

# Mémoires mortes et logique programmable

## VI.1 Mémoires mortes

Nous avons consacré le chapitre précédent à l'étude des mémoires vives, avec accès en lecture et écriture mais volatiles. Cependant pour certaines applications, il est nécessaire de pouvoir conserver des informations (programmes, données, ...) de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou mémoires à lecture seule (ROM : Read Only Memory). Dans ce cas les informations contenues en mémoire ne peuvent être accédées qu'en lecture. Les informations ne peuvent être modifiées ni intentionnellement, ni accidentellement. L'intérêt de ces mémoires réside dans le fait que cette mémoire permanente est accessible à la grande vitesse propre aux circuits électroniques à semi-conducteur.

### VI.1.a Principe

Extérieurement une mémoire morte ressemble à un bloc mémoire sans ligne d'entrée ni commande d'écriture. Ce système peut se restreindre à  $k$  lignes d'adresse,  $n$  lignes de sortie et un signal de sélection du circuit.

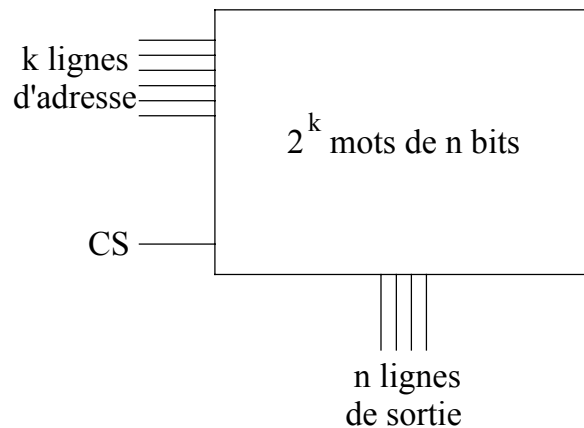


Figure 1

Une telle mémoire morte contient  $2^k$  mots de  $n$  bits. Nous pouvons considérer que chacun de ces mots porte un code. Ainsi une mémoire morte peut être réalisée avec un encodeur de  $2^k$  lignes et  $n$  sorties associé à un décodeur pour activer une de ces  $2^k$  lignes (fig. 2). Nous avons vu qu'un

encodeur peut être obtenu avec des diodes. La figure 3 donne l'exemple simple d'une mémoire morte de 4 mots de 4 bits, correspondant à la table de vérité suivante :

$X_1$	$X_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	1	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	1	1	1	1	1

Table 1

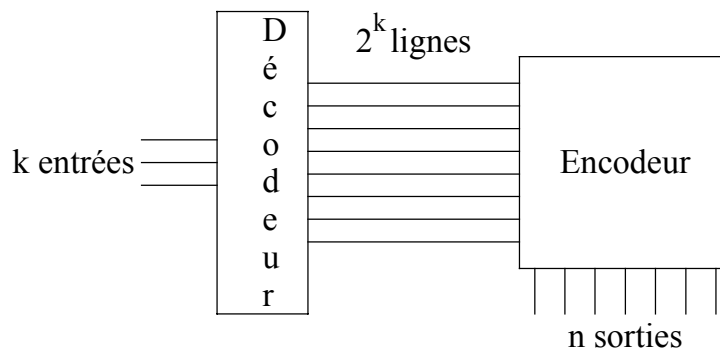


Figure 2

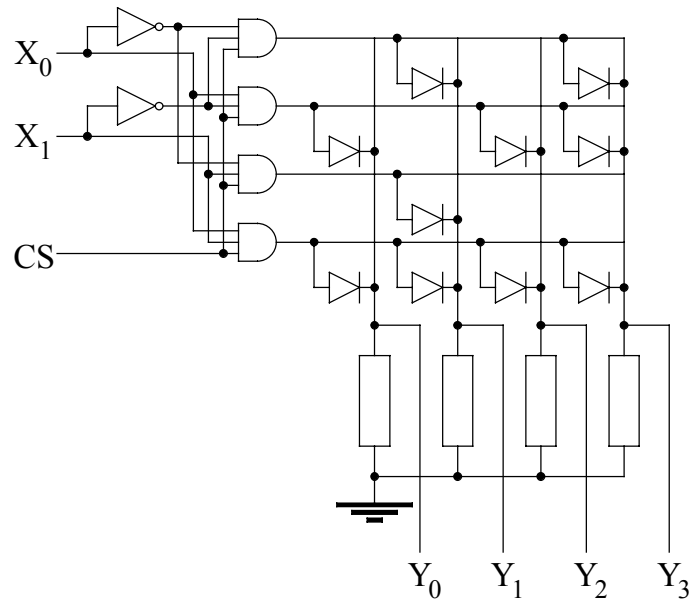


Figure 3

Tout comme pour les mémoires vives cette organisation n'est pas la plus économique. On utilise plutôt une organisation matricielle, une partie de l'adresse servant à identifier une ligne et le reste correspondant à une colonne. Tout ce qui a été dit sur l'assemblage des blocs mémoires pour augmenter la capacité ou la longueur des mots reste valable pour les mémoires mortes.

### VI.1.b Application des ROMs

Les domaines d'application des mémoires mortes sont très nombreux. Certains ont déjà été rencontrés dans ce cours, comme la conversion de code. La génération de caractères, pour un terminal ou une imprimante, en est un exemple. Un caractère peut être considéré comme une matrice de points, chaque point à afficher étant défini par un 1 dans la matrice. Chaque ligne de cette matrice est représentée par un mot de n bits, n étant le nombre de colonnes. Pour un certain nombre de polices permanentes, les matrices peuvent être stockées en mémoire morte dans le périphérique.

Les ROMs peuvent être également utilisées pour conserver certains programmes et les données associées. C'est le cas pour les calculateurs de poche ou pour les programmes de démarrage et de chargement qui s'exécutent automatiquement lors de la mise sous tension d'un ordinateur.

Une mémoire morte peut également être utilisée pour générer une fonction numérique. Considérons par exemple la fonction sinus :  $\sin(x)$  pour x variant entre 0 et  $\pi/2$ . Nous pouvons diviser ce domaine de variation  $[0, \pi/2]$  en  $N = 2^k$  intervalles, la variable x prenant les valeurs centrales, c'est-à-dire :

$$x_i = \left(i + \frac{1}{2}\right) \delta x \quad \text{avec } i \in [0, N-1] \quad \text{et} \quad \delta x = \frac{\pi}{2N}$$

Chacune de ces valeurs est identifiée par l'indice i qui peut être utilisé comme une adresse. Si nous codons dans la cellule de mémoire associée le nombre :

$$f_i = \text{Partie Entière} [(2^n - 1) \sin x_i]$$

nous obtenons une grandeur proportionnelle à  $\sin x_i$ , codée sur la totalité de la gamme dynamique des n bits. Nous obtenons ainsi une table de la fonction sinus (Look-up table). Une telle table associée à un compteur et à un convertisseur Numérique-Analogique peut servir à la génération de signaux périodiques.

Une mémoire morte peut également être utilisée pour générer des fonctions logiques. Supposons que pour une application particulière nous ayons besoin de huit fonctions logiques  $f_0$  à  $f_7$  de deux variables a et b, par exemple :

$$\left\{ \begin{array}{l} f_0(a, b) = a \bullet b \\ f_1(a, b) = a \bullet \bar{b} \\ f_2(a, b) = \overline{a \bullet b} \\ f_3(a, b) = a + b \\ f_4(a, b) = a + \bar{b} \\ f_5(a, b) = \overline{a + b} \\ f_6(a, b) = a \oplus b \\ f_7(a, b) = a \oplus \bar{b} \end{array} \right.$$

Il est facile d'écrire la table de vérité de ces huit fonctions par rapport aux quatre combinaisons de a et b. Nous obtenons ainsi  $8 \times 4 = 32$  valeurs possibles. Celles-ci peuvent être mémorisées dans une ROM 32 bits comptant 5 lignes d'adresse et une sortie. Trois bits servent à identifier la fonction  $f_i$ , les deux autres lignes d'adresse correspondent aux variables a et b.

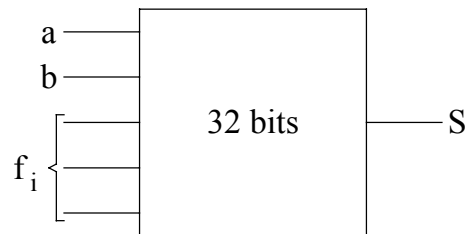


Figure 4

## VI.2 Mémoires programmables par l'utilisateur : PROM

Pour faire fabriquer une mémoire morte le client doit fournir la table de vérité au fabricant qui se charge de réaliser les masques, puis de produire les circuits. La réalisation des masques est une opération onéreuse qui suppose une production importante pour être amortie. Par ailleurs dans la phase de développement d'un système, on peut être amené à modifier le contenu d'une ROM, pour corriger des erreurs ou apporter des modifications. Cela peut s'avérer très coûteux. D'autre part la fabrication des premiers circuits demande quelques mois.

La mémoire programmable (PROM pour Programmable Read Only Memory) a été inventée pour offrir un gain de temps et d'argent durant la phase de mise au point. Une mémoire PROM, fabriquée en grande série, donc à bas prix, est programmée par l'utilisateur lui-même. Lorsque la table de vérité est validée, la ROM peut être réalisée industriellement pour la fabrication en grande série. Sinon la PROM est un composant financièrement intéressant pour les petites séries.

## VI.2.a Techniques de programmation

Plusieurs procédés sont possibles pour l'enregistrement de l'information binaire. Citons pour commencer la méthode des fusibles. Pour chaque bit ou nœud de la matrice existe une diode et un fusible. Initialement les fusibles sont conducteurs, les bits sont donc tous à 1. La définition des 0 se fait à l'aide d'un programmeur de PROM qui, en fonction de la table de vérité, détruit les fusibles en indiquant l'adresse du mot contenant le bit à modifier et en injectant un courant sur la sortie correspondante. Cette méthode de programmation présente quelques risques pour l'ensemble du circuit. Le schéma de la figure 5 est identique à celui de la figure 3, les fusibles encore présents assurant les mêmes connexions.

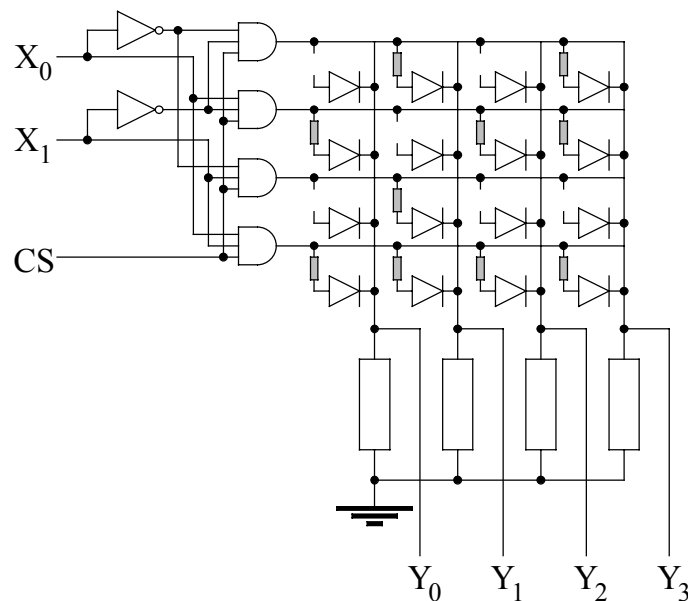


Figure 5

Avec cette technique la programmation est irréversible. C'est également le cas pour la technique des anti-fusibles. Aucune connexion n'est initialement programmée, mais il est possible de créer un contact à l'intersection de chaque ligne et de chaque colonne en détruisant un diélectrique. Au moins deux technologies sont utilisées.

Dans la première (fig. 6) on fait fondre un diélectrique en forçant un courant important (environ 5 mA) dans une zone de très faibles dimensions située à l'intersection de deux électrodes l'une en silicium polycristallin et l'autre réalisée par un implant fortement dopé n+. Ce diélectrique est constitué de trois couches alternées d'oxyde et de nitrure de silicium :  $\text{SiO}_2$  -  $\text{Si}_3\text{N}_4$  -  $\text{SiO}_2$ . La présence du nitrure permet de réduire l'épaisseur du diélectrique. Lors de la fusion du diélectrique il y a diffusion de dopants, ce qui diminue la résistance du contact réalisé.

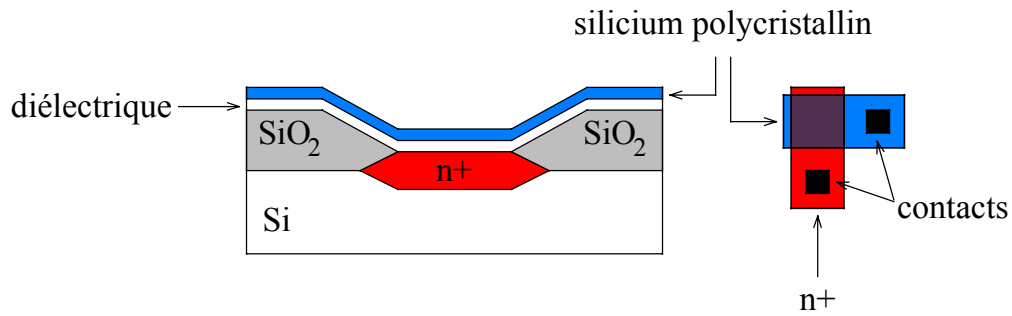


Figure 6

La seconde technologie (fig. 7) permet de réduire cette résistance, car le contact est réalisé par un alliage de tungstène, titane et silicium. Le diélectrique est cette fois du silicium amorphe entre deux électrodes métalliques (m1 et m2). Cette technique requiert moins de place que la première, mais elle est assez délicate à mettre en œuvre.

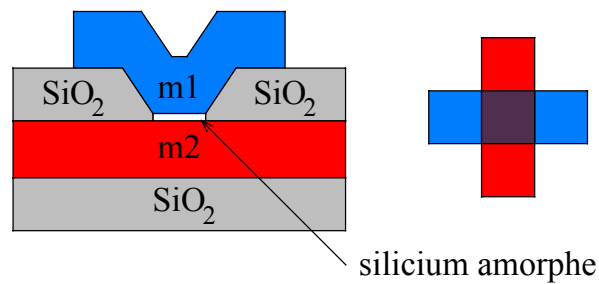


Figure 7

Dans toutes les techniques précédentes la programmation irréversible, il est impossible de revenir en arrière. Toute modification suppose la programmation d'un nouveau circuit. Une autre technique offre la possibilité de multiprogrammation.

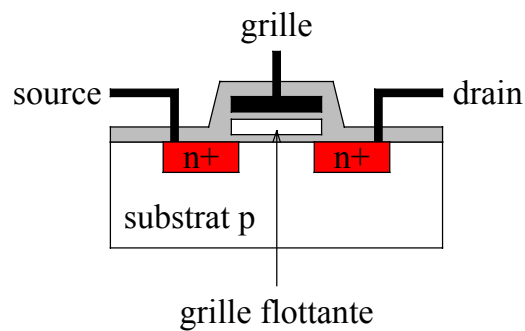


Figure 8

On utilise pour cela des transistors à effet de champ à structure MOS (Métal-Oxyde-Semiconducteur) avec une grille supplémentaire flottante. Dans un transistor MOS classique la grille est utilisée pour induire un canal entre la source et le drain.

Dans l'exemple de la figure 8 une tension positive de la grille va attirer des électrons et repousser des trous. Par ailleurs une partie des électrons attirés se recombinent avec des trous. Une inversion de population se crée induisant un canal n entre les implants de la source et du drain. Le transistor devient passant. Pour programmer un tel nœud on fait alors circuler un courant intense entre la source et le drain. Certains électrons acquièrent une énergie leur permettant d'atteindre la grille flottante. Ils y sont alors piégés. Lorsque la charge piégée est suffisante elle masque le champ électrique induit par la grille et le transistor est bloqué. Le courant de fuite étant très faible cette charge peut se conserver très longtemps.

La programmation est réversible, il suffit de décharger cette grille flottante. Une première solution consiste à exposer le circuit à un rayonnement ultraviolet pendant quelques dizaines de minutes. Ces circuits sont équipés d'une fenêtre en quartz et encapsulés dans un boîtier en céramique pour résister à l'échauffement. Cette mise en œuvre augmente le prix des composants, mais cela n'est nécessaire que pour le développement. Pour la production le même circuit existe sans fenêtre et en boîtier plastique. Cette technique correspond aux mémoires mortes effaçables : EPROM (Erasable Programmable Read Only Memory). Il est également possible de décharger la grille flottante par effet tunnel en appliquant des tensions suffisamment élevées entre la grille, la source et le drain. Cet effacement est plus rapide et n'impose pas de retirer le circuit du système dans lequel il est installé. Cela correspond à la famille des mémoires mortes programmables et effaçables électriquement : EEPROM (Electrically Erasable Programmable Read Only Memory).

La différence entre les EEPROM et les mémoires vives réside dans la vitesse d'écriture. Le cycle d'écriture d'une EEPROM est environ 1000 fois plus long que celui d'une RAM. Par contre rappelons que les temps d'accès en lecture des mémoires RAM, ROM, PROM, EPROM et EEPROM sont comparables.

Citons enfin un composant hybride qui associe dans un même boîtier une RAM et une EEPROM de même capacité : la mémoire NOVDRAM (Non Volatile RAM). L'EEPROM permet de réaliser en moins de 10 ms une sauvegarde globale de la RAM. Cela permet une sauvegarde du contenu de la mémoire en cas de coupure d'alimentation électrique.

### **V.3 Réseaux logiques programmables**

Les mémoires mortes programmables furent les premiers membres d'une famille de composants : les circuits logiques programmables (CLP ou PLD : Programmable Logique Device). Cette famille est aujourd'hui très nombreuse et très variée, comme l'illustre la multitude d'acronymes : PAL, SPLD, EPLD, CPLD, FPGA, etc. La figure 9 donne une représentation symbolique simplifiée d'une PROM de 16 mots de 4 bits. Les portes ET reçoivent 4 variables directes ou inverses. Les connexions de gauche, correspondant au décodeur et représentées par des disques noirs, sont figées. Les croix à droite représentent des connexions programmables. Les

portes OU peuvent recevoir jusqu'à 16 variables. Chaque sortie d'une PROM est ainsi représentée comme une somme canonique de produits.

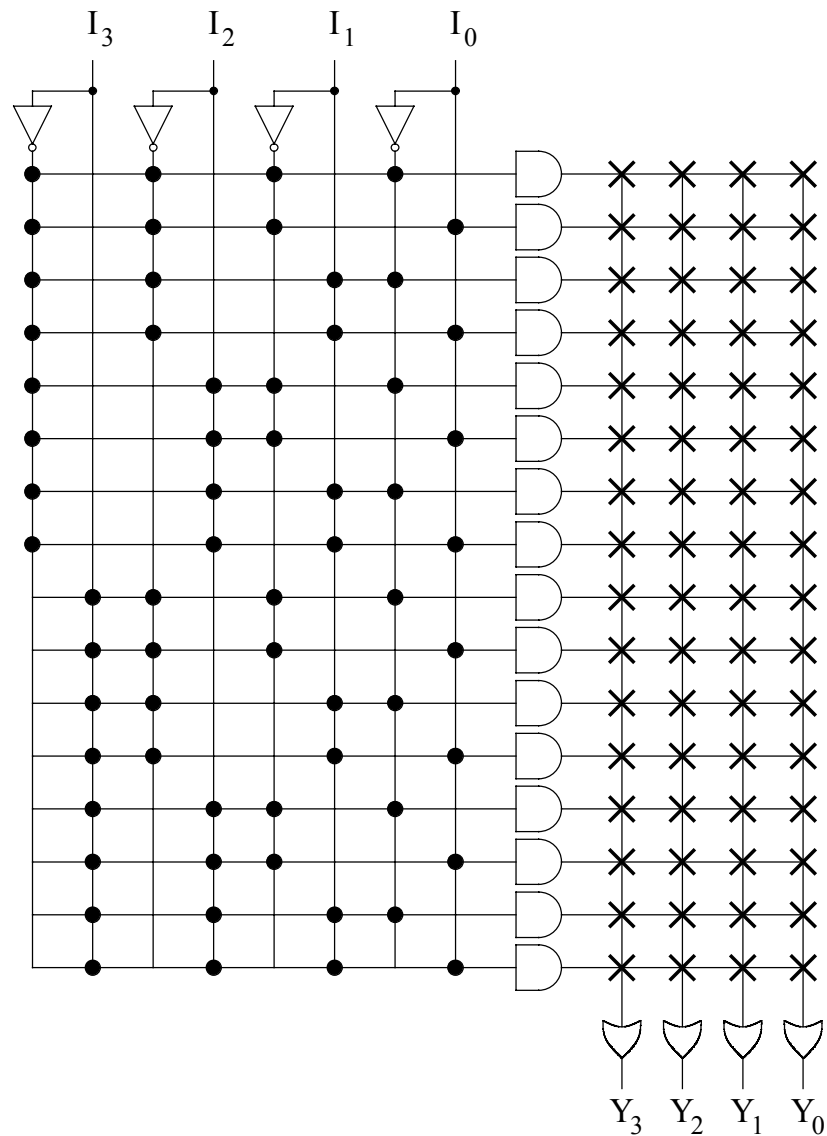


Figure 9

Dans un Programmable Array Logic (PAL) la matrice des connexions OU est figée et la matrice des connexions ET est programmable (fig. 10). Un Programmable Logic Array (PLA ou FPLA : Field Programmable Logic Array) offre le maximum de souplesse car les deux matrices des OU et ET sont programmables comme l'indique la figure 11. Ces circuits constituent la famille de SPLDs (Simple Programmable Logic Device).

Ce type de circuits n'a cessé de se sophistication au fil du temps. Certains canaux d'entrée/sortie peuvent être bidirectionnels, les deux portes "3 états" en commandant le sens



pouvant être programmable. Par ailleurs, des sorties peuvent être injectées dans le réseau de portes ET, offrant ainsi la possibilité d'utiliser des variables intermédiaires.

On peut également trouver un réseau de bascules de type D avec une horloge commune. L'entrée d'une bascule peut être programmée au niveau des réseaux de portes ET et OU. Sa sortie peut être accessible à l'extérieur et utilisée comme variable interne injectée dans le réseau de connexions programmables. Il est alors possible de programmer des fonctions séquentielles.

La figure 12 illustre quelques unes de ces possibilités. On y trouve une bascule D dont la sortie est injectée dans le réseau programmable, matérialisé par les petits carrés. La sortie doit être validée par le signal OE (Output Enable).

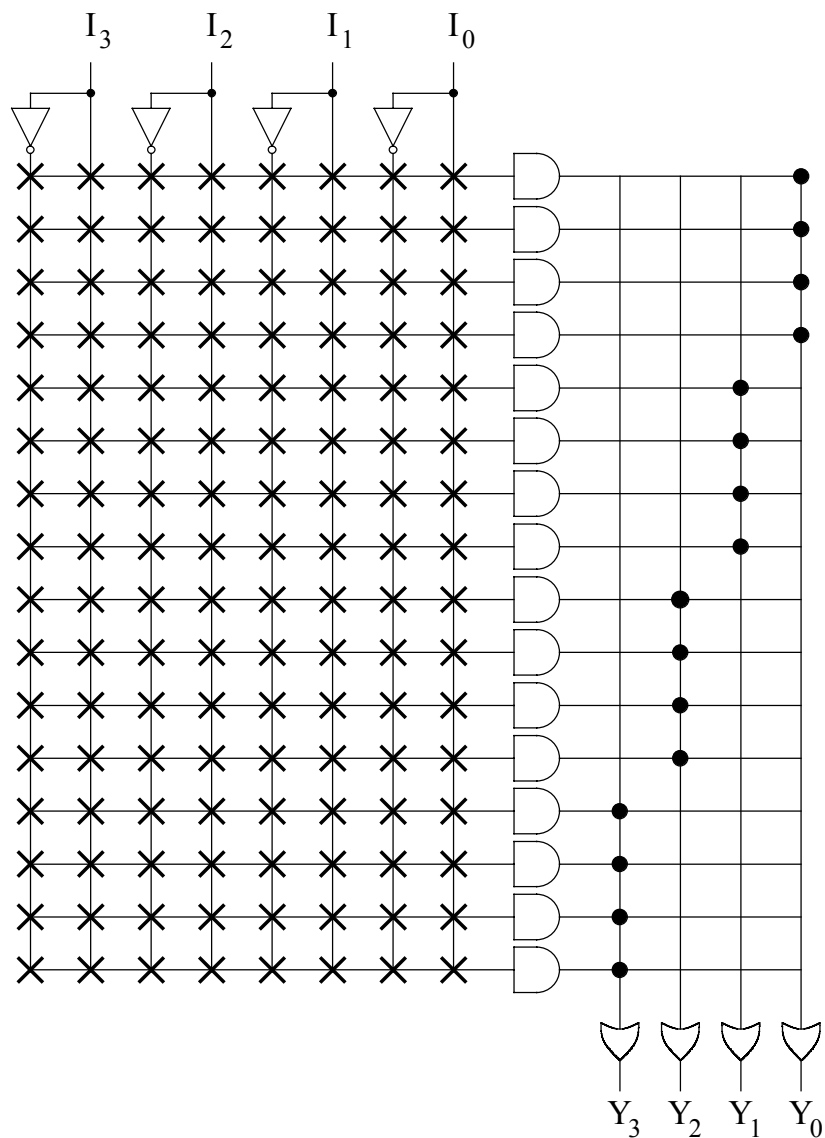


Figure 10

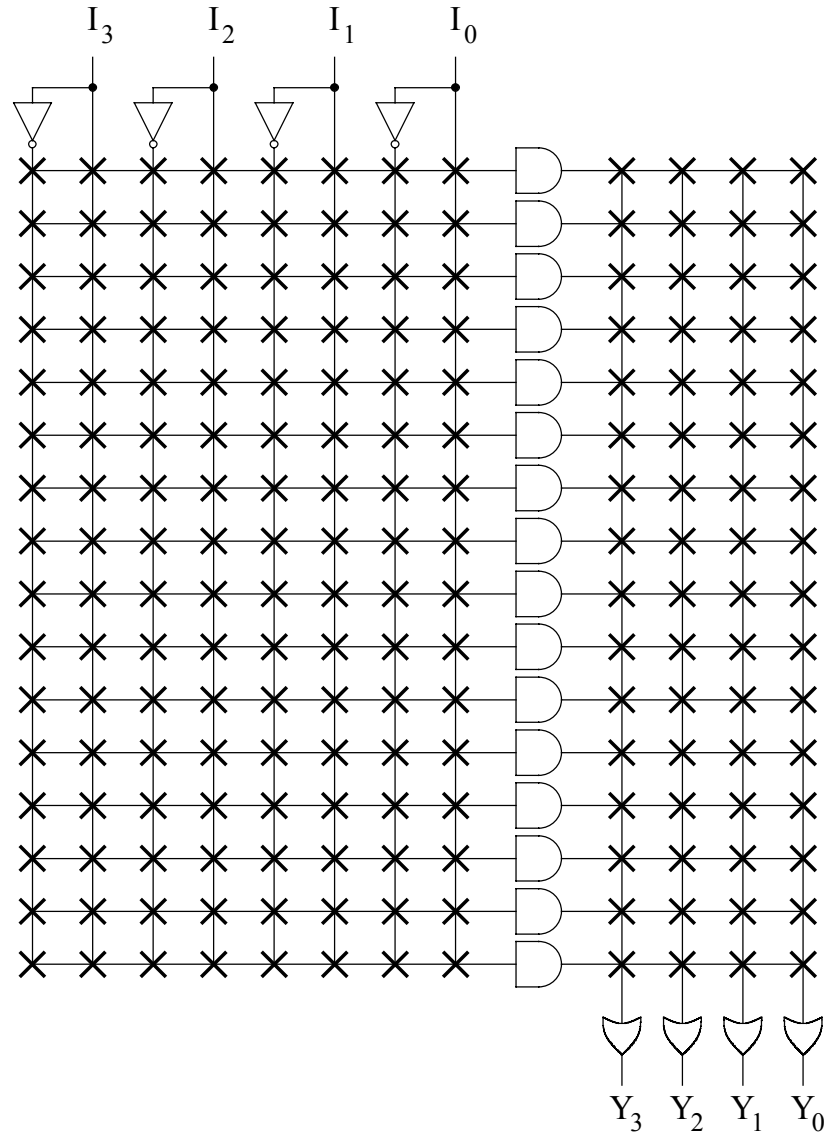


Figure 11

L'évolution a conduit à des composants plus complexes. Ce sont d'abord les CPLDs (Complex Programmable Logic Device), puis les FPGA (Field Programmable Gate Array). Ce sont de relativement de gros ensembles de cellules logiques programmables (comparable à ce que nous venons de présenter) qui peuvent être connectées de différentes manières.

Un CPLD est composé d'un certain nombre de PALs, ou macrocellule. La figure 12 donne un exemple de ce que peut être une macrocellule. Ces macrocellules sont regroupées en blocs logiques, qui peuvent être reliés via une matrice de connexion programmable (fig. 13). Les macrocellules d'un même bloc sont généralement interconnectées.

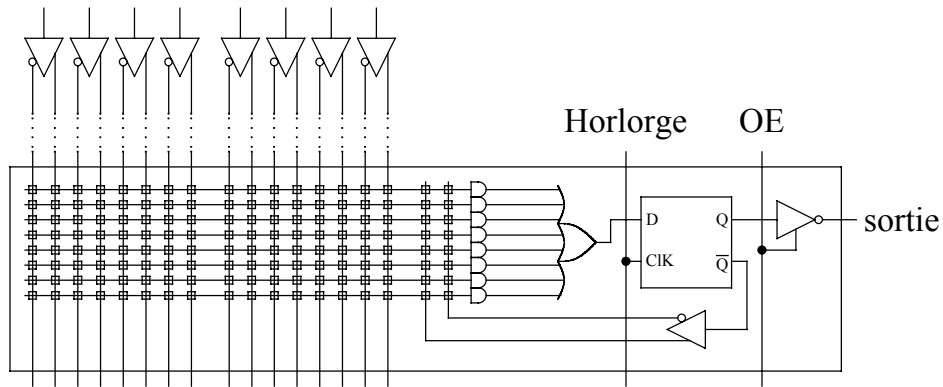


Figure 12

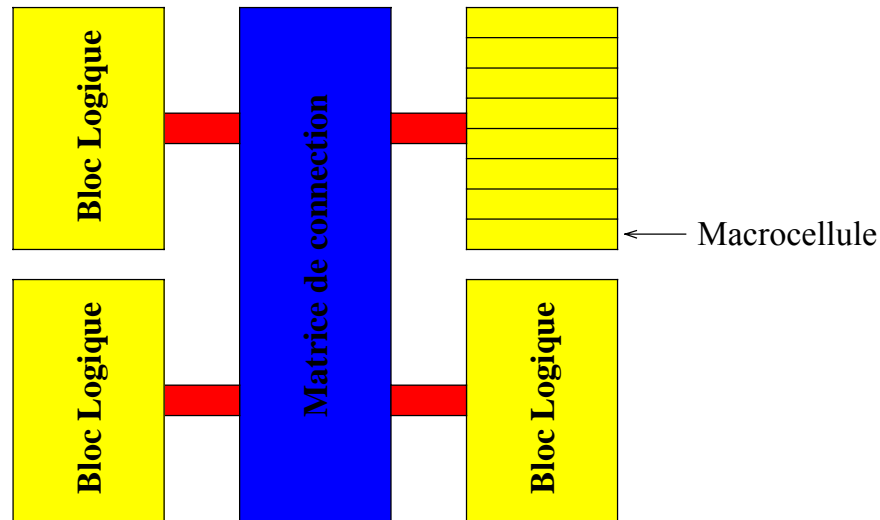


Figure 13

Un FPGA offre le maximum de capacité logique. Il comprend des blocs d'entrée/sortie, des blocs logiques et des matrices de connexion, le tout étant programmable (fig. 14). Les architectures de ces circuits sont très variées.

A une exception près, les techniques de programmation des FPGAs sont identiques à celles que nous avons décrites pour les mémoires mortes programmables. Il existe une technique supplémentaire pour les FPGAs. La programmation de certains FPGAs se fait à l'aide d'une mémoire statique incorporée. La sortie d'un point mémoire de cette RAM peut commander l'état d'un transistor pour assurer ou non une connexion. Cela permet une mise à jour de la programmation au sein du système. Par contre, il faut charger cette mémoire à chaque démarrage et tenir compte du temps nécessaire à ce chargement. Celui-ci peut se faire à partir d'une ROM ou être effectué par un ordinateur. Ces composants complexes sont nécessairement accompagnés de logiciels de programmation, routage et de simulation.

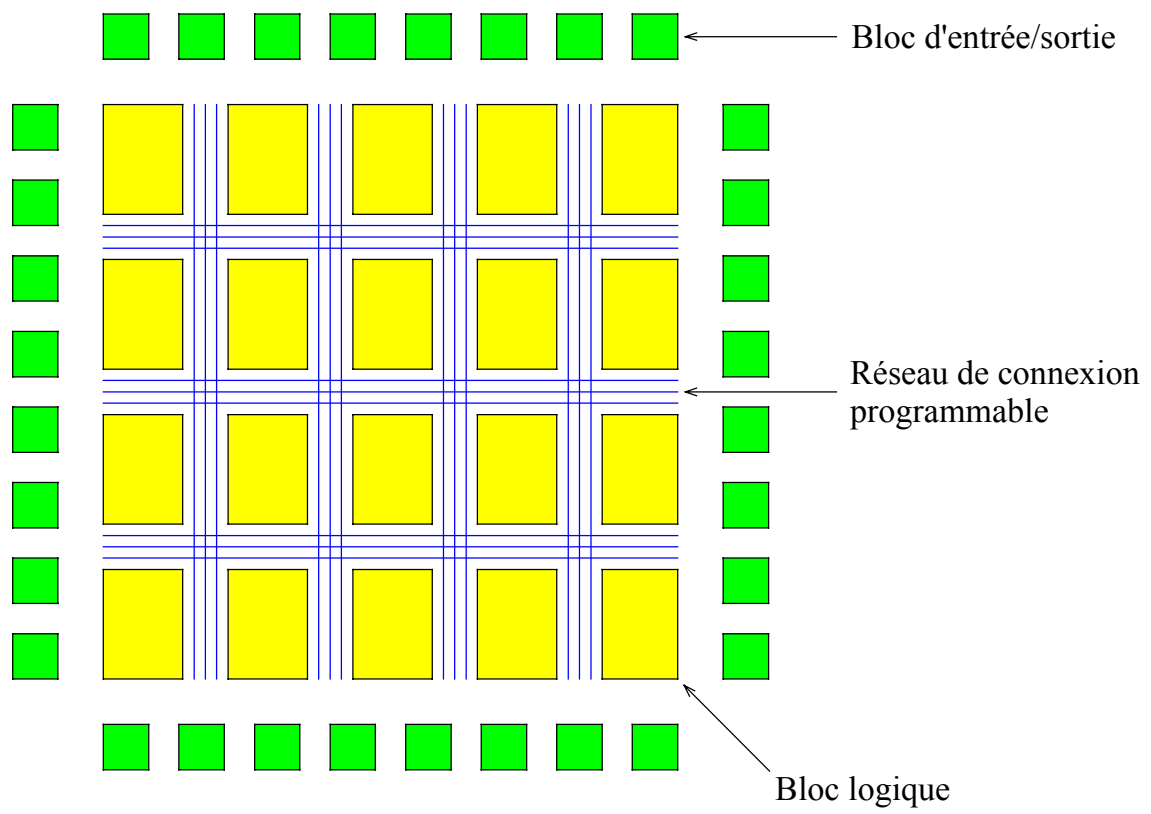


Figure 14