.

Les bords droit et gauche sont adjacents.

En conséquence les coins sont adjacents (deux à deux ou en combiné)

Figure 6 : illustration des règles de repliement

Couvrir tous les « 1 » même avec chevauchements

Une ou plusieurs cases peuvent être communes à plusieurs regroupements. On dit que l'on effectue des **chevauchements** pour augmenter la taille des groupes. Cependant les inclusions ne sont pas autorisées.

La confection des groupes cesse lorsque tous les 1 appartiennent à au moins l'un d'entre eux. Toutes les expressions trouvées sont sommées (OU logique) pour constituer l'équation de la sortie considérée.

> **Remarque** : suivant la forme du résultat souhaité (canonique 1 ou 2), on extrait les « 1 » (équation de la fonction) ou les « 0 » (équation du complément logique de la fonction).

III.2. Réalisation à l'aide d'opérateurs logiques

Divers modes de synthèse seront abordés autour de la fonction exemple $S = (\overline{a+b}).c + a.\overline{d}$.

III.2.1. Synthèse directe

Dans ce mode, on utilise directement les opérateurs correspondant à l'expression logique pour fournir le logigramme de la ***Figure 7***.

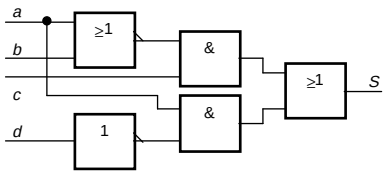


Figure 7 : résultat de la synthèse directe.

III.2.2. Emploi exclusif d'opérateurs NAND

L'opérateur NAND est universel, c'est à dire qu'une fonction peut s'exprimer uniquement avec ces opérateurs. Pour ne faire apparaître que des NAND, on écrit l'expression sous la première forme canonique (somme de produits) puis on complémente. En appliquant les théorèmes de De Morgan plusieurs fois de suite, on obtient une expression exclusivement en NAND.

$$S = (\overline{a+b}).c + a.\overline{d} = \overline{a}\,\overline{b}\,c + a.\overline{d} = \overline{\overline{\overline{a}\,\overline{b}\,c + a.\overline{d}}} = \overline{\overline{\overline{a}\,\overline{b}\,c}.\overline{a.\overline{d}}} = (\overline{a} \uparrow \overline{b} \uparrow c) \uparrow (a \uparrow \overline{d})$$

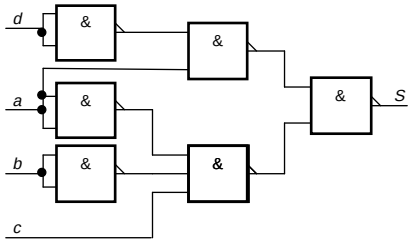


Figure 8 : représentation à l'aide unique de NAND.

III.2.3. Emploi exclusif d'opérateurs NOR

L'opérateur NOR est lui aussi universel. On écrit l'expression sous la deuxième forme canonique (produit de sommes) puis on complémente. En appliquant les théorèmes de De Morgan plusieurs fois de suite, on obtient une expression exclusivement en NOR.

$$S = (\overline{a+b}).c + a.\overline{d} = \overline{a}\,\overline{b}\,c + a.\overline{d} = (\overline{a}+a)(\overline{b}+a)(c+a)(\overline{a}+\overline{d})(\overline{b}+\overline{d})(c+\overline{d})$$

$$S = \overline{\overline{(\overline{b}+a)} + \overline{(c+a)} + \overline{(\overline{a}+\overline{d})} + \overline{(\overline{b}+\overline{d})} + \overline{(c+\overline{d})}} = (a \downarrow \overline{b}) \downarrow (a \downarrow c) \downarrow (\overline{a} \downarrow \overline{d}) \downarrow (\overline{b} \downarrow \overline{d}) \downarrow (c \downarrow \overline{d})$$

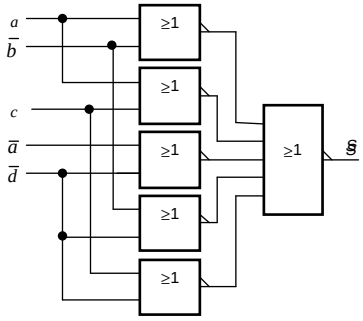


Figure 9 : représentation unique à l'aide de NOR.

III.3. Réalisation à l'aide de multiplexeurs

Le multiplexeur rend disponible en sortie l'une des 2^n entrées (données de la table de vérité) choisie par une adresse fixée par le sélecteur. Ces adresses construisent les 2^n mintermes qui sont validés par les données pour apparaître en sortie.

A titre d'exemple synthétisons la fonction S décrite par la table de vérité du ***Tableau 2***. Le résultat apparaît à la ***Figure 10***. On remarquera l'inutilité d'extraire l'expression logique et la nécessité d'ajouter une porte NON pour prendre en compte l'activité à l'état bas du multiplexeur.

a	*b*	*c*	*S*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tableau 2

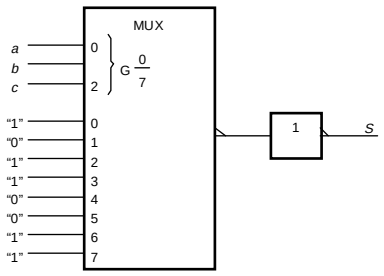


Figure 10 : la fonction synthétisée par multiplexeur '152.

III.4. Utilisation de circuits programmables spécialisés

III.4.1. Présentation et définitions

Les besoins en fonctions combinatoires toujours spécifiques au cahier des charges à résoudre, relayés par les progrès de l'intégration des circuits intégrés et la pression des utilisateurs ont incités les constructeurs à développer des produits de faible encombrement, faciles à mettre en œuvre et répondant à une large étendue de besoins.

L'éventail de ce type de circuits programmables est très large. En voici la liste classée par ordre de complexité décroissante et leur principe de réalisation technologique.

- circuits **ASIC** (*Application Specific Integrated Circuit*), les substrats sont préparés pour une application spécifique. Ce choix est particulièrement justifié dans les grandes séries.
- circuits **précaractérisés**, résultant d'un assemblage de fonctions prédéfinies que l'utilisateur assemble pour être intégrées par le fabricant. Le circuit livré ne nécessite plus d'opération.
- circuits **prédiffusés masquables**, des blocs de fonctions logiques ont été implantées par le constructeur qui, à la demande de l'utilisateur, diffuse une dernière couche d'aluminium matérialisant les liaisons spécifiées.
- circuits **prédiffusés programmables** (FPGA, *Field Programmable Gate Array*), reprend le principe du prédiffusé masquable mais les interconnexions sont réalisées par destruction ou programmation de liaisons fusibles.
- Circuits **PLD** (*Programmable Logic Device*), comportant des éléments logiques dont le câblage est terminé par rupture de liaisons fusibles.
- Circuits **EPLD** (*Erasable Programmable Logic Device*), pour lesquels la liaison est reprogrammable (bien souvent par le biais d'un logiciel) ;
- **PLA** (*Programmable Logic Array*), ce sont des circuits comportant des portes ET programmables et des OU fixes ;
- **GAL** (*Generic Array Logic*), ce sont des PAL CMOS reprogrammables.

III.4.2. Principe de programmation d'un PAL

Les liaisons programmables sont fusibles. Lorsque le PAL est vierge toutes les liaisons existent. Programmer le composant consiste à faire fondre certains fusibles pour enlever les liaisons associées. Cette opération est effectuée à l'aide d'un programmateur qui reçoit les informations d'un logiciel. Bien souvent celui-ci offre la possibilité de réaliser son étude en effectuant les schémas et leur simulation.

Exemple : programmer la fonction $S = I_1\bar{I}_2 + \bar{I}_1 I_2$ (XOR) donne le résultat de la ***Figure 12***.

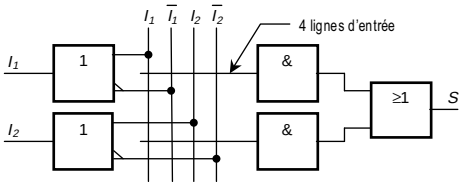


Figure 11 : extrait de PAL. Les intersections sont des liaisons destructibles (fusibles).

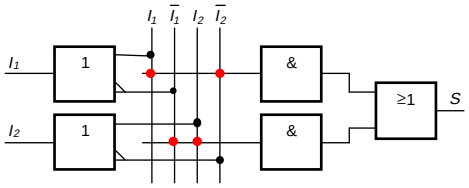


Figure 12 : les liaisons conservées sont marquées (opération XOR).

III.4.3. Eléments sur le PAL TIB PAL16L8-5C (Texas Instruments)

Ce document ne fournit que des éléments d'information. La notice des composants est plus complète. Ici, le symbole IEC apparaît à la ***Figure 13*** et la structure interne à la ***Figure 14***.

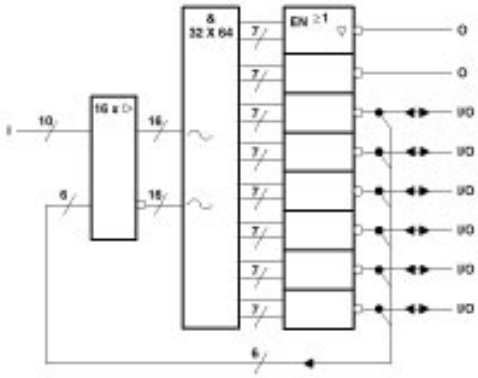


Figure 13 : symbole IEC du PAL16L8-5C

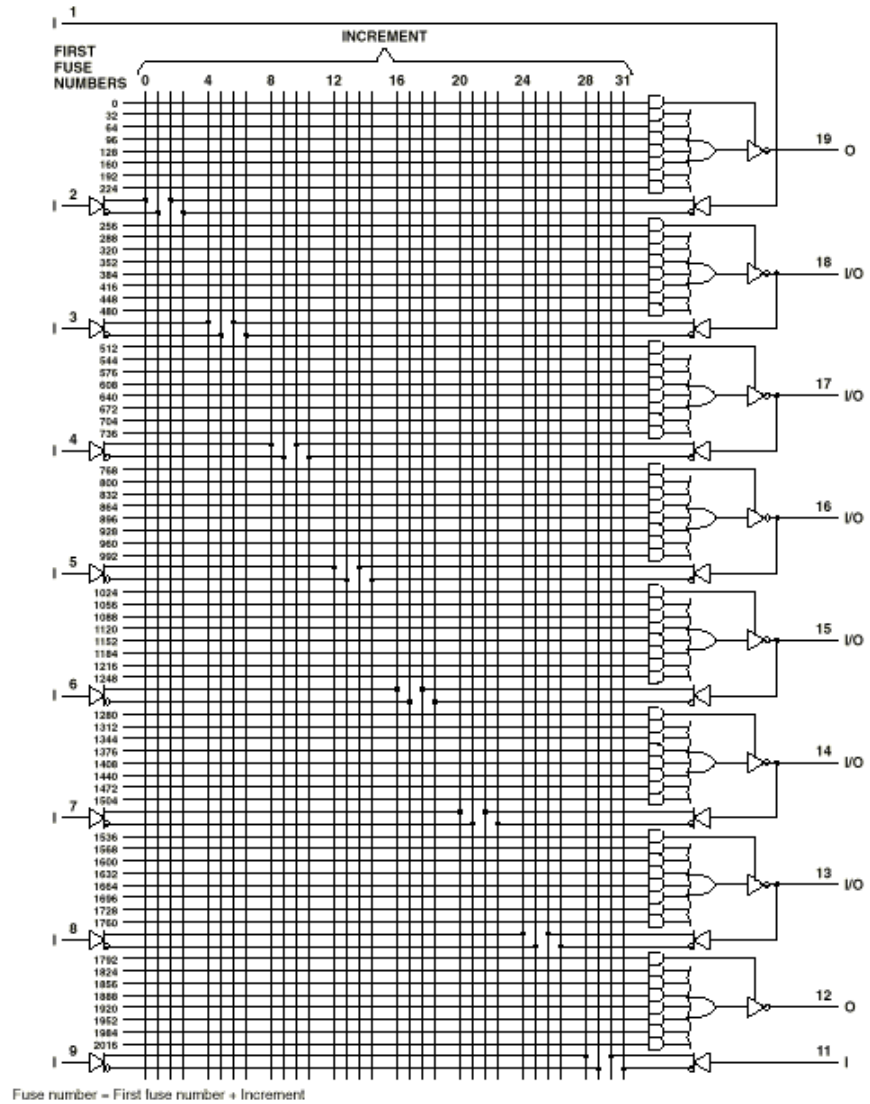


Figure 14 : structure interne du PAL16L8-5C.