RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ MOHAMED SEDIK BENYAHIA JIJEL

FACULTÉ DES SCIENCES ET DE LA TECHNOLOGIE

DÉPARTEMENT D'ÉLECTRONIQUE

Du langage VHDL aux circuits programmables

Support de cours pour les modules ST04 et ST07 du master système et télécommunication

Proposé par

Dr Alioua Chehla épouse Benhadji

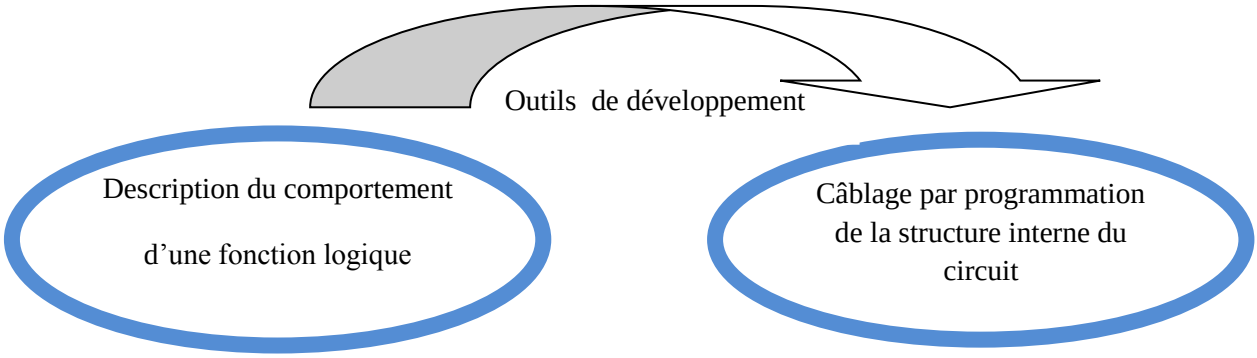
Partie I : Premiers Pas En Langage VHDL

Sommaire :

I- Introduction

De nos jours l'électronique numérique est en développement continu et tente même de remplacer les circuits analogiques. Le progrès des systèmes numériques a commencé par la découverte des fonctions logiques de base contenues dans les circuits intégrés des familles 74xxx ou CD4000. La réalisation de circuits expérimentaux à l'époque été limitée aux fonctions offertes par ces circuits. Ensuite la logique à circuit intégrés a cédée la place aux circuits PLDs (Programmable Logic Device), aux circuits ASICs (Application Specific Integrated Circuit) et aux circuits FPGAs (Field Programmable Gate Array), qui ont permis de s'affranchir des limitations de la logique à circuits intégré. Tous ces composants sont des circuits programmables pour des applications précises et leur utilisation nécessite un logiciel adéquat.

Cependant les fonctions numériques à réaliser deviennent de plus en plus importantes en raison du nombre de variables d'entrées représentant des opérandes à 64 bits et même plus. L'ampleur de ces fonctions nous impose l'utilisation de nouveaux outils de description autre que le schéma.



Dans la majorité des cas, la description du comportement d'une fonction logique est réalisée par l'utilisation de langage dit ''langage de description comportementale''. Au départ on utilisait des langages de bas niveaux tel que le langage ABEL et le langage PALASM, mais la densité d'intégration des fonctions logiques dans les circuits programmables est très importante de nos jours, à savoir plusieurs milliers de portes. C'est pour cette raison qu'il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui.

Pour surmonter les contraintes engendrées par les anciennes technologies de circuits, on a créé des langages dit de haut niveau utilisés comme outils de description du système logique à intégrer dans un composant, et permettent ainsi de matérialiser les structures électroniques de différents circuits. Parmi ces langages on cite le langage VHDL et le langage VERILOG [1].

Cette approche par langage de description comportemental présente l'avantage de s'intéresser plutôt au comportement de la fonction logique qu'à la manière dont elle est réalisée. Ceci peut être simulé avec un outil informatique (ordinateur) où les instructions écrites dans ces langages peuvent se traduire par une configuration logique de portes et de bascules intégrée à l'intérieur d'un circuit logique programmable (comme par exemple un PLD ou un FPGA) et procéder ainsi à l'expérimental.

Dans ce document nous allons voir comment faire une description d'une fonction ou d'un circuit en utilisant le langage VHDL puis nous donnerons ensuite quelques éléments de ce langage.

II- Définition

Le langage VHDL qui est l'abréviation de ``Very High Speed Integrated Circuit Hardware Description Language´´ a été créé pour le développement de circuits intégrés logiques complexes. Il doit son succès, essentiellement, à sa standardisation par l'organisme IEEE (Institute of Electrical and Electronics Enggineers) sous la référence IEEE1076 qui a permis d'en faire un langage unique pour la description, la modélisation, la simulation, la synthèse et la documentation. Ces intérêts majeurs sont :

- Des niveaux de description très divers et surtout un niveau comportemental élevé.
- C'est un langage qui devient commun à de nombreux systèmes.

III- Structure d'une description en langage VHDL :

La structure générale d'un programme en VHDL est constituée de deux parties [2]:

III-1 Entité (entity)

C'est la construction qui décrit l'extérieur d'une fonction logique et permet ainsi de définir ses entrées et ses sorties. Pour déclarer l'entité on donne un nom et on précise la liste des différents signaux d'entrées/sorties.

```
entity circuit is
   Port (entrée 1: in bit;
      entrée2, entrée 3: in bit;
      entrée4: in bit_vector (0 to 3);
      sortie1, sortie2: out bit;
      sortie3: in bit_vector (0 to 5));
end circuit;
```

III-2 Architecture

Une architecture comporte en générale une partie de déclaration et un corps du programme où sont notées les instructions requises pour le fonctionnement d'un circuit. VHDL offre l'avantage de pouvoir effectuer les descriptions en se plaçant à différents niveaux, du niveau comportementale le plus abstrait au niveau structurel le plus détaillé [1].

Il faut noter qu'il est possible d'associer plusieurs architectures à une même entité. Chacune d'elle va décrire l'entité d'une manière différente.

```
architecture exemple of circuit is
    partie déclaration : types, constantes, signaux, composants
    Begin
    Corps de l'architecture
    Suite d'instructions paralléles;
    Affectation de signaux;
    Processus ; blocs ;
    instantiation de composants ;
    end exemple ;
```

Exemple :

workspace	
name	
simprim	Library
adder_1	Entity
V2	Architecture
V1	Architecture

IV- Eléments du langage [2]

IV-1 Les données

Toutes les données ont un type dans le langage VHDL qui doit être déclaré avant l'utilisation. Par exemple pour passer du type entier au type bit_vector il faut faire appel à une fonction de conversion.

IV-1-1 Les classes

Une donnée appartient à une classe qui définie son type et son comportement. Les classes spécifiques dans le langage VHDL sont :

- Signaux : Représentent les données physiques échangées entre les blocs logiques d'un circuit.
  Signal nom1, nom2 : type ;
- Variables : Elles représentent des éléments qui servent à stocker un résultat intermédiaire pour faciliter la construction d'un algorithme séquentiel.

  Variable nom1, nom2 : type [:=expression];

  La différence entre les variables et les signaux est que les premières n'ont pas d'équivalent physique dans le schéma contrairement aux seconds.

- Constantes : Ce sont des objets dont la valeur est fixée une fois pour toute. Comme par exemple '18', '011011', '2014', 'AZERTY'.
  On peut créer des constantes nommées :

  Constant nom1 : type [:=valeur_constante] ;

IV-1-2 Les types (Adaptés à l'électronique numérique)

- Les entiers : Le langage VHDL peut manipuler des valeurs entières qui correspondent à des mots de 32 bits.

  Déclaration :
  Signal nom : integer;
  variable nom : integer ;
  constant nom : integer ;

- Les réels (real) : Ce type permet de définir un nombre réel.

  Déclaration :
  variable nom : real;

- Les bits : Un objet de type bit peut prendre deux valeurs ´0´ ou ´1´.

  Déclaration :
  Signal nom : bit ;
  Variable : bit ;

- Les booléens : Un opérateur de ce type peut prendre deux valeurs ´true´ et ´false´.
- Les tableaux : On peut créer des tableaux à partir de chaque type de base.

  Exemple :
  La première instruction permet de définir un tableau ou un vecteur, nommé VEC, de dix éléments binaires.

```
Signal VEC : bit_vector (0 to 9) ;

Signal SOUS_VEC1 : bit_vector (0 to 3) ;

Signal SOUS_VEC2 : bit_vector (0 to 5) ;

VEC<= SOUS_VEC1 & SOUS_VEC2;
```

La dernière instruction permet de réaliser la concaténation entre les deux tableaux VEC1 et VEC2.

On peut noter :

bit_vector (0 to 9) : pour le sens croissant.

bit_vector (9 down to 0) : Pour le sens décroissant.

- Les enregistrements : ils définissent une collection d'objets de types ou de sous types différents.

Exemple [2]

- Définition d'un type clock_time.
- Déclaration d'un objet de ce type.
- Utilisation de l'objet.

```
-- définition
Type clock_time is record
        hour: integer range o to 12;
        minute, second: integer range 0 to 59;
end record;
-- déclaration
variable time: clock_time;
-- utilisation
time.hour:=6;
time.minute:=45;
time.second:=57;
```

- Le type composite record (article):
Un article est une collection d'éléments de même Type ou de type différents.
La déclaration d'un signal nommé (sig1) du type article est donnée comme suit :

Signal sig1 : article ;

Pour donner une valeur au signal, par exemple l'adresse est 3 et la donnée Data est 1011 on procède comme suit :

sig1 <= (3,``1011``) ;

Si on veut donner la valeur 2 au champ adresse de sig1 :

sig1.adresse <= 2 ;

```
Exemple :
Type article is record
        Adresse : integer range (0 to 4) ;
        Data : bit_vector (0 to 3);
End record;
```

- Le type composite array (Tableau):

Ce type est consacré pour des tableaux simples

(une seule rangée).

Si on veut donner la valeur 2014 à année_en_cours:

année_en_cours :=(2,0,1,4);

Si on veut remplacer 4 par 5 par exemple :

année_en_cours(3) :=5; -- Année 2015

```
Exemple :
Type année is array (0 to 3) of integer;
-- declaration
Variable année_en_cours: année;
```

- Le type composite array à plusieurs rangées :

  On donne l'exemple1 pour définir un tableau à plusieurs rangées où chaque rangée représente l'année.

1	9	2	8
1	9	3	8
2	0	1	4

Exemple1:

Type année **is array** (0 **to** 3) **of integer**;

Type date **is array** (2 **down to** 0) **of** année;

Constant
cste_date :date :=((1,9,2,8),(1,9,3,8),(2,0,1,4)) ;

Variable année_en_cours: année;

  L'exemple 2 pour définir un tableau de 5 lignes d'éléments binaires.

S_1	S_2
0	0
1	1
0	1
1	1
0	1

Exemple2:

type table1 is array (0 **to** 4, 1 **down to** 0) **of** bit;

constant code :table1:=

((´0´,´0´), (´1´,´1´), (´0´,´1´), (´1´,´1´), (´0´,´1´));

Ou encore

type table2 **is array** (0 to 4) **of** bit_vector (1 **down to** 0);

Constant code2 : table2:=

("00", "11", "01", "11", "01");

- Le type array of records
  On donne l'exemple 3 suivant pour définir un type composite d'enregistrements.

D	I	M	1	9	2	8
L	U	N	1	9	3	8
M	A	R	2	0	1	4

Exemple3:

type DATE **is record**

jour : string (0 to2);

heure : integer range (0 to 23);

minute: integer range (0 to 59);

end record;

type table3 **is array** (0 to 2) **of** DATE;

constant cste : table3:=

((DIM,19,28), (LUN,19,38), (MRA,20,14));

IV-2 Différentes méthodes d'association

IV-2-1 Association élément valeur par position :

Il suffit de donner les valeurs dans l'ordre choisi lors de la déclaration du type, à savoir selon l'ordre de l'indice pour un record, selon l'ordre des champs pour un array.

Exemple :

```
année_en_cours := (2,0,1,4);

Sig1<= (3,''1011'');
```

IV-2-2 Association élément valeur par nom:

Il suffit d'indiquer pour chaque élément, par le symbole ''=>'' la correspondance, indice => valeur, pour un record, champ => valeur, pour un array.

```
année_en_cours := (0 =>2,1=>0,2=>1,3=>4);

Sig1<= (adresse=>3, donnée=>''1011'');
```

Exemple :

IV-2-3 Association mixte :

Les valeurs des premiers éléments sont données selon l'association par position. Les valeurs des derniers éléments de l'objet sont données selon l'association par nom.

Exemple :

```
année_en_cours := (2,0,2=>1,3=>4);

Sig1<= (3, donnée=>''1011'');
```

Remarque :

Une valeur affectée à un bit est exprimée entre simple apostrophe, alors qu'une valeur affectée à un objet de type bit_vector est exprimée entre double apostrophes. Plusieurs bases sont possibles :

Pour une variable de type bit_vector, les affectations ci-contre sont valides.

V:= x ''B4''; base hexadécimal et majuscule.

V:= x ''b4'' base hexadécimal et minuscule.

V:= o ''374'' base octale.

V:= B ''101110111001'' base binaire.

V:= ''101110111001'' base binaire prise par défaut.

Dans l'écriture (V:= B''1011_1010'') le trait sert uniquement à aérer la suite de bits.

Donc un nombre quelconque de ''_'' peut être utilisé à condition de spécifier la base par sa lettre caractéristique.

V- Les attributs :

Les attributs sont des propriétés qui portent un nom, associés à une entité, une architecture, un type ou un signal. Cette propriété une fois définie, peut être utilisée dans des expressions.

nom_objet 'nom_de_l'attribut

V-1 Attributs spécifique à un système:

Chaque système de développement fournit des attributs qui aident à piloter l'outil de synthèse ou le simulateur associé au compilateur VHDL.

V-2 Attributs définis par l'utilisateur :

L'utilisateur peut lui-même définir un attribut puis l'utiliser au besoin comme suit :

attribute att_nom : type ;

nom_objet'att_nom;

V-3 Attributs prédéfinis dans le langage:

Les attributs prédéfinis permettent de déterminer les contraintes qui pèsent sur des objets ou des types : domaine de variation d'un type scalaire, bornes des indices d'un tableau, élément voisin d'un objet de type énuméré, etc. ils permettent également de préciser les caractéristiques dynamiques de signaux. Les attributs prédéfinis sont récapitulés sur le tableau suivant [2]:

Attribut	Type	Valeur retournée
'left 'left(n)	type scalaire type tableau	élément de gauche borne gauche de l'indice de la dimension n
'right 'right(n)	type scalaire type tableau	élément de droite borne droite de l'indice de la dimension n
'high 'high(n)	type scalaire type tableau	élément le plus grand borne maximum de l'indice de la dimension n
'low 'low(n)	type scalaire type tableau	élément le plus petit borne minimum de l'indice de la dimension n
'length(n)	type tableau	Nombre d'éléments de la dimension n
'pos(v)	type scalaire	Position de l'élément dans le type
'val(p)	type scalaire	Valeur de l'élément de position p dans le type
'succ(v)	type scalaire	Valeur qui suit l'élément de valeur v dans le type
'pred(v)	type scalaire	Valeur qui précède l'élément de valeur v dans le type
'leftof(v)	type scalaire	Valeur de l'élément juste à gauche de l'élément de valeur v
'rightof(v)	type scalaire	Valeur de l'élément juste à droite de l'élément de valeur v
'event	Signal	Valeur booléenne ''TRUE'' si la valeur du signal vient de changer
'base	tous types	Renvoie le type de base d'un type dérivé
'rang(n)	type tableau	Renvoie la plage de variation de l'indice de dimension n
'reverse-rang(n)	type tableau	Renvoie la plage de variation retournée (to/downto) de l'indice de dimension n

VI- Les opérateurs élémentaires :

Les opérateurs élémentaires du langage sont récapitulés comme suit :

Classes	Opérateurs	Types d'opérandes	Résultat
Opérateurs logiques	AND, OR, NAND, NOR, XOR	Bits ou booléens	Bit ou booléen
Opérateurs relationnels	=, /=, <, <=, >, >=	Tous types	Booléen
Opérateurs additifs	+, - &	Numériques Tableaux	Numérique Tableau
Signe	+,-	Numériques	Numérique
Opérateurs multiplicatifs	*, / mod, rem	Numériques Entiers	Numérique Entier
Opérateurs divers	NOT ABS **	Bits ou booléens Numériques Numériques	Bit ou booléen Numérique Numérique

On donne un exemple sur l'opérateur &, qui représente l'opération dite : concaténation, et qui est utilisé surtout dans les signaux.

Exemple :

```
Signal A,B : bit_vector (0 to 3) ;
Signal C : bit_vector (0 to 7) ;
A <= ''1011''; B <= ''0001'';
C <= A & B;
```

Le résultat de la concaténation est donc : C=''10110001''.

VII- Les instructions concurrentes :

Elles interviennent à l'intérieur d'une architecture, dans la description du fonctionnement d'un circuit. Les principales instructions sont :

- Les affectations de signaux.
- Les processus.
- Les instanciations de composants.
- Les définitions de blocs.
- Les instructions ''generate''.

VII-1 Les affectations de signaux

a) Affectation simple :
Elle représente une interconnexion simple entre deux objets.

```
nom_signal <= expression;
```

b) Affectation conditionnelle :
Elle permet de déterminer la valeur de la cible en fonction des résultats de tests logiques.

```
cible <= source1 when condition1 else
         source2 when condition2 else
         ----------
         source n;
```

c) Affectation sélective :
En fonction des valeurs possibles d'une expression, on peut choisir la valeur à affecter à un signal.

```
with expression select
cible <= source1 when valeur1;
         source2 when valeur 2;
         ----------
         source n when others ;
```

VII-2 Les processus :

Un processus décrit une partie d'une description VHDL d'un circuit, dans laquelle les instructions sont exécutées les une après les autres séquentiellement. Comme par exemple, pour décrire une fonction combinatoire on peut avoir recours à un processus. L'exécution d'un processus est déclenchée par changement d'états des signaux logiques. Tous les signaux lus par ce processus doivent apparaître dans sa liste de sensibilité.

Exemple :

```
[Nom_process:] process (liste de sensibilité)
        Begin
                -- Instruction du process
        End process [Nom_process:] ;
```

Certaines règles doivent être appliquées lors de l'utilisation d'un processus :

- Les instructions d'un processus s'exécutent séquentiellement.
- Tous les signaux lus par un processus doivent apparaître dans sa liste de sensibilité.
- L'exécution d'un processus est prise en compte à chaque changement d'état d'un signal de la liste de sensibilité.
- Les changements d'état des signaux par les instructions du processus sont pris en compte à la fin du processus.

VII-3 Instanciation de composants :

On peut utiliser plusieurs types de composants dans une description en VHDL, la syntaxe de déclaration d'un composant est la suivante :

```
Component nom_composant
Port (liste des ports);
End nom_composant ;
```

Le mécanisme qui consiste à utiliser un sous-ensemble (une paire entité-architecture), décrit en VHDL, comme composant dans un autre ensemble est connu sous le nom d'instanciation. Trois opérations sont nécessaires [3]:

- Le couple entité-architecture du sous ensemble doit être créé et annexé à une librairie de l'utilisateur.
- Le sous ensemble doit être déclaré comme composant (component) dans l'ensemble qui l'utilise.
- Chaque exemplaire du composant que l'on souhaite inclure dans le schéma en cours d'élaboration doit être connecté aux équipotentielles de ce schéma.

La syntaxe de l'instanciation est la suivante :

```
< Nom de l'instance> : <nom_composant> Port map (liste d'association);
```

La liste d'association établie la correspondance entre les équipotentiels du schéma et les ports d'entrées et de sorties du composant. Cette association peut se faire par position. Les noms des signaux à connecter doivent apparaitre dans l'ordre des ports auquel ils doivent correspondre.

Exemple :

```
entity composant is port
        (x, y, z : in bit ; S_OUT : out bit) ;
end composant ;
architecture structure of composant is
        component et
                port (a,b: in bit; z :out bit) ;
        end component;
        component ou
                port (a, b, c : in bit; z :out bit) ;
        end component;
        signal S1, S2, S3: bit;
        begin
        P1 : ET port map (x, y, S1) ;
        P1 : ET port map (x, z, S2) ;
        P1 : ET port map (y, z, S3) ;
        P1 : OU port map (S1, S2, S3, S_OUT) ;
end structure;
```

VII-4 Les blocs :

On peut diviser une architecture en blocs de manière à constituer une hiérarchie interne dans la description d'un composant complexe. La syntaxe est la suivante :

```
[Etiquette] : block [expression de garde]
-- Zone de déclaration de signaux, composants…etc
Begin
-- instructions concurrentes
End block [Etiquette];
```

VII-5 Instructions ''generate'' :

Elles permettent de créer de façon compacte des structures régulières comme les registres ou les multiplexeurs. Elles permettent aussi de dupliquer un bloc d'instructions concurrentes un certain nombre de fois. La syntaxe de déclaration est donnée comme suit :

```
-- structure répétitive :
[Etiquette]: for variable in début to fin generate
        Instructions concurrentes
End generate [Etiquette];
```

```
-- structure conditionnelle:
[Etiquette]: if condition generate
        Instructions concurrentes
End generate [Etiquette];
```

VIII- Les instructions séquentielles :

Les principales instructions séquentielles sont :

a) L'affectation séquentielle d'un signal qui utilise l'opérateur ''<='' à une syntaxe qui est identique à celle de l'affectation concurrente simple.
b) L'affectation d'une variable qui utilise l'opérateur '': =''
c) Les instructions de tests (if) et (case) :
   Elles permettent de sélectionner une ou plusieurs instructions à exécuter en fonction des valeurs prises par une ou plusieurs expressions.

- l'instruction if :

if expression **then**

instructions séquentielles

elseif expression **then**

instructions séquentielles

else

instructions séquentielles

end if ;

-- l'instruction case :

case expression **is**

when valeur1 => Instructions séquentielles;

when valeur2 => Instructions séquentielles;

---

when others => Instructions séquentielles;

end case;

Valeur1, valeur2…etc représentent les différentes valeurs que peut prendre l'expression à tester.

d) Les instructions de contrôle de boucles :

Trois catégories de boucles, qui permettent de répéter une séquence d'instructions, existe en VHDL selon le schéma d'itération choisi :

- Les boucles (for) dont le schéma d'itération précise le nombre d'exécution.

- La boucle for :

[Etiquette] **for** parameter **in** minimum **to** maximum **loop**

Séquences d'instructions

end loop [Etiquette] ;

- Les boucles (while) dont le schéma d'itération précise la condition de maintien dans la boucle.

- La boucle while :

[Etiquette] **while** condition **loop**

Séquences d'instructions

end loop [Etiquette] ;

- Les boucles simples, sans schéma d'itération, dont on ne peut sortir que par une instruction (exit), et l'instruction (next) qui permet de passer à l'itération suivante dans une boucle.

exit [Etiquette] [**when** condition];

next [Etiquette] [**when** condition];

IX- Les bibliothèques

Pour faire la synthèse d'une description en VHDL nous avons besoin de bibliothèques ou librairies. L'organisme IEEE (Institute of Electrical and Electronics Engineers) a fait une normalisation des bibliothèques pour leur utilisation. Ces bibliothèques fournissent des définitions et des informations sur les signaux, les types, les fonctions… etc permettant ainsi la réalisation de n'importe quelles fonctions (logiques, arithmètique…).

a) La librairie *WORK*

Sur certains systèmes la librairie *WORK* est quelque peu virtuelle, elle n'existe pas physiquement dans le système de fichiers, mais recouvre des liens vers des répertoires prédéfinis : vers le répertoire de travail fixé pour la session, vers le fichier en cours de compilation. Le concepteur na pas à se préoccuper de ces aspects qui sont définis à l'installation des logiciels. Sur d'autres systèmes l'utilisateur est invité à créer explicitement cette librairie, lors de la première session de travail sur un projet [4].

b) La librairie *STD*

Tout système de développement en VHDL est assorti d'une librairie *STD*. Cette librairie contient deux paquetages : *STD.STANDARD* et *STD.TEXTIO*. Les codes sources de ces deux paquetages sont fournis à titre de documentation et de conformité à la norme, mais la majorité des compilateurs incluent certaines des fonctions définies dans ces paquetages dans le noyau du compilateur. Il est formellement déconseillé de modifier quoi que ce soit dans le contenu de la librairie *STD*, sous peine de s'exposer à des incohérences graves de fonctionnement des logiciels [4]. .

Exemple

```
Library ieee ;

Use ieee.std_logic_1164.all;

Use ieee.numeric_std.all;

Use ieee.std_logic_unsigned.all;
```

X- Références

[1] Philippe Larcher, ''VHDL Introduction à la synthèse logique'', EYROLLES 2000.

[2] Jaques Weber et Maurice Meaudre, '' Circuit numériques et synthèse logique, un outil VHDL'', MASSON, Paris 1995.

[3] Michel Aumiaux, ''Initiation au langage VHDL'', MASSON 1996.

[4] Sébastien Moutault, Jacques Weber et Maurice Meaudre, ''Le langage VHDL, du langage au circuit, du circuit au langage'', DUNOD 2011.

[5] Thierry Schneider, ''VHDL : méthodologie de design et techniques avancées'', DUNOD, Paris 2011.

Partie II : Circuits Logiques Programmables

Sommaire :

I- Introduction

La réalisation d'un montage en électronique numériques, demande l'utilisation d'un nombre important de circuits intégrés logiques, ce qui conduit à une mise en œuvre complexe, un circuit imprimé de taille importante, et donc un prix de revient assez élevé. En effet, de nos jours, un circuit intégré comporte plus de 100000 portes, voire plusieurs millions de portes, sur la même puce de silicium.

Pour tirer avantage des structures VLSI (very large scale integration), les fabricants développent de nouvelles familles :

a) Les micro-processeurs et les mémoires RAM, ROM : qui sont très intéressants pour les fabricants de composants de base pour les systèmes informatiques.
b) Les ASSP (application specific standard procuct) : ce sont des produits sur catalogue qui sont fabriqués en très grande série. La fonction réalisée est figée par le constructeur, mais le domaine d'utilisation est spécifique à une application.
c) Les ASICs (application specific integrated circuit) : le terme ASIC est employé pour décrire l'ensemble des circuits spécifiques à une application. le circuit est conçu par l'utilisateur, puis réalisé chez le fondeur.

Les circuits spécifiques offrent une densité d'intégration très grande et l'impossibilité de modifier la fonction à réaliser. L'inconvénient majeur de ces circuits c'est le passage chez le fondeur, qui introduit un délai de quelques mois dans le processus de fabrication. Ceci a conduit les fabricants de circuits numériques, à proposer de nouveaux circuits programmables par l'utilisateur (sans passage par le fondeur), et qui sont devenus au fil des années, de plus en plus évolués, rassemblés sous le terme générique PLD.

d) Les circuits PLD (Programmable Logic Device) : Le terme PLD désigne l'ensemble des circuits programmables par l'utilisateur. Ils offrent une grande souplesse d'utilisation, un délai de mise en œuvre très faible et la possibilité de reprogrammer le circuit sans le retirer de la carte sur laquelle il est câblé.

Les circuits programmables par l'utilisateur se décomposent en deux familles :

- Les PROM, les PAL, les PLA, les GAL et les EPLD.
- Les FPGA.

On donne sur la figure (1), une classification des circuits intégrés numériques en générale, et sur la figure (2) les circuits logiques programmables [1].

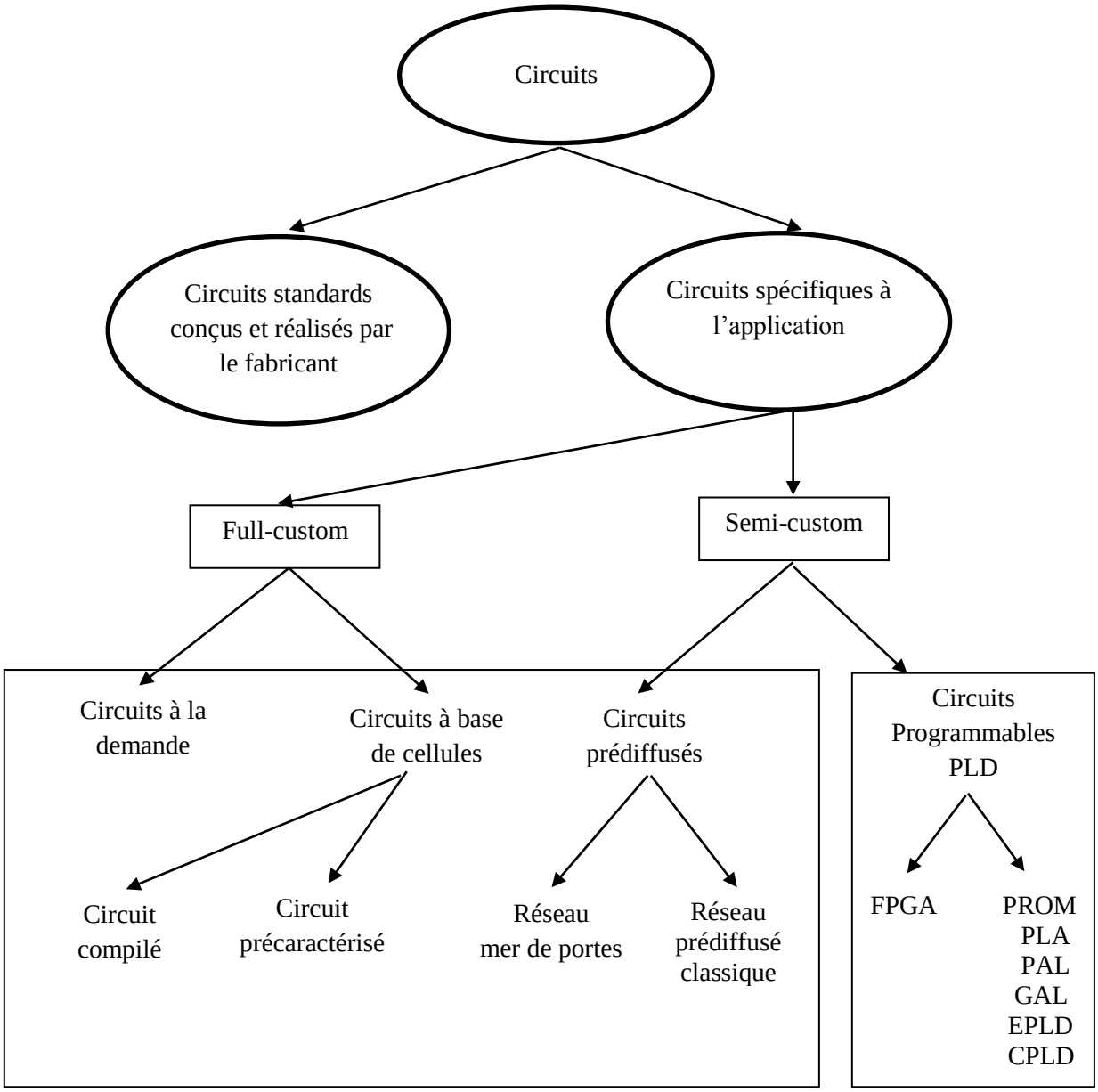


Figure1 : Classification des circuits intégrés numériques.

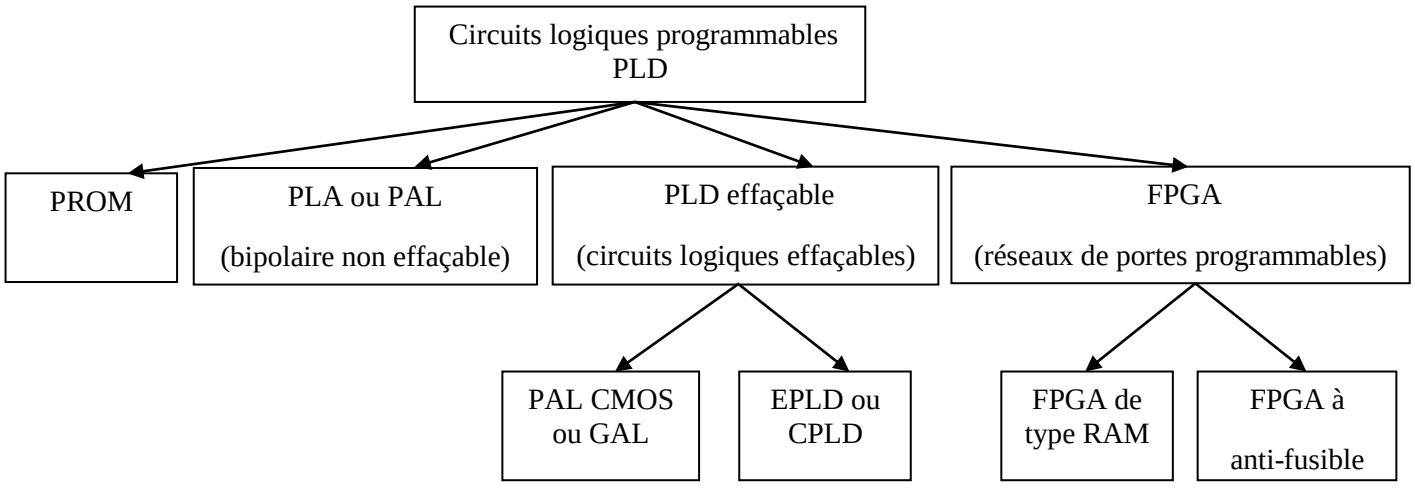


Figure 2 : Circuits logiques programmables.

II- Structure de base d'un PLD

La réalisation des fonctions combinatoires pour des composants du type PLD est obtenue en deux étapes : la réalisation des termes produits puis leur sommation. Ce qui conduit à une structure matricielle telle que, la première matrice permet de créer des mintermes et la deuxième réalise les fonctions. En effet la pluparts des circuits PLD suivent une structure possédant :

a) Un ensemble d'opérateurs ''ET'' (AND) sur lesquels viennent se connecter les variables d'entrée et leurs compléments.
b) Un ensemble d'opérateurs ''OU'' (OR) sur lesquels les sorties des opérateurs ''ET'' sont connectés.
c) Une éventuelle structure de sorties (portes inverseuses, logique 3 états, registres…).

Ce qui revient à dire, qu'un circuit PLD est formé essentiellement de deux matrices (AND) et (OR), dont les interconnexions doivent être programmables comme il est indiqué sur la figure 3 [2].

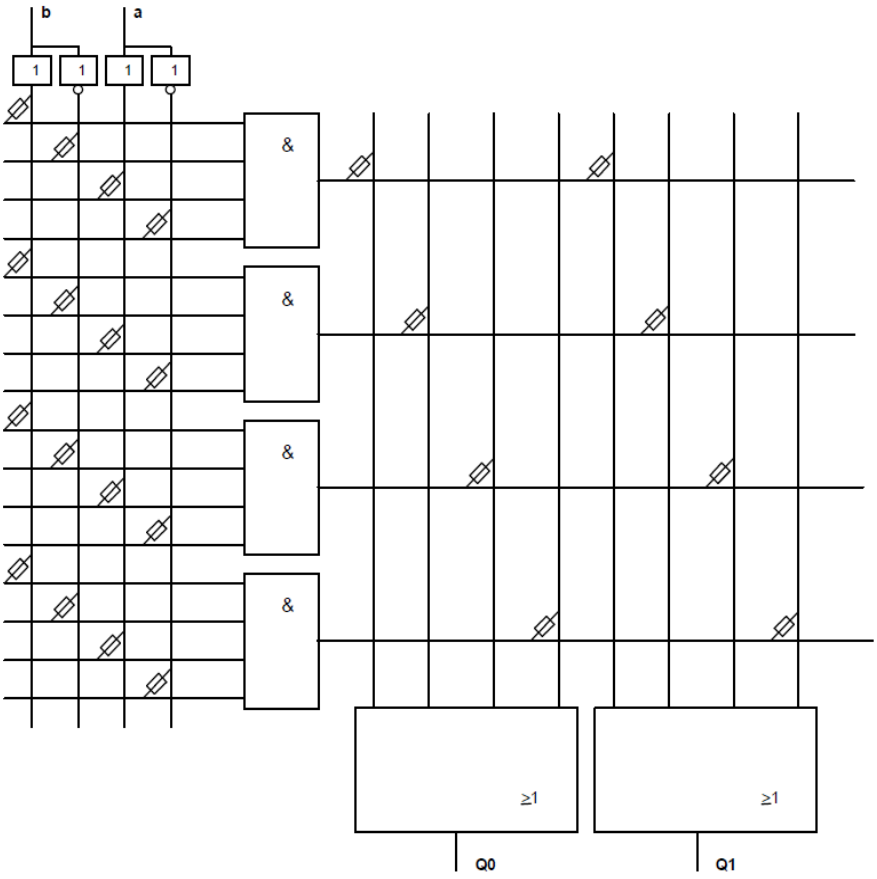


Figure 3 : Structure de base d'un PLD.

III- Technologies utilisées pour les interconnexions :

La technologie utilisée pour matérialiser les interconnexions détermine les aspects électriques de la programmation : maintien (ou non) de la fonction programmée en absence de l'alimentation, possibilité (ou non) de modifier la fonction programmée, nécessité (ou non) d'utiliser un appareil spécial (un programmateur). Parmi les technologies qui existent, on site :

III-1 Interconnexion Fusibles :

Première méthode employée, la connexion par fusibles, est en voie de disparition. On ne la rencontre plus que dans quelques circuits de faible densité et de conception ancienne. La figure 4 illustre ce genre d'interconnexions.

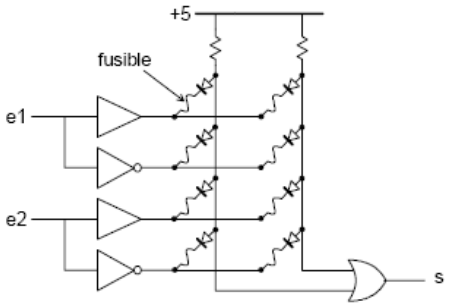


Figure-4 - : PLD élémentaire à fusibles.

Toutes les connexions sont établies à la fabrication. Lors de la programmation le circuit est placé dans un mode particulier par le programmateur, mode dans lequel des impulsions de courant sont aiguillées successivement vers les fusibles à détruire.

III-2 Interconnexion par anti-fusible :

Avec cette technique, c'est l'opération inverse qui est réalisée. On ne coupe pas une liaison, mais on l'établit. L'anti-fusible isole deux lignes métalliques placées sur deux niveaux différents grâce à une fine couche d'oxyde de silicium comme indiqué sur la figure 5. Si on applique une impulsion élevée (≈21V) calibrée en temps (moins de 5 ms), la couche d'oxyde est trouvée et les deux lignes se retrouvent en contact. Le boîtier n'est programmable qu'une seule fois par l'utilisateur. Cette méthode est peu utilisée.

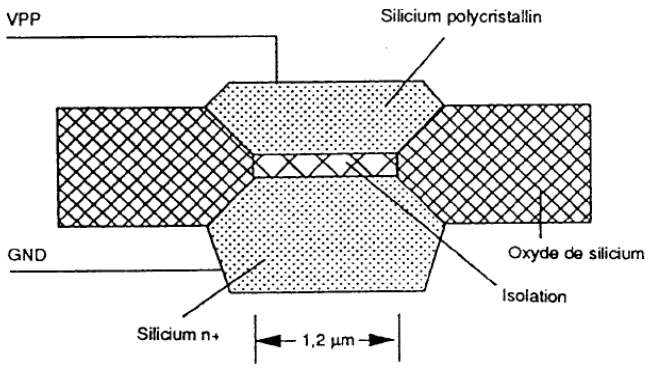


Figure-5- : Principe de l'anti-fusible.

IV- Mode de codage

Trois principes sont utilisés pour coder les fonctions combinatoires :

- PLA : On réalise des termes produits dans un réseau d'interconnexions, généralement deux matrices de portes logiques AND et OR sont utilisées.
- LUT : Des mémoires (look up table) qui contiennent l'équivalent d'une table de transposition sont utilisées.
- MUX : Des multiplexeurs câblés constituent l'essentiel des ressources de codage. La façon dont sont connectées les entrées des multiplexeurs permettent de réaliser un certain nombre de fonctions.

Sur les figures 6 et 7 on donne des exemples de programmation (par fusible) des matrices OR et AND respectivement.

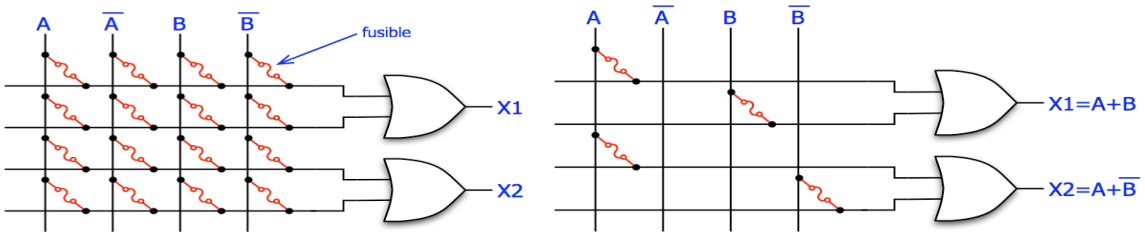


Figure 6 : Exemples de matrice OR non programmée et de matrice OR programmée.

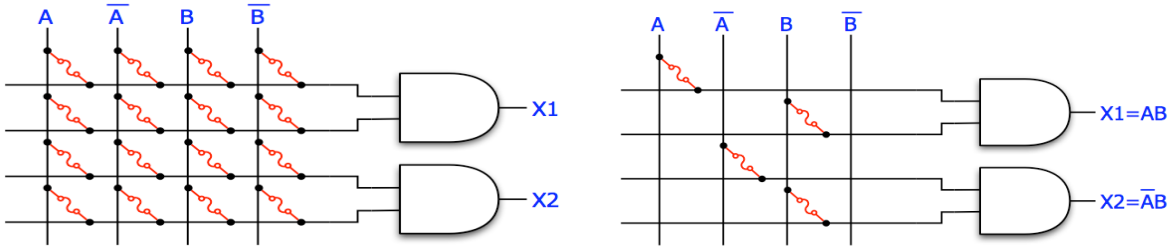


Figure 7 : Exemples de matrice AND non programmée et de matrice AND programmée.

La figure 8 illustre un exemple de représentation de la fonction OU EXCLUSIF sur un PLD.

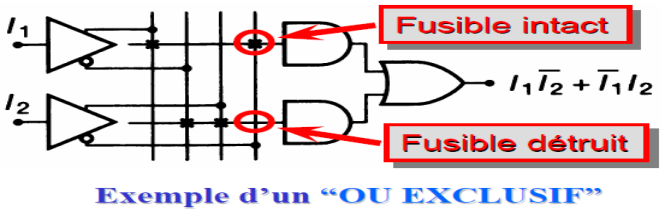


Figure 8 : Représentation de la fonction OU EXCLUSIF.

V- Familles des PLDs

Il existe plusieurs familles de PLDs qui diffèrent selon leurs architectures internes. La différence réside surtout, dans la manière de la programmation des deux matrices AND et OR. Certains circuits PLDs possèdent, en plus de cette structure matricielle, une logique séquentielle et parfois même combinatoire, qui intervient au niveau des sorties du circuit.

Un circuit logique programmable est caractérisé par:

• Le nombre d'entrées.

• Le nombre de sorties.

• Le nombre de termes produits par sortie.

• Le retard de propagation (vitesse).

• La consommation de puissance.

• La technologie.

La plupart de ces paramètres apparaissent dans le nom du circuit comme le montre l'exemple de la figure 9.

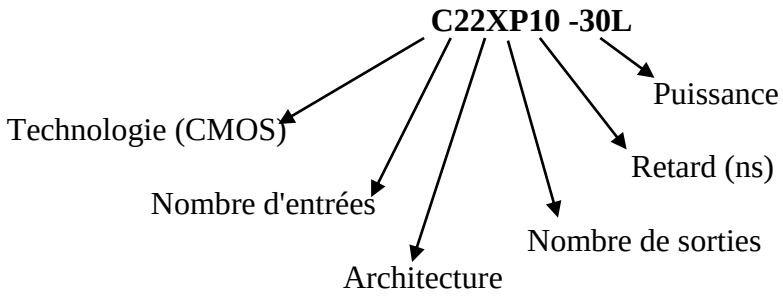


Figure 9 : Exemple d'un circuit PLD

Architecture : Indique la façon avec laquelle est conçue la sortie du circuit programmable.

V-1 Circuit PROM (Programmable read only memory):

Les premiers circuits programmables apparus sur le marché sont les PROM bipolaires à fusibles. En effet les constructeurs se sont inspirés du circuit ROM, qui permettait uniquement le stockage de données, pour le rendre plus utile en lui ajoutant la possibilité d'être programmé, ce qui donne naissance au circuit PROM constitué d'un réseau ''AND'', fixe et d'un réseau ''OR'' programmable.

On donne l'exemple d'une structure logique d'une PROM bipolaire à fusibles sur la Figure 10. Dans cet exemple, Chaque sortie O_i peut réaliser une fonction OU de 16 termes produits de certaines combinaisons des 4 variables d'entrées A, B, C et D. Avec les circuits PROM, les fonctions logiques programmées sont spécifiées par les tables de vérités. Il suffit de mettre les variables d'entrées sur les adresses et de récupérer ensuite la fonction logique sur le bit de donnée correspondant. Le temps de propagation est indépendant de la fonction implantée (c'est le temps d'accès de la mémoire) [2].

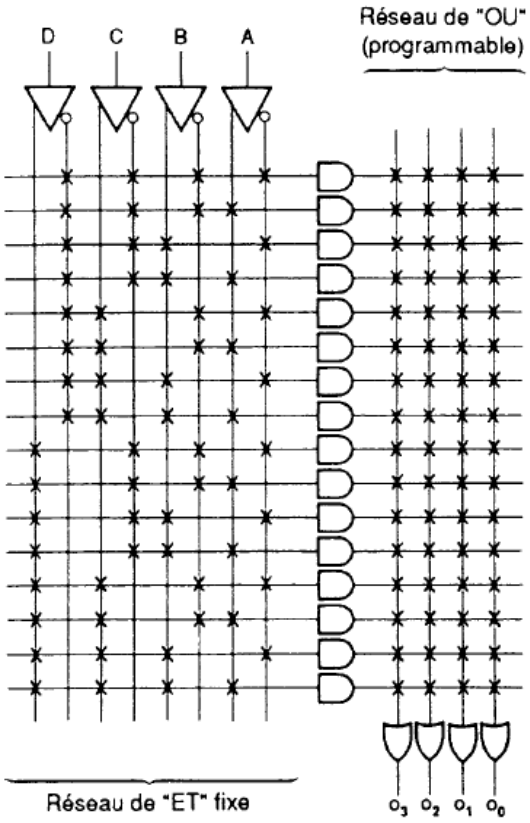


Figure 10: Structure d'un circuit PROM.

V-2 Circuit PLA (Programmable Logic Array):

Juste après la création du circuit PROM, le concept du PLA a été introduit en se basant sur la technique des fusibles des PROM bipolaires. La programmation consiste à faire sauter ces fusibles pour la réalisation des fonctions logiques. Un circuit PLA est constitué d'un réseau de ''ET'' programmable et d'un réseau de ''ou'' programmable ce qui nous offre une possibilité de programmation assez large et assez souple du moment qu'on peut manipuler les deux matrices AND et OR. Cependant, il s'avère inutile d'avoir autant de possibilité de programmation d'autant que les fusibles prennent beaucoup de place sur le silicium. Ces circuits n'ont pas réussis à entrer dans le marcher des circuits programmables et la demande c'est dirigée plutôt vers les circuits PAL.

V-3 Circuit PAL (Programmable Array Logic) :

Ce type de circuit a été introduit par le constructeur AMD dans les années 80. Son architecture a été conçue à partir d'observations indiquant qu'une grande partie des fonctions logiques ne demande que quelques termes produits par sortie. Sa structure est donc obtenue par un réseau AND qui est programmable, et un réseau OR fixe. La fusion des fusibles est obtenue en appliquant à leurs bornes une tension de 11.5v pendant un laps de temps de 10 à 50 μs . L'avantage de cette architecture est l'augmentation de la vitesse par rapport aux PLA. En effet, comme on diminue au niveau du nombre de connexions programmables, on arrive à faire réduire la longueur des lignes d'interconnexion, et donc le temps de propagation entre une entrée et une sortie devient beaucoup plus faible.
Cependant, et comme nous l'avons déjà mentionné, les PAL sont programmés par destruction de fusibles. Ils ne sont donc **programmables qu'une fois**, ce qui peut être gênant en phase de développement.

On représente sur le schéma de la figure 11, une partie de la structure interne d'une PAL. En effet on représente deux entrées et leurs inverses et une sortie totalement combinatoire. Des fusibles permettent, par fusion (programmation), de déconnecter les entrées non désirées sur l'entrée d'une porte ET, donnant ainsi une équation logique de type somme de produits.

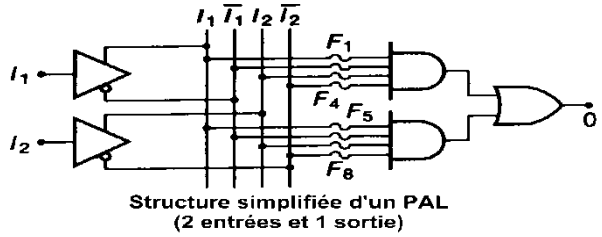


Figure 11: Partie d'une structure interne d'un circuit PAL.

L'inconvénient dans cette représentation est qu'elle demande beaucoup d'espace pour représenter un PAL en entier, ce qui a conduit les industriels à adopter une autre représentation comme on l'indique sur la figure 12. En effet, on représente les différentes lignes d'entrées d'une porte ET, par une seule ligne sur laquelle chaque fusible intact est représenté par une croix.

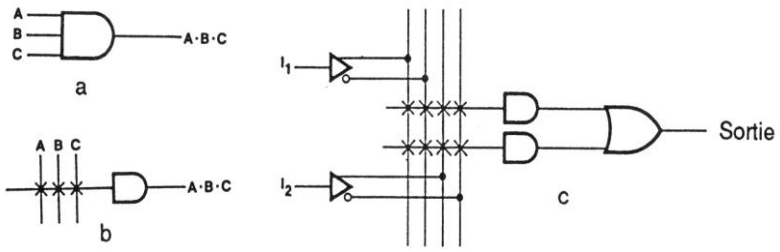


Figure 12 : Représentation de portes logiques

- Sur la figure 12. a : Représentation classique de la porte ET à 3 entrées.
- Sur la figure 12. b : Représentation PAL de la porte ET à 3 entrées. Les croix représentent les fusibles intacts.
- Sur la figure 12.c : Représentation d'une structure interne d'un circuit PAL.

On donne sur la figure 13 un exemple d'une structure de base d'un PAL où chaque sortie intègre 4 termes produits de 4 variables d'entrées. Il s'agit d'une zone logique 'ET' programmable et d'une zone logique 'OU' fixe en sortie.

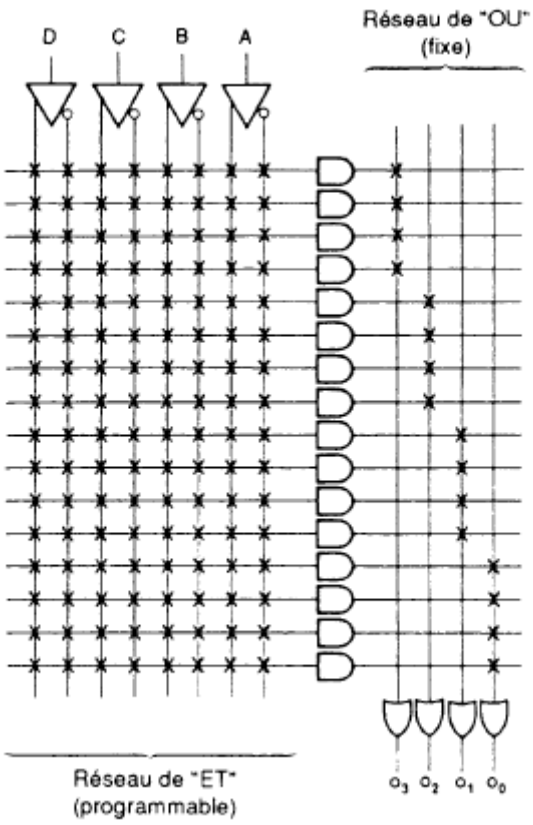


Figure 13 : Exemple d'une structure de base d'un PAL.

Parfois on n'arrive pas à représenter une sortie car elle demande un nombre plus élevé de termes produit, donc il s'avère nécessaire d'utiliser des broches spéciales, qui peuvent être programmées en :

- Entrée simple.
- En sortie réinjectée sur le réseau de AND.

L’architecture des PAL a ensuite évoluée vers les PALs séquentiels (à registres) où la sortie du réseau de ’’OR’’ sera connectée sur l’entrée d’une bascule.

Un circuit PAL est donc constitué :

- De broches d’entrées.
- De broches de sorties ou d’entrés/sorties.

Et on peut trouver aussi :

- Une entrée d’horloge.
- Une entrée de validation des sorties trois états.
- Une entrée de remise à zéro des registres.

V-3-1 Référence des circuits PAL :

Les constructeurs de circuits numériques ont définis une nomenclature permettant de décoder assez facilement les circuits PAL. On la représente sur la figure 14.

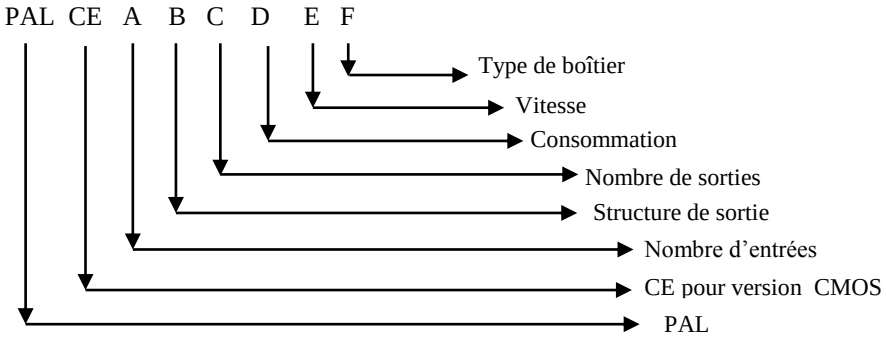


Figure 14 : Références d’un circuit PAL.

Les lettres utilisées pour représenter la structure de sortie sont L, H, C, R, RA, X, V.

V-3-2 structure de sorties des circuits PAL :

La structure de sortie d’un circuit PAL peut être constituée d’une porte ’’NOT’’, d’une porte Ou exclusive, d’une bascule D ou d’une combinaison des trois. Vu la complexité de ce genre de circuit on distingue trois types de structures de base :

a) Combinatoire :

Il existe trois types:

- H :(High) Porte ET suivit d'une Porte OU. Sortie active à l'état haut.
- L : (Low) Porte ET suivit d'une Porte NON OU. Sortie active à l'état bas.
- C : (Combinée) programmable en type H ou L.

La figure 15 représente la forme la plus simple d'un bloc élémentaire d'une PAL (PAL combinatoire).

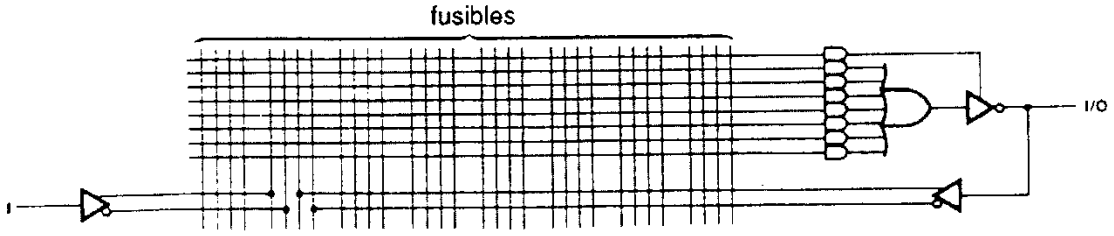


Figure 15 : Bloc élémentaire d'un circuit PAL combinatoire.

b) Séquentielle :

Il existe trois types:

- R : Registre synchrone (D)
- RA : Registre asynchrone
- X : Registre et ou exclusif

Dans ce genre de structure des bascules D, pour effectuer la logique séquentielle, sont placées à la sortie du réseau de ''OR''. Les sorties de bascules Q sont amenées, via un buffer sur une broche de sortie du circuit, quand aux sorties $\overline{Q}$ elles peuvent être réinjectées via un inverseur ou un non inverseur comme dans un circuit PAL combinatoire.

On trouve aussi :

- Des sorties séquencées sur une horloge H.
- Une commande OE (Output/Enable) qui permet d'activer ou de désactiver la sortie.

Le schéma de la figure 16 illustre cette structure.

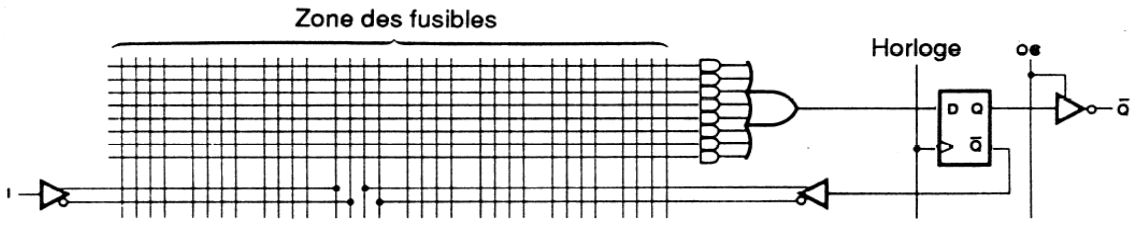


Figure 16: Bloc élémentaire d'un circuit PAL Séquentiel.

Il existe aussi des circuits PALs à sortie 'OU exclusif', que l'on représente sur la figure 17, et qui sont un peu moins utilisés.

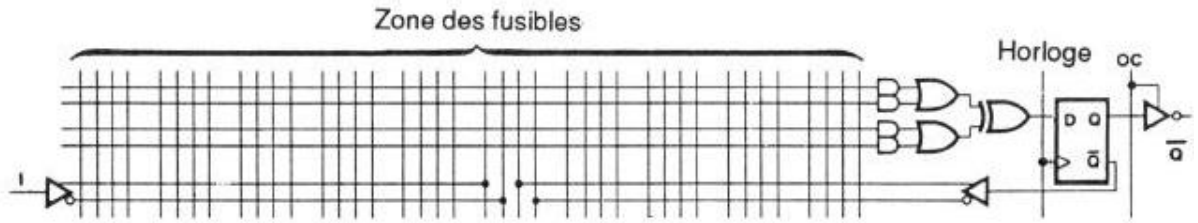
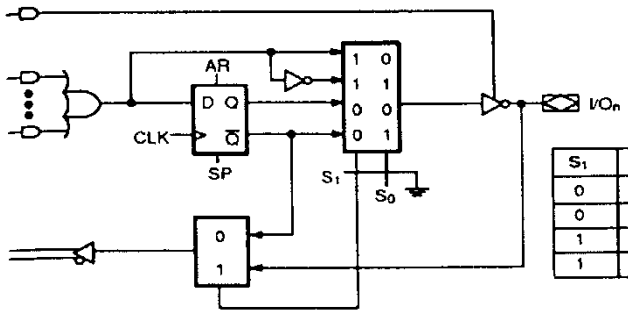


Figure 17: Bloc élémentaire d'un circuit PAL Séquentiel.

c) Versatile :

- Désignés par la lettre V.

Les circuits PAL ont évolués après vers des circuits programmables de haut niveau. La différence entre ces derniers et les PAL classiques réside dans la structure de leurs cellules de sortie. En effet les broches de sortie de tels circuits peuvent être configurées par programmation, ce qui va donner une diversité dans leurs utilisations (parfois en mode combinatoire et parfois en mode séquentiel d'où le nom versatile). La structure de sortie ainsi que sa configuration sont représentées sur la figure 18.



S_1	S_0	Configuration des sorties
0	0	Registre, actif bas
0	1	Registre, actif haut
1	0	Combinatoire, actif bas
1	1	Combinatoire, actif haut

Figure 18: Bloc élémentaire d'un circuit PAL Séquentiel.

Cette configuration permet la réalisation de nouveaux circuits que l'on nomme PAL CMOS, ces derniers éliminent l'inconvénient des PAL classiques qui ne peuvent être programmés qu'une seule fois et engendrés ainsi un gaspillage important lors du développement du circuit. En effet les fusibles sont remplacés par des transistors **MOS** pouvant être régénérés,

On donne comme exemple de ces circuits (PAL CMOS) une représentation de la structure interne du PAL CMOS 22V10 sur la figure 19 [3].

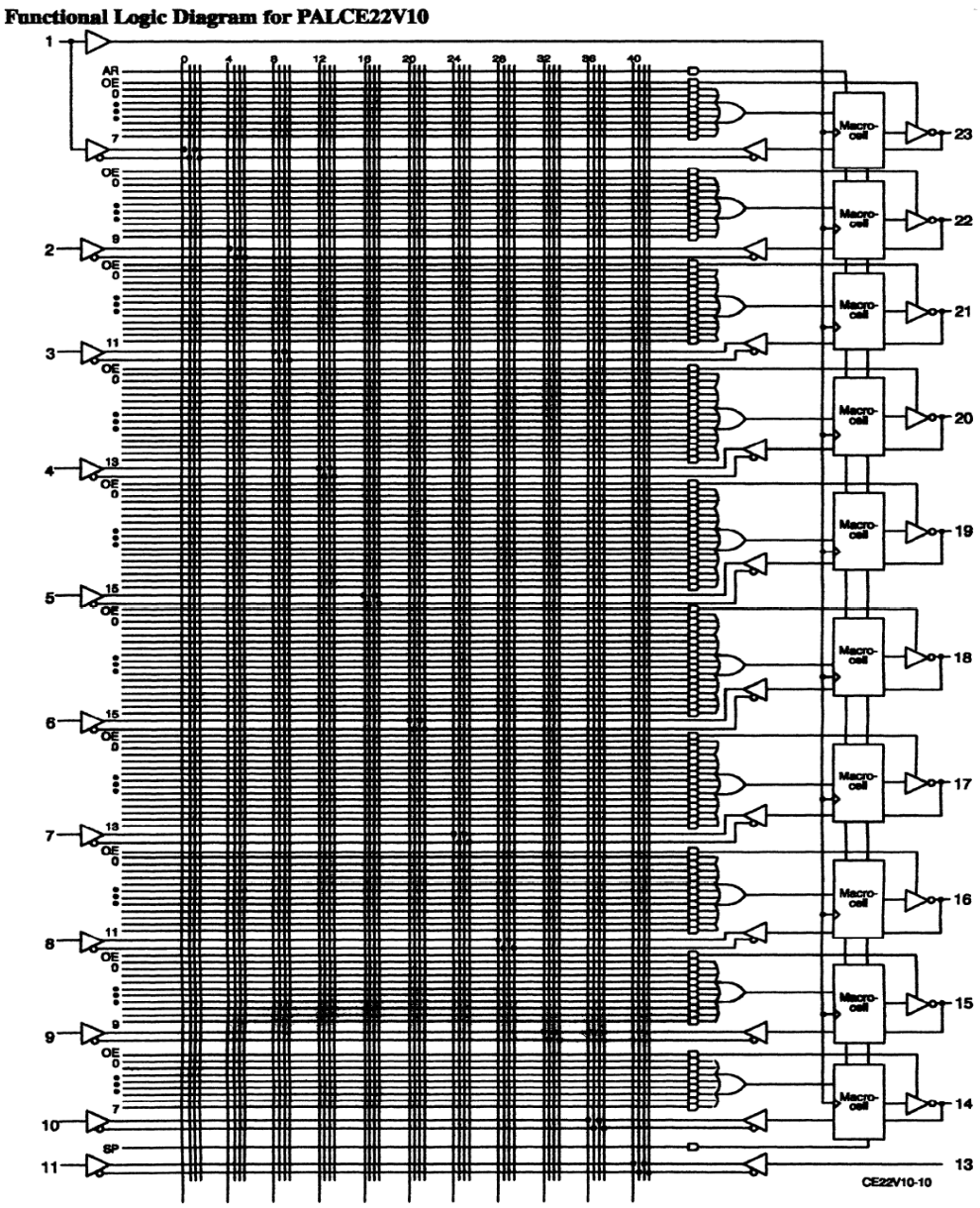


Figure 19: Structure interne du PAL CMOS 22V10.

V-4 Circuit GAL (Generic Array Logic) :

D'après la structure des PAL, il est clair que ces derniers présentent l'inconvénient d'être programmable une seule fois, ce qui donne un gaspillage important, lorsqu'on veut développer un nouveau circuit. Pour remédier à ce problème, la firme LATICE a pensé à remplacer les fusibles irréversibles des PAL, par des transistors MOSFET pouvant être régénérés et atteindre les objectifs suivants :

- Proposer des produits fabriqués selon une technologie qui permet des tests sur la zone de programmation lors de la fabrication du circuit, ce qui été impossible pour les PAL bipolaires puisque cela consistait à faire sauter des fusibles.
- Permettre de remplacer les PAL bipolaires dans n'importe quelle application.
- Offrir une plus faible consommation par rapport au PAL bipolaires.
- Avoir une plus grande souplesse de configuration des entrées/sorties par rapport à celle des PAL.

Tout ceci a donc donné naissance aux circuits GAL ''Réseaux Logiques Génériques'', qui peuvent être reprogrammés à volonté, et qui sont en fait des circuits PAL CMOS programmables, mais surtout effaçable électriquement. Les GAL ont été commercialisés pour la première fois en 1985 par la société américaine Lattice Semi-conducteur. Depuis, certains constructeurs des circuits PAL, comme AMD, Texas et Cypres, ont commercialisé eux aussi des circuits analogues aux GAL et ont appelé leurs produits PAL CMOS ou E2PAL ou bien PAL EECMOS puisque il s'agit en fait de circuits en technologie CMOS programmable et effaçable électriquement. Entre temps la firme LATTICE a continuée à développer sa famille propre de GAL et propose des références n'ayant, à ce jour, pas de concurrents. Depuis 1995 cette même société propose des circuits appelés pLSI et surtout ispLSI ''in system programmable LSI'' et qui signifie, circuits logiques à grande échelle, que l'on peut programmer (et effacer) sur le circuit où ils sont installés.

V-4-1 Réseau logique de portes :

Pour résoudre le principal désavantage du PAL, le constructeur Lattice Semiconductors a introduit le Gate Array Logic (GAL) (Réseau logique de portes), que l'on pourrait traduire aussi par ''Réseau logique Générique'' en remplaçant les fusibles irréversibles des PAL par des dispositifs appelés ''Floating Gate Transistors'' (FGMOS). Ces circuits peuvent donc être reprogrammés à volonté sans pour autant avoir une durée de vie restreinte. La structure de base est donc la même que celle des PAL, matrice ET programmable matrice OU fixe comme indiqué sur la figure 20.

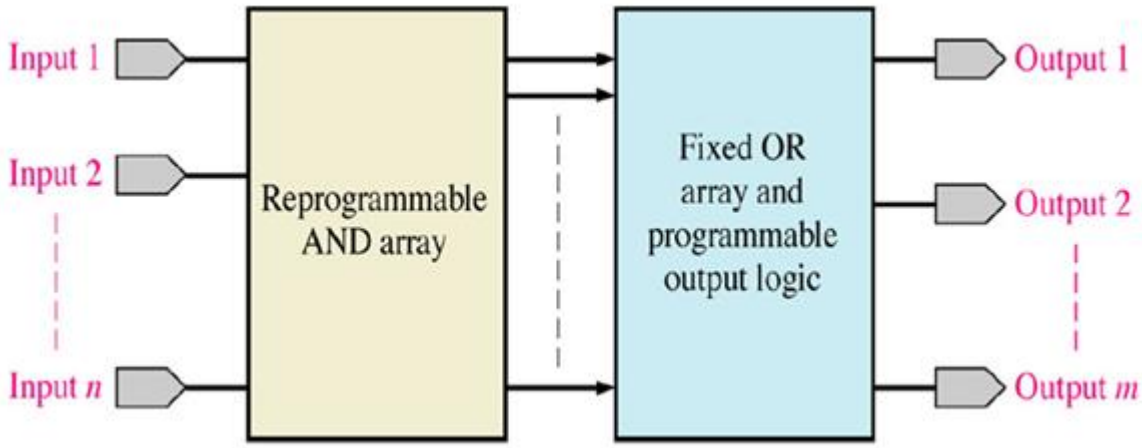


Figure 20: Structure interne d'un circuit GAL.

Au niveau de la structure interne, les GAL sont constitués de transistor CMOS alors que les PAL classiques sont constitués de transistors bipolaires. Il reste encore un inconvénient important dans les circuits PAL qui vient du fait que leurs structures de sorties soit figée pour chaque composant. Il est donc nécessaire de choisir le type de circuit PAL selon les fonctions à réaliser. Pour cela on a introduit dans la structure de sorties des circuits GAL des macros cellules programmables qui rendent le circuit **versatile**, ce qui veut dire qu'il est possible, par programmation, de choisir entre une configuration de sortie combinatoire ou une configuration de sortie séquentielle.

Donc un circuit GAL est constitué de :

- Une matrice AND reprogrammable et effaçable électriquement (EECMOS).
- Une matrice OR fixe (comme celle des PAL).
- Une configuration de sortie programmable à base de macro cellule (OLMC).

V-4-2 Matrice AND :

Dans un circuit PAL la matrice AND contient des fusibles (figure 21), alors que dans un circuit GAL elle est reprogrammable et effaçable électriquement et contient des cellules E2CMOS qui ont pour élément de base le transistor MOSFET (figure 22).

Figure 21 Matrice AND dans un PALClassique.

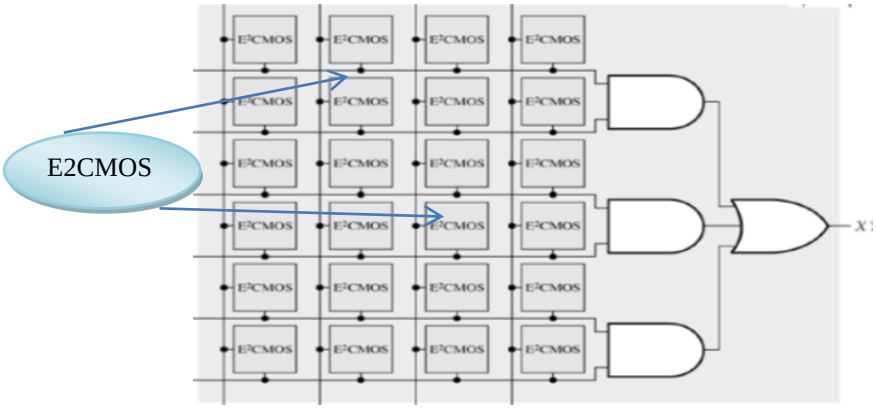


Figure 22 : Matrice AND dans unGAL.

V-4-3 Architecture des Cellules E2CMOS :

La possibilité de multiprogrammation des GAL est obtenue en utilisant des transistors à effet de champ à structure MOS (Métal-Oxyde-Semi-conducteur) avec une grille supplémentaire flottante.

Dans un transistor MOS classique la grille est utilisée pour induire un canal entre la source et le drain et les transistors MOS sont des interrupteurs, commandés par une charge électrique stockée sur leur électrode de grille. Si, en fonctionnement normal, cette grille est isolée, elle conserve sa charge éventuelle éternellement. Il reste au fondeur de trouver un moyen pour modifier cette charge et programmer l'état du transistor.

Le dépôt d'une charge électrique sur la grille isolée d'un transistor fait appel à un phénomène connu sous le nom d'effet tunnel : un isolant très mince (une cinquantaine d'angströms, 1 Å = 10-10 m) soumis à une différence de potentiel suffisamment grande (une dizaine de volts, supérieure a 3,3 ou 5 volts des alimentations classiques) est parcouru par un courant de faible valeur, qui permet de déposer une charge électrique sur une électrode normalement isolée. Ce phénomène, réversible, permet de programmer et d'effacer le circuit. [4]

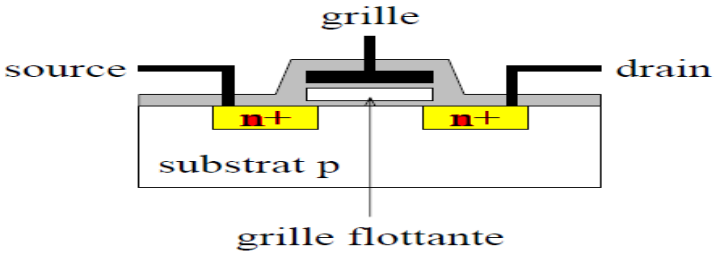


Figure 23 : Transistor MOS utilisé dans les circuits GAL**.**

Les transistors disposent de deux grilles, dont l'une est isolée. Une charge négative (des électrons) déposée sur cette dernière, modifie la tension de seuil du transistor commandé par la grille non isolée. Quand cette tension de seuil dépasse la tension d'alimentation, le transistor est toujours bloqué (interrupteur ouvert). Une variante (plus ancienne) de cette structure consiste à mettre deux transistors en séries, l'un à grille isolée, l'autre normal. Le transistor à grille isolée est programmé pour être toujours conducteur ou toujours bloqué ; on retrouve exactement la fonction du fusible et la réversibilité en plus.

Le contrôle des dimensions géométriques des transistors permet actuellement d'obtenir des circuits fiables, programmables sous une dizaine de volts, reprogrammables à volonté (plusieurs centaines de fois), le tout électriquement.

Les puissances mises en jeu lors de la programmation sont suffisamment faibles pour que les surtensions nécessaires puissent être générées par les circuits eux-mêmes. Vu par l'utilisateur, le circuit devient alors programmable in situ, c'est à dire sans appareillage accessoire. Dans ces circuits, un automate auxiliaire gère les algorithmes de programmation et le dialogue avec le système de développement, via une liaison série. [5]

La figure 24 montre la structure d'un PLD élémentaire, dans lequel les fusibles sont remplacés par des transistors à grille isolée (technologie FLASH).

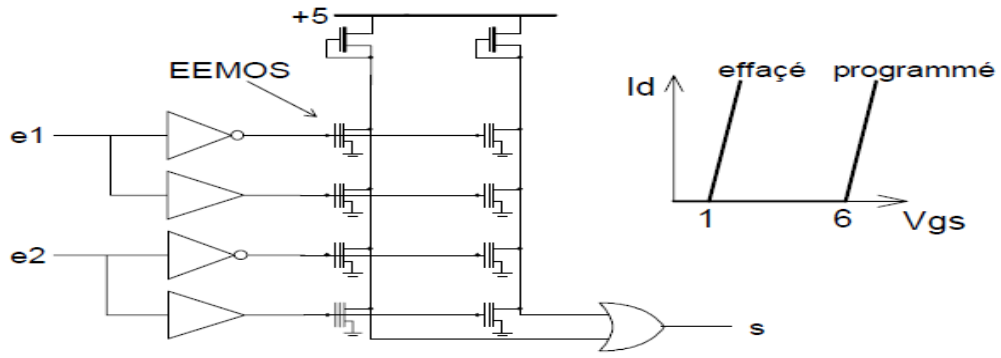


Figure 24 : Transistors CMOS dans la matrice AND

V-4-4 Macro-cellule de sortie (OLMC) :

La macro-cellule programmable OLMC (Output Logic Macro Cell), permet la réalisation de divers structures de sorties et ainsi d'émuler n'importe quel circuit PAL. Cette cellule est ajoutée après la matrice OR comme il est indiqué sur la figure 25.

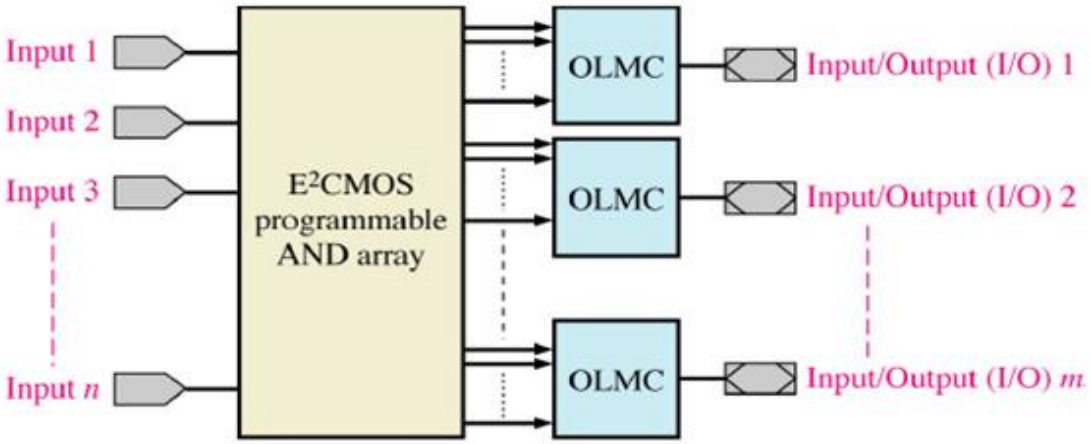


Figure 25: Structure interne du PAL CMOS 22V10.

On donne un exemple de représentation de la structure d'une macro-cellule OLMC sur le schéma de la figure 26, pour certains circuits GAL (GAL22V10, GAL18V10 et GAL26CV12).

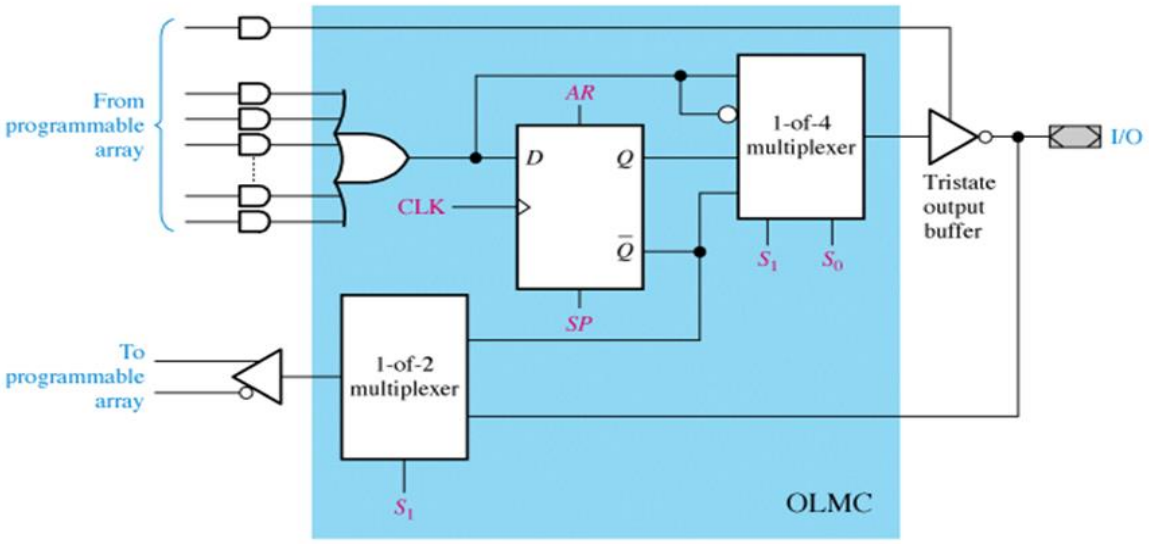


Figure 26 : Structure d'une macro-cellule OLMC.

Selon la configuration des multiplexeurs on obtient différentes structures de sorties résumées sur le schéma de la figure 27.

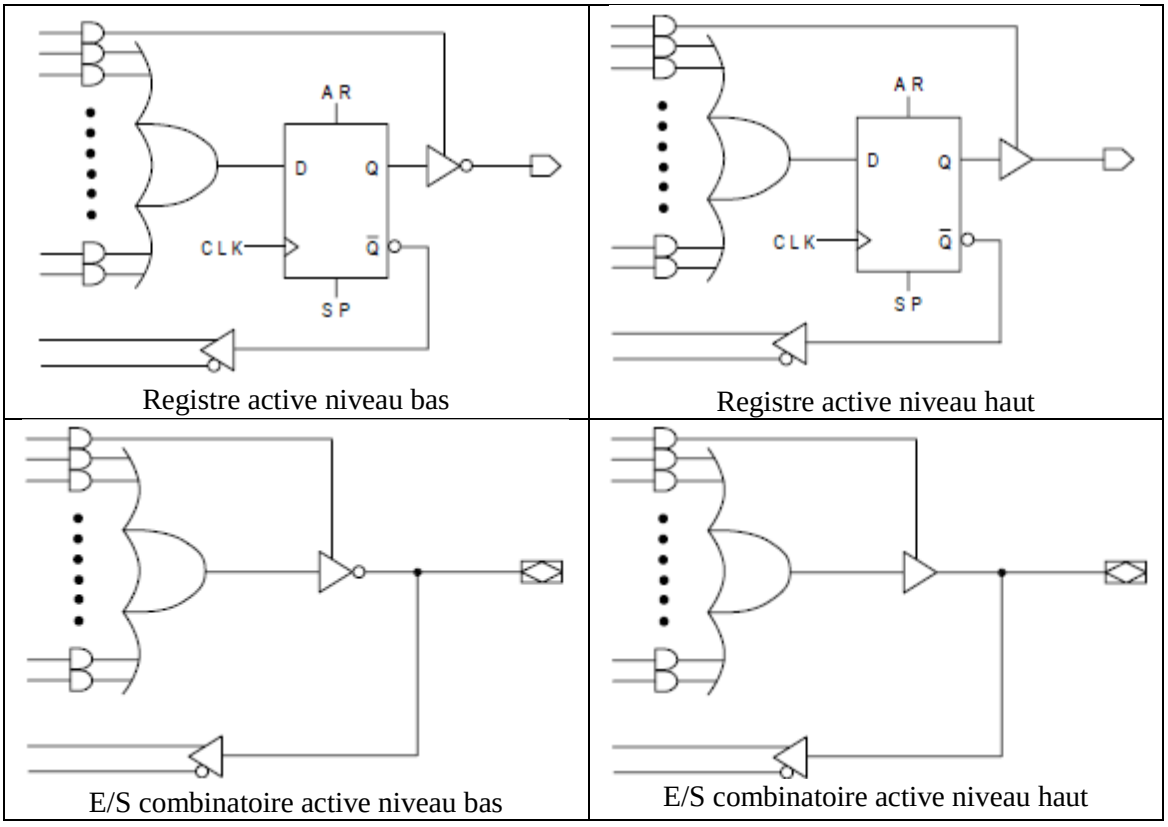


Figure 27 : Configurations possibles d'une OLMC

On donne comme exemples les différentes configurations possibles de la macro cellule du circuit GAL 16V8 :

- En registre synchrone sortie 3 états on peut émuler un PAL 16R8.
- En entrée/Sortie combinatoire sortie 3 états on peut émuler un PAL 16R4 et un PAL 16R6.
- En entrée et/ou Sortie combinatoire on peut émuler un PAL 10L8 et un PAL 12H6.
- En entrée combinatoire on peut émuler un PAL 12L6.
- En entrée/Sortie combinatoire sortie 3 états on peut émuler un PAL 16L8 et un PAL 16H8.

Remarque :

- La programmation des cellules de sortie est transparente pour l'opérateur. C'est le logiciel de développement qui, en fonction de certaines indications (sortie, entrée, registre ou combinatoire), effectue la configuration des structures de sortie.
- Les PAL à registres étaient appelés FPLS.

V-4-5 Référence des Circuits GAL

Chaque constructeur de circuits programmables pose une nomenclature pour identifier son propre produit. On donne sur la figure 28 un exemple d'identification d'un circuit GAL de la firme AMD.

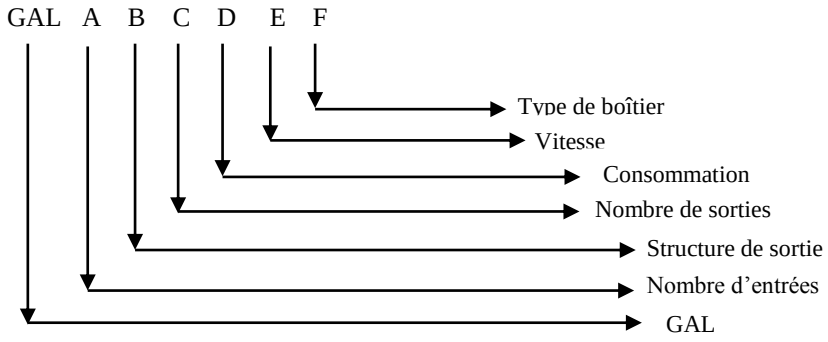


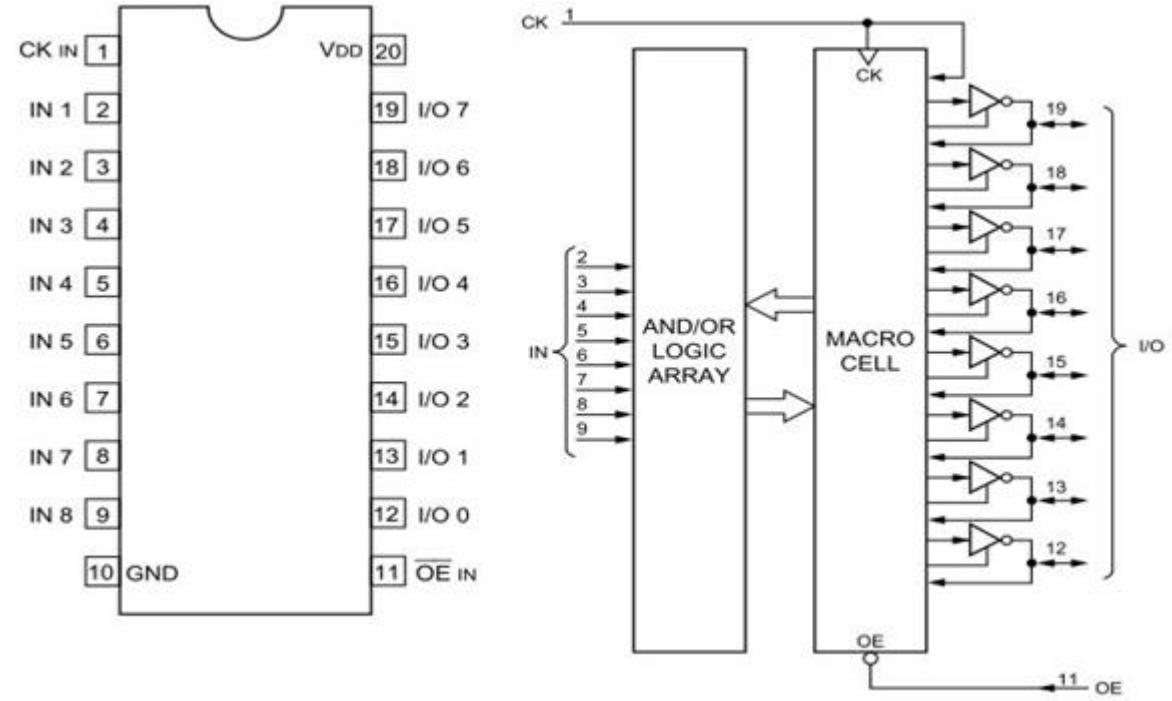
Figure 28 : Références d'un circuit GAL par AMD.

On résume sur le tableau suivant quelques types de circuits GAL commercialisés.

Référence	Nombre de broches	Vitesse (ns)	Consommation (mA)	Structure de sortie
GAL 16V8	20	10, 15 ou 20	55 ou 115	Macro cellule
GAL 18V10	20	15 ou 20	115	Macro cellule
GAL 20V8	24	10, 15 ou 25	55 ou 115	Macro cellule
GAL 20R10	24	15 ou 20	115	Registre asynchrone
GAL 22V10	24	15, 20 ou 25	130	Macro cellule
GAL 26V12	28	15 ou 20	130	Macro cellule
GAL 6001	24	30 ou 35	150	Macro cellule type FPLA
IspGAL16Z8	24	20 ou 25	90	Macro cellule programmable en circuit

Tableau 1 : Circuits GAL commercialisés

On donne, à titre d'exemple, sur la figure 29 le brochage du circuit GAL 16V8 qui utilise une architecture de matrice ET programmable (64x32) comportant 64 termes produit de 32 variables complémentées ou non. La matrice OU est pré-câblée et les 64 termes produit sont répartis entre les 8 macro-cellules [4].



a. Brochage b. structure générale

Figure 29 : Brochage du circuit GAL 16V8.

V-4-6 Protection contre la duplication :

Les circuits GAL sont dotés d'un bit de sécurité qui peut être activé lors de la programmation empêchant ainsi toute lecture du contenu du circuit. Ce bit est remis à zéro seulement en effaçant complètement le circuit. Il est aussi constitué d'un ensemble de huit octets, appelé signature électronique, pouvant contenir des informations diverses sur le produit [5].

À peut près à la même époque que celle de la première commercialisation des GAL, la technologie des PAL CMOS a aussi évolué dans une autre direction sous l'impulsion de la société ALTERA qui a proposé les premiers PAL CMOS effaçable aux UV, appelés à l'époque EPLD.

VI- Circuit EPLD (Erasable Programmable Logic Device) :

Les circuits EPLD (Erasable Programmable Logic Device), c'est à dire circuits logiques programmables et effaçables soit par exposition aux UV soit électriquement, sont une évolution importante des PAL CMOS. En effet ils sont basés sur le même principe et utilisent la macro cellule dans leurs structures pour la réalisation des fonctions logiques combinatoires ou séquentielles.

Un exemple de schéma d'une macro-cellule de base d'un EPLD est présenté sur la figure 30. On remarque que le réseau logique est composé de 3 sous ensembles :

- Le réseau des signaux d'entrées provenant des broches d'entrées du circuit.
- Le réseau des signaux des broches d'entrées/sorties du circuit.
- Le réseau des signaux provenant des autres macro-cellules.

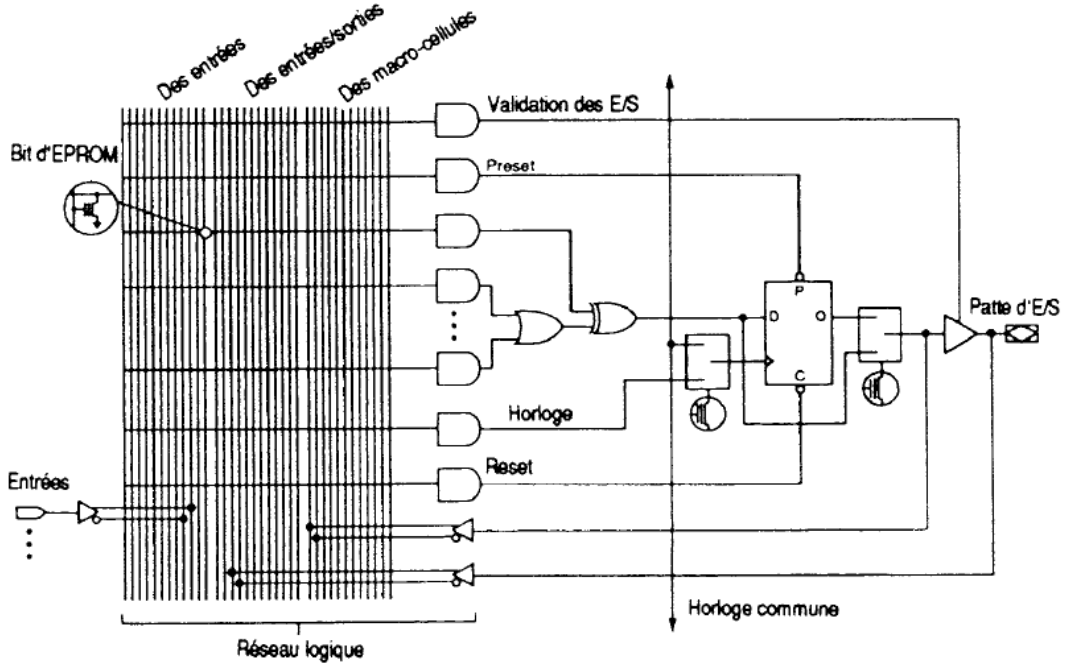


Figure 30 : Macro cellule de base d'un EPLD.

Outre la logique combinatoire, la macro-cellule possède une bascule D configurable. Cette bascule peut être désactivée par programmation d'un multiplexeur. Le signal d'horloge peut être commun à toutes les macro-cellules, ou bien provenir d'une autre macro-cellule via le réseau logique. Quelque soit la famille d'EPLD, la fonctionnalité de la macro-cellule ne change guère.

Plusieurs types d'EPLD existent en technologie CMOS, on site :

- Les circuits programmables électriquement et non effaçables et qui sont du type OTP (One Time Programmable).
- Les circuits programmables électriquement et effaçables aux UV (obsolètes).
- Les circuits programmables électriquement et effaçables électriquement dans un programmateur.
- Les circuits programmables électriquement et effaçables électriquement sur la carte (ISP : In System Programmable), utilisant une tension unique de 5 V.

Les EPLD offrent des procédés physiques d'intégration beaucoup plus importants que ceux autorisés par les PAL CMOS. En effet, les plus gros EPLD actuellement commercialisés intègrent jusqu'à 24000 portes logiques dont 12000 sont réellement accessibles à l'utilisateur. On peut ainsi loger dans un seul boîtier, l'équivalent d'un schéma logique utilisant jusqu'à 50 à 100 PAL classiques [6].

On rencontre parfois le terme CPLD (Complex Programmable Logic Device), et qui est utilisé pour désigner des EPLD ayant un fort taux d'intégration.

VII- Circuit FPGA

Lancé sur le marché en 1984 par la firme XILINX, le FPGA (Field Programmable Gate Array) est un circuit pré-diffusé programmable. Le concept du FPGA est basé sur l'utilisation d'une unité appelée LUT (look up table) comme élément combinatoire de la cellule de base. La LUT peut être vu comme une mémoire (en générale 16 bits) qui permet de créer n'importe quelle fonction logique combinatoire à 4 variables d'entrées. On l'appel aussi générateur de fonctions (Function generator).

VII-1 Cellule type de base

Elle contient une LUT à 4 entrées et une bascule D qui permet la réalisation de fonctions logiques séquentielles. En sortie nous avons un multiplexeur 2 vers 1 de sorties autorise la sélection des deux types de fonctions combinatoire ou séquentielle. Nous avons sur la figure 31 un exemple de représentation d'une cellule LUT [7].

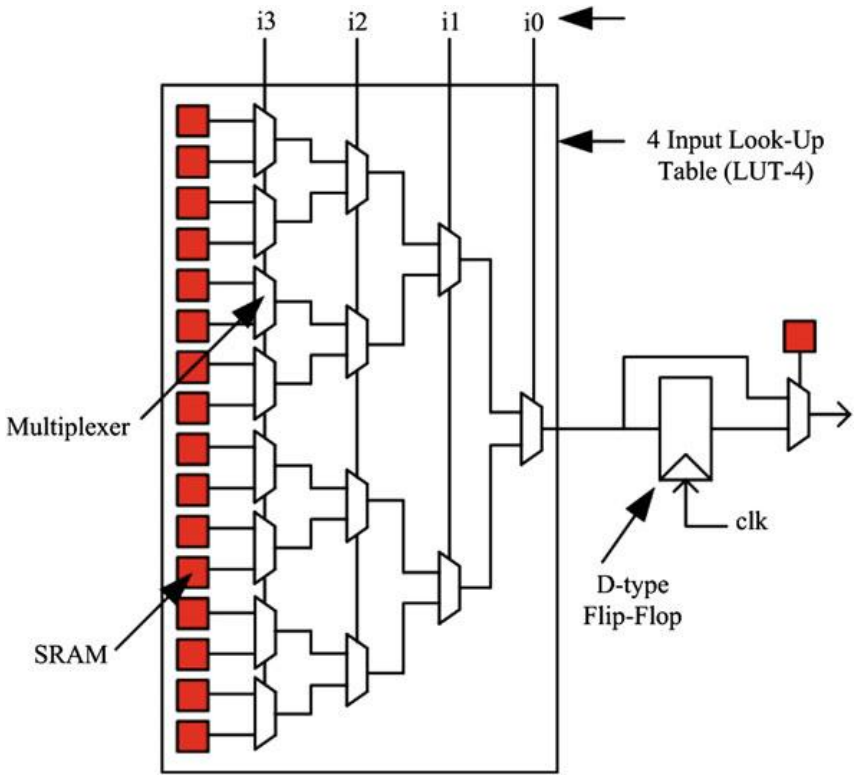


Figure 31 : Exemple d'une LUT à entrées.

Les cellules de base d'un FPGA sont disposées en lignes et en colonnes. Des lignes d'interconnexions programmables qui traversent le circuit, horizontalement et verticalement, entre les différentes cellules, et des lignes d'interconnexions permettant de relier les cellules entre elles et avec les ports d'entrées/sorties. Les connexions programmables sur ces lignes sont réalisées par des transistors MOS dont l'état est contrôlé par des cellules mémoires SRAM. Ainsi, toute la configuration d'un FPGA est contenue dans des cellules SRAM.

Les circuits FPGA sont constitués d'une matrice de blocs logiques programmables entourés de blocs d'entrée sortie programmable. L'ensemble est relié par un réseau d'interconnexions programmable comme montré sur la figure 32.

Contrairement aux circuits EPLD, on ne peut pas prédire la fréquence de travail maximale d'une fonction logique avant son implémentation. En effet cela dépend fortement du résultat de l'étape de placement-routage.

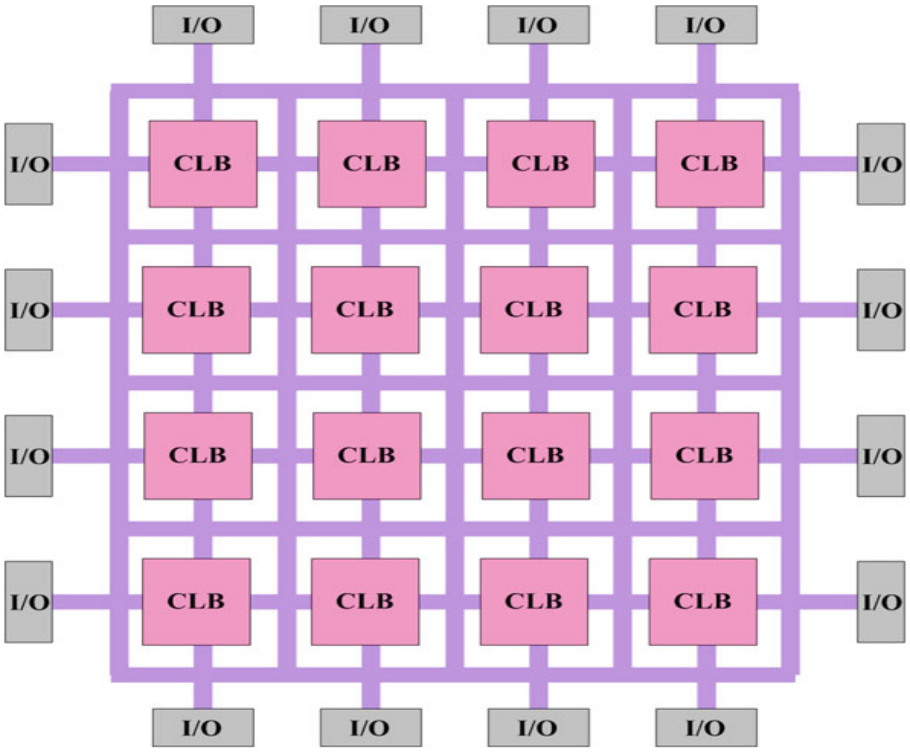


Figure 32: Structure d'un circuit FPGA.

VIII- Programmation des circuits PLD

La programmation des circuits PLD nécessite un logiciel adéquat et un programmateur pour griller les fusibles d'interconnexions. Elle demande une mise en équations logiques, ou en organigramme du problème et l'utilisation d'un fichier JEDEC.

Le fichier JEDEC est un format standard et international de programmation très répandu et qui est accepté par la plupart des programmateurs de PLD. C'est un fichier de données binaires indiquant au programmateur les fusibles à « griller ».

On résume sur le l'organigramme suivant (figure 33) les étapes de programmation de n'importe quel circuit PLD [2].

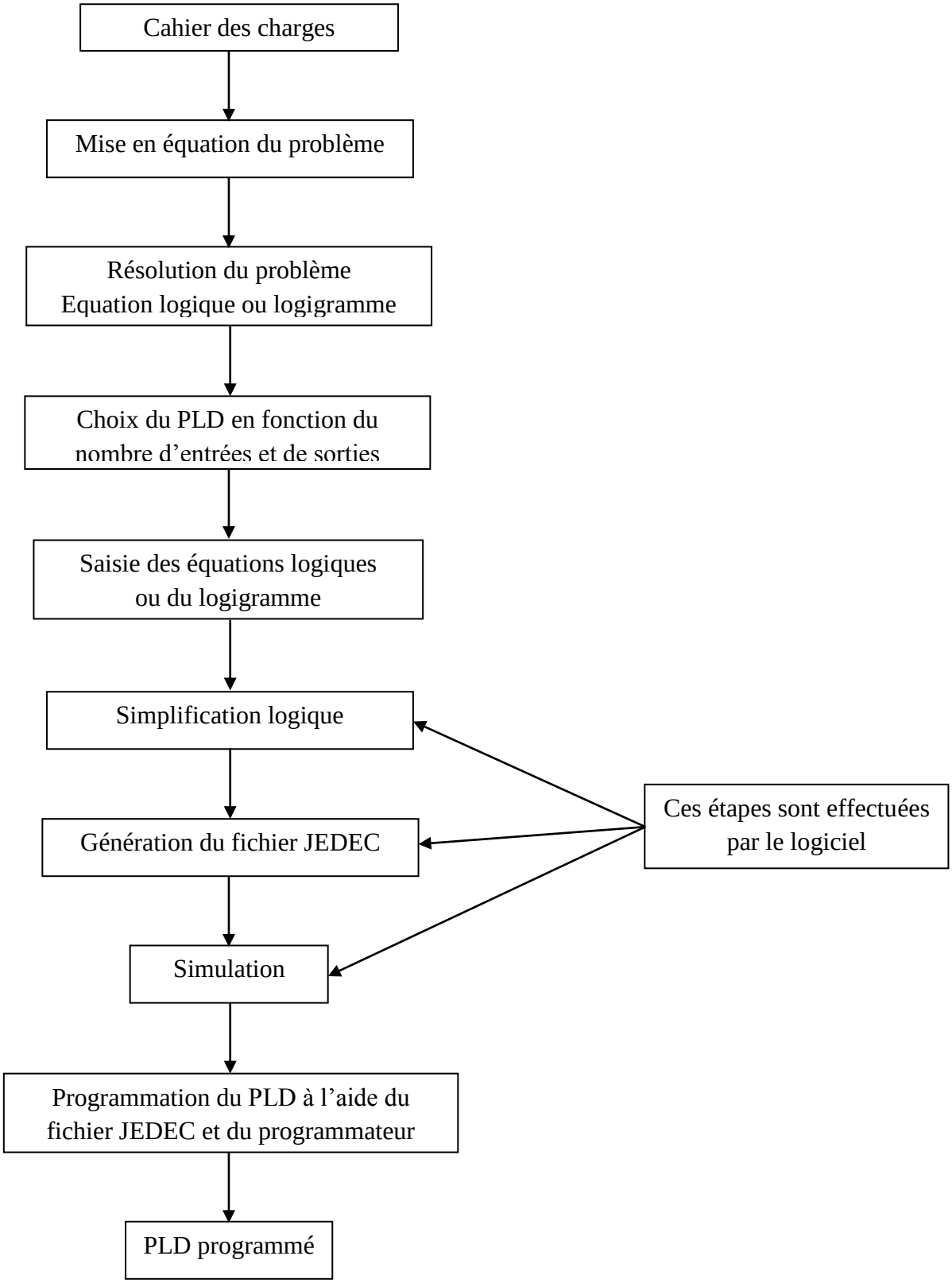


Figure 33 : Etapes de programmation d'un circuit PLD.

IX- Avantages et inconvénients des circuits PLD:

Les circuits PLD présentent certains avantages par rapport aux circuits spécifiques on site :

- Le coût et le temps de développement qui est assez court.
- Le prototypage et temps de simulation accéléré.
- La prise de risque est réduite. (on peut effectuer des modifications).
- Des supports logiciels peu chers.

Par contre ils présentent aussi certains inconvénients :

- Un niveau d'intégration moindre par rapport aux circuits ASIC.
- Une consommation plus élevée.
- Des prix importants et une faible disponibilité pour de grandes séries.

X- Lexique

- ASIC (Application Specific Integrated Circuit) : Circuit non programmable configuré lors de sa fabrication pour une application spécifique.
- CPLD (Complex Programmable Logic Device) : Désigne des PLD ayant un haut niveau d'intégration.
- EEPROM ou E2PROM (Electrical Erasable Programmable Read-Only Memory) Mémoire programmable à lecture seule, effaçable électriquement.
- EPLD (Erasable Programmable Logic Device) : Circuits logiques reprogrammables.
- EPROM (Erasable Programmable Read-Only Memory) : Mémoire programmable à lecture seule, effaçable par ultraviolets.
- FPGA (Forecasting Programmable Gate Array) : Réseau de portes programmables à la demande. Technologie qui utilise des circuits encapsulés comportant des réseaux de portes logiques non reliées : l'utilisateur réalise les interconnexions nécessaires par programmation.
- FPLS (Field Programmable Logic Sequencer) : Ancien nom donné aux PAL à registres.
- GAL (Generic Array Logic) : Circuits logiques PAL reprogrammables à technologie CMOS.
- ISP (In System Programmable) : Circuit que l'on peut programmer (et donc effacer) même lorsqu'il est en place sur l'application.
- JEDEC : Format de fichier de programmation des circuits logiques (image des fusibles à griller).
- LSI (Large Scale Integration) : Intégration à grande échelle : Circuits regroupant quelques centaines à quelques milliers de portes logiques (CI de télécommande, décodeur de code à barre, etc …).
- MSI (Medium Scale Integration) : Intégration à échelle moyenne : Circuits regroupant quelques dizaines de portes logiques (décodeurs, multiplexeurs, bascules …).
- OTP (One Time Programmable) : Circuits programmables électriquement et non effaçables.
- PAL (Programmable Array Logic) : Circuits logiques programmables dans lesquels seules les fonctions ET sont programmables, les fonctions OU ne le sont pas.
- PAL CMOS ou PAL EECMOS : Qui font référence aux circuits GAL.
- PLA (Programmable Logic Array) : Circuits logiques programmables dans lesquels les deux fonctions ET, OU sont programmables.
- PLD (Programmable Logic Device) : Famille des circuits programmables qui comprend les PAL, GAL, EPLD et FPGA.
- SSI (Small Scale Integration) : Intégration à petite échelle : Circuit ne regroupant que quelques portes logiques (fonctions de base des séries 74 ou 4000).
- VHDL : Langage de programmation utilisé pour programmer les PLD.
- VLSI (Very Large Scale Integration) : Intégration à très grande échelle : Circuits regroupant quelques dizaines de milliers de portes logiques (microprocesseurs …).

XI- Bibliographie

[1] Christian Tavernier, ''Circuit logiques programmables'', DUNOD, Paris, 1996.

[2] C. Cimelli, R. Bourgeron, '' Guide du technicien en électronique'', HACHETTE, Paris, 1995.

[3] ''Programmable logic devices (PLD)", Data Hand Book, Philips semi-conductors, 1994.

[4] ''Lattice Semiconductor'' Data Book 1996, pp.365-392.

[5] D. Sellers, ''Quality and Reliability", Hand Book, Space Electronics Inc.

[6] Fabrice CAIGNET, '' Etude des circuits logiques programmables; Les FPGA'', PDF.

[7] Haibo Wang, '' FPGA Programmable Interconnect and I/O Cells'', PDF.

[8] Jean-Luc Danger, '' Les circuits logiques programmables FPGAs'', PDF.

[9] http://proxacutor.free.fr/

[10] http://embeddedmicro.com/

[11] www.lattice.com.