الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Akli Mohand Oulhadj - Bouira -
ⵜⴰⵙⴷⴰⵡⵉⵜ ⴰⴽⵍⵉ ⵎⵓⵃⴻⵏⴷ ⵓⵍⵃⴰⵊ - ⵜⵓⴱⵉⵔⴻⵜ -

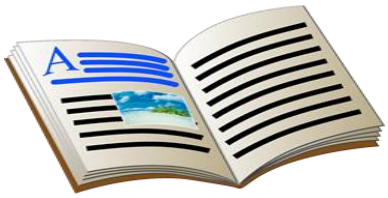
وزارة التعليم العالي والبحث العلمي
جامعة أكلي محند أولحاج
- البويرة -

Faculté des Sciences et des Sciences Appliquées

كلية العلوم والعلوم التطبيقية

Département de génie électrique

Polycopié de cours/TD



En :Electronique

Spécialité : Electronique des systèmes embarqués

Electronique numérique avancée : FPGA et VHDL

Par Fekik Arezki

Année : 2021

Avant-propos

Ce document est un support du cours qui traite le programme détaillé du module « Electronique numérique avancée : FPGA et VHDL ». Il est destiné aux étudiants de première année master inscrit dans la filière électronique, et spécialité électronique des systèmes embarqués. Ce guide est inséré dans la spécialité électronique des système embarqués, filière électronique pour donner les connaissances nécessaires sur les fonctionnalités du VHDL en conception de systèmes numériques, ainsi que l'implémentation d'une description VHDL sur une carte FPGA. Ces connaissances sont donc indispensables pour toute personne désirant une insertion dans cette spécialité dans le domaine professionnel ou de recherche. En outre ce travail de synthèse présente une étude sur les Réseaux Logiques Programmables (PLD). Il évoque l'architecture des FPGA, il donne aussi un exemple d'une carte FPGA « Spartan-6 FPGA LX9 » du constructeur Xilinx. Cependant, il est important de signaler que ce document traite des connaissances de base sur la programmation avec le VHDL. Quelques applications pour l'implémentation des circuits logiques dans la carte FPGA ont été décrites. Dans cette optique, et vu les avantages des carte FPGA dans le domaine professionnel ou de recherche, l'affectation de ce document peut être élargie à d'autres spécialités.

Objectifs de l'enseignement :

L'utilisation d'un composant à « logique programmable » (FPGA) – dont l'architecture correspondra à celle voulue par le concepteur – devient alors incontournable.
Dans cette matière, les étudiants seront capables de :

- Comprendre et utiliser un composant à logique programmable ;
- Appréhender les fonctionnalités du VHDL en conception de systèmes numériques ;
- Être capable de décrire un système numérique simple en VHDL ;
- Être capable d'implémenter une description VHDL sur une carte FPGA.

Connaissances préalables recommandées :

Electronique numérique (combinatoire et séquentielle)

FICHE DE CONTACT

Enseignant de la matière : FEKIK Arezki,

Contacts : arezkitdk@yahoo.fr

Coefficient :02· **Crédits** : 04

Volume horaire global :15 semaines

Volume horaire de travail requis/semaine: Cours: 1h30, TD: 1h30

Mode d'évaluation:

Contrôle continu: 40% ; Examen: 60%.

Table des Matières

Références
Annexes I : Travaux dirigés
Annexes II : Solution des travaux dirigés

Chapitre I :

Réseaux Logiques Programmables : PLD

I. Les Réseaux Logiques Programmables : PLD

I.1 INTRODUCTION

La réalisation de montages électroniques numériques peut se faire en utilisant des circuit à portes logiques (SSI, MSI,LSI) telle que les portes logique combinatoire ou séquentielles (AND, OR, NOT, NAND, NOR, XOR, MUX, DECODEUR, Bascules et LATCH…….). Cette approche nécessite en plus la réalisation de conceptions basées sur des schémas d'implantations, ce qui accroit la complexité et le coût.

La solution serait l'utilisation de dispositifs programmables. Ce type de produit peut intégrer dans un seul circuit plusieurs fonctions logiques programmables par l'utilisateur. Sa mise en œuvre se fait très facilement à l'aide d'un programmateur, d'un micro-ordinateur et d'un logiciel adapté.

I.2 Structure de base d'un PLD

La plupart des PLDs suivent la structure suivante :

- Un ensemble d'opérateurs « ET » sur lesquels viennent se connecter les variables d'entrée et leurs compléments.
- Un ensemble d'opérateurs « OU » sur lesquels les sorties des opérateurs « ET » sont connectées.

I.2.1 Matrice ET

On utilise le principe de ports ET à diodes. La programmation s'effectue grâce à des fusibles placés en série avec des diodes, comme illustré sur la figure.I.1

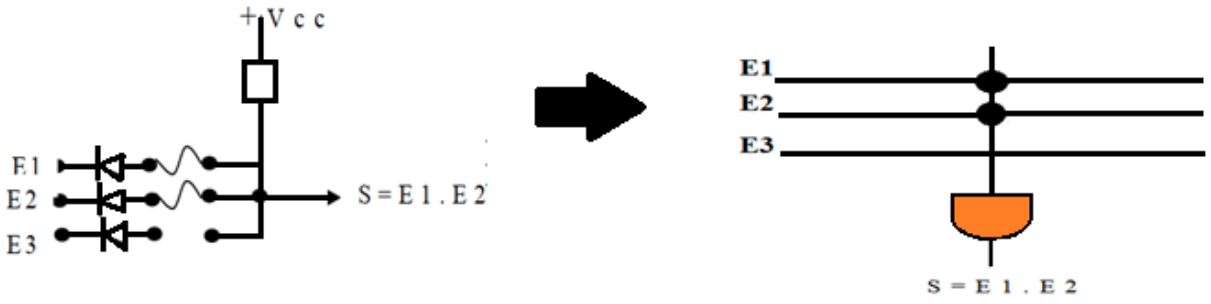


Figure.I.1 : Matrice ET

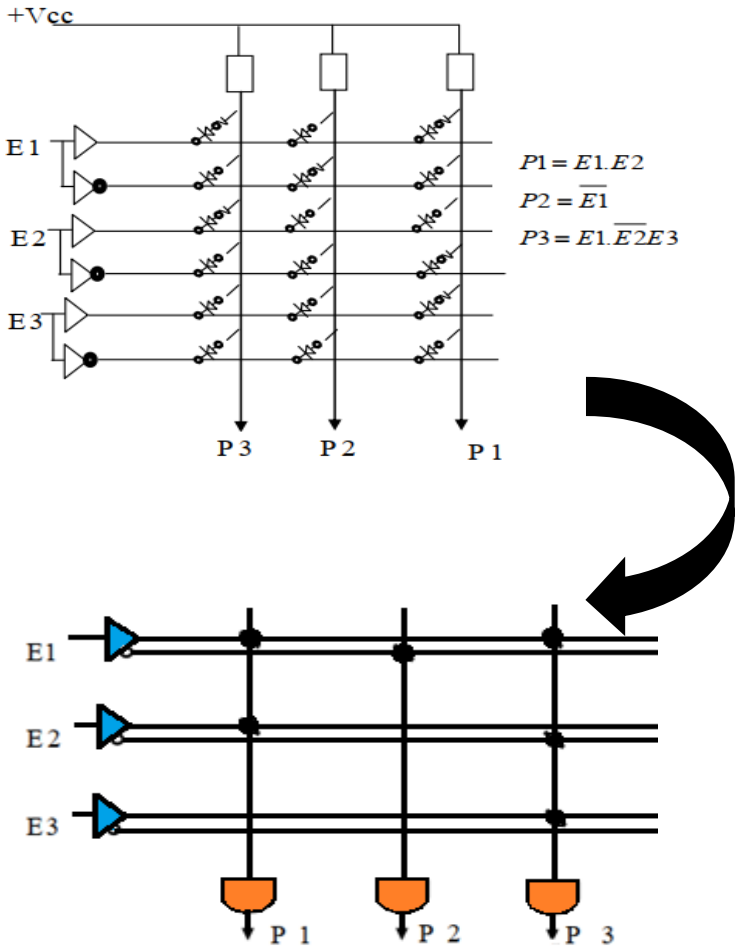


Figure.I.2 : Structure d'un PLD avec la matrice ET.

I.2.2 Matrice OU

On utilise le principe d'une porte OU à diode. La programmation se fait de la même manière que pour le ET, comme montre la figure.I.3

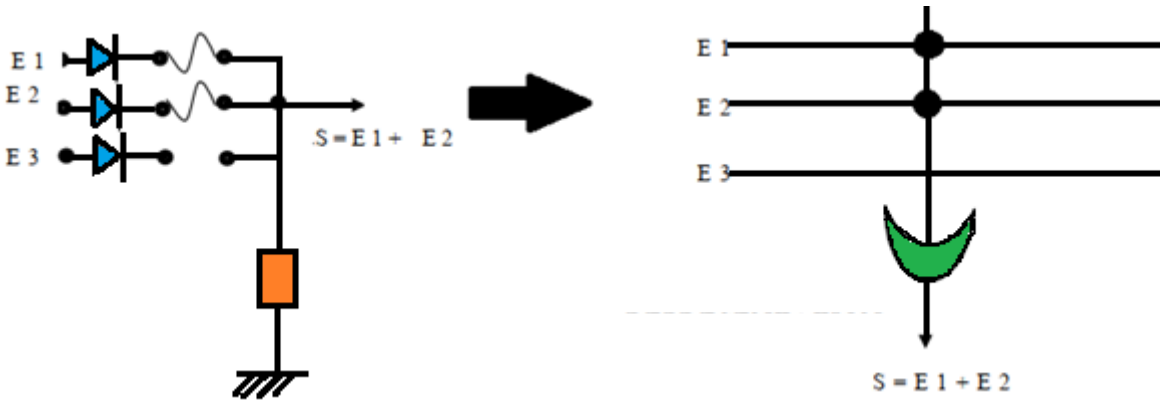


Figure.I.3 : Matrice OU

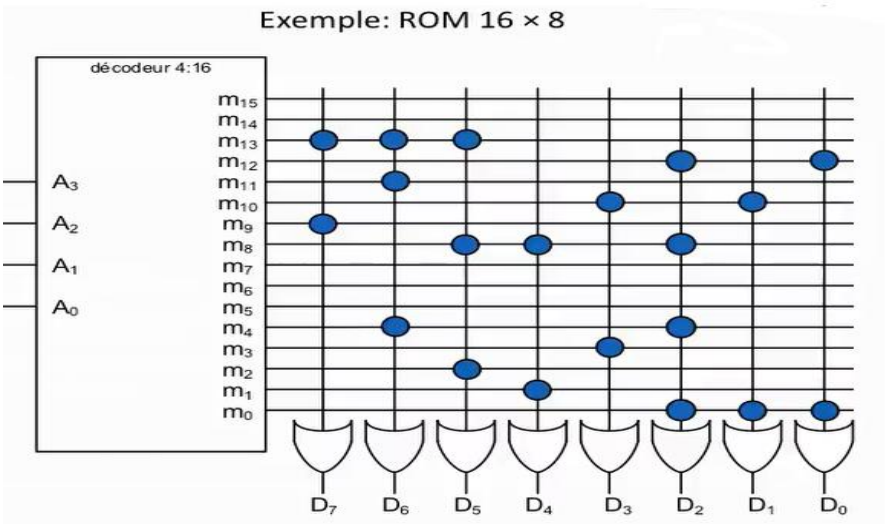
REMARQUE
D'après le cours de systèmes logiques : Toute fonction logique se représente sous forme d'une expression de **somme de produits** ou de **produit sommes**.

I.3 La mémoires morte (PROM)

Cette partie est consacrée à :

- La structure interne d'un mémoire morte programmable PROM ;
- Analyse d'une fonction logique à implémenter sur une PROM ;
- Implémentation d'une fonction logique sur PROM ;
- EPROM, EEPROM : les mémoires mortes programmable plusieurs fois.

La figure suivante montre la structure interne d'une mémoire morte programmable une seul fois (Programmable Read Only Memory PROM)

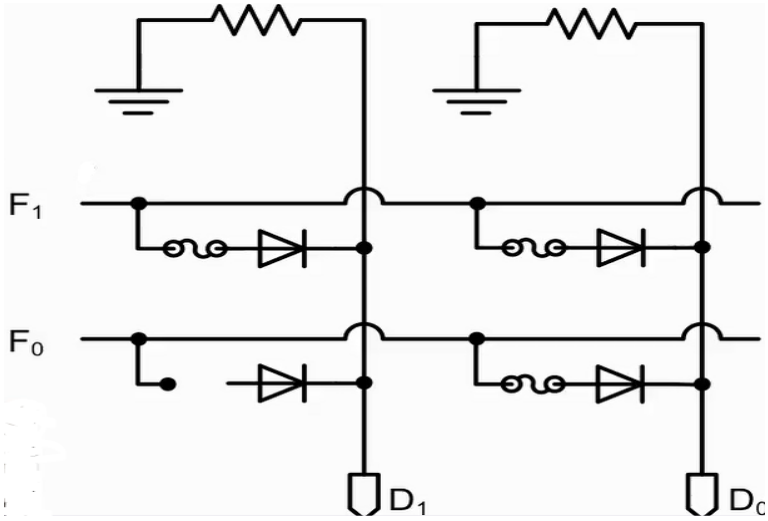


Une PROM est composée de :

- Décodeur avec *n* signaux d'entrée et 2^n sorties
- Réseau d'interconnexion programmable entre 2^n rangées et *m* colonnes
- *m* portes OU à 2^n entrées.

Remarque : on indique une connexion entre une verticale et horizontale par un Point pour éviter la complexité du diagramme.

Le fonctionnement de PROM ou exactement le fonctionnement du réseau d'interconnexion , sachant que les lignes horizontale et verticale sont reliées par des diodes et des fusibles connectés en série comme le montre la figure suivante :



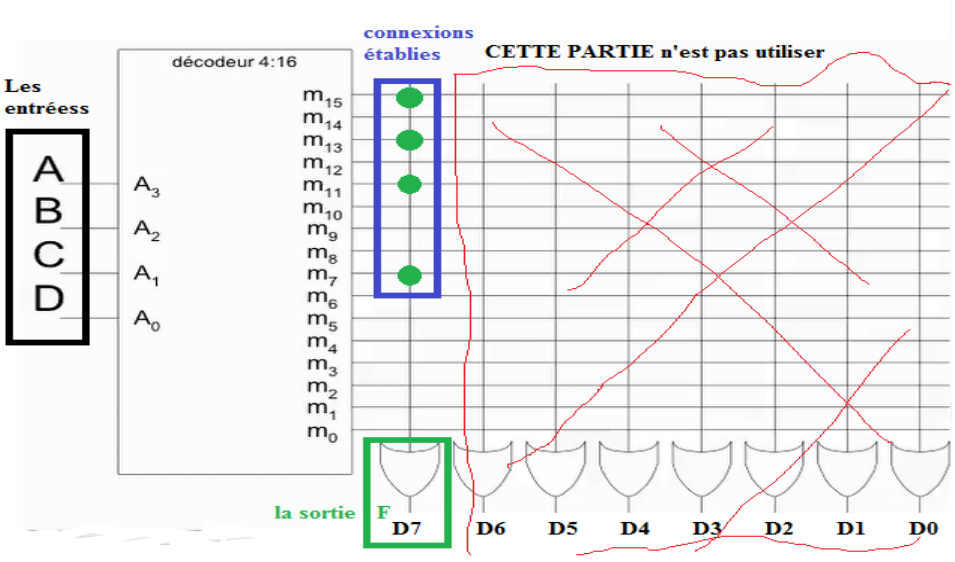
Au départ tous les fusibles sont en place, après on effectuera la programmation en faisant passer un fort courant dans les fusibles de connexion pour faire les fondre, ainsi que les lignes verticale sont mises à la masse à travers des résistance (configuration pull-down)

Exemple d'application

Soit l'équation suivante : $F = \bar{A}\boldsymbol{BCD} + \boldsymbol{A}\bar{\boldsymbol{B}}\boldsymbol{CD} + \boldsymbol{AB}\bar{\boldsymbol{C}}\boldsymbol{D} + \boldsymbol{ABC}\bar{\boldsymbol{D}} + ABCD$

Pour implémenter cette équation dans une PROM 16x8 il faut suivre les étapes suivantes :

1- Choisir les ports d'entrées/ sortie
2- Ecrire l'équation de la sortie en somme de produit
3- Indiquer les connexions établies



I.4 Classification des réseaux logiques combinatoires

Dans cette partie on va détailler quatre types des circuit logiques :

- ✓ Réseaux de logique programme PLA « ***Programmable Logic Array -PLA*** »

- ✓ Circuit PAL ***« programmable Array Logic – PAL »***
- ✓ Circuit Gal ***« Generic Array Logic -GAL »***
- ✓ Circuit logique programmable complexe CPLD « ***Complex Programmable Logic Device-CPLD »***

I.3.1 Réseaux de logique programme PLA

- Un PLA est similaire à une ROM, mais il ne réalise pas tous les produits de termes comme une ROM ;
- Un PLA à *n entrées* et *m sorties* peut réaliser ***m fonctions*** de *n variables*, en autant que chacune requiert un nombre limite de produit des variables en entrés ;
- Un PLA est composé de deux réseaux programmables ET, et OU, le réseau ET programmable est effectivement un décodeur programmable incomplet.

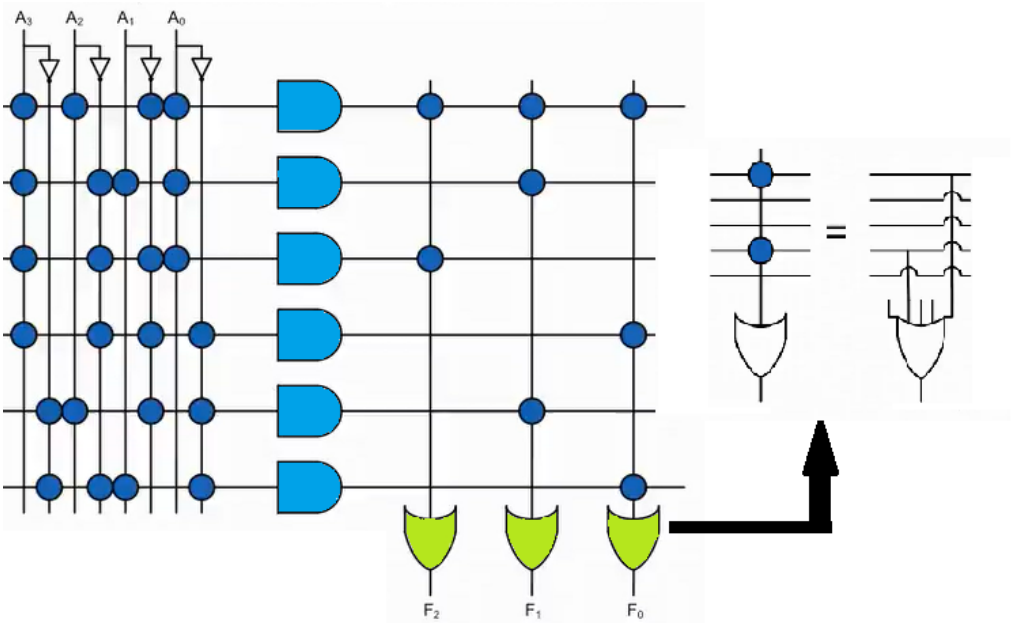


Figure.I.4 : Exemple PLA à 4 entrées, 3 sorties et 6 termes.

Remarques :

- ➢ Chaque intersection d'une ligne horizontale et d'une ligne verticale est programmable ;
- ➢ Seuls 6 termes (Produits-ET logique) peuvent être réalisés à partir des 4 entrées et de leurs compléments ;
- ➢ Seuls 3 fonctions de sorties peuvent être réalisées ;
- ➢ Chaque fonction peut utiliser n'importe quel des 6 termes programmés.

Exemple :

Soit la logique suivante fonction :

$$F = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}BCD + ABCD$$

Pour implémenter cette équation dans un PLA de 4-3-6 il faut suivre les étapes suivantes :

1. Choisir les portes d'entrées et de sortie ;
2. Écrire les équations de sortie en somme de produits ;
3. Indiquer quelles connexions établir.

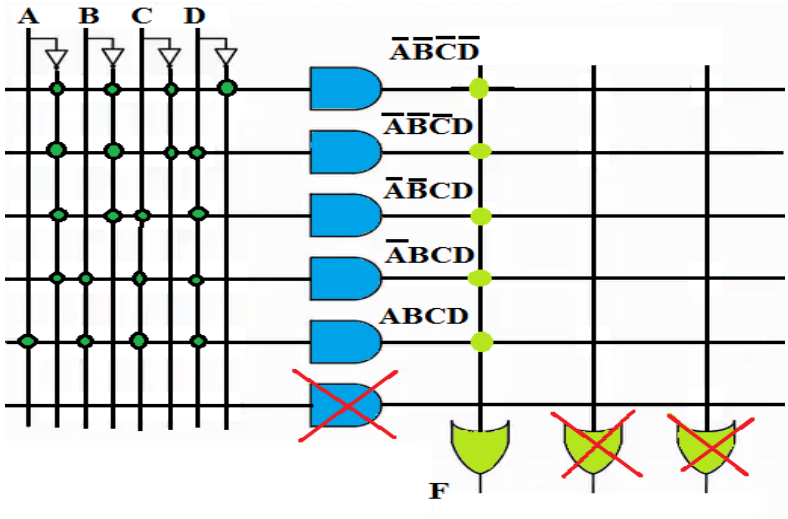


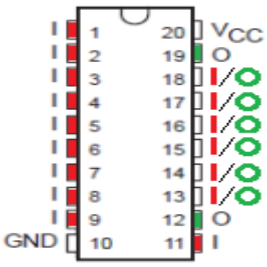
Figure.I.5 : Exemple PLA

Remarques :

- La minimisation des équations n'est pas toujours utile.
- L'ordre dans lequel on place les entrées est crucial.
- L'utilisation d'un circuit PAL4-3-6 est mieux qu'une ROM 16 × 8 pour ce circuit.

I.3.2 Circuit PAL « programmable Array Logic – PAL »

- Dans un circuit PAL le réseau ET est programmable et le réseau OU est fixe ;
- Chaque intersection d'une ligne horizontale et d'une ligne verticale est programmable ;
- Les portes ET ont une sortie de zéro par défaut ;
- Chaque patte de sortie est menée d'un tampon inverseur contrôlé par une fonction logique.



Soit un circuit PAL16L8 qui comporte :

- **10** entrées dédiées (Pattes 1-9 et 11) ;
- **2** sorties dédiées (Pattes 12 et 19) ;
- **6** Pattes peuvent être utilisées comme entrée ou sortie (13-18).

Exemple :

Soit la fonction logique suivante:

$$F = AB + C$$

Pour implémenter cette équation dans un PAL 16L8, il faut suivre les étapes suivantes :

1. Choisir les portes d'entrées et de sortie.
2. On programme un PAL en établissant des connexions entre les lignes verticales et horizontales pour former des sommes de produits.
3. Il faut tenir compte des sorties qui sont toujours inversées.

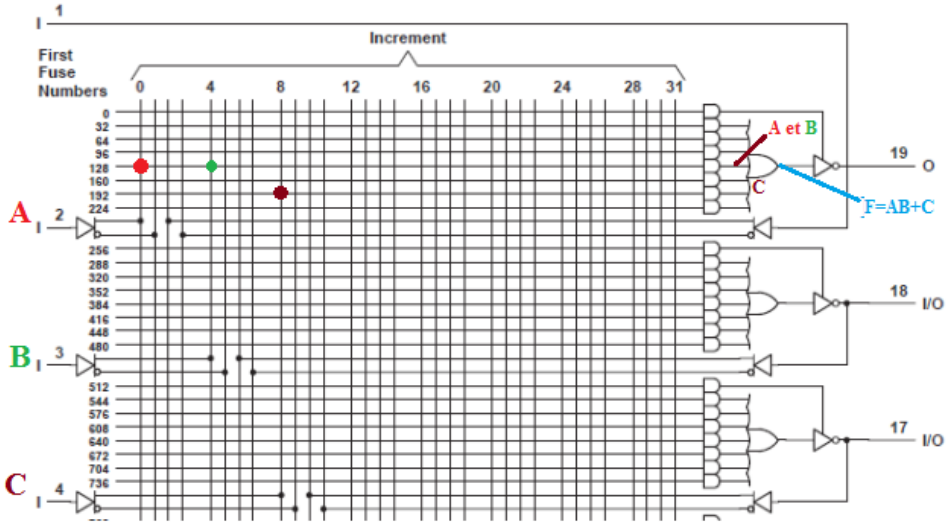


Figure.I.6 : Exemple d'un PAL.

I.3.3 Circuits GAL (Generic Array Logic)

Ils sont l'amélioration des circuit PLA et PAL et sont :

- Dispositifs programmables par l'utilisateur ;
- Mis en marché par Lattice Semiconductors en 1985 ;
- Peuvent émuler différents types de PAL.

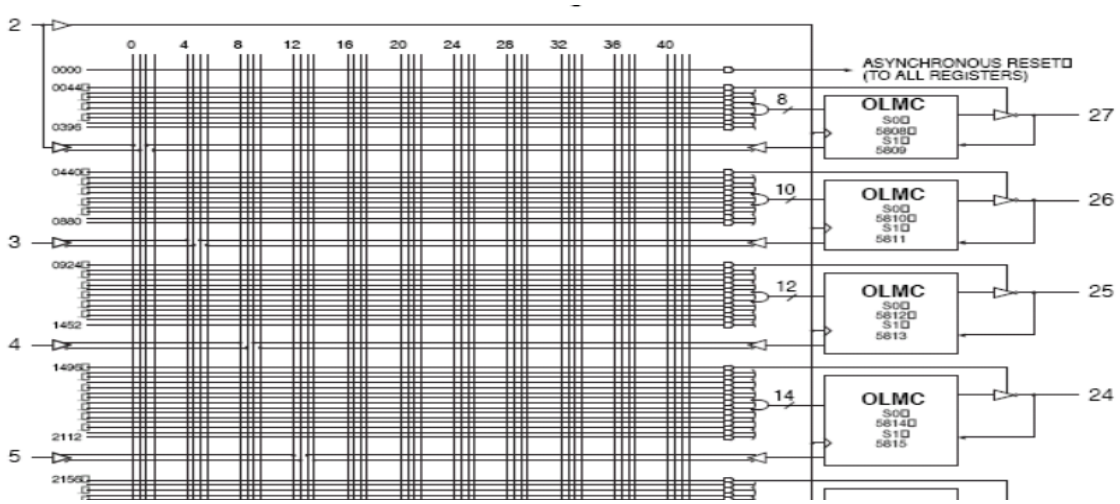


Figure.I.7: Circuit GAL.

Il ressemble vraiment au circuit PAL la seule différence c'est qu'à la sortie il y a un bloc additionnel par rapport aux PAL OLMC

Le schéma de la figure.I.8 montre une macro-cellule (*Output Logic Macro Cell* – OLMC) qui permet, par programmation, de réaliser de nombreuses fonctions logiques de base. Ils sont caractérisés par leur densité d'intégration qui est nettement supérieure à celle offerte par les PALs et une vitesse de fonctionnement égale, ou du moins comparable, à celle des PALs bipolaires.

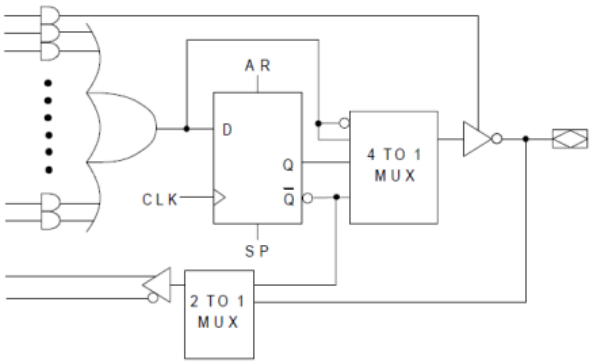
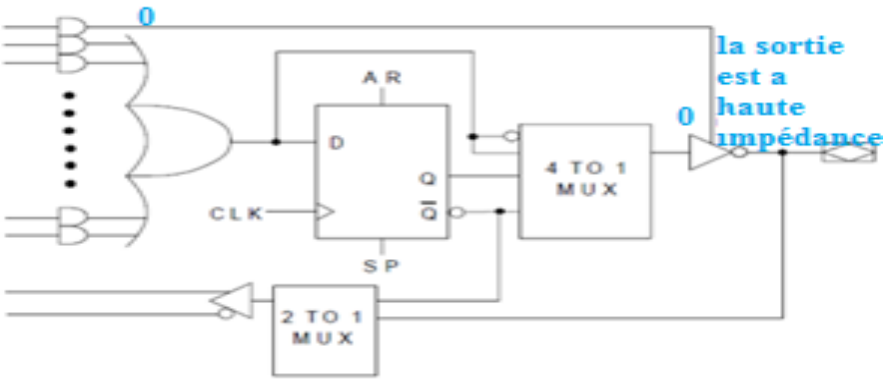
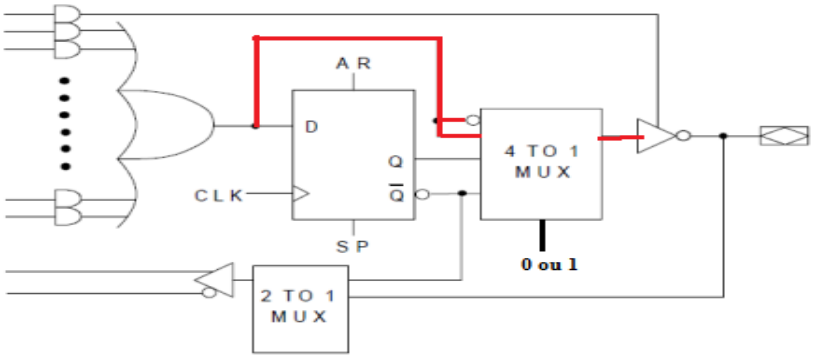


Figure.I.8 : Macro-cellule.

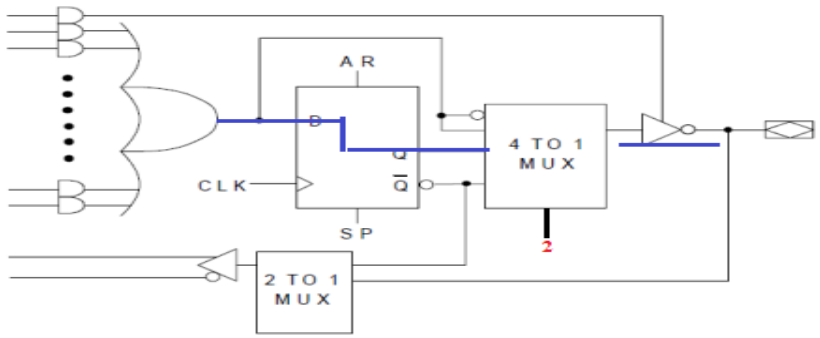
• La sortie de la OLMC peut être : 1- En haute impédance : si on place un Zéro sur l'entrée en voie que la sortie est à haute impédance



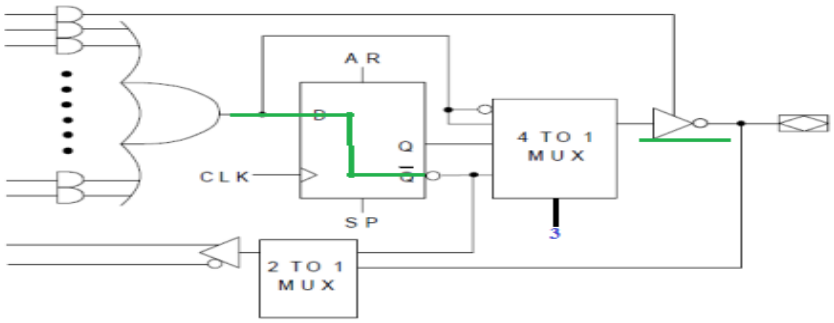
2- Sortie combinatoire inversée ou non : Si on met la valeur de multiplexeur à 0 ou 1 sa sera une sortie combinatoire inverse ou non qui sera renvoyée vers la sortie



3- sortie de bascule inversée ou non : Si on donne au multiplexeur la valeur d'entrée à 2 donc la sortie combinatoire passera tout d'abord par la bascule comme indiqué sur la figure :



Si on donne au multiplexeur la valeur d'entrée à 3 donc la sortie combinatoire passera tout d'abord par la bascule mais elle sera une sortie inverse de la bascule comme indiqué sur la figure :



4- renvoyée dans le réseau programmable : dans tous les cas les sorties logiques combinatoires peut être renvoyées vers l'entrée pour réaliser des fonctions logiques très complexes à travers le multiplexeur 2 vers 1

I.3.4 Circuits logiques programmables complexes (CPLD)

Les ROM, PLA, PAL et GAL sont parfois appelés des circuits logiques programmables simples (*Simple Programmable Logic Devices* – SPLD).

- Les *Complex Programmable Logic Devices* – CPLD – sont une extension naturelle des circuits PA ;
- Un CPLD incorpore plusieurs PAL sur une seule puce avec un réseau d'interconnexion ;
- Le réseau permet de relier les pattes de la puce à différents blocs internes et de relier les blocs entre eux.

Exemple : famille CPLD XC9500XL de Xilinx

- Chaque bloc fonctionnel est un PAL à 54 entrées et 18 sorties ;
- Les macro-cellules contiennent un élément programmable à mémoire ;

- Le circuit peut comprendre de 2 à 16 blocs fonctionnels ;
- Le réseau d'interconnexion permet d'établir des connexions entre les blocs d'entrées-sorties reliés aux pattes de la puce et les blocs fonctionnels.

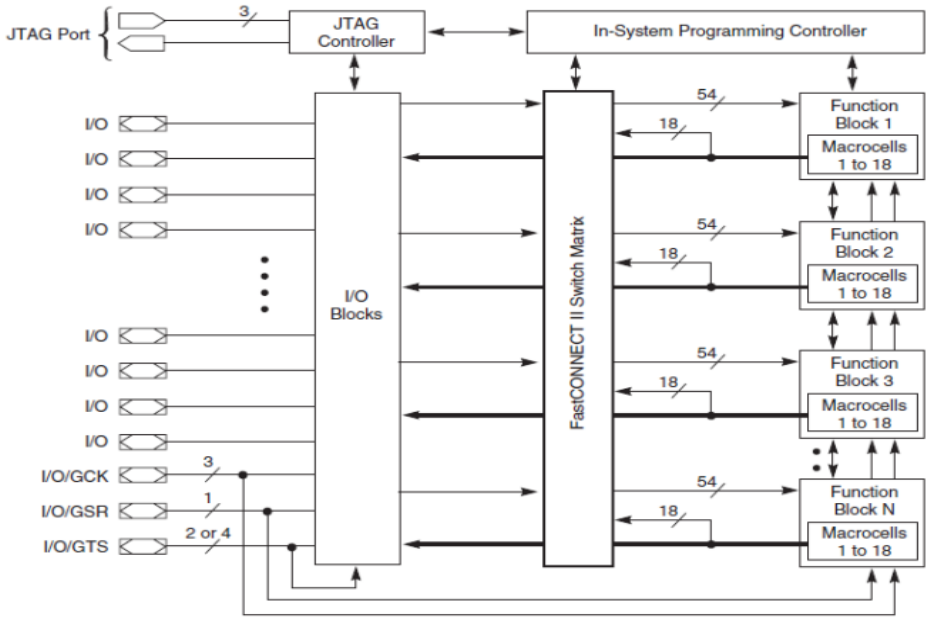


Figure.I.9 : Circuit logique programmable complexe.

I.5 Conclusion :

Dans ce chapitre nous avons abordé la description des Réseaux Logiques Programmables (PLD). Nous nous sommes intéressés l'explication de l structure interne des PLA, PAL, GAL et CPLD, pour montrer comment programmer ces dispositifs, afin de réaliser un circuit combinatoire.

Chapitre II :
Les technologies des éléments programmables

II. Les technologies des éléments programmables

II.1 Introduction

Les éléments programmables se trouvent dans les blocs logiques des PLDs afin de leur donner une fonctionnalité, mais aussi dans les matrices d'interconnexions entre ces blocs. Un élément programmable peut être considéré comme un interrupteur. Afin de respecter les contraintes imposées à l'ingénieur, les éléments programmables doivent posséder plusieurs qualités :

- Ils doivent occuper une surface la plus petite possible (Ce point s'explique pour des raisons évidentes de coût. Ceci est d'autant plus vrai que l'on désire en disposer d'un grand nombre).
- Ils doivent posséder une résistance de passage faible et une résistance de coupure très élevée.
- Ils doivent apporter un minimum de capacité parasite. Les deux derniers points s'expliquent quant à eux pour des raisons de performance en termes de fréquence de fonctionnement du PLD. Plus la résistance et la capacité sur le chemin d'un signal sont faibles, plus la fréquence de ce signal peut être élevée (RC effet)

II.2 Technologies à fusibles

La technologie à fusible est caractérisée par :

- ✓ Circuits de faible densité. -La programmation permet de supprimer la connexion.
- ✓ Toutes les connexions sont établies à la fabrication.
- ✓ Son principe est d'appliquer une tension de 12 à 25v (tension de claquage) aux bornes du fusible.
- ✓ La programmation dans cette technologie est irréversible

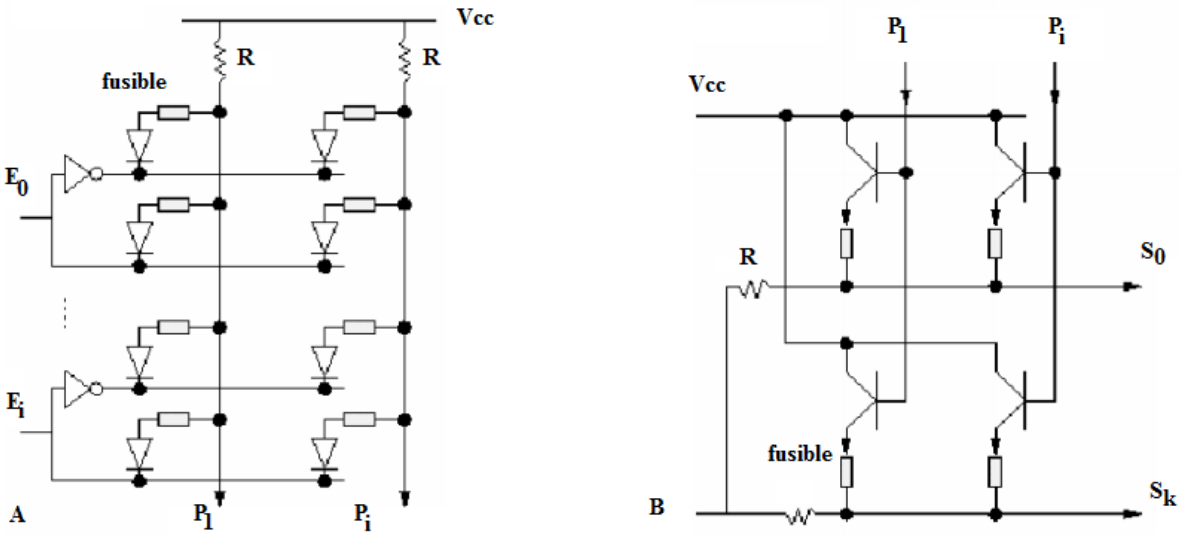


Figure.II.1 : Technologie a fusibles : A. Fusible-diode ; B. Fusible-transistor.

Ces technologies abandonnées pour manque de fiabilité. Les fusibles qui sautent provoquent du bruit qui peut affecter le reste du circuit. Aussi, cette programmation est irréversible et ne peut donc pas être reprogrammée.

II.3 Technologies à anti fusible et SRAM

Bien que le terme reprogrammation ne soit plus compris dans les circuits FPGA, il existe deux classifications différentes des FPGA, soit à base d'anti-fusible, soit à base de mémoire. Cette technologie est classée en deux catégories :

II.3.1 Anti fusibles

La technologie anti-fusible développée par ACTEL consiste à isoler deux Ligne de connexion en fine couche d'oxyde.

- Lorsqu'une impulsion haute tension (environ 20 volts) est appliquée à ce fusible, un trou est fait dans la couche d'oxyde et les deux fils sont connectés.
- La programmation permet d'établir la connexion mais dans cette technologie est irréversible. Par déduction cette technique est non reprogrammable.

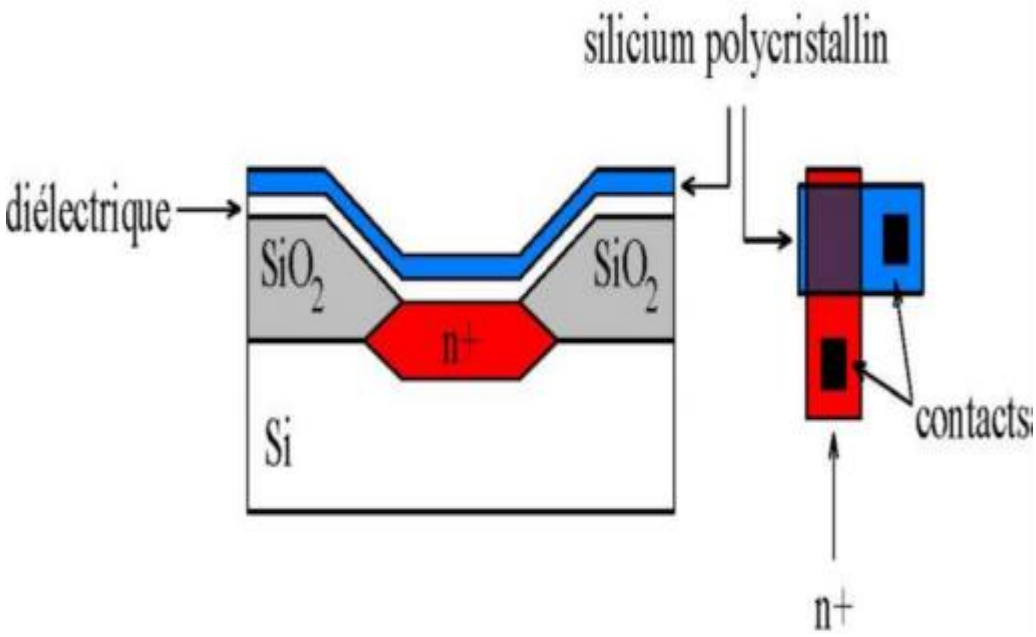


Figure.II.2 : Technologie anti fusibles.

II.3.2 SRAM

Les circuits SRAM-FPGA **(Static Random Access Memory- Field Programmable Gate Arrays)** deviennent des alternatives avantageuses pour une haute intégration numérique. Les cycles de développement et de prototypage (tests réels et vérification) sont accélérés et peuvent même prêter à confusion.

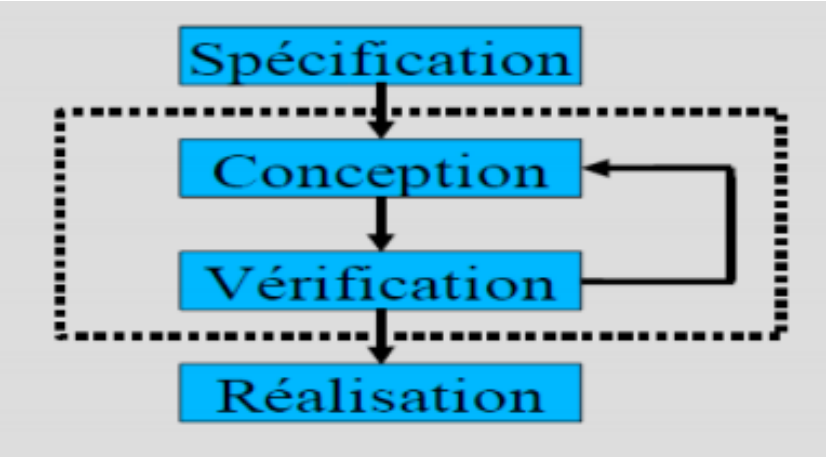


Figure.II.3 : Technologie de FPGA SRAM.

II.4 Technologies à EPROM/FLASH

Il y a deux variantes EPROM,

- Erasable Programmable Read Only Memory Classique **(EPROM)**
- Electrically Erasable Programmable Read Only Memory **(EEPROM).**

La technologie EEPROM

- ✓ Les PLD avec EPROM sont programmés électriquement pour éliminer les UV. Les PLD avec EEPROM, par contre, sont programmés presque instantanément et restent configurés jusqu'à ce que de nouveaux programmes soient disponibles (même en l'absence de tension).
- ✓ La technologie EEPROM est rapide et facile à programmer et la configuration est la puissance.

II.5 Technologies utilisées par les différents fabricants

Le tableau suivant nous offre les différents producteurs du monde qui utilise ces technologies des éléments programmables.

FABRICANTS	TECHNOLOGIES UTILISEES
ACTEL	Anti fusible, SRAM
ALTERA	EPROM, EEPROM,SRAM
AMD	EEPROM
ATMEL	SRAM
LATTICE	EPROM,EEPROM
XILINX	SRAM, Anti fusible, EPROM, EEPROM

Tableau II.1 : Technologies utilisées par les différents fabricants.

II.6 Conclusion :

Dans ce chapitre nous avons effectué une étude sur les différentes technologies des éléments programmables afin d'expliquer le principe et les avantage de chaque technologie à savoir la technologie anti-fusible et SRAM, puis nous avons exposé le principe de la technologie à EPROM/FLASH, pour faire la déférence entre toutes ces technologies utilisées par les différents fabricants.

Chapitre III :
Architecture des FPGA

III. Architecture des FPGA

III.1 Introduction

Les FPGA, sigle anglais qui signifie « Field Programmable Gates Arrays » traduit en français par réseau de portes programmables, sont des circuits intégrés reprogrammables. Ils offrent la possibilité de réaliser des fonctions numériques plus ou moins complexes, tout comme leurs homologues figés : les ASIC. Les FPGA sont des circuits numériques matériels configurables dédiés à l'électronique numériques.

Pour utiliser un FPGA il faut le relier à différents composants par des connexions électriques d'une façon à l'utiliser sur une planchette de développement comme par exemple la planchette de développement la Nexys4 DDR de Digilent équipée d'un FPGA Artix-7 de Xilinx qui facilite l'utilisation du FPGA en ajutant plusieurs interfaces comme le montre sur l'image ci-dessous :

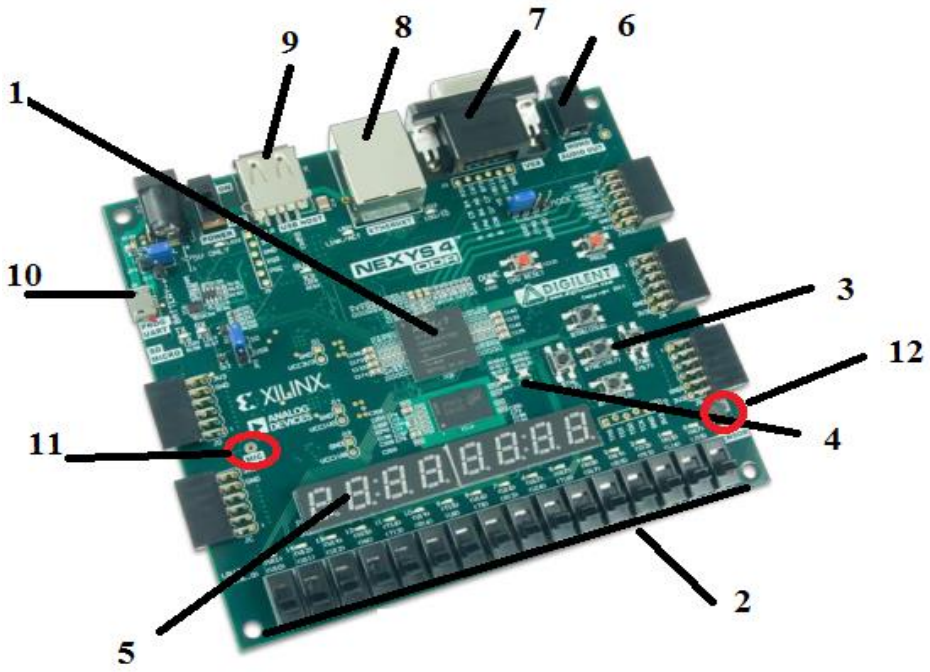


Figure.III.1 : Exemple planchette de développement.

1- *FPGA Artix-7*

2- *16 commutateurs et 16 LED*

3- *5 boutons-poussoirs*

4- *Deux LED à trois couleurs*

5- *Deux groupes de 4 affichages à 7 segments*

6- *Sortie audio*

7- *Sortie VGA*

8- *Ethernet 10/100*

9- *Port USB*

10- *Connecteur Micro SD*

11- *Microphone*

12-*Capteur de température intégré*

III.2 Architecture générale d'un FPGA

On trouve à l'intérieur d'une carte FPGA trois composants principaux :

- Un réseau de blocs de logique programmable (*Configurable Logic Block – CLB*), chaque bloc pouvant réaliser des fonctions complexes de plusieurs variables, et comportant des éléments à mémoire ;
- Un réseau d'interconnexions programmables entre les blocs ;
- Des blocs spéciaux d'entrée et de sortie avec le monde extérieur (*Input/Output Block – IOB*).

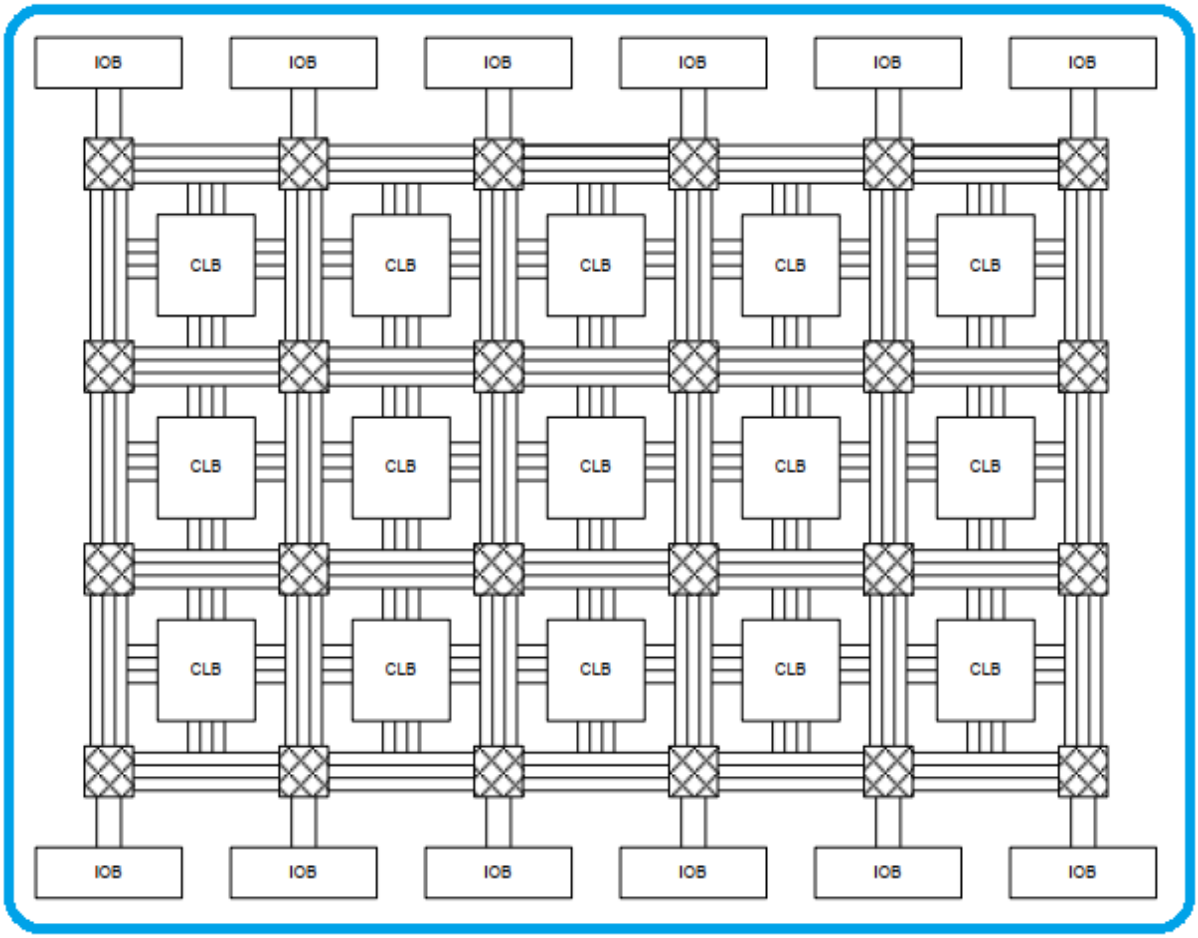


Figure.III.2 : Modèle d'un FPGA.

Dans la figure.III.2, on a : 12 IOBs, 15 CLBs, la plus part des puces moderne contient beaucoup plus des IOBS comme, la carte FPGA XC7A100T-1CSG324C a plutôt: 210 IOBs, 7925 CLBs

III.2.1 Blocs de logiques programmables

Les blocs de logiques programmables appelé aussi Eléments Logiques sont composés de deux tranches (*Slices)*, on peut voir sur la figure.III.3 qu'ils sont numérotées à partir du point gauche inférieur de la puce ou FPGA (X_0 , Y_0) le premier chiffre X_0 indique le numéro de la colonne et Y_0 indique le numéro de la ligne ou de la rangée. Chacune des tranches est reliée au réseau d'interconnexions comme montre la figure.III.3 (en rouge), tout comme elles sont reliées entre elle verticalement (Bleu)

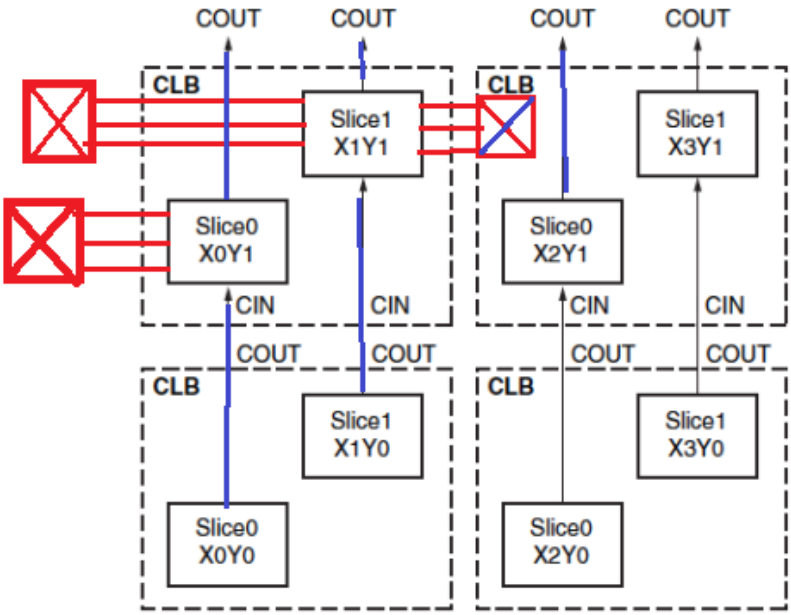


Figure.III.3 : Modèle d'un CLB.

Une tranche du FPGA de Xilinix série 7 montrée sur la Figure.III.4 comprend :

- Quatre tables de correspondance (Look-up Table –LUT encerclé en vert) à 6 entrées (A6:A1) et deux sorties O6 et O5, pouvant réaliser une fonction logique à 6 entrées ou deux fonctions logiques à 5 entrées.
- Trois multiplexeurs (en rouge, verticaux) qui permettent de combiner les quatre tables de correspondances pour réaliser des fonctions logiques très complexes de 7 ou 8 entrées
- Des portes logiques pour réaliser l'addition rapide qui sont montrées par la colonne de couleur bleu. On peut constater qu'elle a quatre OU exclusive et des multiplexeurs.
- Huit éléments à mémoire :
  - 4 (au centre de couleur oranges) on les utilise toujours comme des bascules
  - 4 (à droite de couleur Maron) peuvent être programmés soit comme des bascules
- Des multiplexeurs programmables (la colonne en violet) pour router les signaux à l'intérieur de la tranche, on peut voir que chacun des multiplexeurs a différentes entrées et une sortie vers l'extérieur ou vers l'élément mémoire en ajoutant un bit de programmation à ces multiplexeurs. On peut déterminer laquelle des entrées sera routée vers l'élément mémoires ou vers la sortie.
- Il existe aussi des tranches de type M qui peuvent implémenter de la mémoire et des registres à décalage.

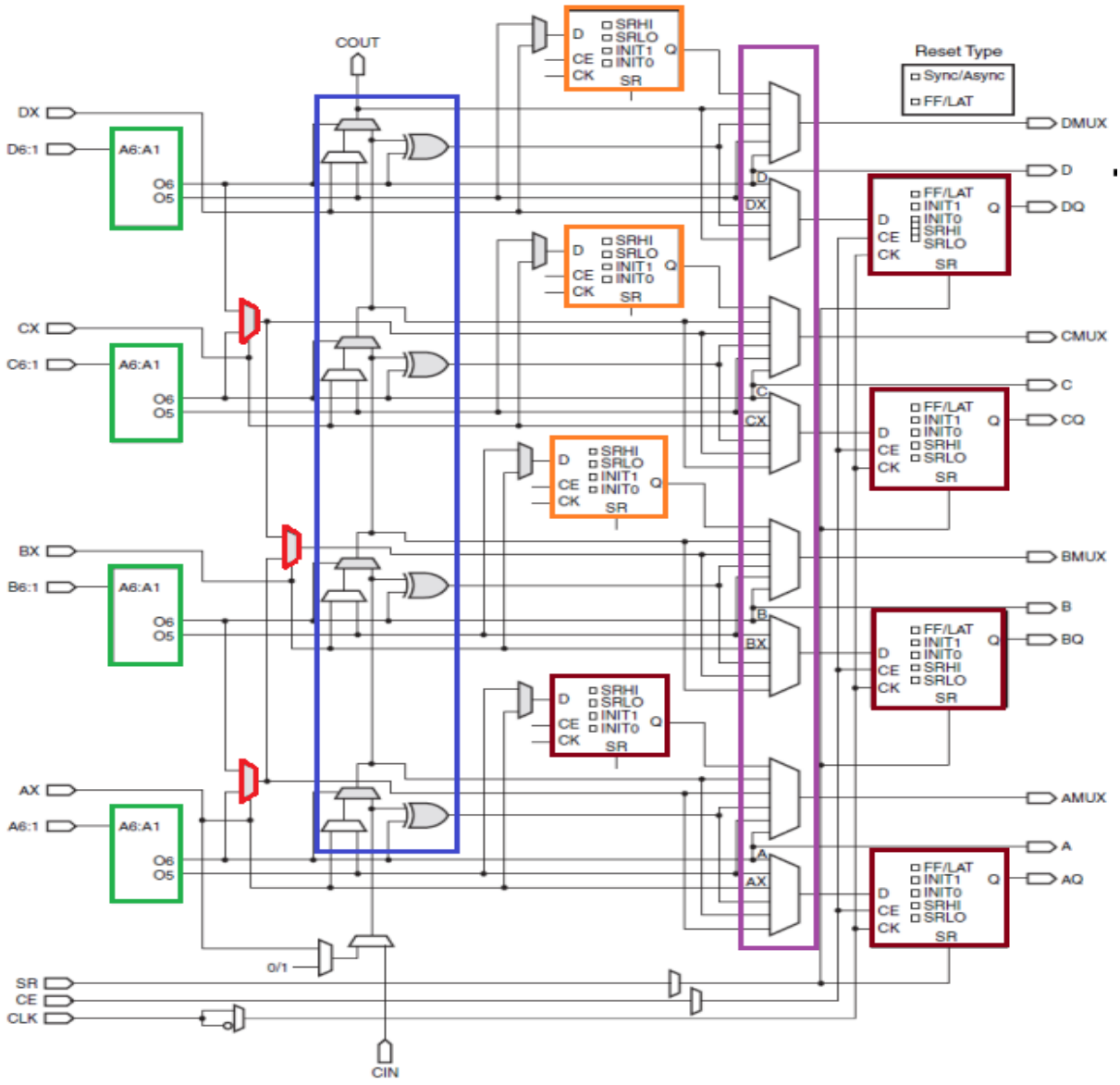


Figure.III.4 : Tranche pour FPGA de Xilinix série 7.

La disposition des portes logique à la partie souligner en bleu a pour objectif de propager la retenue rapidement. Ce qui fait que cette partie est la plus intéressante qui caractérise les cartes FPGA, par rapport aux autres dispositifs programmables.

La figure.III.5 montre le modèle simplifié d'une tranche du Virtex 2 Pro dans le but d'implémenter une équation logique sur cette tranche qui est équipée de :

Deux tables de correspondance à 4 entrées :
– Fonction logique
– Mémoire RAM
– Mémoire ROM
– Décalage
• Deux éléments à mémoire, bascule.
• Des multiplexeurs pour router les signaux.

Exemple : Soit la fonction logique suivante :

$$F = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}BCD + ABCD$$

Pour programmer la tranche du FPGA pour implémenter l'équation logique précédente il faut suivre les étapes suivantes:
1. Choisir les ports d'entrées et de sortie dans notre exemple les entrées sont A, B,C, D qui sont affectée respectivement à G4 G3 G2 G1, la sortie est F qui affectée pour Y.

2. Indique quelles connexions ont été établies pour l'entrée. Après cela, nous acheminons la sortie de la table correspondante vers la sortie Y, comme indiqué en rouge sur la Figure II.5, et programmons le bit S1 du multiplexeur de routage dans la table Sortie vers Sortie Y

3. Donner le contenu des tables de correspondance. (Ici: la table de correspondance contient la table de vérité désirée de l'équation)

A	B	C	D	**F**
0	0	0	0	**1**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**0**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**1**

Table.III.1 : Table de vérité.

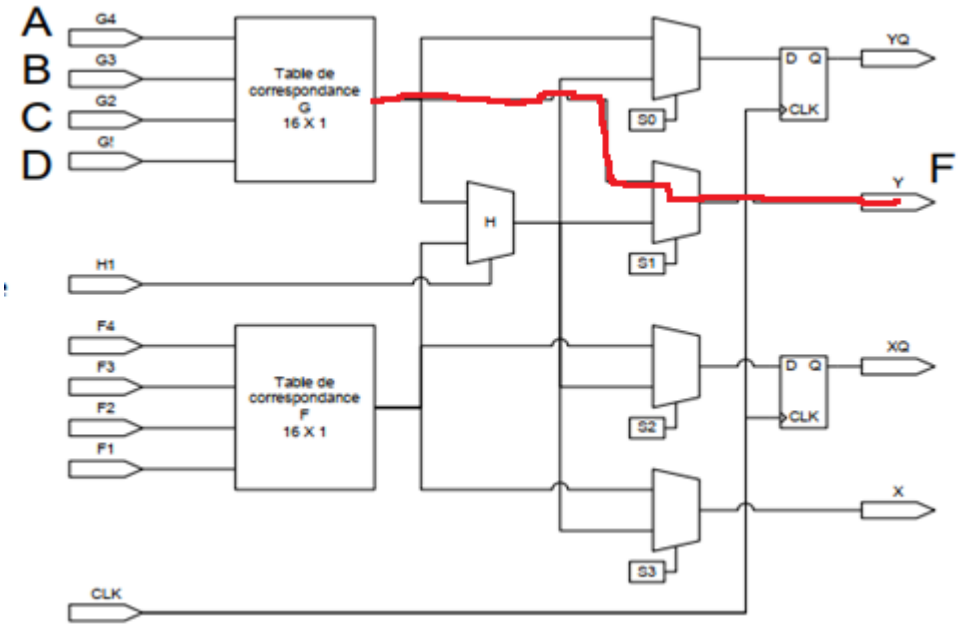


Figure.III.5 : Modèle simplifier d'une tranche du Virtex 2 Pro.

Remarques:
• La minimisations des équations n'est pas toujours utile.
• L'ordre dans lequel en place les entrées est important.
• Des fonctions à plus de 4 entrées nécessiterait plus d'un niveau de logique.

III.2.2 Réseaux d'interconnexion programmable

Ce réseau d'interconnexion permet de connecter une CLB avec une autre CLB ou avec une cellule d'entrée/sortie, et pour cela il existe un ensemble de lignes horizontales et verticales et un ensemble de points de connexion.

On distingue plusieurs types de lignes qui sont définies par leur longueur relative et qui sont :

- Les interconnexions ou lignes segmentées à usage générale, de longueur la plus courte.
- Les lignes directes ou interconnexions directes, de longueur double des lignes courtes.
- Les lignes longues.

Chaque CLB est entourée de ces lignes et des points de connexion et tout s'est détaillé sur la figure suivante :

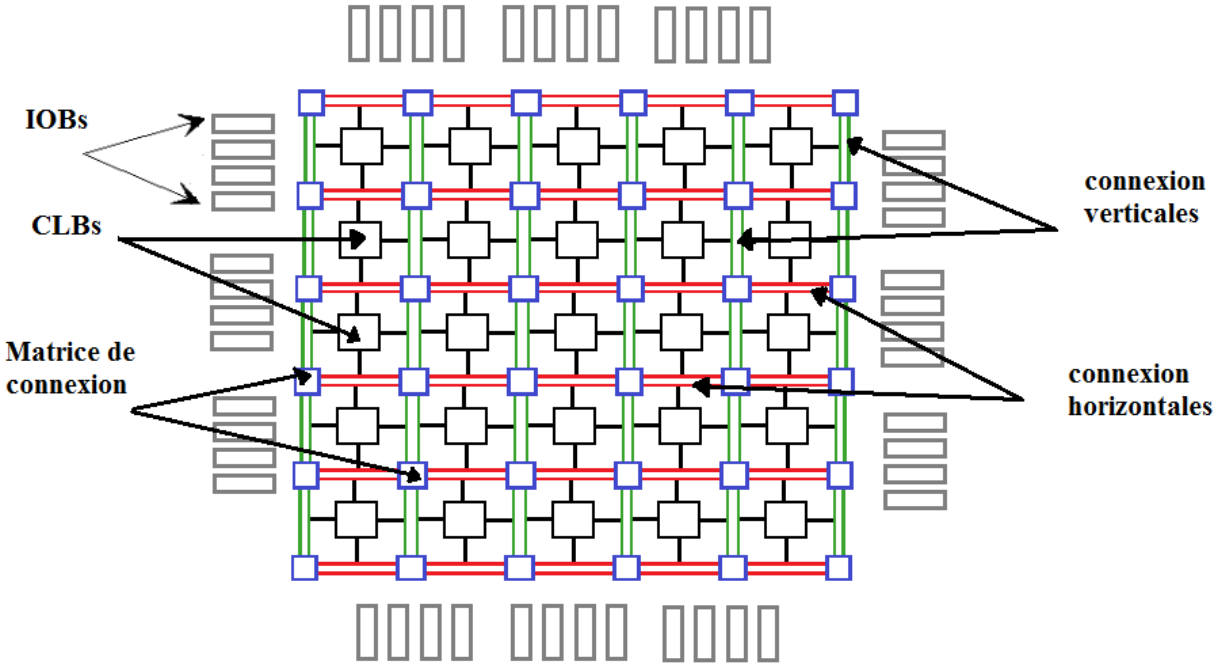


Figure.III.6 : Concept architectural de base des FPGAs

III.2.3 Blocs d'entrées-sorties :

Ils constituent l'interface entre les branches de sortie du circuit et les CLB. Ils sont présents sur toute la périphérie du circuit FPGA car ces cellules sont des intermédiaires par lesquelles les données transitent depuis les blocs logiques internes jusqu'aux ressources externes et vice versa.

Chaque bloc IOB contrôle une broche du composant et peut être défini en entrée, en sortie, en entrée/sortie ou être inutilisé qui peut prendre les états suivant 0,1 ou haute impédance. Le rôle principal des interfaces d'entrées/sorties est de transmettre et de recevoir des données. Néanmoins l'interface d'entrée/sortie peut être dotée d'options telles que des registres, impédances et buffers.

Chaque fabricant a sa propre appellation pour désigner l'interface d'entrées/sorties mais la fonction reste toujours la même.

Altera, les nomme IOE (Input Output Element). L'IOE remplit toujours son rôle d'interface d'entrés/sorties, elle dispose d'une résistance de rappel pull-up et un temporisateur du signal.

Chez XILINX, les interfaces d'entrées/sorties sont nommées IOB pour Input Output Blocks. L'IOB est constitué de registres, de diviseurs de tension, des résistances de rappel pull up et autres ressources spécifiques.

Remarque :les FPGAs contiennent aussi plusieurs composantes secondaire très intéressantes qui sont :

- Des blocs mémoires intégrées ;
- Des fonctions arithmétiques avancées ;
- Des modules de génération d'horloge ;
- Des réseaux de distribution d'horloge :
- Des microprocesseurs fixes.

1- **Blocs mémoires intégrées :** les FPGAs obéissent à la loi de MOORE c'est-à-dire le nombre de transistors intégrés dans un FPGA suit une progression exponentielle avec le temps. Les constructeurs depuis plusieurs années au lieu d'augmenter le nombre de bloc de logique programmable sur les FPGAs ajoutent à la place des blocs mémoires, ces blocs mémoires imbriqués sont placés entre les colonnes des blocs de logique programmable ou configurables comme le montre la figure.III.7

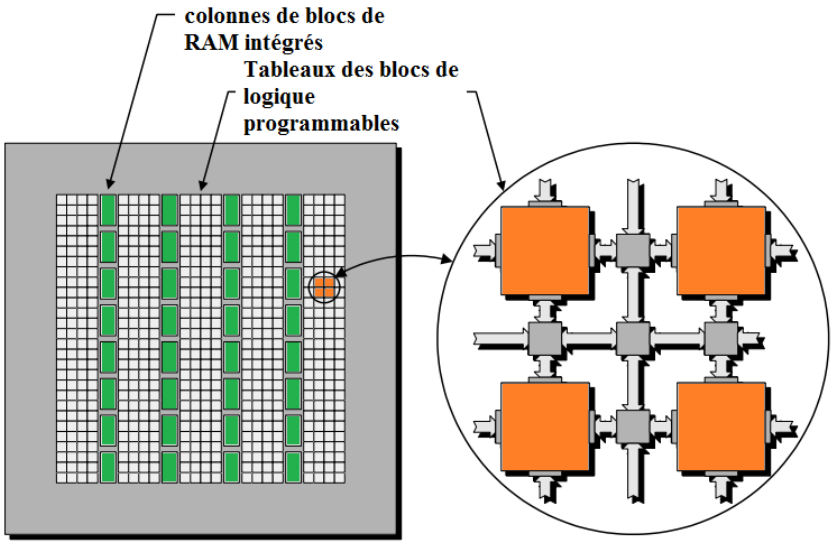


Figure.III.7 : Mémoire RAM intégrée.

2- **Fonctions arithmétiques avancées :**

La multiplication est une opération fondamentale dans les applications de traitement du signal pour lesquelles les FPGAs sont très populaires. Les fabricants de FPGAs ont donc ajouté à leurs architectures des blocs spéciaux pour réaliser cette opération. Les multiplicateurs sont en général disposés près des blocs mémoire RAM pour des raisons de l'efficacité de routage des signaux et de disposition des ressources du FPGA sur une puce comme montre la figure.III.8 :

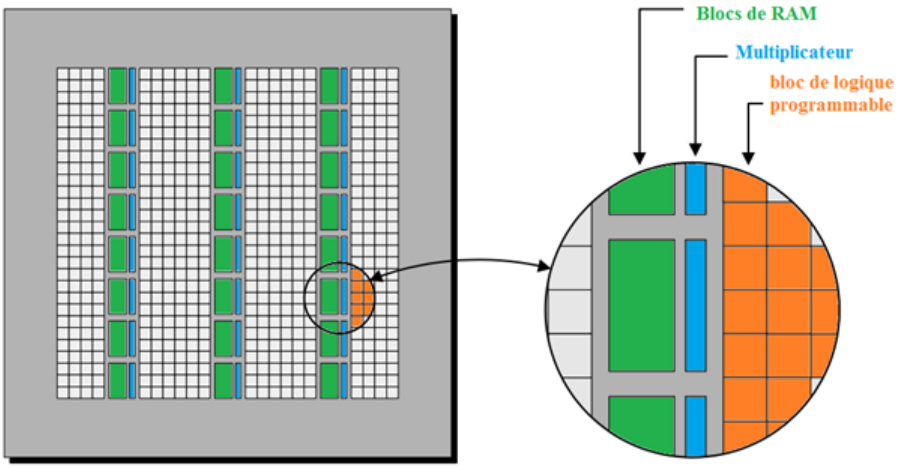


Figure.III.8 : Multiplicateurs intégrés

3- Modules de génération d'horloge

La génération et la distribution du signal d'horloge est un problème difficile, les concepteurs des FPGAs proposent un bloc qui permet de générer plusieurs fréquences d'horloge différents à partir de la même horloge de référence c'est à dire que le générateur accepte en entrée une horloge externe et génère une ou plusieurs horloges internes comme le montre la figure.III.9

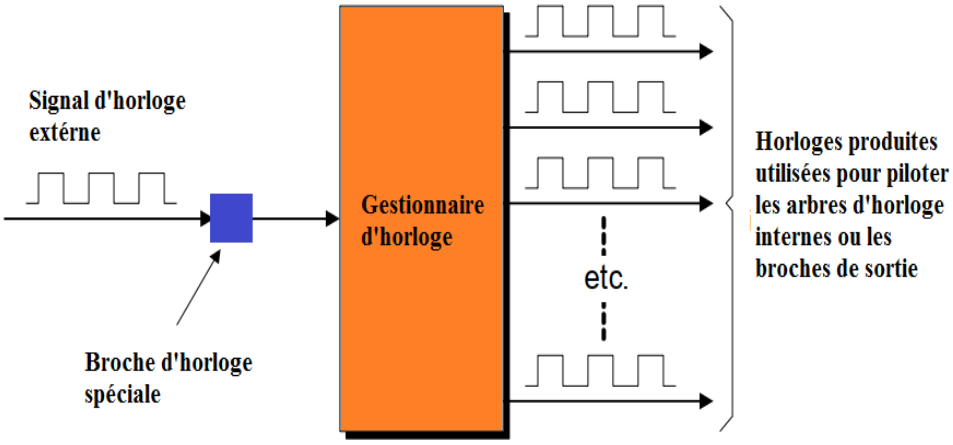


Figure.III.9 : Génération de signaux d'horloge à partir d'une référence externe.

4- Réseaux de distribution d'horloge

Pour distribuer l'horloge à travers la puce tout en minimisant le déphasage d'horloge, on utilise les réseaux de distribution d'horloge. Ce réseau est alimenté soit par une patte spéciale du FPGA à laquelle est associé un amplificateur dédié, ou bien par des signaux internes qui peuvent être routés au même amplificateur.

5- Microprocesseurs fixes

Avec le nombre de transistors intégrables sur une puce doublant tous les 18 mois, et les besoins important en contrôle dans plusieurs applications, les fabricants de FPGA intègrent des processeurs fixes (*hard*) à plusieurs familles de FPGAs, l'architecture résultante de

l'ajout des microprocesseurs fixe est très performante, elles ont un accès rapide entre ces microprocesseurs et le reste des composant (CLB, multiplicateur, et mémoire)

III.3 Exemples de constructeurs Xilinx

Le constructeur **Xilinx** est classé parmi les plus grands détenteurs mondiaux de brevets et il propose des produits et des services se distinguant par leur qualité et leur réputation afin de satisfaire les clients

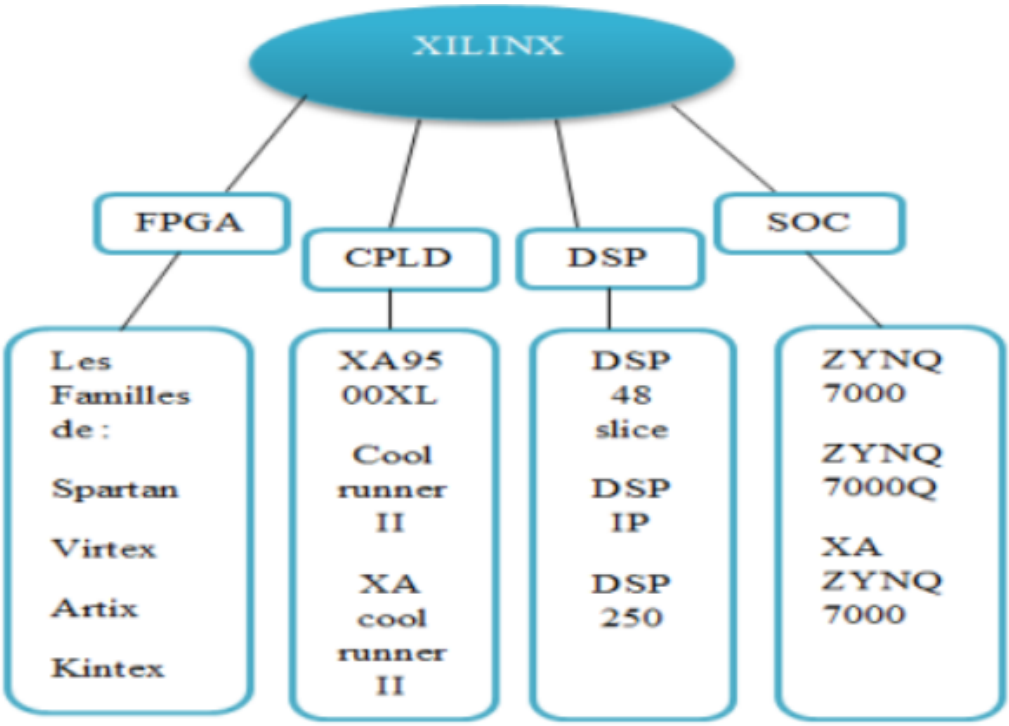


Figure.III.10 : Ensemble de produit de constructeur Xilinx.

Les circuits *FPGA* (Fields Programmable Gate Array) de la famille *Spartan®-6* de Xilinx présentent les avantages suivants :

- Un parfait équilibre entre moindres risques
- Un bas coût
- Offre une technologie de gestion de la puissance avancée
- Une prise en charge de la mémoire avancée, des *DSP* (Digital Signal Processing)

III.3.1 Présentation des capacités :

Facilité d'utilisation : Design plus facile et plus rapide des blocs intégrés

Plusieurs niveaux de performance : Une capacité logique augmentée de cellules de logique allant jusqu'à 147 Ko offrant plus de performances pour les systèmes de traitement de signal numériques pour la vidéo, la radio et sans fil et beaucoup d'autres applications avec efficacité

Connectivité plus rapide, Plus complète

Deux fois la Capacité, la moitié de la puissance : Processus plus rapide avec une augmentation du bloc RAM, capacité de logique 2X, 50% de plus DSP48A1

La figure.III.11 illustre le Micro Board Xilinx Spartan-6 FPGA LX9 qui fournit une plate-forme de développement complète pour la conception et la vérification d'applications basées sur la famille FPGA Xilinx Spartan-6 LX. Disponible avec le Spartan-6 LX9, le kit permet aux concepteurs de créer facilement des modèles polyvalents.

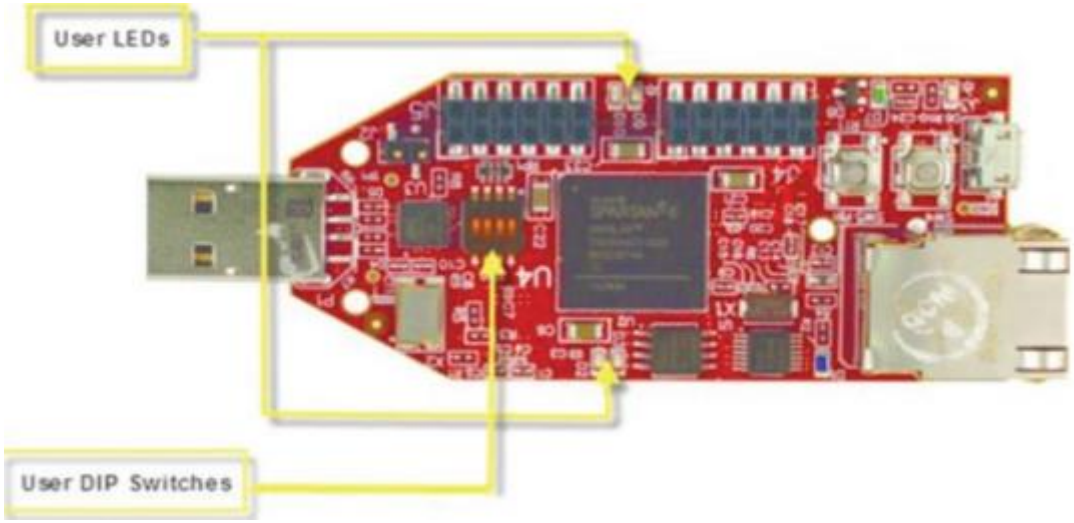


Figure.III.11 : Micro Board Xilinx Spartan-6 FPGA LX9.

La micro-carte Spartan-6 FPGA LX9 est disponible avec le FPGA XC6SLX9-2CSG324C. La carte comprend :

- Une mémoire SDRAM LPDDR,
- Une mémoire Flash SPI multi-E / S,
- Une connexion Ethernet PHY 10/100
- Un port série USB.

Les autres caractéristiques de la carte incluent un port : USB JTAG, une horloge programmable, des commutateurs utilisateurs et des LED. La carte fournit également deux embases d'extension 2x6 pour un usage général ou pour ajouter l'un des nombreux modules périphériques Pmod ™. Les Pmod sont de petites cartes d'interface d'E / S qui offrent un moyen idéal pour étendre les capacités de la micro-carte Spartan-6 FPGA LX9.

La figure.III.12 montre la structure interne de la micro carte SPARTAN6.

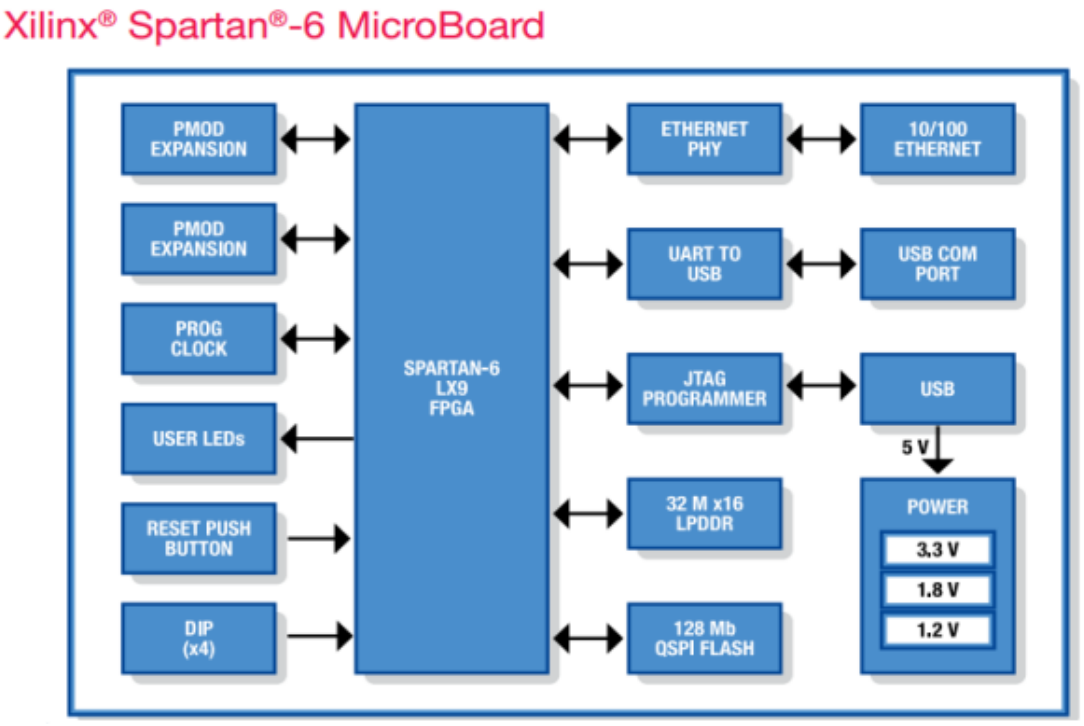


Figure.III.12 : Structure interne de la micro carte SPARTAN6.

III.4 Conclusion :

Nous devrons maintenant être capable de :

- ✓ Décrire, à l'aide d'un diagramme, la structure interne d'un FPGA et de ses composantes principales, et expliquer leur fonctionnement et comment les utiliser.
- ✓ Enumérer et décrire les composants secondaires qu'on retrouve dans un FPGA
- ✓ Montrer comment programmer un FPGA pour réaliser un circuit combinatoire ou séquentiel.

Chapitre IV :
Programmation VHDL

IV. Programmation VHDL

IV.1 Introduction.

VHDL est l'acronyme de **VHSIC HDL** (***V**ery **H**igh **S**peed **I**ntegrated **C**ircuit **H**ardware **D**escription **L**anguage*), c'est un langage de description matérielle qui a été créé dans les années 1980 à la demande du département de la défense américaine (**DOD**).

- La première version du **VHDL** accessible au public a été publiée en 1985, et a fait l'objet d'une norme internationale en **1986** par l'institut des ingénieurs électriciens et électroniciens (**IEEE**).
- De nos jours, le langage **VHDL** devient un outil indispensable pour la conception des systèmes électroniques intégrés, il est proposé par la grande majorité des sociétés de développement et la commercialisation d'**ASIC** et de **FPGA** telle que la société américaine **Xilinx**.
- Avec un langage de description matérielle et un **FPGA** (***F**ield **P**rogrammable **G**ate **A**rray*), un concepteur peut développer rapidement et simuler un circuit numérique sophistiqué, de l'implémenter sur une carte de prototypage, et de vérifier son fonctionnement.

IV.2 Description générale :

Prenons comme exemple le circuit intégré 7400 (**Figure .IV.1*(a)*).** Ce dernier est constitué de quatre portes logiques NAND à deux entrées (**Figure.IV.1*(b)***)

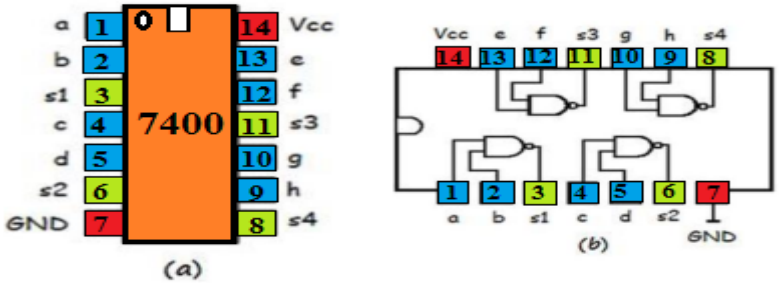


Figure.IV.1: Circuit intégré 7400 (Vue externe *(a)* et vue interne *(b)*).

La description du circuit intégré 7400 en langage VHDL est donnée par le **listing III.1 :**

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity CI_7500 is
        Port (a, b, c, d, e, f, g, h : in std_logic ;
              s1, s2, s3, s4: out std_logic);
end CI_7400;
architecture behavioral of CI_7400 is
begin
s1 <= a nand b;
s2 <= c nand d;
s3 <= e nand f;
s4 <= g nand h;
end behavioral ;
```

Listing IV.1 : Description VHDL du circuit intégré 7400.

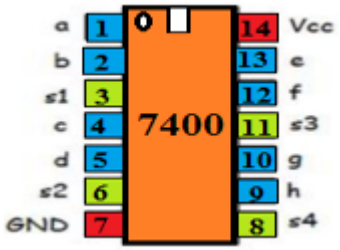
III.2.1 Déclaration des bibliothèques.

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. L'IEEE (Institut of Electrical and Electronics Engineers) a normalisées et plus particulièrement la bibliothèque IEEE1164. Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,..

Library *ieee*;
Use *ieee.std_logic_1164.all*;
Use *ieee.numeric_std.all*;
Use *ieee.std_logic_unsigned.all*;

Cette dernière bibliothèque est souvent utilisée pour l'écriture de compteurs. La directive **Use** permet de sélectionner les bibliothèques à utiliser.

III.2.2 Déclaration de l'entité et des entrées / sorties (I/O).

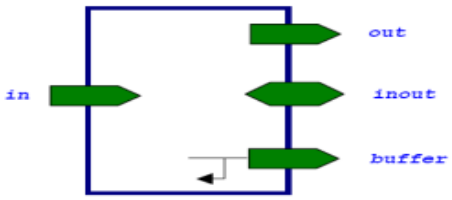


L'entité correspond au circuit vu de l'extérieur (Figure.I.1*(a)*), et comprend les entrées-sorties du circuit. La première ligne d'une entité (ligne 4 du code **Listing I.1**) indique le nom du circuit, ''CI_7400''dans notre exemple

La première ligne d'une entité (ligne 4 du code **Listing.IV.1**) indique le nom du circuit (nom de l'entity), ''CI_7400''dans notre exemple

Les entrées - sorties du circuit sont des ports qui doivent avoir le format suivant :

NOM_DU_PORT : MODE TYPE ;



Le mode d'un port peut être :

IN : Il s'agit d'une entrée
OUT : Il s'agit d'une sortie
INOUT : Il s'agit d'une entrée - sortie (port bidirectionnel)
BUFFER : Il s'agit d'une sortie rebouclée en entrée

TYPE du port : il faut savoir que le langage VHDL est un langage typé, c'est-à-dire que chaque objet manipulé doit avoir un type de données. Si le type du signal utilisé est un bus de données, on utilise alors le type **STD_LOGIC_VECTOR** qui est défini comme étant un tableau unidimensionnel d'éléments de type **STD_LOGIC.**
Dans le listing.IV.1, le type du port utilisé est STD_LOGIC. Il est défini dans le paquetage *STD_LOGIC_1164*, et possède neuf états comme montrer sur le tableau.IV.1 :

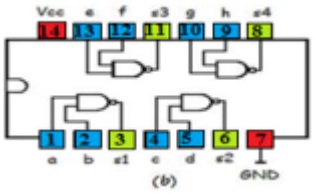
Tableau.IV.1 : Etats du paquetage STD_LOGIC_1164

Etat	Définition
0	Niveau logique 0, forçage faible
1	Niveau logique 1, forçage fort
Z	Haute impédance
U	Niveau non initialisé
X	Niveau inconnu, forçage fort
-	Niveau quelconque (Don't care)
L	Niveau logique 0, forçage faible
H	Niveau logique 1, forçage faible
W	Niveau inconnu, forçage faible

III.2.3 Déclaration de l'architecture

L'architecture du circuit correspond au circuit vu de l'intérieur (Figure.IV.1 *(b)*), et comprend essentiellement la description des opérations du circuit. Nous pouvons associer plusieurs architectures à une entité, mais pas le contraire.

```
architecture behavioral of CI_7400 is
begin
s1 <= a nand b;
s2 <= c nand d;
s3 <= e nand f;
s4 <= g nand h;
end behavioral ;
```



La description principale d'une architecture comprend toujours les deux mots réservés ''**BEGIN**'' et ''**END**'' (ligne 11 et 16 successivement). Les expressions des opérations du circuit se trouvent entre elles. Ces expressions sont des instructions qui s'exécutent en concurrence contrairement à un langage informatique comme le langage C, où les instructions s'exécutent séquentiellement.

III.3 Différents styles de description d'une architecture

III.3.1 Description par flot de données :

C'est une description qui décrit la façon dont les données transitent à l'intérieur d'un circuit.
Prenons comme exemple d'un multiplexeur 4 vers 1 (**Figure III.2**) et décrivons-le avec le style de description flot de données (**Listing III.2**).

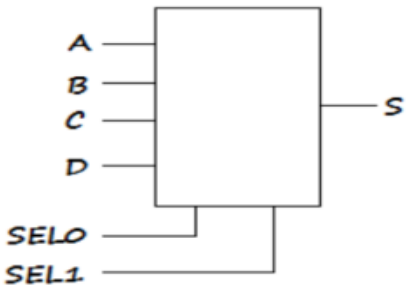


Figure IV.2 : Multiplexeur 4 vers 1.

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity mux_4v1 is
        Port (A, B, C, D, SEL0, SEL1: in std_logic ;
                S: out std_logic);
end mux_4v1;
architecture Données_Mux of mux_4v1 is
begin
S <= (not (SEL0) and not (SEL1) and A) or (not (SEL0)
and (SEL1) and B) or (SEL0 and not (SEL1) and C) or
(SEL0 and SEL1 and D);
end Données_Mux ;
```

Listing IV.2 : Description VHDL par flot de données d'un multiplexeur 4 vers 1.

Dans ce type de description par le flou de données, on modélise le circuit par un ensemble d'équations logiques et arithmétiques, car elle consiste à décrire chaque sortie par une équation en fonction des entrées.

III.3.2 Description comportementale :

Appelée aussi description procédurale, elle décrit le comportement d'un circuit selon des conditions (**IF**), des cas (**CASE, WHILE**), et des boucles (**FOR**).
C'est une description qui comporte des processus (**PROCESS**) qui s'exécutent en parallèles.
Les instructions à l'intérieur d'un processus s'exécutent séquentiellement.

A- Définition d'un PROCESS.

Un **process** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement c'est-à-dire les unes à la suite des autres. Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standard de la programmation structurée comme dans les systèmes à microprocesseurs. L'exécution d'un **process** est déclenchée par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans **la liste de sensibilité** lors de la déclaration du **process.**

B- Syntaxe du PROCESS

[Nom_du_process :] process(Liste_de_sensibilité_nom_des_signaux)
Begin
-- instructions du process
end process [Nom_du_process] ;

C- Règles de fonctionnement d'un process :

1) L'exécution d'un ***process*** a lieu à chaque changement d'état d'un signal de la liste de sensibilité ;
2) Les instructions du ***process*** s'exécutent séquentiellement ;

3) Les changements d'état des signaux par les instructions du ***process*** sont pris en compte à la **fin** du ***process***.

III.3.2.1 Description avec l'instruction if :

La description avec l'instruction if commence avec le mot réservé if et se termine avec les deux mots réservés end if ;(remarquer l'espace entre end et if suivie d'un point-virgule ;)ensuite vient la condition, suivie par le mot réservé then.

A- Syntaxe de l'instruction « if »

if condition_1 **then**
séquence_instructions_1 **;**
elsif condition_2 **then**
séquence_instructions_2 **;**
else
séquence_instruction_3 **;**
end if ;

Soit l'exemple de la description par l'instruction if suivant :

if signal_x = '1' **then**
y <= '0' ;
end if ;

La condition est vraie quand la valeur du signal « **signal_x** » est à l'état logique '**1**', et elle est fausse dans le cas contraire. Quand la condition est vraie, les instructions entre « **if »** et « **end if** » sont exécutées.
Dans notre exemple, le signale « **y »** recevra le niveau logique '**0**', si la valeur du signal « **signal_x** » vaudra le niveau logique '**1**', dans le cas contraire les instructions entre « **if** » et « **end if** » ne seront pas exécutées.

B- Exemple pratique :

On étudie le même circuit vu précédemment à savoir le multiplexeur 4 vers 1 , pour qu'on puisse faire la différence entre les différents styles de descriptions d'une architecture en VHDL.

Soit un multiplexeur 4 vers 1 donnée à la **figure III.2**. La description de ce circuit en VHDL est donnée par le **listing III.3**

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity mux_4v1 is
        Port (E1, E2, E3, E4: in std_logic ;
                SEL: in std_logic_vector (1 downto 0);
                S: out std_logic);
```

```
end mux_4v1;
architecture comportemental of mux_4v1 is
begin
process (E1, E2, E3, E4)
begin
if (SEL=”00”)   then
S <= E1;
elsif (SEL=”01”)   then
S <= E2;
elsif (SEL=”10”)   then
S <= E3;
else
S <= E4;
End if ;
End process;
end comportementale ;
```

Listing IV.3 Multiplexeur 4 vers 1 avec la description « **if** ».

III.3.2.2 Description avec l'instruction CASE :

La description avec l'instruction **case** est utilisée lorsque la valeur d'un signal peut être utilisée pour choisir entre un certain nombre d'actions. Cette description débute avec le mot réservé « **case** » suivi par *une instruction*, et le mot-clé « **is** ». L'*instruction* entre « **case** » et « **is** », renvoie une valeur qui correspond à un des choix qui se trouvent dans les déclarations « **WHEN** » et « **OTHERS** ».

A- Syntaxe de l'instruction « CASE»

case instruction **is**
when choix_1 => instruction_1 **;**
when choix_2 => instruction_2**;**
when choix_3 => instruction_3**;**
when others => instruction_n**;**

La description avec l'instruction **case** doit obligatoirement avoir le choix « **others** » à la fin, si toutes les valeurs de l'expression ne sont pas énumérées.

B- Exemple pratique :

On donne dans le **listing III.4** une autre architecture de l'entité **mux_4v1**, décrit avec la description « **case** ».

```
architecture comportemental_2 of mux_4v1 is
begin
process (E1, E2, E3, E4)
begin
case SEL is
```

```
when “00”=> S<= E1;
when “01”=> S<= E2;
when “10”=> S<= E3;
when “11”=> S<= E4;
when others => S<= ‘0’;
End case ;
End process;
end comportementale _2;
```

Listing IV.4 Multiplexeur 4 vers 1 avec la description « **Case** ».

III.3.2.3 Description avec la boucle de répétition «FOR» :

La boucle **for** est basée sur un indice, et répète les instructions séquentielles pour un nombre fixe d’itérations (intervalle_boucle).

A- Syntaxe de la boucle « FOR » : La syntaxe simplifiée d’une boucle **for** est :

For indice **in** intervalle_boucle **loop**
instructions séquentielles **;**
end loop;

B- Exemple

L’utilisation de la boucle **for** est montrée par un exemple simple : description d’un circuit **xor** de 4 bits (**listing III.5**)

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity xor_4bit_loop is
        Port (A, B: in std_logic_vector (3 downto 0);
                S: out std_logic_vector (1 downto 0));
end xor_4bit_loop;
architecture boucle of xor_4bit_loop is
begin
process (A, B)
begin
for I  in 3 downto 0 loop
S (I) <= A (I) xor B (I)  ;
End loop ;
End process;
end boucle ;
```

Listing IV.5 Description VHDL d’un circuit numérique **xor** de 4 bits décrit avec la boucle « **for** ».

Le nombre d’itérations de la boucle est 4 (**3 downto 0**). L’indice de la boucle est « **I** », ce dernier est local (c’est-à-dire qu’il n’appartient qu’à la boucle) et n’a pas besoin d’être déclarée.

L'indice prends toujours la valeur qui est à gauche dans l'intervalle d'itération, dans notre exemple, l'indice « **I** » vaudra **3** dans la première itération, **2** dans la deuxième itération, **1** dans la troisième itération, et enfin **0** dans la quatrième itération.

III.3.2.4 Description avec la boucle de répétition «WHILE» :

La boucle **while** est une variante de la boucle **for**, mis à part que l'incrémentation de l'indice de la boucle **while** se fait tant que la condition booléenne associée est vraie.

A- Syntaxe de la boucle « WHILE » : La syntaxe simplifiée d'une boucle **while** est :

while condition_booléenne **loop**
instructions_séquentielles **;**
end loop;

B- Exemple
L'utilisation de la boucle **while** est démontré par le même exemple vu avec la boucle **for.**
La description d'un circuit **xor** de 4 bits (**listing III.6**)

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity xor_4bit_loop is
        Port (A, B: in std_logic_vector (3 downto 0);
              S: out std_logic_vector (1 downto 0));
end xor_4bit_loop;
architecture boucle of xor_4bit_loop is
begin
process (A, B)
Variable I : integer :=0;
begin
While I < 5  loop
S (I) <= A (I) xor B (I)  ;
End loop ;
End process;
end boucle ;
```

Listing IV.6 Description VHDL d'un circuit numérique **xor** de 4 bits décrit avec la boucle « **while** ».

L'indice de la boucle est la variable locale **I** qu'on a initialisé auparavant à 0 :
variable I : integer := 0 ;
Le nombre d'itérations de la boucle est 4 (**I< 5**).
L'instruction **S(I) <= A(I) xor B(I) ;**va s'exécuter tant que la condition sera vraie (c'est-à-dire tant que la variable **I** sera strictement inférieure à **5**).

III.3.3 Description Structurelle

Lorsqu'on a un système électronique composé, on peut structurer et séparer ces composants en des blocs plus petits pour les décrire plus simplement. L'avantage d'une telle description est la simplicité et la rapidité de la synthèse. Pour comprendre le principe de cette description, prenons comme exemple l'additionneur complet 1 bit. Cet additionneur est composé de deux demi-additionneurs et une porte logique OR (**figure IV.3**)

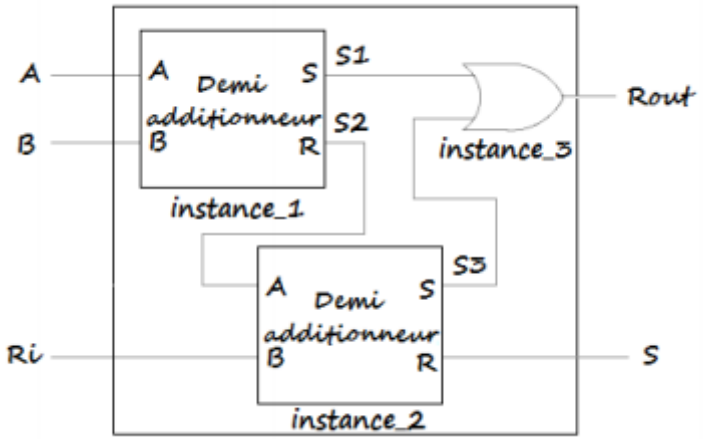


Figure IV.3 Schéma d'un additionneur complet 1 bit.

Les deux demi-additionneurs et la porte logique OR peuvent être considérés comme des blocs. Chaque bloc est appelé « **component** » (composant), et est une instance du modèle. Pour différencier ces dernières, chaque instance a un nom distinct.

Chaque bloc doit être décrit séparément (c'est-à-dire avoir son entité et l'architecture associée) (**listing IV.7** et **listing IV.8**).

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity porte_ou is
        Port (E1, E2: in std_logic ;
              Y: out std_logic);
end porte_ou;
architecture Archi_ou of porte_ou is
begin
Y<= E1 or E2;
end Archi_ou ;
```

Listing IV.7 Description VHDL d'une porte logique **OR**.

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity half_adder is
        Port (A, B: in std_logic ;
```

```
                S, R: out std_logic);
end half_adder;
architecture Archi_addi of half_adder is
begin
S<= A xor B;
R<= A and B;
end Archi_addi;
```

Listing III.8 Description VHDL d'un demi-additionneur 1 bit.

Dans l'architecture de l'additionneur complet 1 bit, il faut déclarer les composants (**component**) utilisés, ainsi que leurs broches d'entrée-sortie pour faire les interconnexions nécessaires (**listing III.9**).

```
component half_adder
        Port (A, B: in std_logic ;
                S, R: out std_logic);
end component;

component porte_ou
        Port (E1, E2: in std_logic ;
                Y: out std_logic);
end component;
```

Listing III.9 Déclaration des composants de l'additionneur complet 1 bit.

Nous avons besoin des signaux intermédiaires pour décrire l'assemblage des instances utilisées.

III.3.1 Instanciation :

Il s'agit de mettre en correspondance chaque broche de chacune des instances des composants avec les ports auxquels il est connecté.
Il y a trois types d'instanciation :
- Instanciation par position.
- Instanciation par nom.
- Instanciation mixte

A- Instanciation par position :

L'instanciation par position consiste à interconnecter pour chaque instance, les signaux connectés à ses broches exactement selon l'ordre choisi dans la déclaration « **component** » (**listing III.10**).

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity full_adder is
        Port (A, B, Ri: in std_logic ;
```

```
                    S, Rout: out std_logic);
end full_adder;
architecture struct_addi of full_adder is
component half_adder
        Port (A, B: in std_logic ;
                    S, R: out std_logic);
end component;

component porte_ou
        Port (E1, E2: in std_logic ;
                    Y: out std_logic);
end component;
signal S1, S2, S3 : std_logic;
begin
instance_1: half_adder port map (A, B, S1, S2);
instance_2: half_adder port map (S2, R1, S3, S);
instance_3: half_adder port map (S1, S2, Rout);
end struct_addi;
```

Listing III.10 Description structurelle d'un additionneur complet 1 bit avec instanciation par position.

B- Instanciation par nom :

L'instanciation par nom consiste à interconnecter pour chaque instance, les signaux connectés à ses broches sans respecter l'ordre choisi dans la déclaration « **component** » (**listing III.11**). Il faut cependant utiliser le symbole « => » pour la correspondance entre les broches de l'instance et le signal auquel elle est connectée.

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity full_adder is
        Port (A, B, Ri: in std_logic ;
                    S, Rout: out std_logic);
end full_adder;
architecture struct_addi of full_adder is
component half_adder
        Port (A, B: in std_logic ;
                    S, R: out std_logic);
end component;

component porte_ou
        Port (E1, E2: in std_logic ;
                    Y: out std_logic);
end component;
signal S1, S2, S3 : std_logic;
begin
instance_1: half_adder port map (A=>A, S=>S1, B=>B, R=>S2);
instance_2: half_adder port map (A=>S2, B=>Ri, S=>S3, R=>S);
instance_3: half_adder port map (E1=>S1, Y=>Rout, E2=>S3);
end struct_addi;
```

Listing III.10 Description structurelle d'un additionneur complet 1 bit avec instanciation par nom.

C- Instanciation mixte (listing III.12):

L'instanciation mixte permet d'utiliser les instanciations par position et par nom en même temps, mais dans ce cas-là, l'instruction « **port map** » autorise une association par position pour commencer, ensuite une association par nom pour finir.

On ne peut pas faire le contraire.

```
begin
instance_1: half_adder port map (A, B, S1, R=>S2);
instance_2: half_adder port map (S2, Ri, S=>S3, R=>S);
instance_3: half_adder port map (S1, E2=>S3, Y=>Rout);
end struct_addi;
```

Listing III.12 Description structurelle d'un additionneur complet 1 bit avec instanciation mixte.

III.4 Conclusion :

Nous devrons maintenant d'être capable de :

- ✓ Expliquer les différents styles de description d'une architecture ;
- ✓ D'exploiter les déférentes bibliothèque et les paquetages du VHDL ;
- ✓ Décrire, un circuit numérique à l'aide du langage VHDL.

Chapitre IV :

Applications : Implémentation de quelques circuits logiques dans les FPGA

V. Applications : Implémentation de quelques circuits logiques dans les circuits FPGA

V.1 INTRODUCTION

Dans ce chapitre, nous allons d'abord décrire en VHDL quelques circuits logiques de base à savoir les multiplexeurs, et décodeurs puis quelques circuits séquentiels tel que les bascules et les compteurs.

Dans la deuxième partie de ce chapitre, la procédure de l'implémentation réelle des exemples pratiques de base (portes logiques :AND, OR, XOR) sur la carte FPGA Micro Board Xilinx Spartan-6 FPGA LX9 sera détaillée.

V.2 Décodeur 1 parmi 4

Un décodeur « 1 parmi 2^n » (une sortie parmi n entrées), est un circuit logique à n entrées et 2^n sorties, qui fournissent tous les produits Pi qui identifient toutes les combinaisons de n variables d'entrée.

Les sorties sont actives à l'état 0 (vraies au niveau bas). On a donc une seule sortie à l'état 0, celle qui décode la combinaison présente sur les entrées; toutes les autres sont à l'état 1.

Pour pouvoir activer toutes les 4 voies, on a besoin de 2 bits à l'entrée car c'est $2^2=4$ pour le décodeur 1 parmi 4 :

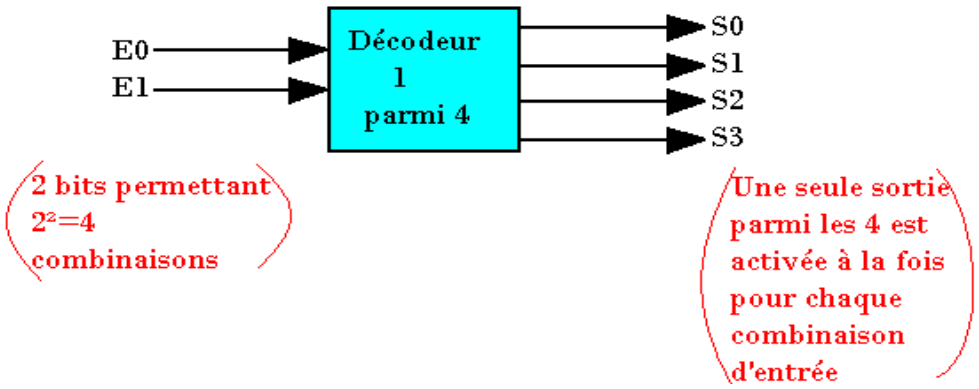


Figure V.1 : Schéma de principe du décodeur 1 parmi 4

V.2.1 Table de vérité

Le tableau suivant montre la table de fonctionnement ou de vérité du décodeur 1 parmi 4

Code binaire d'entrée		Code 1 parmi 4 sorties			
E1	**E0**	**S3**	**S2**	**S1**	**S0**
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Table V.1 : Table de fonctionnement du décodeur 1 parmi 4.

A partir de cette table on peut déduire les équations logiques de décodeur

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

V.2.2 Description en langage VHDL

A. Description par flot de données

La description en langage VHDL du décodeur 1 parmi 4 en utilisant la description par flot de données est montrée par le listing suivant :

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity DECOD1_4 is
port(IN0, IN1: in std_logic;
D0, D1, D2, D3: out std_logic);
end DECOD1_4;

architecture DESCRIPTION of DECOD1_4 is
begin
D0 <= (not(IN1) and not(IN0));
D1 <= (not(IN1) and IN0);
D2 <= (IN1 and not(IN0));
D3 <= (IN1 and IN0);
end DESCRIPTION;
```

B. Description comportementale

La description en langage VHDL du décodeur 1 parmi 4 en utilisant la description comportementale avec l'instruction **CASE** est donnée par le listing suivant :

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity DECOD1_4 is
port(Entrée: in std_logic_vector( 1 downto 0);
Sortie: out std_logic_vector(3 downto 0));
end DECOD1_4;

architecture DESCRIPTION of DECOD1_4 is
begin
process (Entrée)
begin
Case Entrée is
When "01" => Sortie<= "0010";
When "10" => Sortie<= "0100";
When "11" => Sortie<= "1000";
When Others => Sortie<= "0001";
end Case;
end process;
end DESCRIPTION;
```

V.3 Multiplexeur 4 vers 1

Un multiplexeur ou sélecteur de données est un commutateur qui va pouvoir, à l'aide de n bits d'adresse, sélectionner une de ses 2^n entrées et la mettre en communication avec sa sortie. Le schéma ci-dessous donne une image d'un multiplexeur 4 voies (E3 à E0) vers une (S) sélec table à l'aide des bits d'adresse A1 et A0.

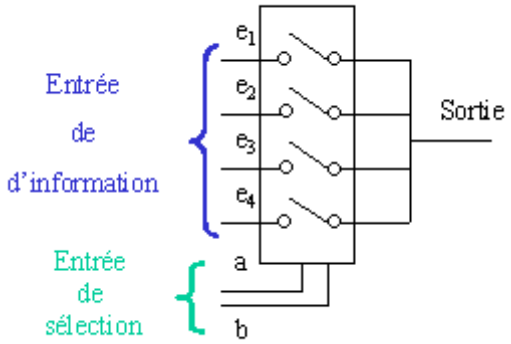


Figure V.2 : Schéma de principe d'un multiplexeur 4 vers 1.

V.3.1 Table de vérité

Pour concevoir cette fonction, il serait possible d'établir la table de vérité du circuit en tenant le raisonnement suivant:

(1) - Lorsque A1,A0 = 00 si E0 = 0 →S = 0, si E0 = 1 → S = 1 et ceci quelles que soient les entrées E1,E2,E3

(2) - Lorsque A1,A0 = 01 si E1 = 0 → S = 0, si E1 = 1 → S = 1 et ceci quelles que soient les entrées E0,E2,E3 Etc...

Le tableau suivant montre la table de fonctionnement ou de vérité d'un multiplexeur 4 vers 1.

Adresse		**Sélection**			
A1	**A0**	**S3**	**S2**	**S1**	**S0**
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Table V.2 : Table de fonctionnement d'un multiplexeur 4 vers 1.

A partir de cette table on peut déduire les équations logiques du décodeur :

$$s0 = \overline{A1} \cdot \overline{A0}$$
$$s1 = \overline{A1} \cdot A0$$
$$s2 = A1 \cdot \overline{A0}$$
$$s3 = A1 \cdot A0$$

V.3.2 Description en langage VHDL

A. Description avec flot de données

La description en langage VHDL d'un multiplexeur 4 vers 1 en utilisant la description par flot de données est montrée par le listing suivant :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX is
        port( E0, E1, E2, E3, SEL0, SEL1: in std_logic;
                S: out std_logic);
End MUX;

architecture FLOT_MUX of MUX is
        begin
        S <= ((not SEL0) and (not SEL1) and E0) or
             (SEL0 and (not SEL1) and E1) or
             ((not SEL0) and SEL1 and E2) or
             (SEL0 and SEL1 and E3);
end FLOT_MUX;
```

B. Description comportementale

La description en langage VHDL du multiplexeur 4 vers 1 en utilisant la description comportementale avec l'instruction **CASE** est donnée par le listing suivant :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Multiplexer_VHDL is
port ( a, b, c, d, e, f, g, h : in std_logic;
        Sel : in std_logic_vector(2 downto 0);
        Output : out std_logic );
end Multiplexer_VHDL;
architecture Behavioral of Multiplexer_VHDL is
begin
        process (a, b, c, d, e, f, g, h, Sel) is
        begin
        case Sel is
                when "000"  => Output <= a;
                when "001"  => Output <= b;
                when "010"  => Output <= c;
                when "011"  => Output <= d;
                when "100"  => Output <= e;
                when "101"  => Output <= f;
                when "110"  => Output <= g;
                when others => Output <= h;

        end case;
        end process;
end Behavioral;
```

La figure V.3 illustre le diagramme de simulation du multiplexeur 4 vers 1 sous le model Sim

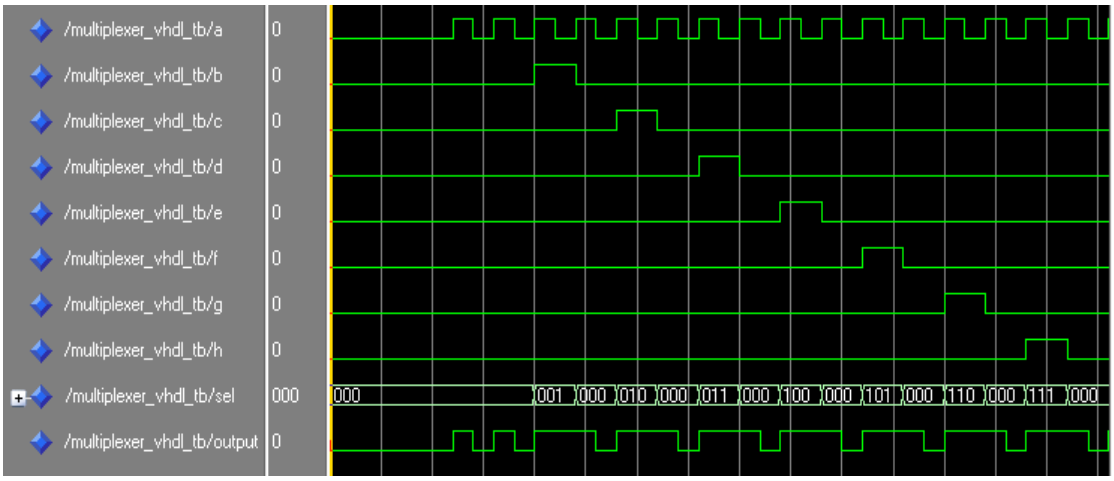


Figure V.3 : Diagramme de simulation du multiplexeur 4 vers 1.

On peut cependant utiliser la syntaxe « when », qui indique la valeur que doit prendre le signal ou la sortie dans tel ou tel cas. Cette écriture est déjà plus lisible, elle correspond beaucoup plus directement à la définition en français du multiplexeur.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity MUX is port( E0, E1, E2, E3, SEL0, SEL1: in std_logic;
S: out std_logic);
end;
architecture FLOT_MUX of MUX is
begin
S <= E0 when (SEL0='0' and SEL1='0') else
      E1 when (SEL0='1' and SEL1='0') else
     E2 when (SEL0='0' and SEL1='1') else
     E3;
end FLOT_MUX;
```

V.4 Bascules

V.4.1 Bascule D

Pour modéliser une bascule, il est nécessaire de pouvoir décrire le fait que le changement d'état se produit sur une *transition* d'un signal d'horloge et non sur sa *valeur*.

- Pour ce faire, on peut utiliser les attributs d'évènements (*event attribute*) définis en VHDL.
- L'exemple démontre l'utilisation de l'attribut event sur le signale CLK, dénoté par *CLK' event.*

La condition *CLK='1'* dénote alors un font montant comme montré sur la figure V.4.

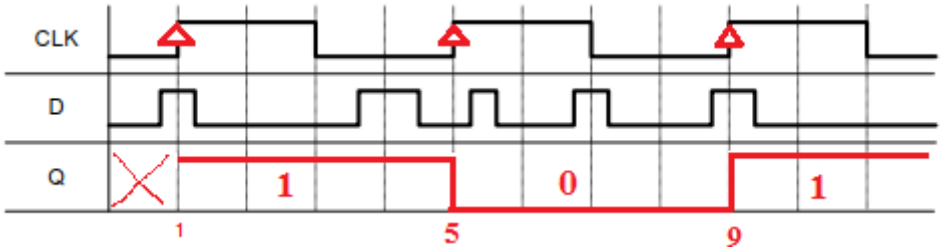


Figure V.4 : Diagramme de simulation de la bascule D.

Sur l'intervalle [0 1] on peut savoir quel est l'état de la sortie Q.
A partir de1 jusqu'à 5 on a un front montant sur le signal d'horloge et D=1 donc on a Q=D=1.
A partir de 5 jusqu'à 9 on a un autre front montant su le signal d'horloge et D=0 don on a Q=D=0.
A partir de 9 on a un front montant sur le signal d'horloge et D=1 donc on a Q=D=1 jusqu'à une autre éventuelle transition d'horloge/
Le listing suivant donne la description VHDL de la bascule D en utilisant l'attribut *CLK' event* .

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity basculeDRA is
        Port (rest, CLK, D: in std_logic ;
                Q: out std_logic);
end basculeDRA;
architecture basculeDRasynch of basculeDRA is
begin
process (CLK, rest) is
begin
if (rest='0')  then
Q <= '0';
Elsif (rising_edge(CLK))  then ;
Q <= D;
End if;
End process;
end basculeDRasynch ;
```

Remarque :

Le package std_logic_1164 contient aussi deux fonctions qui combinent ces conditions, *rising_edge()* et *falling_edge().* Ces deux fonctions retournent des valeurs booléennes.

comportement désiré	option 1	option 2
front montant	CLK'event and CLK = '1'	rising_edge(CLK)
front descendant	CLK'event and CLK = '0'	falling_edge(CLK)

Table V.3 : Table de déclaration des fronts montant et descendant sous VHDL.

Il existe deux types de signaux d'initialisation de la bascule D pour la ramener à l'état désirée indépendamment ou dépendamment du signal d'horloge.

Le listing suivant montre l'initialisation asynchrone c-à-d indépendamment du signal d'horloge

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity basculeDRA is
        Port (rest, CLK, D: in std_logic ;
                Q: out std_logic);
end basculeDRA;
architecture basculeDRasynch of basculeDRA is
begin
process (CLK, rest) is
begin
if (rest='0')  then
Q <= '0';
Elsif (rising_edge(CLK))  then ;
Q <= D;
End if;
End process;
end basculeDRasynch ;
```

Dans le process on a deux signaux (un signal d'horloge et un autre pour l'initialisation), et une condition **if** pour la transition du signal d'horloge, et à l'extérieur de la condition du signal d'horloge on retrouve ce qui va se passer dans le cas où le signal d'initialisation est à zéro. A ce moment-là on va mettre zéro à la sortie logique Q.

Le listing suivant montre l'initialisation synchrone c'est-à-dire une initialisation qui dépend du signal d'horloge.

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity basculeDRA is
        Port (rest, CLK, D: in std_logic ;
                Q: out std_logic);
end basculeDRS;
architecture basculeDRsynch of basculeDRS is
begin
process (CLK, rest) is
begin
if (rising_edge(CLK))  then ;
if (rest='0')  then
Q <= '0';
else
Q <= D;
End if;
End process;
end basculeDRsynch ;
```

Dans ce listing on remarque que la condition sur le signal d'initialisation est à l'intérieur de la condition d'horloge, cette condition ne sera évaluée que lors de la transition du signal d'horloge,

donc il sera impossible de remettre la bascule à zéro s'il n'y a pas en même temps une transition sur le signal d'horloge et le signal d'initialisation pour remettre la bascule à zéro.

V.4.2 Bascule RS

Un circuit à bascule peut être construit à partir de deux portes NAND ou de deux portes NOR. Ces bascules sont illustrées à la figure V.5. Chaque bascule a deux sorties, Q et Q 'et deux entrées, set et reset. Ce type de bascule est appelé bascule RS.

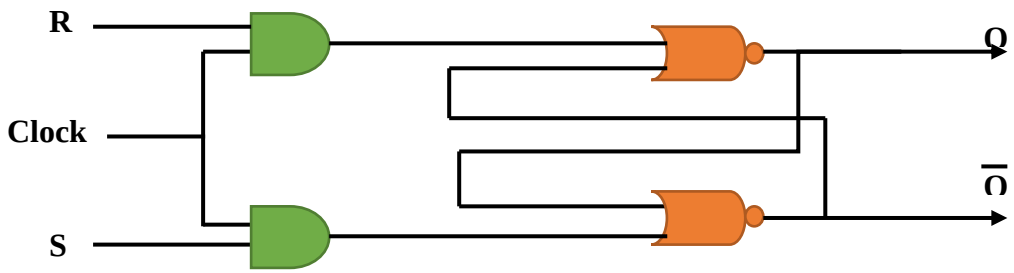


Figure V.5: Schéma logique de la bascule RS.

Le tableau V.4 suivant montre la table de fonctionnement ou de vérité de la bascule RS

H	R	S	Q	$\overline{Q}$
0	0	0	**Q**	$\overline{Q}$
0	0	1	**Q**	$\overline{Q}$
0	1	0	**Q**	$\overline{Q}$
0	1	1	**Q**	$\overline{Q}$
1	0	0	**Q**	$\overline{Q}$
1	0	1	**1**	**0**
1	1	0	**0**	**1**
1	1	1	**Etat indéterminé**	**Etat indéterminé**

Table V.4 : Table de vérité de la bascule RS.

La description en code VHDL de la bascule RS est donnée par le listing suivant avec des variables interne :

```
Library ieee ;
Use ieee.std_logic_1164.all;
Entity RSH is
        Port (R, S, H: in std_logic ;
                Q, Q_bar: out std_logic);
end RSH;
architecture basc of RSH is
Signal SIG : std_logic :=0 ;
Signal SIG : std_logic_vector (1 downto 0) ;
Begin
RS(1) <= R;
RS(0) <= S;
process (RS, SIG, H)
if (H='1')  then
Case RS is
When "00" => SIG<= SIG;
When "01" => SIG<= '1';
When "10" => SIG<= '0';
When others  => SIG<= '-';
End case;
else
SIG <= SIG;
End if;
Q <= SIG;
Q_bar <= not SIG;
End process;
end basc ;
```

V.5 Compteurs

Ils sont très utilisés dans les descriptions **VHDL**. L'écriture d'un compteur peut être très simple comme très compliquée. Ils font appels aux ***process***.

Soit la description VHDL du compteur simple suivante :

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP4BITS is
PORT (
CLOCK : in std_logic;
Q : inout std_logic_vector(3 downto 0));
end CMP4BITS;
architecture DESCRIPTION of CMP4BITS is
begin
process (CLOCK)
begin
if (CLOCK ='1' and CLOCK'event) then
Q <= Q + 1;
end if;
```

end process;
end DESCRIPTION;

La description ci-dessus est très simple. Le déclenchement du ***PROCESS*** se fera sur un changement d'état du signal ***CLOCK***, l'incrémentation de la sortie ***Q*** se fera aussi sur le front montant de l'horloge ***CLOCK***.

- L'incrémentation du compteur est réalisée par l'opérateur **+** associé à la valeur **1**,parce que les entrées et sorties ainsi que les signaux sont déclarés de type ***std_logic*** ou ***std_logic_vector***, et par conséquent on ne peut pas leur associer de valeurs entières décimales.
- Un signal peut prendre comme valeur les états **'1'** ou **'0'** et un bus de n'importe quelle valeur, du moment qu'elle est écrite entre deux guillemets **"1010"** ou **X"A"** ou **o"12",** mais pas une valeur comme par exemple **1,2,3,4**. Ces valeurs décimales sont interprétées par le synthétiseur comme des valeurs entières (**integer**), on ne peut pas par défaut additionner un nombre entier **1** avec un bus de type électronique (***st_logic_vector***), c'est pour cela que l'on rajoute dans la partie déclaration des bibliothèques les lignes :

  Use ieee.numeric_std.all;
  Use ieee.std_logic_unsigned.all;

Ces deux bibliothèques ont des fonctions de conversions de types et elles permettent d'associer un entier avec des signaux électroniques. Elles permettent d'écrire facilement des compteurs, décompteurs, additionneurs, soustracteurs, …..

Remarque :

Le signal ***Q*** est déclaré dans l'entité de type ***inout***, cela est logique car il est utilisé à la fois comme entrée et comme sortie pour permettre l'incrémentation du compteur. Ce type d'écriture est peu utilisé car elle ne permet pas au code d'être portable, on préférera utiliser un signal interne, celui-ci peut être à la fois une entrée et une sortie.

V.6 Implémentation réelle des circuits logiques

Cette partie détailleras la procédure de l'implémentation réelle de quelques circuits logiques de base (AND, OR, XOR) sur la carte FPGA Micro Board Xilinx Spartan-6 FPGA LX9 et pour ce faire il faut suivre les étapes suivantes :

V.6.1 Création d'un nouveau projet dans ISE

Dans un premier lieu il faut créer un nouveau projet ISE qui cible le FPGA Spartan 6, pour lancer, sélectionnez Start > Programs > Xilinx ISE design suite 14.5 > ISE Design Tools > Project Navigator. Puis pour créer un projet dans ISE, il faut cliquez sur File > New Project. Dans la fenêtre *New Project Wizard*, tapez Exemple comme nom de projet sélectionnez *HDL* pour le champ *Top-Level Source Type* et cliquez sur Next puis entrez les valeurs suivantes dans la fenêtre *New Project Wizard – Project Settings*.

a. Top-level source type : HDL
b. Evaluation development board: None Specified
c. Product Category: All
d. Family: Spartan6
e. Device: XC6SLX16
f. Package: CSG324

g. Speed: -3
h. Synthesis Tool: XST (VHDL/Verilog)
i. Simulator: modelsim-SE VHDL
j. Preferred Language: VHDL

Enfin, cliquez sur **Next**. Cliquez enfin sur Finish dans la fenêtre *Project Summary* comme montré sur la figure V.6 suivantes :

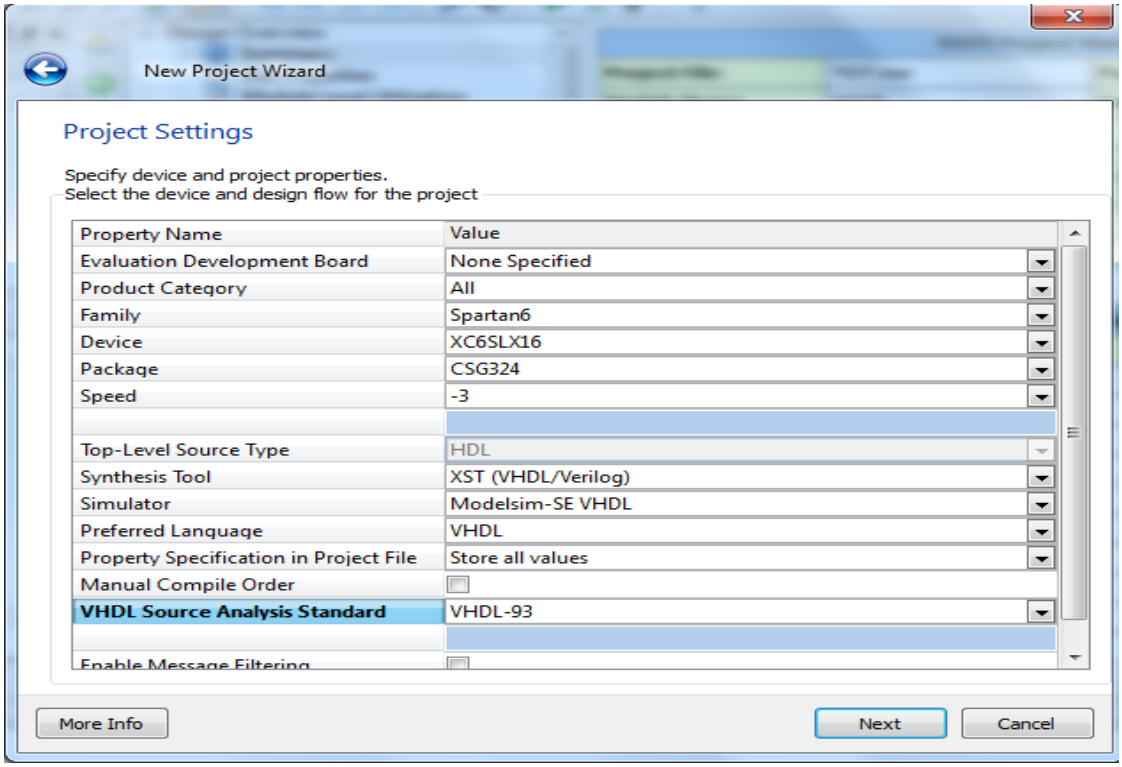


Figure V.6 : Configuration de la carte FPGA.

À ce stade, vous obtenez un nouveau projet ISE ne contenant encore aucun fichier HDL.

V.6.2 Création d'un HDL top level

Pour créer le fichier HDL il faut cliquez sur **Project > New Source** et sélectionnez **VHDL module** dans la fenêtre *Project Wizard – Select Source Type*, tapez **Test** comme nom de fichier et vérifiez que la case **Add to project** est cochée puis cliquez sur **Next comme illustrer sur la figure V.7 :**

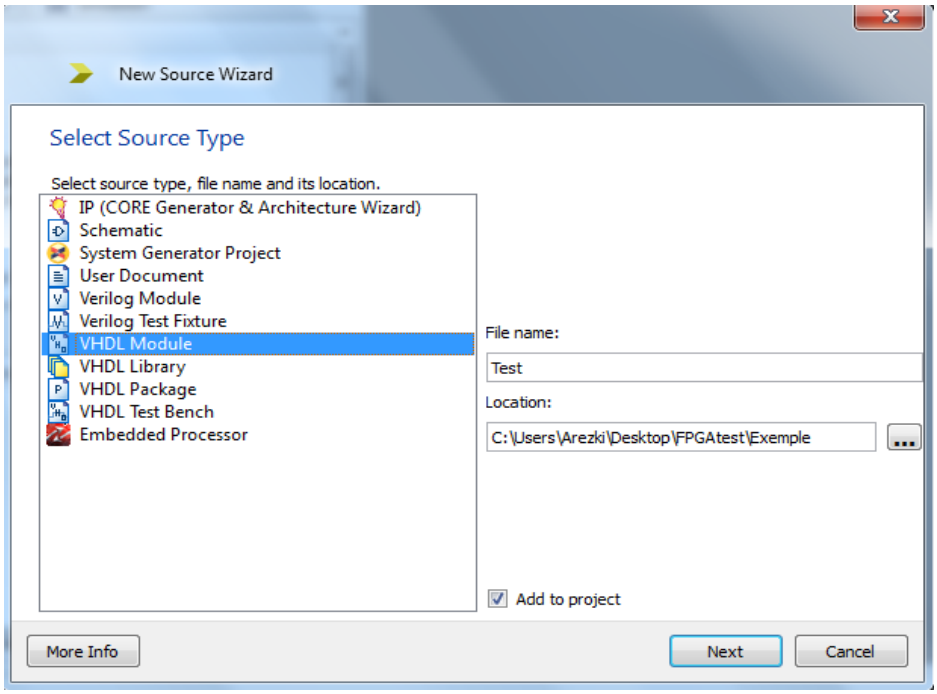


Figure V.7 : Création du module VHDL.

Dans un deuxième lieu on va créer les ports d'entrée **A, B** et le port de la sortie **D** comme illustré sur la figure V.8

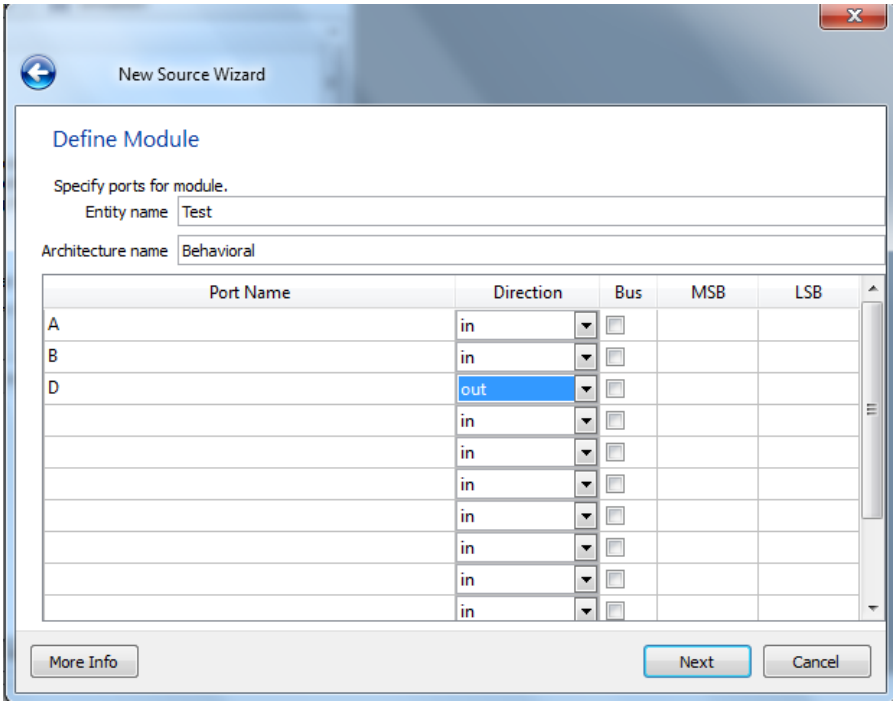


Figure V.8 : Configurations des ports entrées/sorties.

Entrez **A, B**, **D** dans les 3 premiers champs Port Name puis dans le champ Direction, choisissez **in** pour A, B, et **out** pour D, puis cliquez sur **Next** et ensuite sur **Finish** pour compléter cette étape de configuration. Une boîte de dialogue affiche la description VHDL générée par le Wizard. Cliquez sur **Finish**.

Le fichier VHDL vide décrivant le module Test s'ouvre dans l'espace de travail comme montré sur la figure ci-dessous (Figure V.9). Ce fichier contient l'entité du module mais présente une architecture vide à compléter.

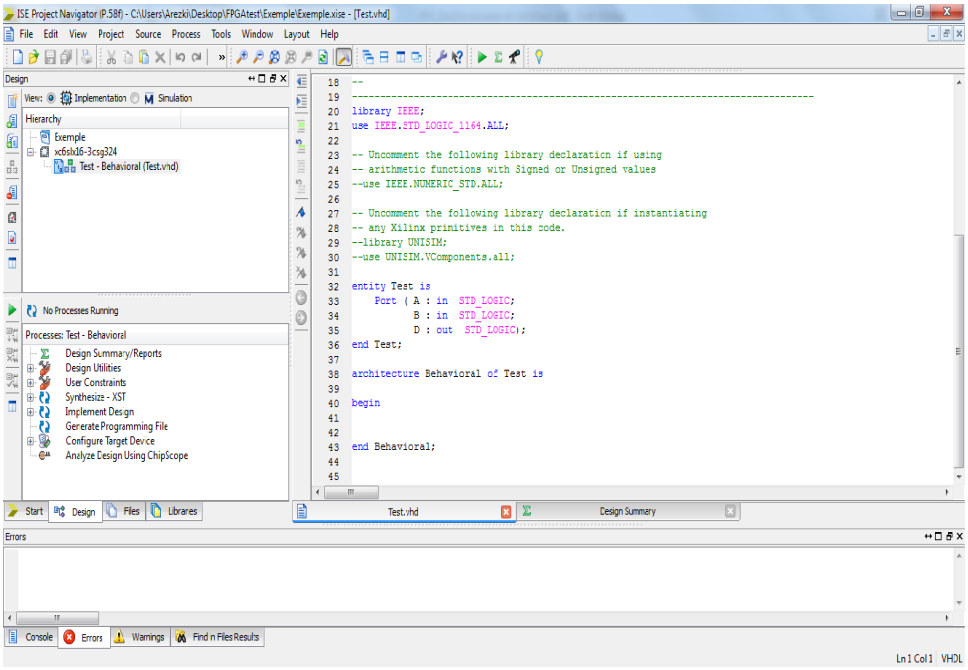


Figure V.9 : Espace de travail.

Pour compléter la description VHDL du module Test, effectuer les étapes suivantes :
1. Dans le fichier Test ouvert dans l'espace de travail du Navigateur de Projet ISE, effacez toutes les lignes de code situées sous la ligne architecture Behavioral of Test is.
2. Supposons que le circuit de Test soit similaire à la porte AND, alors, le comportement du circuit sera écrit comme :

```
38  architecture Behavioral of Test is
39
40  begin
41  D<= A and B;
42
43  end Behavioral;
```

Le code VHDL du module du Test est maintenant complété, donc il reste à cliquer sur **File > Save** , après sélectionnez l'item **Test** dans l'espace Hierarchy et double-cliquez sur **Synthesize** dans l'espace **Processes**, puis vérifiez les résultats dans la Console et corrigez les erreurs de syntaxe s'il y a lieu.

V.6.3 Création d'implémentation de contraintes

Les contraintes d'implémentation du design sont entrées dans un fichier UCF (User constraints file). UCF utilisé à définir le mapping entre les ports logiques et les ports physiques de la carte (ports du circuit conçu et ports de FPGA de la Spartan 6 LX9). Nous allons utiliser deux Switches pour A et B pour donner des valeurs d'entrée et une LED pour afficher le résultat de D.

Pour ajouter le fichier UCF au projet, il faut suivre les étapes décrites ci-dessous :
1. Copiez l'UCF à partir du simple dossier et collez-le dans le dossier du projet Exemple
2. Cliquez sur **Project > Add Source**
3. Sélectionnez UCF et cliquez sur Open.
4. Double-cliquez sur le Avt_S6LX9_MicroBoard_UCF_110804.ucf pour l'ouvrir
Pour utiliser le fichier UCF dans l'implémentation ci-dessous, nous décommenterons les lignes 100 et 101 et changerons les noms par A et B pour nos entrées, pour la sortie, D, nous utiliserons LED1. Nous allons donc décommenter la ligne 108 et remplacer le nom par D, puis cliquez sur **File > Save.**

```
85  #NET CLOCK_Y2 TNM_NET = CLOCK_Y2;
86  #TIMESPEC TS_CLOCK_Y2 = PERIOD CLOCK_Y2 66666.7 kHz;
87  #NET CLOCK_Y3 TNM_NET = CLOCK_Y3;
88  #TIMESPEC TS_CLOCK_Y3 = PERIOD CLOCK_Y3 100000 kHz;
89
90  ##############################################################################
91  # The following oscillator is not populated in production but the footprint
92  # is compatible with the Maxim DS1088LU
93  ##############################################################################
94  #NET BACKUP_CLK        LOC = R8   | IOSTANDARD = LVCMOS33;              # "MAIN_CLK"
95
96  ##############################################################################
97  # User DIP Switch x4
98  #   Internal pull-down required since external resistor is not populated
99  ##############################################################################
100 NET "A"          LOC = B3   | IOSTANDARD = LVCMOS33 | PULLDOWN;    # "GPIO_DIP1"
101 NET "B"          LOC = A3   | IOSTANDARD = LVCMOS33 | PULLDOWN;    # "GPIO_DIP2"
102 #NET "C"          LOC = B4   | IOSTANDARD = LVCMOS33 | PULLDOWN;    # "GPIO_DIP3"
103 #NET GPIO_DIP4         LOC = A4   | IOSTANDARD = LVCMOS33 | PULLDOWN;    # "GPIO_DIP4"
104
105 ##############################################################################
106 # User LEDs
107 ##############################################################################
108 NET "D"          LOC = P4   | IOSTANDARD = LVCMOS18;              # "GPIO_LED1"
109 #NET "s2"          LOC = L6   | IOSTANDARD = LVCMOS18;              # "GPIO_LED2"
110 #NET "S"          LOC = F5   | IOSTANDARD = LVCMOS18;              # "GPIO_LED3"
111 #NET "G"          LOC = C2   | IOSTANDARD = LVCMOS18;              # "GPIO_LED4"
112
```

V.6.4 Implémentation du projet

L'implémentation du projet consiste à effectuer les étapes d'interprétation (translation), de mapping, de placement & routage et de génération du fichier de programmation BIT. Tous les outils d'implémentation du projet sont intégrés au Navigateur ISE.

1. Sélectionnez l'onglet Design dans l'espace Hierarchy situé dans la partie supérieure gauche du Navigateur de Projet.
2. Assurez-vous que l'item Implementation est sélectionné dans la liste View au dessus de l'espace Hierarchy.
3. Sélectionnez l'item top level Test dans l'arborescence du projet.
4. Double-cliquez sur l'item Implementing Design.
5. Consultez les messages dans la Console et assurez-vous que l'implémentation du projet s'est bien déroulé.

V.6.5 Création du fichier de programmation

Vous allez générer le fichier de programmation BIT pour configurer la carte Spartan 6 LX9.

1. Sélectionnez **Test** dans l'arborescence du projet.

2. Double-cliquez sur l'item **Generate Programming File**. ISE lance le programme intégré Bit Gen pour générer le fichier BIT qui servira à programmer le FPGA.

3. Consultez les messages dans la Console et assurez-vous que la création du fichier s'est bien Déroulée, un crochet devrait s'afficher à côté de **Generate Programming File**

V.6.6 Programmez le FPGA en mode JTAG

Vous allez utiliser le logiciel ADEPT pour programmer le FPGA en mode JTAG avec le fichier BIT généré à l'étape précédente. Connectez le câble USB entre votre poste de travail et la prise USB de la carte Spartan 6 LX9 .

Procédez comme suit pour programmer le FPGA: Lancer le logiciel ADEPT, retrouver le fichier Test.bit dans votre espace de travail et ensuite cliquer sur Program. Dans l'exemple ci-dessus, vous avez vu comment créer un nouveau projet, comment ajouter la ressource VHDL, comment écrire votre architecture, comment ajouter UCF, comment synthétiser, implémenter, générer le fichier BIT et programmer le FPGA

V.7 Conclusion

Nous devrons maintenant d'être capable de :

- ✓ Décrire, un circuit numérique (combinatoire et séquentiel) à l'aide du langage VHDL ;
- ✓ D'exploiter le fichier UCF de la carte Spatran 6 LX9 et de générer le fichier BIT ;
- ✓ Expliquer les différentes étapes de la programmation de la carte FPGA Spartan 6LX9.

Références :

Serti, Abdelghani, "Conception d'un logiciel d'optimisation et de synthèse de réseaux logiques complexes et de leur réalisation à l'aide de PLD", (1993).

Volnei A. Pedroni, “Circuit Design with VHDL”, MIT Press, 2004

Jacques Weber , Sébastien Moutault, Maurice Meaudre, “Le langage VHDL : du langage au circuit, du circuit au langage“, Dunod, 2007

Christian Tavernier, “Circuits logiques programmables“, Dunod 1992

Chang KC, “Digital systems design with VHDL and synthesis”, IEEE computer society press, 1999 May 1.

Avnet Electronics Marketing , “Xilinx® Spartan®-6 FPGA LX9 MicroBoard User Guide“, AVNET, 2011.

Chu, Pong P, “RTL hardware design using VHDL: coding for efficiency, portability, and scalability”, John Wiley & Sons, 2006.

La Meres, Brock J, “Introduction to logic circuits & logic design with VHDL”, Springer, 2019.

V.Tourtchine et M.Izouin, “Modelisation des systemes numeriques dans l'environnement Xilinx ISE 13.2 (vhdl) ”, université M Bougara de Boumerdes, 2012.

ABBAD, Houari, "Méthodologie de développement et d’implantation sur puce FPGA d’algorithme de commande", thèse de doctorat., Université Mohamed Khider-Biskra, 2016.

Annexes I : Travaux dirigés

Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Série TDN°1

Exo N°1 :

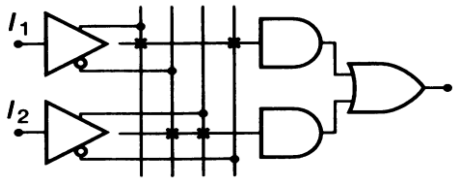
A- On souhaite réaliser une fonction :

1- Déterminer la fonction réalisée par ce PAL ?

B- Soit la fonction logique suivante :

$$S = \overline{A}\overline{B} + \overline{B}C + AC$$

2- Donner le circuit PAL de cette fonction ?



Exo N°2 :

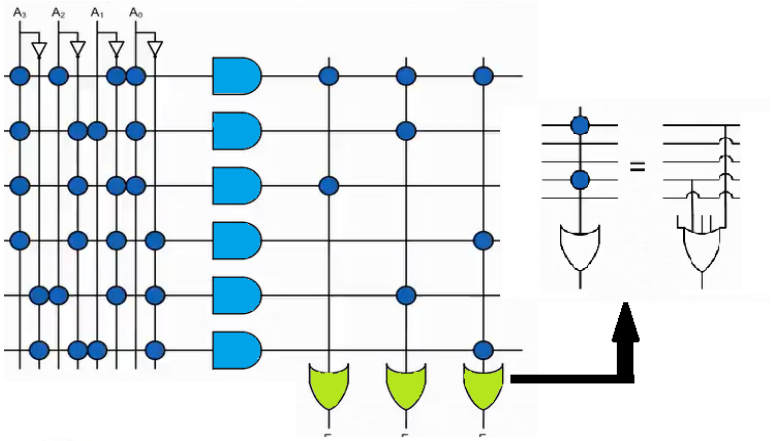
On veut implémenter dans un PLA de 4-3-6, la fonction logique suivante :

$$F = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + ABCD$$

1- Choisir les portes d'entrées et de sortie.
2- Etablir le schéma de circuit PLA qui correspond à cette équation

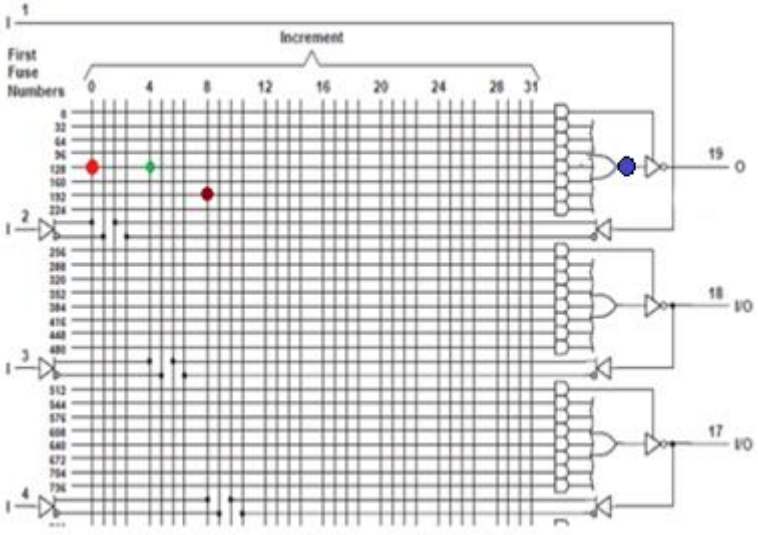
Exo N°3: Soit le schéma du PLA suivant :

1- Déterminer les entrées et les sorties.
2- Etablir les équations de sortie de ce PLA



Exo N°4: A partir des trois points de connexion du schéma du PAL suivant :

1- Donner les portes d'entrées et de sortie ?
2- Déduire l'équation de sortie de ce circuit PAL ?



A partir des équations suivantes et en se basant sur le circuit PAL précédant, établir le schéma de circuit PAL correspond :

$$\overline{F} = AB + C$$

$$\overline{H} = AB + AC + B\overline{C}$$

Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Série TDN°2

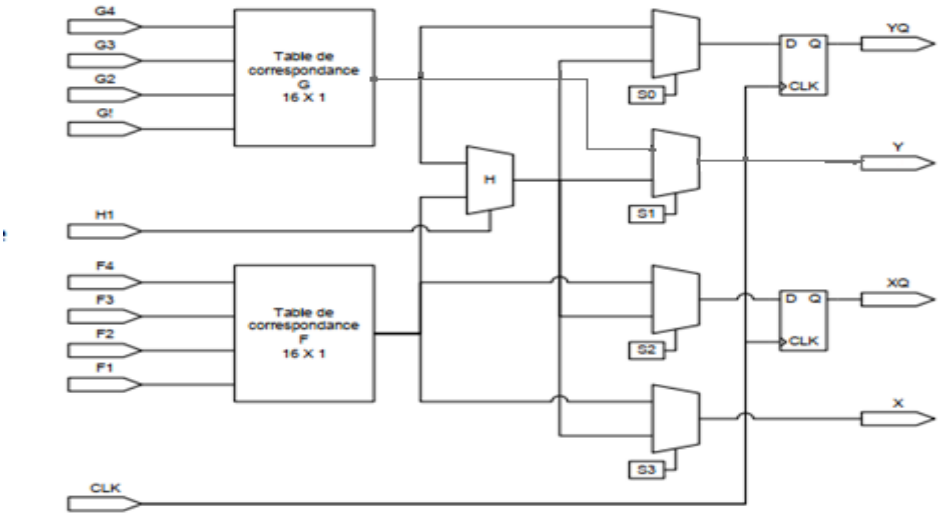
Exo N°1 :

Soit la fonction logique suivante :

$$F = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{B}\bar{C}\bar{D}$$

1- Donner la table de vérité qui correspond à cette fonction ?

En se basant sur le modèle simplifié d'une tranche du Virtex 2 Pro suivant :



2- Choisir les ports d'entrée et de sortie correspondant à la fonction logique ?
3- Etablir les connexions entre les ports d'entrées et sorties de la fonction logique ?
4- Déduire le contenu de la table de correspondance de cette carte FPGA Virtex 2 Pro ?

Exo N°2 :

Soient les fonctions logiques suivantes :

$$F = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}BCD + ABCD$$

$$H = NM + \bar{N}MK + \bar{N}\bar{M}K$$

En se basant sur le **modèle simplifié d'une tranche du Virtex 2 Pro donné dans l'exercice N°1**

1- Donner la table de vérité qui correspond à chaque fonction ?
2- Choisir les ports d'entrée et de sorties correspondant à chaque fonction ?
3- Etablir les connexions entre les ports d'entrée et sortie de chaque fonction ?

Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Série TDN°3

Exo N°1 :

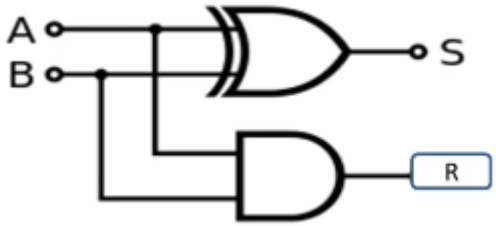
Soit la fonction logique suivante :

$$S = \overline{A}BC + A\overline{B}C + AB\overline{C}$$

1- Donner la table de vérité
2- Donner le schéma logique de cette fonction en utilisant les portes (AND, NOT et OR)?
3- Ecrire le programme VHDL de cette fonction avec description comportemental (**if-then-else)**?

Exo N°2 :

Soit le schéma logique suivant :



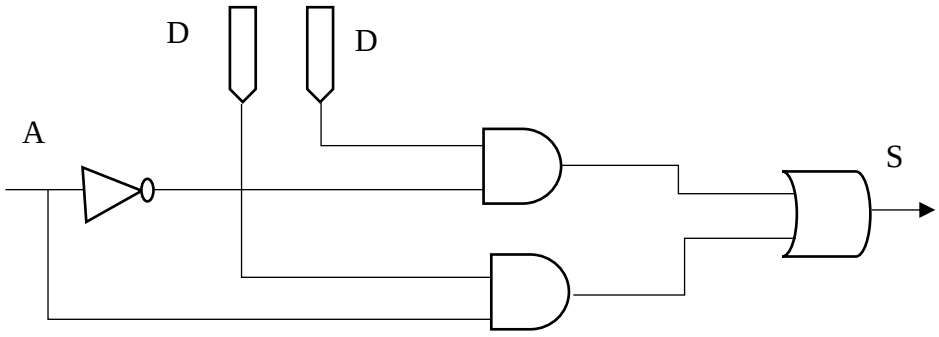
1- Donner la description VHDL par le flot de données et deux portes Xor et And séparément ?
2- Donner la description structurelle VHDL du schéma complet (**Instanciation par nom**)?

Exo N°3 :

Ecrire un programme VHDL décrivant le comportement d'un circuit combinatoire du code binaire 3bits au code Gray, en utilisant une instruction concurrente de sélection **With-Select-When**?

Exo N°4 :

Soit le schéma logique suivant :



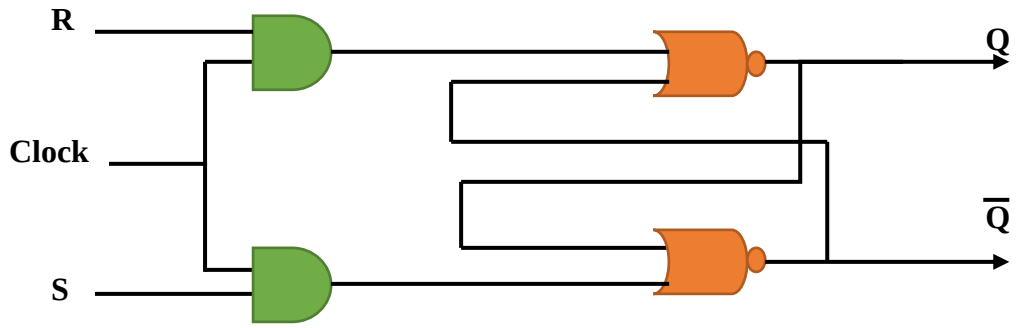
1- Donner l'équation logique de la sortie S ?
2- Donner la table de vérité pour la sortie ?
3- Donner le programme VHDL qui correspond avec la structure comportementale (**if**) et par Flot de données ?

Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Série TDN°4

Exo N°1 :

Soit le schéma de la bascule SR suivant :



1- Donner la table de fonctionnement ou de vérité de la bascule RS ?
2- Donner la description en code VHDL de la bascule RS ?

Exo N°2 :

On veut réaliser un compteur à 3 bits avec une remise à zéro, donner la description VHDL qui permet d'avoir un compteur avec une remise à Zéro synchrone en utilisant les signaux internes ?

Exo N°3 :

On veut implémenter l'équation logique suivante sur la carte FPGA SPARTAN LX09 :

$$S = A.B + \bar{A}.B$$

1- Donner la table de vérité de l'équation ?
2- Donner le programme VHDL qui correspond avec le Flot de données ?
3- En se basant sur l'extrait du fichier Avt_S6LX9_MicroBoard_UCF_110804.ucf , configurer les entrées et les sortie sur ce fichier ?

```
# User DIP Switch x4
#   Internal pull-down required since external resistor is not populated
##########################################################################
#NET "GPIO_DIP1"        LOC = B3  | IOSTANDARD = LVCMOS33 | PULLDOWN
#NET "GPIO_DIP2"        LOC = A3  | IOSTANDARD = LVCMOS33 | PULLDOWN;
#NET "GPIO_DIP3"        LOC = B4  | IOSTANDARD = LVCMOS33 | PULLDOWN;
#NET "GPIO_DIP"'        LOC = A4  | IOSTANDARD = LVCMOS33 | PULLDOWN;
# User LEDs
##########################################################################
#NET "GPIO_LED1" LOC = P4  | IOSTANDARD = LVCMOS18;            #
#NET "GPIO_LED1"LOC = L6  | IOSTANDARD = LVCMOS18;            #
#NET "GPIO_LED1" LOC = F5  | IOSTANDARD = LVCMOS18;            #
#NET "GPIO_LED1" LOC = C2  | IOSTANDARD = LVCMOS18;            #
```

Annexes II : Solution des travaux dirigés

Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Corrigé Série TDN°1

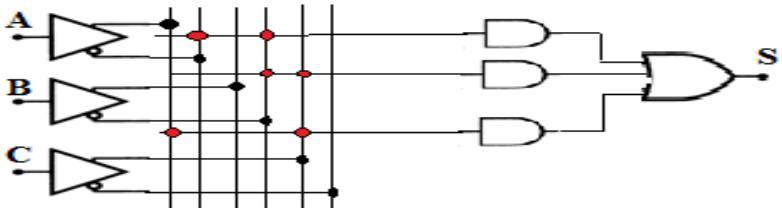
Exo N°1 :

1- La fonction réalisée par ce PAL

$$S = I_1\overline{I_2} + \overline{I_1}I_2$$

2- Le circuit PAL de cette fonction

$$S = \bar{A}\bar{B} + \bar{B}C + A$$

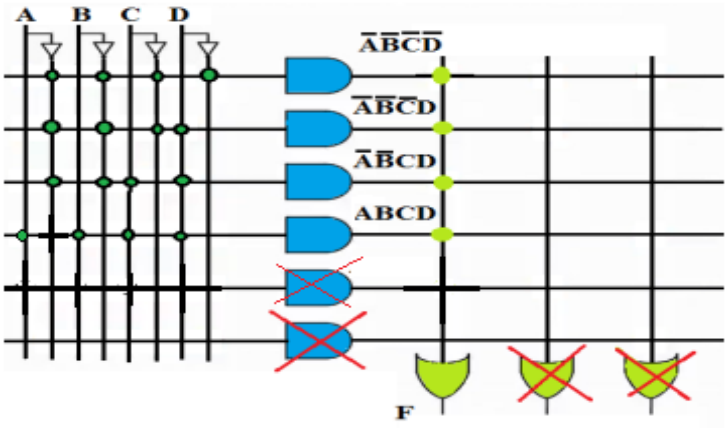


Exo N°2 :

On veut implémenter dans un PLA de 4-3-6, la fonction logique suivante :

$$\boldsymbol{F = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + ABCD}$$

1- Les portes d'entrées et de sortie. : **F : sorite, A,B,C,D : les entrées**
2- Le schéma de circuit PLA qui correspond à cette équation



Exo N°3: Soit le schéma du PLA suivant :

1- Les entrées et les sorties.
Entrées : A0, A1, A2, A3
Sorties : F0, F1,F2

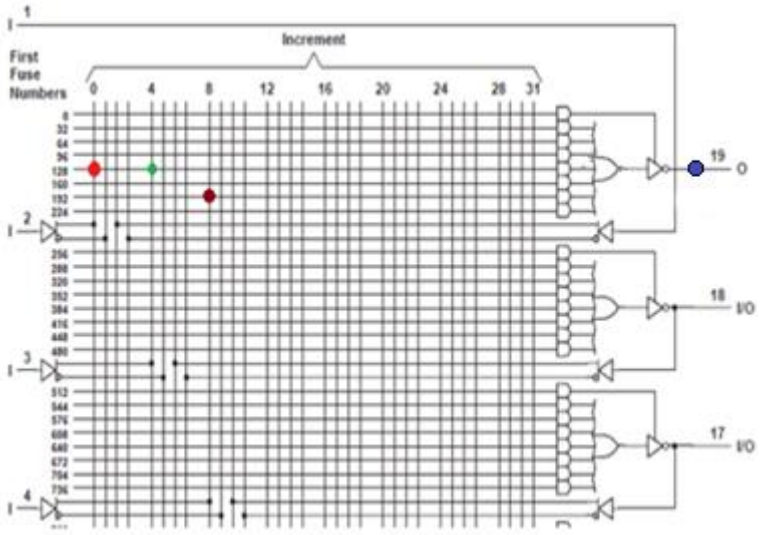
2- Les équations de sortie de ce PLA

$$\boldsymbol{F_0 = A_3A_2\overline{A_1}A_0 + A_3\overline{A_2A_1A_0} + \overline{A_3A_2}\ A_1\overline{A_0}}$$
$$\boldsymbol{F_1 = A_3A_2\overline{A_1}A_0 + A_3\overline{A_2}A_1A_0 + \overline{A_3}\ A_2\overline{A_1}A_0}$$
$$\boldsymbol{F_2 = A_3A_2\overline{A_1}A_0 + A_3\overline{A_2}A_1A_0}$$

Exo N°4: A partir des trois points de connexion du schéma du PAL suivant :

1- Les portes d'entrées et de sortie ?
   Les entrées : I2=A , I3=B, I4=C
   La Sortie : F
2- L' équation de sortie de ce circuit PAL
   $\boldsymbol{F = AB + C}$
3- Le schéma de circuit PAL correspond :

$\boldsymbol{\overline{F} = AB + C}$



Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Corrigé Série TDN°2

Exo N°1 :

Soit la fonction logique suivante :

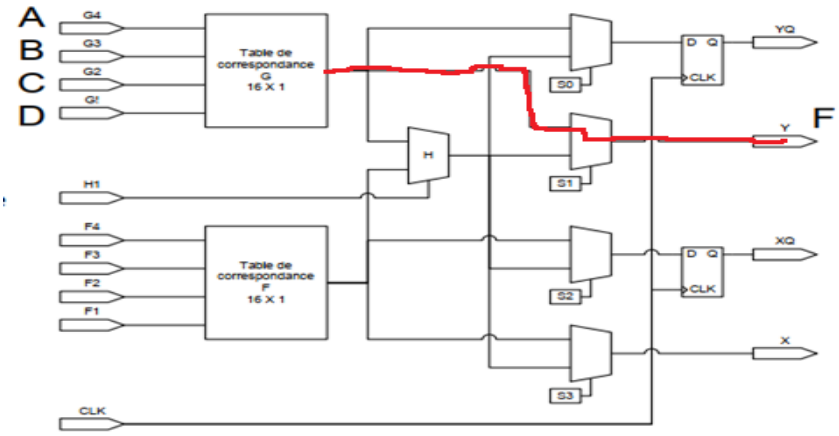
$$F = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + AB\bar{C}\bar{D}$$

1- La table de vérité qui correspond à cette fonction

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

2- Les ports d'entrée et de sortie correspondant à la fonction logique
Les entrées : G4=A, G3=B, G2=C, G1=D
Les sortie : Y=F

3- Les connexions entre les ports d'entrées et sorties de la fonction logique ?



Exo N°2 :

Soient les fonctions logiques suivantes :

$$F = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}BCD + ABCD$$

$$H = NM + \bar{N}MK + \bar{N}\bar{M}K$$

En se basant sur le **modèle simplifié d'une tranche du Virtex 2 Pro donné dans l'exercice N°1**

1- Tables de vérité qui correspond aux fonctions F et H

La fonction F

A	B	C	D	**F**
0	0	0	0	**1**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**0**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**1**

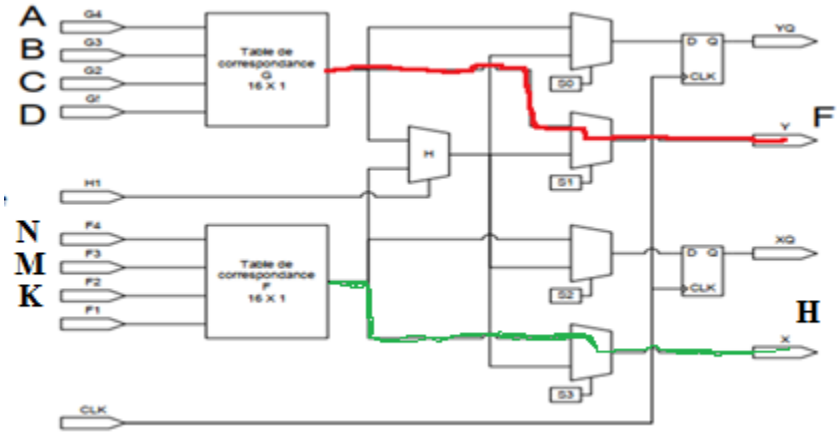
La fonction H

N	M	K	**H**
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

2- Les ports d'entrée et de sorties correspondant à chaque fonction :

La fonction F
Les entrées : G4=A, G3=B, G2=C, G1=D
La sortie : Y=F
La fonction H
Les entrées : F4=N, F3=M, F2=K
La sortie : X=H

3- Les connexions entre les ports d'entrée et sortie de chaque fonction



Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

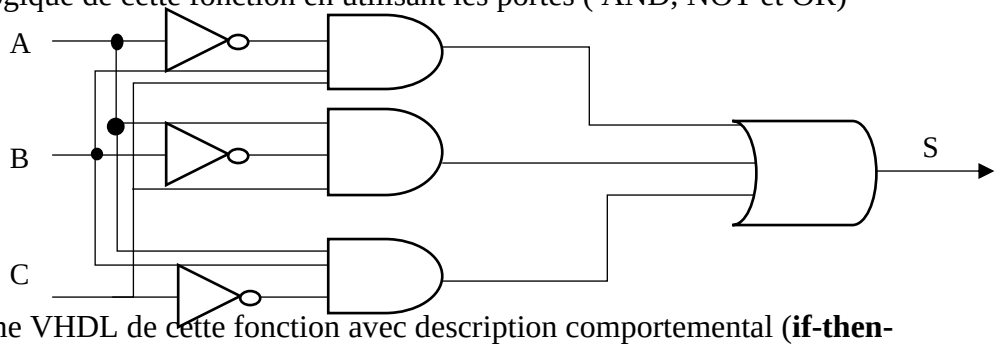
Corrigé Série TDN°3

Exo N°1 :

1- La table de vérité

A	B	C	**S**
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	**1**	**1**	**1**
1	0	0	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	1	1	**0**

2- Le schéma logique de cette fonction en utilisant les portes (AND, NOT et OR)



3- Le programme VHDL de cette fonction avec description comportemental (**if-then-else)**

```
Library ieee;
Use ieee.std_logic_1164.all;
entity Exo_1 is
port ( abc: in std_logic_vector(2 downto 0);
        S: out std_logic);
end Exo_1;
architecture Mon_Exo_1 of Exo_1 is
begin
  Process (abc)
       begin
          if  abc= "011"  then S <= "1" ;
             else if abc= "101"  then S <= "1" ;
             else if abc= "110"  then S <= "1" ;
             else S<="0";
           end if;
    end process;
end Mon_Exo_1;
```

Exo N°2 :

1- La description VHDL par le flot de données pour les deux portes Xor et And séparément

Xor

```
Library ieee;
Use ieee.std_logic_1164.all;
entity Exo_2 is
port ( I1,I2 : in std_logic;
      M: out std_logic);
end Exo_2;
architecture Mon_Exo_2 of
Exo_2  is
begin
  M <=I1 xor I2;
end Mon_Exo_2;
```

AND

```
Library ieee;
Use ieee.std_logic_1164.all;
entity Exo_22 is
port ( E1,E2 in std_logic;
      N: out std_logic);
end Exo_22;
architecture Mon_Exo_22 of
Exo_22  is
begin
  N <=E1 and E2;
end Mon_Exo_22;
```

La description structurelle VHDL du schéma complet (**Instanciation par nom**

```
Library ieee;
Use ieee.std_logic_1164.all;
entity Exo_222 is
port ( A,B: in std_logic;
        R,S: out std_logic);
end Exo_22;
architecture Mon_Exo_222 of Exo_222 is
component Xor_2 is
port(I1 , I2: in Std_logic;
      M: out  Std_logic);
        End component;
component And_2 is
port(E1 , E2: in Std_logic;
      N: out  Std_logic);
        End component;
 begin
Instance_1: Xor_2 port map (I1 <=A, I2 <=B,M<=S);
Instance_2: AND_2 port map (E1 <=A, E2 <=B,N<=R);

end Mon_Exo_222;
```

ExoN°3

```
architecture Behavioral of conv_gray is
begin
with x select
y<="000" when "000" ,
"001" when "001" ,
"011" when "010" ,
"010" when "011" ,
"110" when "100" ,
"111" when "101" ,
"101" when "110" ,
"100" when "111" ,
(others =>'0') when others ;
end Behavioral;
```

Exo N°4 :

1- L'équation logique de la sortie S

$S=\overline{A}*D0 + A*D1$

2- La table de vérité pour la sortie

A	S
0	D0
1	D1

3- Le programme VHDL correspond avec la structure comportementale et par Flot de données

Comportementale if

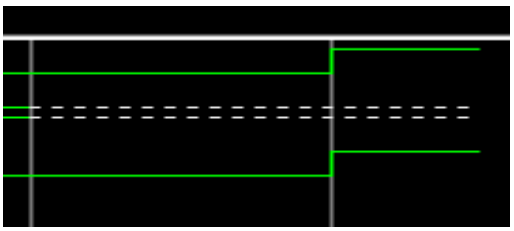
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Exo_4 is
            port ( A, D0, D1 : in std_logic;
            S: out std_logic);
end Exo_4;
architecture Mon_Exo_4 of Exo_4  is
begin
process (A, D0,D1)
begin
    if (A= '1') then
                                    S<= D1;
                                    else
                                    S <= D0;
End if;
End process;
End Mon_Exo_4;
```

Flot de données

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Exo_44 is
            port ( A, B, D0, D1 : in std_logic;
            S: out std_logic);
end Exo_44;
architecture Mon_Exo_44 of Exo_44  is
begin
    S<= (not A AND D0) OR (A AND D1);
End Mon_Exo_44;
```

4- Les courbes de simulation



D0=L	Niveau logique 0, forçage faible
D1=H	Niveau logique 1, forçage faible

Université Akli Mohand Oulhadj-Bouira
Faculté des Sciences et des Sciences Appliquées
Département de Génie Electrique
Module : Electronique Numériques Avancées : VHDL, FPGA

Corrigé Série TDN°4

Exo N°1 :

1- La table de fonctionnement ou de vérité de la bascule RS

H	R	S	Q	$\overline{Q}$
0	0	0	**Q**	$\overline{Q}$
0	0	1	**Q**	$\overline{Q}$
0	1	0	**Q**	$\overline{Q}$
0	1	1	**Q**	$\overline{Q}$
1	0	0	**Q**	$\overline{Q}$
1	0	1	**1**	**0**
1	1	0	**0**	**1**
1	1	1	**Etat indéterminé**	**Etat indéterminé**

2- La description en code VHDL de la bascule RS

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity RSH is
5     port(R, S, H : in std_logic;
6          Q, Q_bar : out std_logic);
7  end RSH;
8
9  architecture basc of RSH is
10       signal SIG : std_logic := '0';
11       signal RS : std_logic_vector (1 downto 0);
12 begin
13    RS(1) <= R;
14    RS(0) <= S;
15    process(RS, SIG, H)
16    begin
17       if (H = '1') then
18          case RS is
19             when "00" => SIG <= SIG;
20             when "01" => SIG <= '1';
21             when "10" => SIG <= '0';
22             when others => SIG <= '-';
23          end case;
24       else
25          SIG <= SIG;
26       end if;
27       Q <= SIG;
28       Q_bar <= not SIG;
29    end process;
30 end basc;
```

Exo N°2 :

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity CMP3BITS is
PORT (
CLOCK : in std_logic;

Q : out std_logic_vector(2 downto 0));
end CMP3BITS;
architecture DESCRIPTION of CMP3BITS is
signal CMP: std_logic_vector (2 downto 0):="000";
begin
process (CLOCK)
begin

if (CLOCK ='1' and CLOCK'event) then
CMP <= CMP + 1;
end if;
end process;
Q <= CMP;
end DESCRIPTION;
```

Exo N°3 :

On veut implémenter l'équation logique suivante sur la carte FPGA SPARTAN LX09 :

$$S = A.B + \bar{A}.B$$

4- La table de vérité de l'équation

A	**B**	**S**
0	0	0
0	1	**1**
1	0	0
1	1	**1**

5- Le programme VHDL qui correspond avec le Flot de données

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Exo_3 is
            port ( A, B: in std_logic;
            S: out std_logic);
end Exo_3;
architecture Mon_Exo_3 of Exo_3  is
begin
    S<= (A AND B) OR (NOT A AND B);
End Mon_Exo_3;
```

6- Configuration les entrées et les sortie sur ce fichier ?

User DIP Switch x4

```
#   Internal pull-down required since external resistor is not populated
##############################################################################
NET "A"                    LOC = B3  | IOSTANDARD = LVCMOS33 | PULLDOWN
NET "B"                    LOC = A3  | IOSTANDARD = LVCMOS33 | PULLDOWN;
#NET "GPIO_DIP3"       LOC = B4  | IOSTANDARD = LVCMOS33 | PULLDOWN;
#NET "GPIO_DIP'"       LOC = A4  | IOSTANDARD = LVCMOS33 | PULLDOWN;
# User LEDs
##############################################################################
NET "S                  " LOC = P4  | IOSTANDARD = LVCMOS18;          #
#NET "GPIO_LED1"LOC = L6  | IOSTANDARD = LVCMOS18;          #
#NET "GPIO_LED1" LOC = F5  | IOSTANDARD = LVCMOS18;          #
#NET "GPIO_LED1" LOC = C2  | IOSTANDARD = LVCMOS18;          #
```