UNIVERSITÉ MOULOUD MAMMERI TIZI-OUZOU

Faculté de Génie Électrique et D'informatique

Département d 'informatique

Mémoire de fin d'études

En vue de l'obtention du diplôme de MASTER II en Informatique

Option : Système Informatique.

Thème

Etude et Réalisation d'un microprocesseur sur une cible reconfigurableFPGA.

Proposé et dirigé par :

Mr. KARA.

Réalisé par :

Mouhoubi Hassina.

KaricheRoza.

Promotion 2013-2014

Remerciements

Dédicaces

Je dédie ce modeste travail

À Toute ma famille, mon père, ma mère, mon mari Mustapha, ma fille Anais ainsi que ma belle famille.

À tous mes ami(e)s.

À tous les membres de laboratoire L2CSP.

À toutes les personnes qui me sont chères.

M. HASSINA

Dédicaces

Je dédie ce modeste travail :

A ma mère.

A mon père.

Aucun hommage ne peut être à la hauteur de l'amour

Et de l'affectation dont ils ne cessent de nous combler

Qu'ils trouvent dans ce travail un modeste témoignage

De mon profonde amour.

Que bien leurs procures bonnes santé et longue vie.

A mes sœurs.

A mes frères.

A tous mes amis.

K. ROZA

Liste des acronymes et des abréviations

ASIC: Application **S**pecific **I**ntegrated **C**ircuit.

CPLD ou EPLD: Erasable**P**rogrammable **L**ogic**D**evice.

CLB:Configurable **L**ogic **B**loc.

CMOS:Complementaire**M**etal **O**xyde**S**emiconductor.

DL:Digital **D**elay **L**ocked **L**oop.

EEP-ROM:Electrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory.

FPGA: Field **P**rogrammable **G**ate **A**rray.

FPLS:Field **P**rogrammable **L**ogic **S**equencer.

GAL:Generic **A**rray **l**ogic.

ISP:In **S**ystem **P**rogrammable.

IOB:Input **O**utput **B**loc.

LUT: Look **U**p **T**ables.

MUX: Multiplexeur.

NMOS :N-type-**M**etal-**O**xyde-**S**emiconductor.

OLMC:Output **L**ogic **M**acro **C**ell.

PAL:Programmable **A**rray **L**ogic.

PLD:Programmable **L**ogic **D**evice.

PLL:Phase-**L**ocked-**L**oop.

PMOS: P-type-**M**etal-**O**xyde-**S**emiconductor.

TTL: Transistor **T**ransistor **L**ogic.

VLSI: Very **L**arge **S**cale **I**ntegration.

Liste des figures

Liste des Figures

Sommaire

Introduction Générale

Chapitre I :Généralités

Chapitre II : Conception du microprocesseur

Chapitre III : Réalisation

Introduction Générale

Introduction générale

La réalisation de processeurs est une tâche passionnante qui, à première vue, peut paraître très compliquée puisqu'elle peut nécessiter des connaissances dans des domaines très variés tels que: la physique, l'informatique et la micro-électronique.
Comme chacun le sait, le processeur est un circuit numérique dédié au traitement d'instructions. Son unique rôle est d'exécuter des instructions, à une certaine cadence. L'ensemble des instructions que doit exécuter le processeur constitue le programme de l'utilisateur. Ce dernier est stocké dans une mémoire, couplée au processeur.

Notre travail consiste à réaliser un simple processeur (Nanoprocesseur) étape par étape avec un jeu d'instructions limites à quelques opérationsarithmétiques : Addition,soustraction et logique : le AND et OR ; de saut inconditionnel : jump; et D'accès à la mémoire : chargement d'une donnée et stockage d'un résultat), dont chaque instruction et codé sur 1 octet, et doté d'une mémoire programmable adressable.
La descriptionmatériel du processeur a été réalisée avec le langage de description de matériel VHDL et sa mise en œuvre a été effectuée sur un circuit logique programmable de type FPGA. Pour ce faire, en a utilisé la carte DE2 d'Altéra et son environnement de programmation Quartus II.
Ce mémoire est organisé en trois chapitres :
Le premier chapitre est une présentation des circuits logiques programmables.
Dans le deuxième chapitre, nous avons, dans une première partie, étudié la conception générale d'un processeur. En deuxième partie nous avons donnéles différentes étapes à suivre pour réaliser notre processeur.
Dans le dernier chapitre, nous avons réalisé en pratique notre système présenté enchapitre II et pour cela nous allons présenter L'environnement de développement Quartus II et le langage de description matériel vhdl.En suite on a simulé notre système sur une carte FPGA.
Et enfin, nous terminons par une conclusion générale.

Chapitre I

Introduction

Il y a quelques années la réalisation d'un montage en électronique numérique impliquait l'utilisation d'un nombre important de circuits intégrés logiques. Ceci avait pour conséquences un prix de revient élevé, une mise en œuvre complexe et un circuit imprimé de taille importante.

Le développement des mémoires utilisées en informatique fut à l'origine des premiers circuits logiques programmables (PLD : Programmable LogicDevice). 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 outil de développement.

I.1 Les Circuits Logiques Programmables [4]:

De nombreuses familles de circuits programmables et reprogrammables sont apparues depuis les années 70 avec des noms très divers suivant les constructeurs**. La figure I.1**donne une classification possible des circuits numériques en précisant où se situe chaque type de circuits dans cette classification.

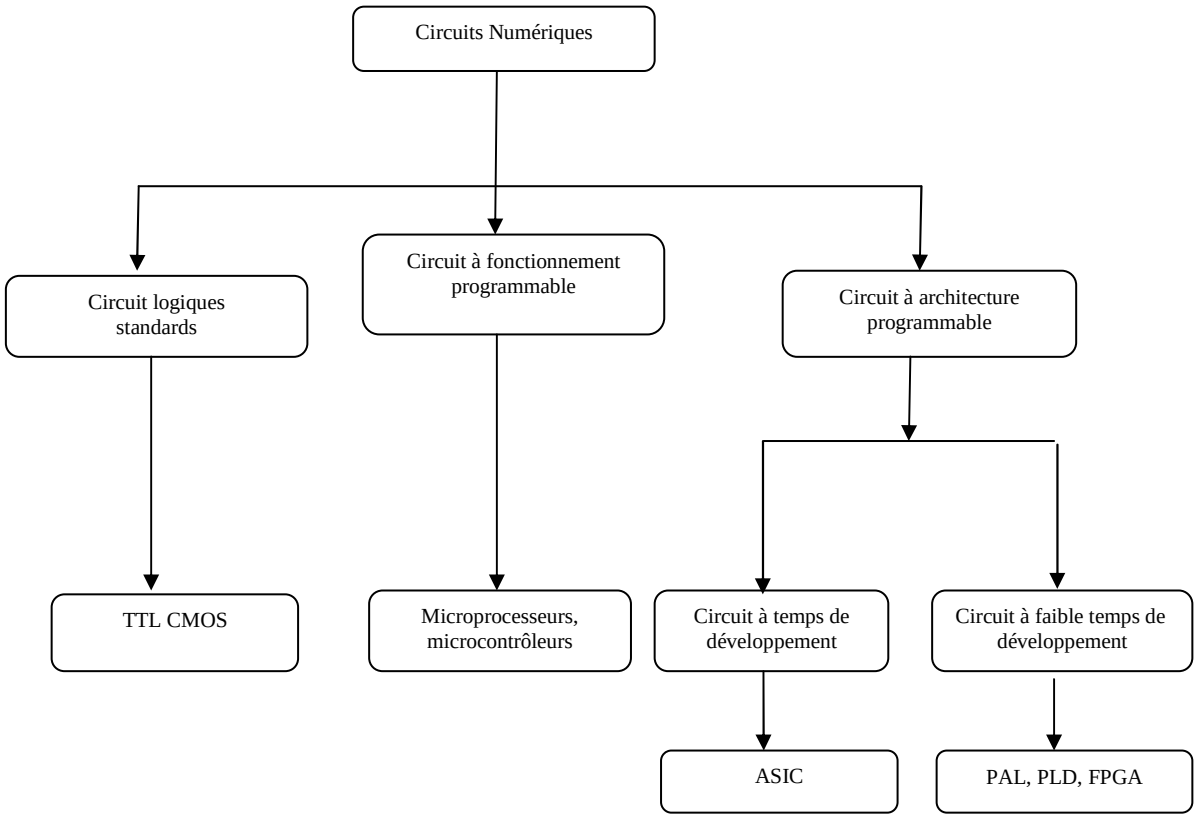

Figure I.1 : Classification des circuits numériques.

La technologie de la logique câblée permet de réaliser des circuits de faible complexité à un coût réduit, en interconnectant des circuits logiques classiques TTL (série 74xx) ou CMOS (série 4xxx). Lorsque la complexité est importante, on préfère utiliser des circuits logiques programmables : FPLSdu type PLD, FPGA ou CPLD. Ces technologies permettent de réaliser des circuits logiques en configurant des matrices de portes logiques et de bascules.

Les circuits logiques programmables et reprogrammables architecturalement sont classifiés en trois grandes familles les PLD, CPLD et FPGA. Le diagramme suivant illustre les différents types suivant la technologie utilisée.

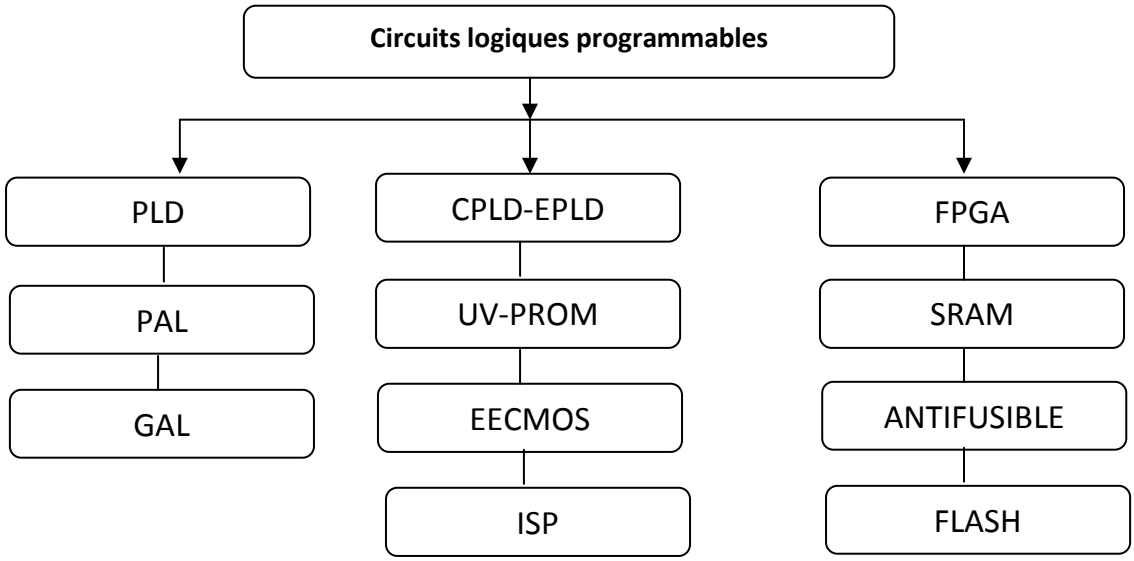

Figure I.2 : le diagramme de différents types de circuits logiques programmables.

1) **Les PLD (p**rogrammable **l**ogic**D**evice**)** : famille des circuits programmables qui comprend les PAL et les GAL.

- PAL (programmable ArrayLogic) : circuits logiques programmables dans lesquels seules les fonctions ET sont programmables, les fonctions OU ne le sont pas.
- GAL (GenericArraylogic) : circuits logiques reprogrammables à technologieCMOS.

- **Structure de base d'un PLD :**

La plupart des circuits PLDs suivent la structure suivante **(Figure I.3)**:

- Un bloc d'entrée qui permet de fournir au bloc combinatoire l'état de chaque entréeet son complément.
- Un ensemble d'opérateurs « ET » sur lesquels viennent se connecter, les variablesd'entrées et leurs compléments.
- Un ensemble d'opérateurs « OU »sur lequel les sorties des opérateurs « ET » sontconnectées.
- Un bloc de sortie.

- Un bloc d'entrée-sortie, qui comporte une porte à 3 états et une broche d'entrée-sortie.

Le deuxième et le troisième ensemble forment chacun ce qu'on appelle une matrice. Lesinterconnexions de ces matrices doivent être programmables. Ceci est réalisé par desfusibles qui sont grillés lors de la programmation. Lorsqu'un PLD est vierge, toutes lesconnexions sont assurées.

Le bloc de sortie est souvent appelé macro-cellule que l'on nomme OLMC (abréviationanglaise de output Logic Macro Cell signifiant macro-cellule logique de sortie). Cette macrocellulecomporte :

- Une porte OU Exclusif, une bascule D.
- Des multiplexeurs qui permettent de définir différentes configurations et un dispositifde rebouclage sur la matrice ET.
- Des fusibles de configuration (dans les FPGAs on utilise plutôt des cellules decommande des points de connexion).

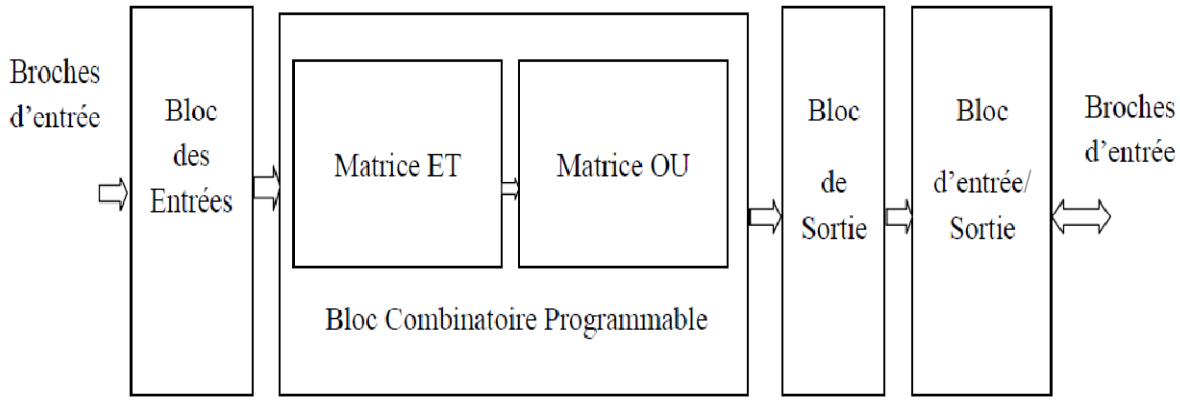

Figure I.3: La structure de base d'une PLDs.

2) **CPLD ou EPLD (E**rasable **P**rogrammable **L**ogic **D**evice**)** : circuits logiques reprogrammables, qui sont des circuits effaçables et reprogrammés électriquement.

- ISP (**I**n **S**ystem **P**rogrammable) : Circuits que l'on peut programmer même lorsqu'il est en place sur l'application.

Les CPLD sont des circuits composés de plusieurs PALs élémentaires reliés entre eux par une zone d'interconnexion (matrice d'interconnexion). Sa physionomie est généralement très structurée. Un certain nombre de macro-cellules de base sont regroupées pour former des blocs logiques. Grâce à leurs structures, ils peuvent atteindre des vitesses de fonctionnement élevées (plusieurs centaines de MHZ). Ces circuits ont une capacité en nombre de portes et en possibilités de configuration très supérieure à celle de PALs.

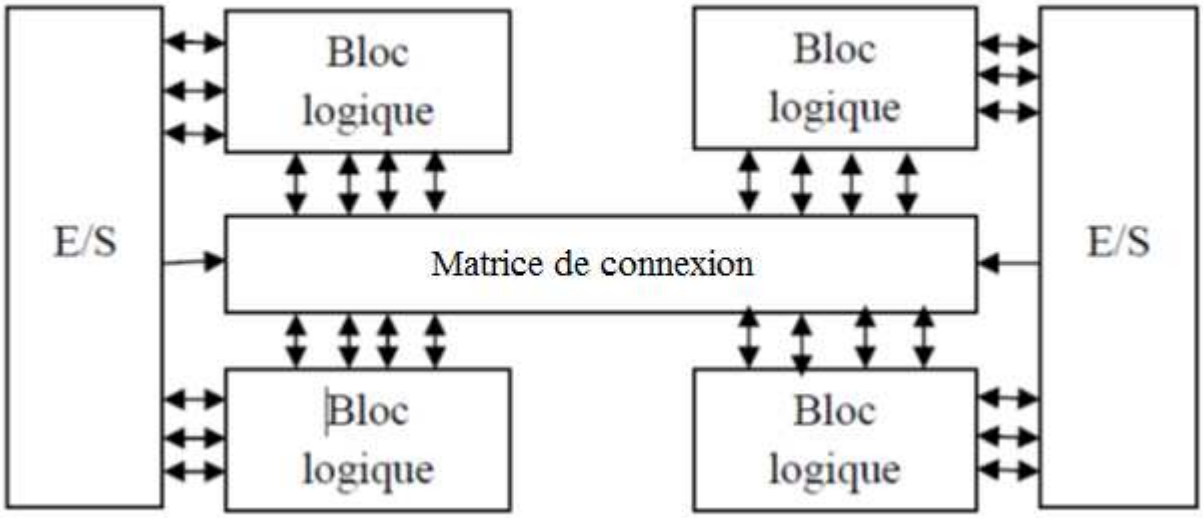

Figure I.4 : Physionomie d'un CPLD.

3-FPGA (Field **P**rogrammable **G**ate**A**rray**) :**Ces circuits sont une évolution des CPLD.

I.1.1 Présentation de FPGAs:

Les circuits FPGAs sont des composants entièrement reconfigurables ce qui permet deles reprogrammer à volonté. Ils se composent d'une matrice de blocs logiques programmables entourée de blocs d'entrées/sorties et l'ensemble est relié par un réseau d'interconnexion programmable. Ces circuits comparés aux CPLDs utilisent des modules logiques beaucoup plus réduits, mais beaucoup plus nombreux.

Les interconnexions de ces modules ne sont pas centralisées comme les CPLDs. Laphysionomie du réseau de routage est vue comme une multitude de segments métalliques(lignes) pouvant être reliés entre eux ou connectés en entrée ou en sortie des blocs logiques.

On distingue plusieurs types de lignes définies par leur longueur relative, on trouve :

- Les interconnexions à usage général qui sont composées de segments verticaux ethorizontaux qui entourent chaque **bloc logique configurable** CLB et qui peuvent êtrereliés entre eux par une matrice de commutation.
- Les lignes directes fournissant des chemins entre les CLB adjacents et entre les CLBet les cellules d'entrée-sorties.
- Les lignes longues qui sont des lignes verticales et horizontales qui n'utilisent pas dematrice de commutation. Elles parcourent toutes les zones d'interconnexion. Ellessont utilisées pour véhiculer les signaux qui doivent parcourir de long trajet. Ceslignes conviennent pour véhiculer les signaux d'horloge.

L'ensemble de points de connexion est appelé PIP (abréviation anglaise de Programmable Interconnect Points) et chaque point de connexion peut être réalisé selon deux techniques définissant deux classes de FPGAs (FPGAs à SRAM et FPGAs à ANTIFUSIBLE).

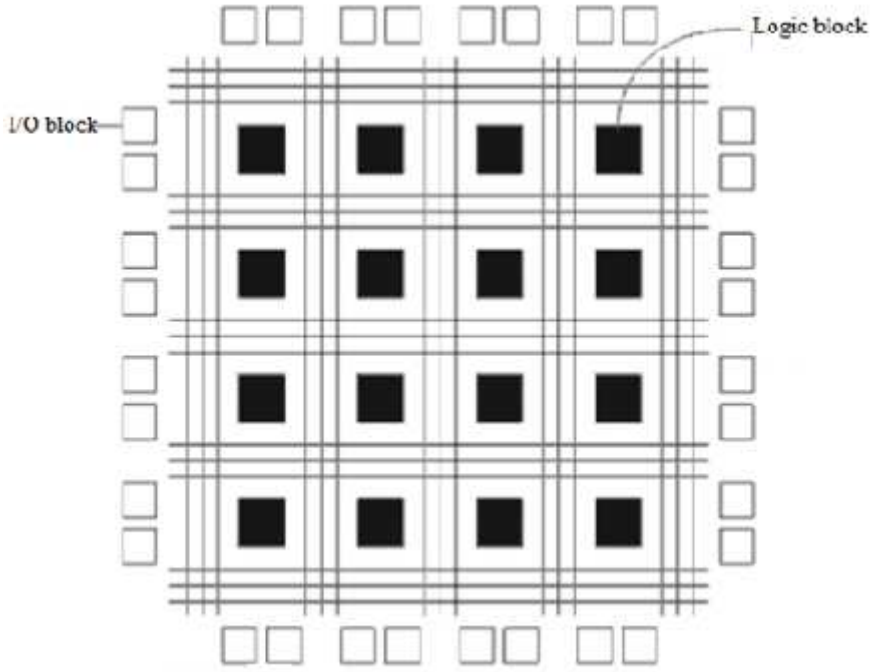

Figure I.5: physionomie d'un FPGA.

I.1.2 Classification des FPGAs :

Selon la technique de réalisation des PIP (point de connexion), on distingue trois technologies d'FPGA:

- Technologie de programmation par RAM.
- Technologie de programmation par EEPROM ou FLASH.
- Technologie de programmation par ANTI-FUSIBLE.

1. Technologie à base de RAM (XLINIX et ALTERA) :

Cette technologie permet d'avoir une reconfiguration rapide des FPGAs, les points de connexions sont des ensembles de transistors commandés.L'inconvénient majeur de cette technologie c'est qu'elle nécessite beaucoup de place et il est nécessaire de sauvegarder le design du FPGA dans une mémoire flash.

2. Technologie à base d'EEPROM ou FLASH (LATTICE et ACTEL) :

Cette technologie garde sa configuration, mais il existe un nombre limité de configurations plus lentes par rapport à SRAM.

3. Technologie à base d'ANTI-FUSIBLE (ACTEL) :

Les points de connexions sont de type ROM, c'est-à-dire que la modification du point n'est pas possible. Pour comprendre le mécanisme de connexion sans rentrer dans les détails des semi-conducteurs, on considère que le point de connexion est le point de rencontre de deux segments conducteurs ou lignes conductrices. Le nom ANTI-FUSIBLE vient du fait que l'état initial du fusible ou la couche isolante est présent et il n'y pas de contact pour l'établir, il faut détruire le fusible ce qui est contradictoire au fonctionnement habituel d'un fusible. Des composants moins génériques, mais plus petits et plus rapides ont été développés.

I.1.3 Etude de la structure générale d'un circuit FPGA [1]:

I.1.3.1 Les cellules logiques de base :

On distingue deux principaux types de cellules de base qui sont :

- **Les cellules 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 ou en entrée/sortie.

Le rôle principal des interfaces d'entrées/sorties est de transmettre et de recevoir des données du milieu extérieur, néanmoins l'interface d'entré/sorties peut être dotée d'options tels 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.

- Chez XILINX, les interfaces d'entrées/sorties sont nommées IOB pour Input Output Blocks. L'IOB est constitué de registres, de diviseurs de tensions, des résistances de rappel pull up et autres ressources spécifiques.
- Altera, les nomme IOE Input Output Element. L'IOE remplie toujours son rôle d'interface d'entrées/sorties, elle dispose d'une résistance de rappel pull-up et d'un temporisateur du signal.

- **Les macro-cellules**

Ces cellules logique sont appelé aussi par :

- Soit CLB (configurable logique bloc), dénomination adoptée par XILINX.
- Soit LC (cellule logique), le nom choisi par CYPRESS.
- Soit LE (élément logique), l'appellation d'ALTERA.

Ces macro-cellules sont plus nombreuses. La macro-cellule est constituée d'une partie **combinatoire** et une partie **séquentielle**.

- **La partie combinatoire** sur laquelle sont réalisées les fonctions de complexité moyenne car les constructeurs ont proposé pour chacun une ou plusieurs solutions desynthèse dont les principaux sont :

- La synthèse de fonctions à 4 ou 5 variables avec des portes classiques ET, OU et NON(not).
- La synthèse de fonction utilisant un multiplieur.
- La synthèse de fonction combinatoire à l'aide de mémoire vive RAM.

Dans ce dernier cas, on dit aussi réalisation de fonction logique par LUT (look-up table ou table d'observation).

- **La partie séquentielle** dans les FPGA sest représentée par des bascules, généralement de type D.

I.1.3.2 Réseau d'interconnexion :

Dans les FPGA à architecture symétrique on peut distinguer une composante d'interconnexion locale et une composante d'interconnexion globale. La composante locale a pour rôle d'interconnecter les éléments des blocs logiques en fonction de la configuration. Ladeuxième assure la circulation des données à l'échelle du FPGA entre les blocs logiques éloignés.

- Au niveau local, on trouve donc des interconnexions configurables au sein des blocs logiques dans et entre les cellules logiques élémentaires. Elles sont réalisées au moyen de multiplexeurs, de buffers trois états et de portes passantes (un simple NMOS ou bien une porte de transmission associant un NMOS et un PMOS en parallèle). On trouve également un groupe d'interconnexions vers les blocs logiques directement adjacents selon le même principe. Il s'agit d'interconnexions courtes présentant de faibles retards si on les compare aux interconnexions à longue distance.
- Au niveau global, on trouve un réseau d'interconnexions horizontales et verticales à l'échelle du circuit situé entre les blocs logiques montrés à la figure ci-dessous.

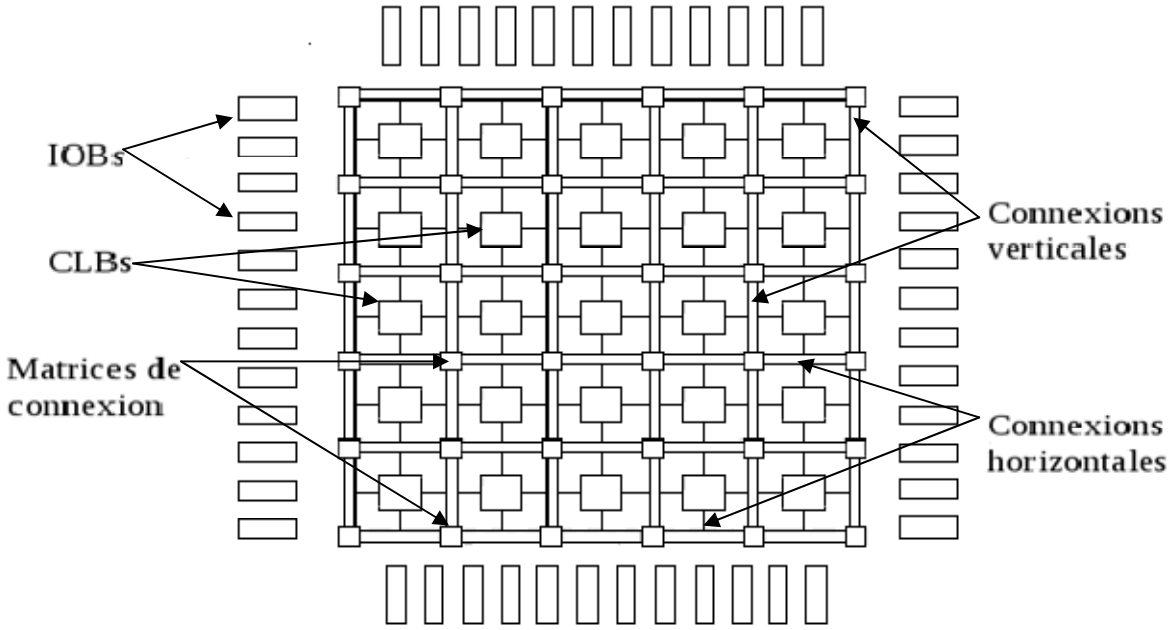

Figure I.6 : Concept architectural de base des FPGA.

On distingue trois types d'interconnexions :

- **Les interconnexions à usage général :**

Sont des lignes verticales et horizontales qui encadrent chaque CLB, auxquelles peut être reliée par une matrice de commutation montées à la **figure I.6**, le rôle de cette dernière est de raccorder les segments de communication entre eux selon diverses configurations, assure aussi la commutation des signaux d'une voie à une autre.

- **Les interconnexions directes :**

Ces interconnexions permettent d'établir des chemins entre les CLB adjacents et les cellules d'entrées/sorties avec un maximum d'efficacité en terme de vitesse et d'occupation de circuit.

- **Les longues lignes :**

Sont des longues lignes verticales et horizontales qui n'utilisent pas les matrices de commutations. Elles parcourent toutes la longueur et la largeur du circuit. Elles permettent aussi de transporter les signaux qui parcourent un long trajet. Elles égalisent les délais entre les signaux de façon à permettre un décalage minimum entre deux points distants de la ligne. Ces lignes conviennent pour transporter les signaux d'horloge.

L'échange des données entre les blocs logiques et ces interconnexions globales se fait par l'intermédiaire de Cellules d'Interconnexions Locales (CIL). L'aiguillage des données au sein du quadrillage d'interconnexions globales est assuré par des Cellules d'Interconnexions Globales (CIG) placées à chacune de ces intersections selon les chéma synoptique de la figure suivante :

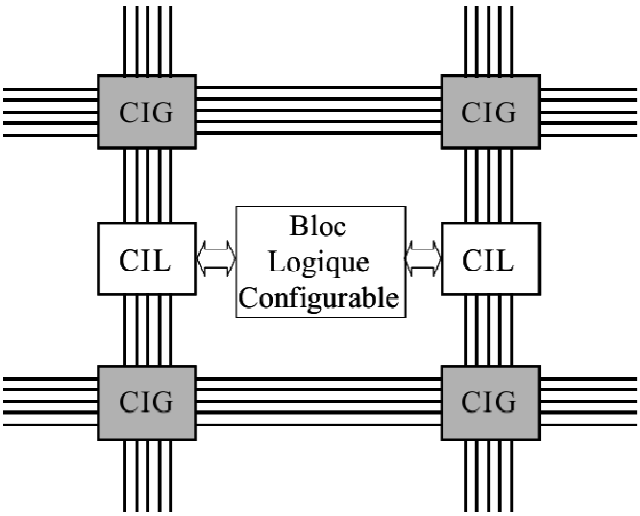

Figure I.7: Réseau d'interconnexions dans l'FPGA.

I.1.3.3Les éléments de mémorisation :

Actuellement, Les FPGAs sont utilisés pour des applications plus complexes et plus importantes qui demandent un espace de stockage plus important et une vitesse élevée.

L'intégration des blocs de mémoire directement dans l'architecture des FPGAs est vite devenue capitale. De cette façon, les temps d'accès à la mémoire sont diminué spuisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

I.1.3.4 Les éléments de contrôle des horloges :

L'horloge est un élément essentiel pour le bon fonctionnement d'un système électronique.

Les circuits FPGA sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routage spécialement adaptées au transport d'horloges sur des longues distances. Les circuits FPGA disposent des éléments d'asservissement des horloges (des PLL ou des DLL) afin d'avoir la même horloge dans tout le circuit (synchronisation des signaux).

I.1.3.4 Les éléments de routage :

Les éléments de routages sont les composants les plus importants dans les FPGA. En fait, ces éléments représentent la plus grosse partie du silicium consommée sur la puce du circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre eux les autres éléments via des matrices de connexions. Le routage de ces ressources est un point critique du développement d'une application sur un FPGA. Ces éléments sont très importants puisqu'ils vont déterminer la vitesse et la densité logique du système. Par exemple, les matrices de routage sont

physiquement réalisées grâce à destransistors pilotés de cellules SRAM, qui ont une résistance et une capacité, ce qui entraînel'existence de constantes de temps.

I.1.4 Critères du choix des FPGAs:

Les FPGAs sont développés récemment grâce aux progrès de la technologie VLSI (Very-Large-ScaleIntegration). L'application de ce type de circuits est une évolution des systèmes ouvrant des perspectivesde traitement numérique inaccessibles auparavant. La fin des années 80 a vu l'apparition despremiers circuits FPGAs qui sont des circuits intégrés que l'on peut configurer en un tempsrelativement court pour réaliser n'importe quelle fonction logique « câblée »à bas coût parune programmation de ses cellules logiques et ses interconnexions avec une restriction de nepas épuiser les ressources du FPGA. Typiquement, un circuit FPGA haute densité peutcontenir jusqu'à plusieurs millions d'éléments programmables.

Pour réussir une application à base d'FPGA et afin d'obtenir un système plusperformant, consommant un minimum de puissance, il est nécessaire de respecter un certainnombre de règles comme :

- Bien connaître les caractéristiques du FPGA cible pour assurer son adéquation avecles besoins du projet.
- Élaborer une méthodologie de conception.
- Maîtriser les outils d'implémentation et choisir des outils de synthèse de qualité.

I.1.5 Déférents domaines d'applications des FPGAs :

Les FPGAs ont fait révolutionner certains domaines de contrôle numérique et de plus en plus utilisé pour intégrer des architectures numériques complexes. Ils sont devenus les pluspopulaires en matière d'implantation et de prototypage des circuits numériques après leurapparition sur le marché en 1984. La clé maîtresse de leurs réussites est l'aspect deprogrammation de ces derniers. Leurs utilisations actuelles couvrent les deux domaines : Civilet Militaire.Parmi ces applications nous citons :

1-Informatique : périphériques spécialisés.

2-Machinerie industrielle : Contrôleur pour machines.

3-Télécommunication : Traitement d'image, filtrage.

4-Instrumentation : Équipement médical, Prototypage.

5-Transport : Contrôle d'avions et métros.

6-Militaire : Radar, communication protégée, la détection et la surveillance.

I.1.6Les familles architecturales des FPGAs :

Les familles des FPGAs peuvent se regrouper en deux groupes :

- **Les Circuits FPGAs à base de « LUT » (Look Up Tables) :** Ressemblent aux tables de vérité des fonctions logiques et réalisables par des mémoires de type SRAM.
- **Les circuits FPGAs à base de Multiplexeurs « MUX » :** Sont des macrocellules à trois entrées capablesde réaliser des fonctions logiques.

I.1.7 Avantages et Inconvénients des FPGAs :

Les avantages et les inconvénients des FPGAs sont multiples. On trouve :

Avantages	Inconvénients
1. Technologie facile à maîtriser. 2. Temps de développement réduit. 3. Reprogrammable. 4. Coût peu élevé. 5. Flexibilité et possibilité de réduire fortement les délais de développement et de commercialisation.	1. Performances non optimisées. 2. Temps de réponse long par rapportaux ASIC.

Figure I.8 : Avantages et Inconvénients des FPGA.

Remarque :

Dans notre travail nous utilisons la carte de développement**ALTERA DE2.(Voir Annexe A).**

Conclusion:

Dans ce chapitre, nous avons introduit dans une première partie les différentes familles des circuits logique programmables. Par la suite les différentes familles de FPGA, et leurs compositions.

Chapitre II

Introduction :

Un microprocesseur seul ne peut absolument rien faire, il doit obligatoirement être associé à d'autre élément pour pouvoir fonctionner, autrement dit il faut l'insérer dans un système minimum de traitement.

II.1 Architecture d'un système à microprocesseur [3] :

L'architecture de base pour tous les systèmes à microprocesseurs est composée des éléments suivants :

- Unité Centrale
- Mémoire Principale
- Interface d'entrées/sorties (E/S)

Les différents organes du système sont reliés par des voies de communication appelées bus.
La figure suivante montre l'architecture générale d'un tel système :

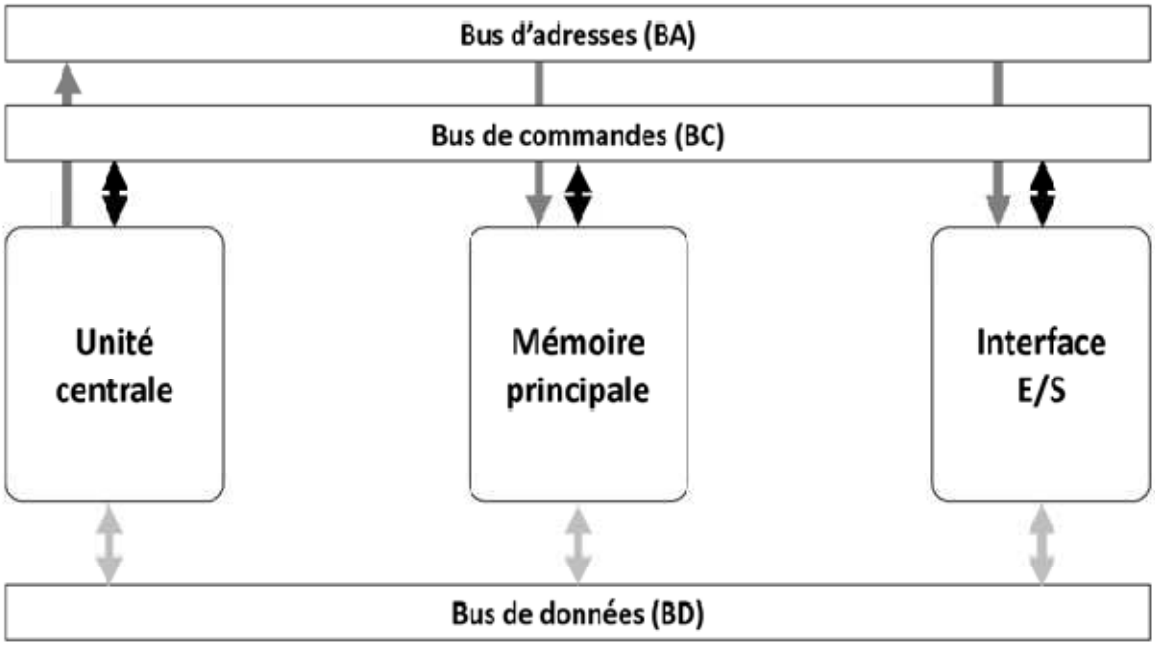

Figure II.1 : Architecture de base d'un système à microprocesseur (Modèle de Von Neumann).

a) L'unité centrale :

Elle est composée d'un microprocesseur qui est chargé d'interpréter et d'exécuter lesinstructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et decommuniquer avec les unités d'échange. Toutes les activités du microprocesseur sont cadencées par une horloge.

a) La mémoire principale :

Elle contient les instructions du ou des programmes en cours d'exécution et les données associées à ce programme.

b) Les interfaces d'entrées/sorties :

Elles permettent d'assurer la communication entre le microprocesseur et les périphériques. (Capteur, clavier, moniteur ou afficheur, imprimante, modem, etc…).

c) Les bus :

Un bus est un ensemble de fils qui assure la transmission du même type d'information. Onretrouve trois types de bus véhiculant des informations en parallèle dans un système detraitement programmé de l'information :

- **un bus de données :** bidirectionnel qui assure le transfert des informations entre lemicroprocesseur et son environnement, et inversement. Son nombre de lignes est égalà la capacité de traitement du microprocesseur.
- **un bus d'adresses :** unidirectionnel qui permet la sélection des informations à traiterdans un espace mémoire (ou espace adressable) qui peut avoir **2n** emplacements, avec**n** = nombre de lignes du bus d'adresses.
- **un bus de commande :** constitué par quelques conducteurs qui assurent lasynchronisation des flux d'informations sur les bus des données et des adresses.

II.2 Le microprocesseur (µP) :

Un µP est un composant électronique, numérique et programmable.

- **Electronique :** un µP est un circuit intégré constitué par des composants électroniques debases.
- **Numérique :** les entrées et les sorties ainsi que les commandes sont de nature numérique.
- **Programmable :** c'est cette qualité qui rend le µP intelligent. On peut modifier le comportement d'un système à base de µP en modifiant le programme qu'il exécute.

Un microprocesseur est construit autour de deux éléments principaux:

- Une unité de commande
- Une unité de traitement

II.2.1 L'unité de commande :

Elle permet de séquencer le déroulement des instructions. Elle effectue la recherche enmémoire de l'instruction. Comme chaque instruction est codée sous forme binaire, elle enassure le décodage pour enfin réaliser son exécution puis effectue la préparation del'instruction suivante.

II.2.2 L'unité de traitement :

C'est le cœur du microprocesseur. Elle regroupe les circuits qui assurent les traitementsnécessaires à l'exécution des instructions.

II.3 Conception d'un microprocesseur étape par étape [2] :

Un microprocesseur est un circuit intégré complexe. Il résulte de l'intégration sur une puce de fonctions logiques combinatoires (logiques et/ou arithmétiques) et séquentielles (registres, compteurs,….etc.). Il est capable d'interpréter et d'exécuter les instructions d'un programme. C'est une machine à calculer (un peu évoluée) programmable. Notre objectif est de concevoir un microprocesseur simple, pour cela, nous allons procéder par étapes.

II.3.1 Microprocesseur basique (version1):

On voudrait réaliser un microprocesseur (**µP)** le plus simple qui soit. Pour cela, le processeur sera relié à une mémoire vive (RAM) stockant **n mots** de **m bits**.

Exemple :RAM**256 mots** de **8 bits.**

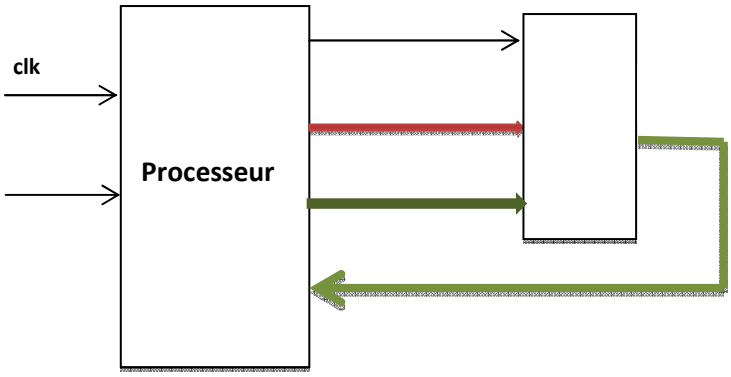

Figure II.2 : Architecture du la première version.

La RAM est reliée au processeur par ses bus de données, adresses et la ligne de WRITE.

A) **Organisation de la mémoire :**

La RAM stocke des mots de 8 bits, ça nous donne 256 instructions possibles, ce qui est largement suffisant pour un processeur basique.

Le microprocesseur doit commencer son exécution à l'adresse 0 de la mémoire.

On part donc du principe qu'on aura donc toujours une instruction à l'adresse 0 de la mémoire, et qu'on aura toujours en mémoire une instruction, puis l'opérande 1, puis l'opérande 2, puis un octet pour stocker le résultat.

Exemple :

On suppose que le programme (opérations à effectuer) ainsi que les données sont déjà chargées dans la mémoire, et qu'ils respectent le formatsuivant :

adresse	Type de mot stocké	exemple
0	instruction	+
1	Donnée(premier opérande)	3
2	Donnée(deuxième opérande)	4
3	Donnée(résultat)	X
4	Donnée(premier opérande)	-
5	Donnée(deuxième opérande)	12
6	Donnée(résultat)	8
7	Donnée(premier opérande)	X

Figure II.3 : Exemple d'un programme chargé en mémoire.

Le "X" indique que la RAM ne contient rien de valide à cet endroit-là. C'est au microprocesseur d'aller y écrire le résultat correct.

A) Les instructions :

Les opérations arithmétiques opèrent sur des nombres de 8 bits, représentant des entiers non signés. Les instructions étant stockées en RAM, il est nécessaire de les coder.

Exemple :

instruction	Code (binaire sur8 bits)
addition	00000010
soustraction	00000001

Figure II.4 : Exemple d'instructions.

B) Fonctionnement :

Pour effectuer un calcul, l'automate doit disposer de trois informations :

- l'instruction (l'opération)
- l'opérande 1
- l'opérande 2

Le processeur doit en disposer en même temps. Mais elles sont stockées en RAM, et ne peuvent être lues que l'une après l'autre. Il faudra donc prévoir un moyen de stockage de ces trois informations à l'intérieur du processeur pour pouvoir effectuer le calcul.

Vu l'organisation de la mémoire, le fonctionnement de processeur est linéaire donc l'ordre des actions effectuées est toujours le même. On peut donc le concevoir comme une machine à quatre états, dont le fonctionnement est circulaire : état 1 → état 2 → état 3 →état 4→état 1 →état 2 → . . .

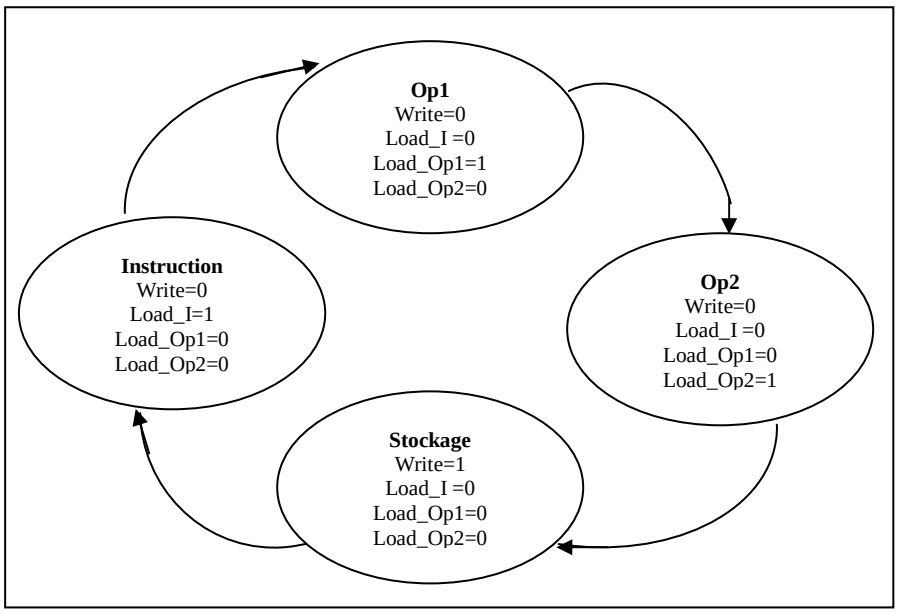

Figure II.5 Graphe d'états de la première version.

État 1 : chercher une instruction :

- Le compteur programme (cp) présente à la RAM une adresse correspondante à une instruction.

Le processeur récupère sur le bus Q[7 :0] le contenu de la RAM à cette adresse, c'est àdire l'instruction à effectuer.

- Il faut stocker cette instruction pour plus tard (quand on effectuera l'opération demandée).

On ajoute donc à l'automate un registre sur 8 bits disposant d'un enable.

L'entrée de ce registre est reliée au bus Q[7 :0] (sortie de la RAM).

Le signal d'enable de ce registre est mis à l'état haut seulement pendant l'état 1 –> stockage de l'instruction dans le registre.

État 2 : chercher le premier opérande :

- Le compteur présente à la RAM une adresse correspondante au premier opérande.
  le processeur récupère sur le bus Q[7 :0] le contenu de la RAM à cette adresse, c'est à dire l'opérande 1.
- Il faut stocker cet opérande, donc, on ajoute un registre8 bits avec enable, relié à la sortie de la RAM (Q[7 :0]). L'enable est mis à l'état haut seulement pendant l'état 2.

État 3 : chercher le deuxième opérande :

- Le compteur présente à la RAM une adresse correspondante au deuxième opérande.
  le processeur récupère sur le bus Q[7:0] le contenu de la RAM à cette adresse, c'est à dire l'opérande 2.
- Comme d'habitude on stocke cet opérande dans un registre 8 bits, dont l'enable est piloté à l'état haut seulement pendant ce cycle-ci.

État 4 : stocker le résultat du calcul :

Le compteur présente à la RAM une adresse correspondante au résultat à stocker.
Le processeur dispose dans ses trois registres de toutes les données pour effectuer le calcul. Il suffit d'ajouter une fonction combinatoirepure, pour produire le résultat.
On obtient donc l'architecture suivante pour notre processeur :

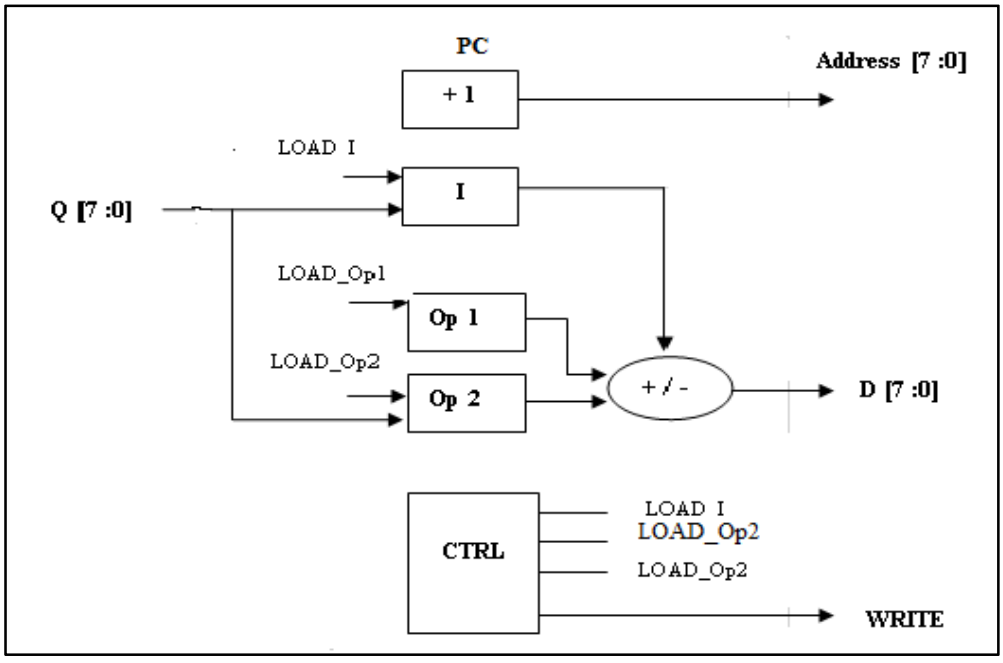

Figure II.6 : Architecture de la première version.

CTRL : machine à état qui séquence les opérations.

Inconvénient :

L'architecture actuelle ne permet pas de chaîner les calculs. Pour le faire, on peut soit :

a) Garder le résultat de chaque opération en mémoire, et définir une nouvelle addition qui opère sur un opérande en RAM et celui qu'on agardé.

L'inconvénient est qu'on rajoute une instruction pour chaque type d'opération, que cette nouvelle opération, ne nécessitant qu'un seulopérande en RAM et pourra être effectuée en 3 cycles au lieu de 4, et que ça risque de compliquer la machine à état si on veut l'optimiser(certaines opération en 3 cycles, d'autres en 4).

b) Définir des opérations de manipulation de la RAM, et grâce à elles recopier le résultat en RAM à l'endroit d'une des deux opérandes de laprochaine instruction.

c) Définir une nouvelle addition qui opère sur un opérande à l'endroit habituel en RAM, et sur un autre opérande situé à l'adresse (instruction - 1).

Solution :

Pour simplifier les choses (et par cohérence) on supprime les opérations sur deux opérandes en RAM. Toutes les opérations (à deux opérandes) se feront entre un opérande en RAM, et un gardé dans un registre interne au processeur. Et pour rendre cela possible, on définit deux nouvelles instructions : chargement de ce registre à partir d'une donnée en RAM, et stockage du contenu de ce registre en RAM.

II.3.2 Microprocesseur avec Accumulateur (version2):

Nous allons doter notre processeur d'un registre interne sur 8bits, que nous appellerons **accumulateur**. Toutes les opérations arithmétiques àdeux opérandes s'effectueront entre l'accumulateur et une donnée en RAM.

On ajoute donc deux instructions à notre processeur :

Load : chargement de l'accumulateur à partir de la RAM

Store : stockage du contenu de l'accumulateur dans la RAM

Parallèlement, les instructions d'addition et de soustraction n'ont plus besoin que d'un seul opérande - le deuxième opérande est dansl'accumulateur.

A) Organisation de la mémoire :

Exemple :

Le contenu de la RAM se présentera donc maintenant ainsi :

adresse	Type de mot stocké	exemple	effet
0	instruction	load	
1	donnée	5	L'accumulateur contient maintenant 5
2	instruction	+	
3	donnée	3	L'accumulateur contient maintenant 8
4	instruction	store	
5	donnée	X	Après exécution du programme cet emplacement en RAM contiendra 8
6	instruction	……	
7	donnée	……	

Figure II.7 : Exemple d'un programme chargé en mémoire.

Pour le **store**, aucun opérande. Par contre, en RAM, à la suite de l'instruction store, il doit y avoir un emplacement libre pour stocker lecontenu de l'accumulateur.

B) Les instructions :

Le nouveau jeu d'instruction devient donc :

Code (binaire sur 8 bits)	instruction	effet
00000010	addition	Additionne le contenu de l'accumulateur à une donnée en RAM
00000001	soustraction	soustrait du contenu de l'accumulateur une donnée en RAM
00000011	load	Charge dans l'accumulateur une donnée en RAM
00000100	store	Stocke le contenu de l'accumulateur en RAM

Figure II.8 : Exemple d'instructions.

C) Fonctionnement :

Comme précédemment, la RAM est parcourue de façon linéaire. On garde donc le compteur d'adresse incrémenté à chaquecycle d'horloge.

Pour effectuer les calculs, le processeur n'a plus besoin de connaître que deux informations : **l'instruction** et **l'opérande**. On garde donc leregistre d'instruction (8 bits) qui stocke l'instruction à effectuer pendant qu'on va chercher l'opérande en RAM.

Auparavant on parcourait 4 emplacements en RAM pour chaque instruction, d'où une machine à états à 4 cycles. Maintenant on ne parcourt que 2 emplacements en RAM, donc une machine à état à 2 cycles devrait convenir.

A chaque instruction, le processeur effectuera ceci :

Pour une opération normale :

1. Aller chercher l'instruction en RAM, la stocker dans le registre d'instruction I.

2. Aller lire l'opérande en RAM, effectuer le calcul et stocker le résultat dans l'accumulateur (opération).

Pour un load :

1. Aller chercher l'instruction en RAM, la stocker dans le registre d'instruction I.

2. Aller lire l'opérande en RAM, et le stocker dans l'accumulateur (opération).

Pour un store :

1.Aller chercher l'instruction en RAM, la stocker dans le registre d'instruction I.

2. Ecrire le contenu de l'accumulateur en RAM à l'adresse courante.

Chaque instruction est donc traitée de façon très similaire :

1. Un cycle de récupération de l'instruction (dans lequel l'enable du registre d'instruction est mis à l'état haut).

2. Un cycle de traitement de l'instruction.

Lors du **second cycle**, l'accumulateur peut subir trois traitements différents :

Pour une opération normale (+, -, AND, OR), l'accumulateur se voit modifié et chargé avec le résultat de l'opération.

Pour un load, l'accumulateur est modifié aussi, et chargé avec la donnée sortant de la RAM.

Pour un store par contre, l'accumulateur n'est pas modifié.

En entrée de l'accumulateur on mettra donc un multiplexeur qui présentera soit le résultat de l'opération en cours (si on exécute uneopération standard), soit le contenu de la RAM (si on exécute un load). La sortie de l'accumulateur est branchée en même temps :

1. Sur le bus d'entrée de la RAM (pour le cas ou on fait un store)

2. Sur l'ALU (qui implémente, selon l'instruction à effectuer, l'addition, la soustraction, le XOR, etc. . .)

On a donc les éléments suivants :

1. Compteur d'adresse (PC)
2. Registre d'instruction
3. Accumulateur avec multiplexeur en entrée
4. Une machine à état générant les signaux LOAD_I, LOAD_ACC, WRITE et le contrôle du multiplexeur.

L'architecture globale est représentée en**figureII.9** et son graphe d'étaten **figureII.10**.

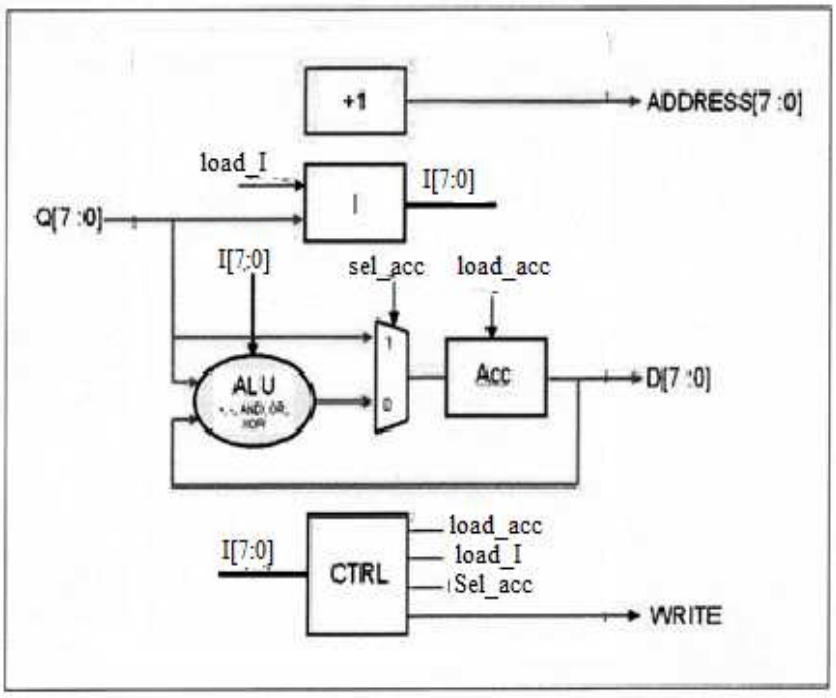

Figure II.9 : Architecture de la deuxième version.

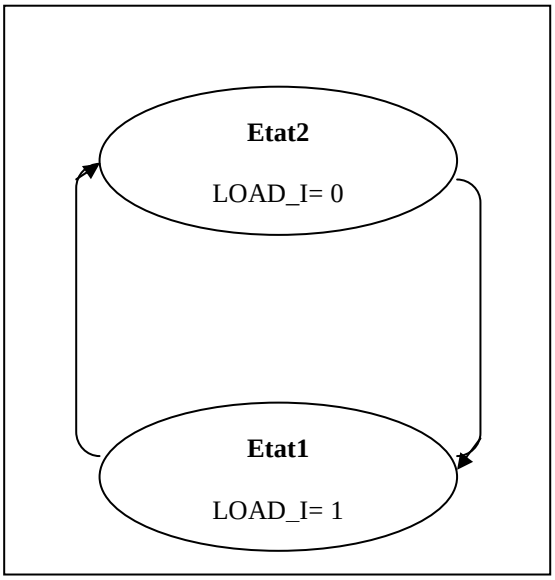

Figure II.10 : graphe d'état de la deuxième version.

Inconvénient :

L'architecture actuelle ne permet pas de :

1. Faire tourner un même code sur des données différentes.

2. Faire tourner différents codes sur des mêmes données.

3. Faire tourner un code sur des données qui ne sont pas connues avant l'exécution du programme (du genre, le début du programmedemande à l'utilisateur d'entrer des valeurs...).

Solution :

Séparer le code et les données dans RAM.

II.3.3 Microprocesseur avec Accumulateur et Indirection (version3):

Le microprocesseur est doté d'un Accumulateur et d'une RAM dont laquelle le code et les données sont séparées de tel sorte qu'on peut :

On aurait donc en RAM, une zone avec les instructions et une zone avec lesdonnées. Il suffirait juste d'aller modifier la zone des données, et d'exécuter le code générique qui saurait, pour chaque instruction, où trouver lesbons opérandes.

Pour cela, on modifie (toutes) les instructions de la façon suivante : au lieu d'avoir en RAM deux octets instruction - opérande, on aura plutôt**Instruction - adresse de l'opérande.**

A) Organisation de la mémoire :

Exemple :

Le contenu de la RAM se présentera donc maintenant ainsi :

adresse	Type de mot stocké	exemple	zone
0	instruction	load	Zone de code
1	Adresse de l'opérande	100	
2	instruction	+	
3	Adresse de l'opérande	101	
4	instruction	store	
5	Adresse de l'opérande	103	
6	instruction	load	
7	Adresse de l'opérande	100	
8	instruction	-	
9	Adresse de l'opérande	102	
10	instruction	store	
11	Adresse de l'opérande	104	
……	…….	…….	
100	donnée	3	Zone de données
101	donnée	4	
102	donnée	1	
103	donnée	X	
104	donnée	x	
……	……..	…….	

Figure II.11 : Exemple d'un programme chargé en mémoire.

B) Fonctionnement :

Le processeur doit maintenant pour chaque instruction :

1. Aller chercher l'instruction (la stocker dans le registre d'instruction)

2. Aller chercher l'adresse de l'opérande (le stocker, dans un registre dit "d'adresse")

3. Aller chercher l'opérande proprement dit, en lisant la RAM à l'adresse stockée au cycle précédent.

On a donc une machine qui possède un état de plus (celui où on va lire en RAM l'opérande proprement dit). Maintenant, on n'accède plus à la RAM de façon linéaire.Les adresses de code sont globalement linéaires (0, 1, 2, . . .), celles des données elles sont arbitraires). Il faut doncprésenter sur le bus d'adresse RAM :

1. Soit le compteur d'adresse pendant les deux premiers cycles (et on l'incrémente à chaque fois).

2. Soit le contenu du registre d'adresse (adresse de l'opérande à aller chercher) pendant le troisième cycle (et ici le compteur d'adresse nedoit pas être incrémenté). Donc, on utilise un multiplexeur.

L'architecture globale est représentée en **figure II.12** et son graphe d'état en **figure II.13** :

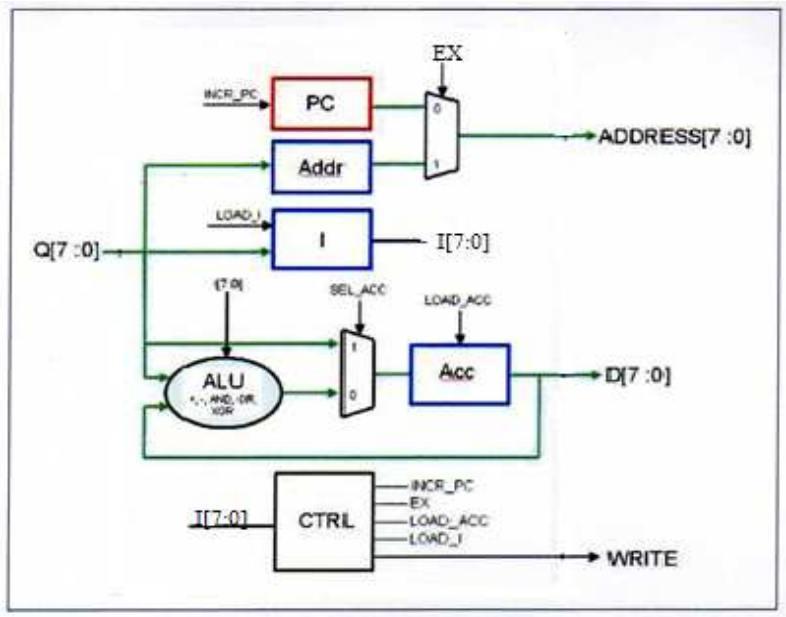

Figure II.12 : Architecture de la troisième version.

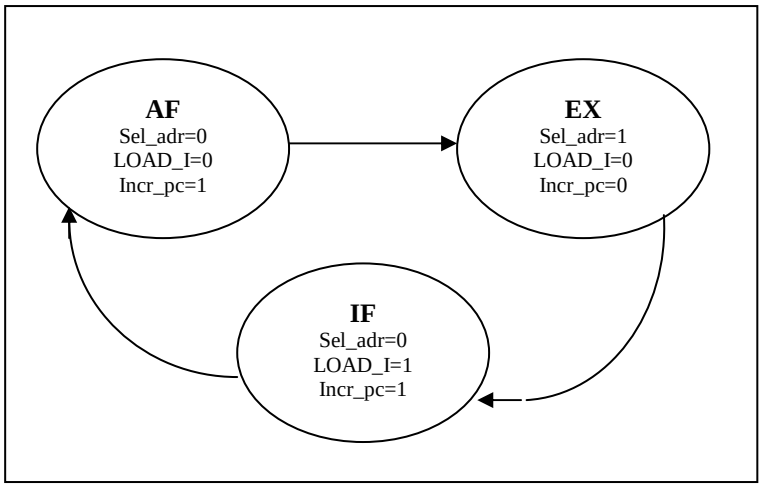

Figure II.13 : Graphe d'état de la troisième version.

Inconvénients :

L'architecture actuelle ne sait effectuer que des calculs linéaires (suite fixe d'instructions), sur des données potentiellement inconnues (maisdont l'adressage de stockage est connue).

Solution :

Ajouter au processeur des instructions des sauts conditionnels et inconditionnels.

II.3.4 Processeur (version finale):

1) Flags :

Pour cela, chaque opération (logique ou arithmétique) va positionner deux signaux devant être mémorisés pour l'instruction suivante, qui nedoivent être modifiés que si on modifie l'accumulateur :

C (comme carry) :

- mise à 1 si l'opération courante est une opération arithmétique et donne lieu à une retenue,
- mise à 0 si l'opération courante est une opération arithmétique et ne donne pas lieu à une retenue,
- mise à 0 si on fait un load

Z (comme zéro) :

- mise à 1 si on charge 0 dans l'accumulateur
- mise à 0 dans tous les autres cas.

La génération de C et Z est combinatoire et peut être effectuée par l'ALU. Il suffit juste de rajouter deux registres 1 bits pour stocker ces deux signaux, pilotés par le même enable que l'accumulateur (LOAD_ACC, qu'on appellera maintenant LOAD_AZC). On considérera donc que Z et C font partie de l'accumulateur (qui devient donc un registre sur 10 bits : 8 de donnée, 1 pour Z, un pour C).

Exemple :

Les opérations ADDC et SUBC, prenant en compte la retenue C de l'opération précédente.

2) Sauts :

Pour implémenter les sauts, on définit trois instructions supplémentaires :

- JMP : saut inconditionnel.

L'exécution de cette instruction fait sauter l'exécution du programme directement à une adresse donnée (passée comme opérande).

- JNC : saut si C est nul.

Idem à JMP, mais seulement si C est nul. Sinon, équivalent à NOP (on continue à l'adresse suivante)

- JNZ : saut si Z est nul.

Idem à JMP, mais seulement si Z est nul. Sinon, équivalent à NOP (on continue à l'adresse suivante).

Pour implémenter les sauts, il suffit de se donner la possibilité de remplacer le contenu de PC par la valeur lue en RAM.

- PC devient donc un peu plus complexe. C'est globalement un compteur, mais ilest incrémenté si son signal de commande INCR_PC = 1
- est chargé avec une nouvelle valeur si un signal de chargement LOAD_PC = 1
- si LOAD_PC et INCR_PC valent 1.

  Ceci peut être implémenté comme suit :

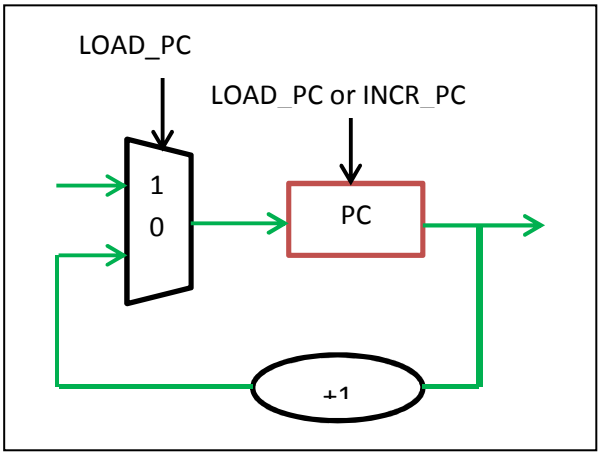

Figure II.14 : Implémentation de PC.

L'architecture globale est représentée en **figure II.15** et son graphe d'état en **figureII.16**:

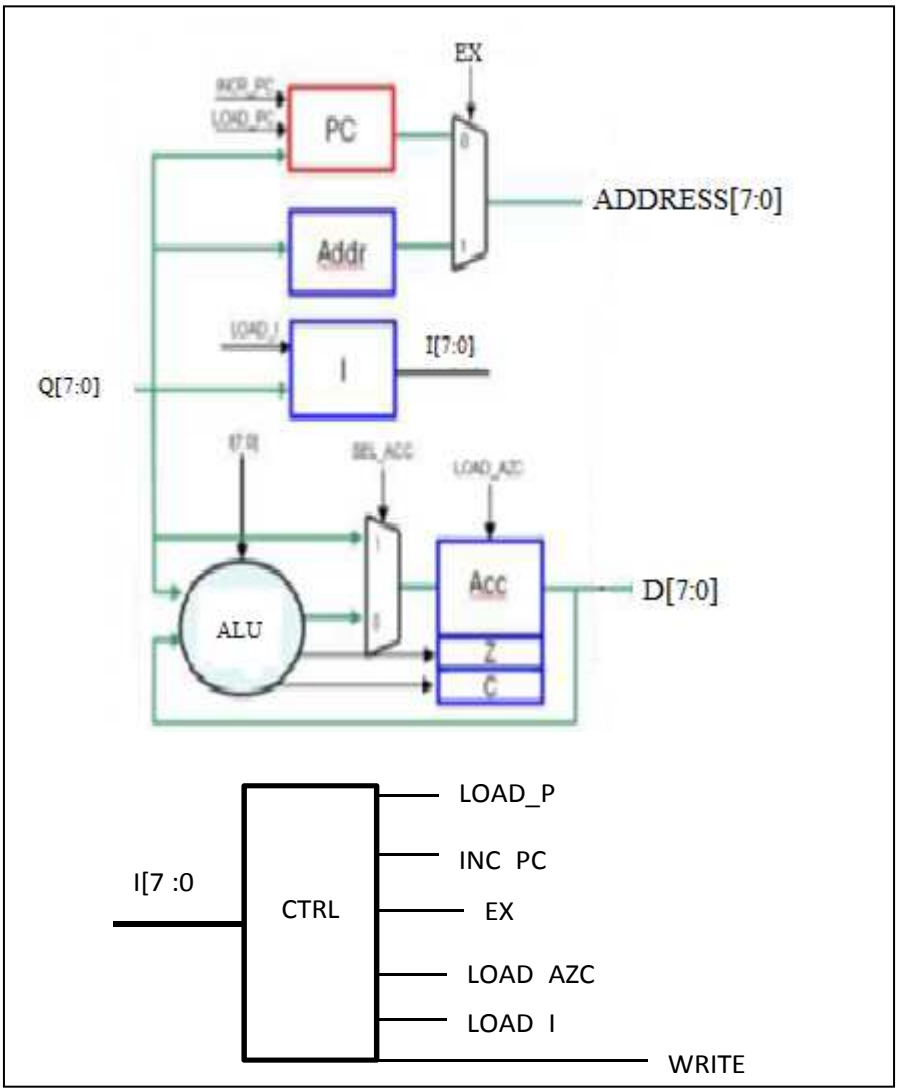

Figure II.15 : Architecture de la version finale.

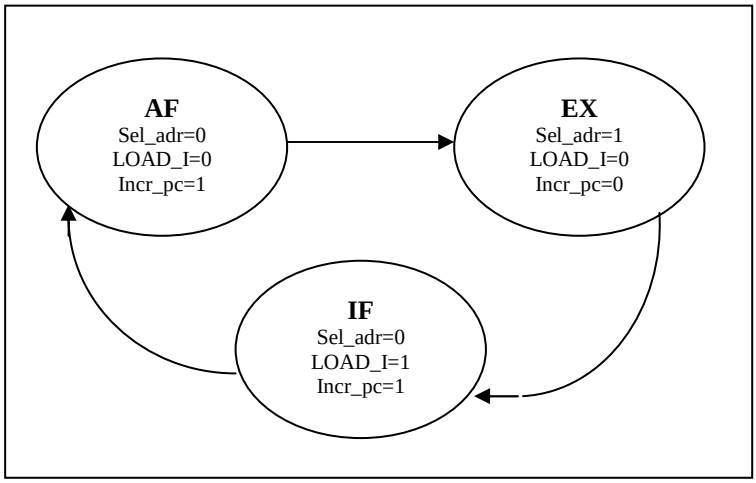

Figure II.16 : Graphe d'état de la version finale.

Conclusion :

Lors de ce chapitre, on a représenté la notion des systèmes à microprocesseur ainsi que les différentes représentations optées pour la réalisation d'un microprocesseur.

Nous intéressons dans ce qui suit à la réalisation d'un microprocesseur avec accumulateur (version2).

Chapitre III

Introduction

La conception théorique de notre microprocesseur étudiée au chapitre précédent nous offre la possibilité de la réaliser en pratique à l'aide d'une carte électronique FPGA DE2.

Ce chapitre est consacré à la réalisation de notre microprocesseur (Nanoprocesseur 4 bit), la première partie est consacrée à la présentation générale de L'environnement de développement Quartus II, et au Langage de description de matériel VHDL**,** en suite nous allons simuler notre système, enfin nous passerons à la visualisation sur la carte électronique(FPGA).

III.1Présentation de Quartus II [1] :

L'environnement de développement permettant de programmer les composants reconfigurables d'Altera se nomme Quartus. Il permet de réaliser l'intégralité des étapes permettant de configurer un composant programmable, alors ce logiciel permet la programmation des circuits programmables ALTERA.

❖ L'environnement Quartus II :

L'environnement de Quartus II est le suivant :

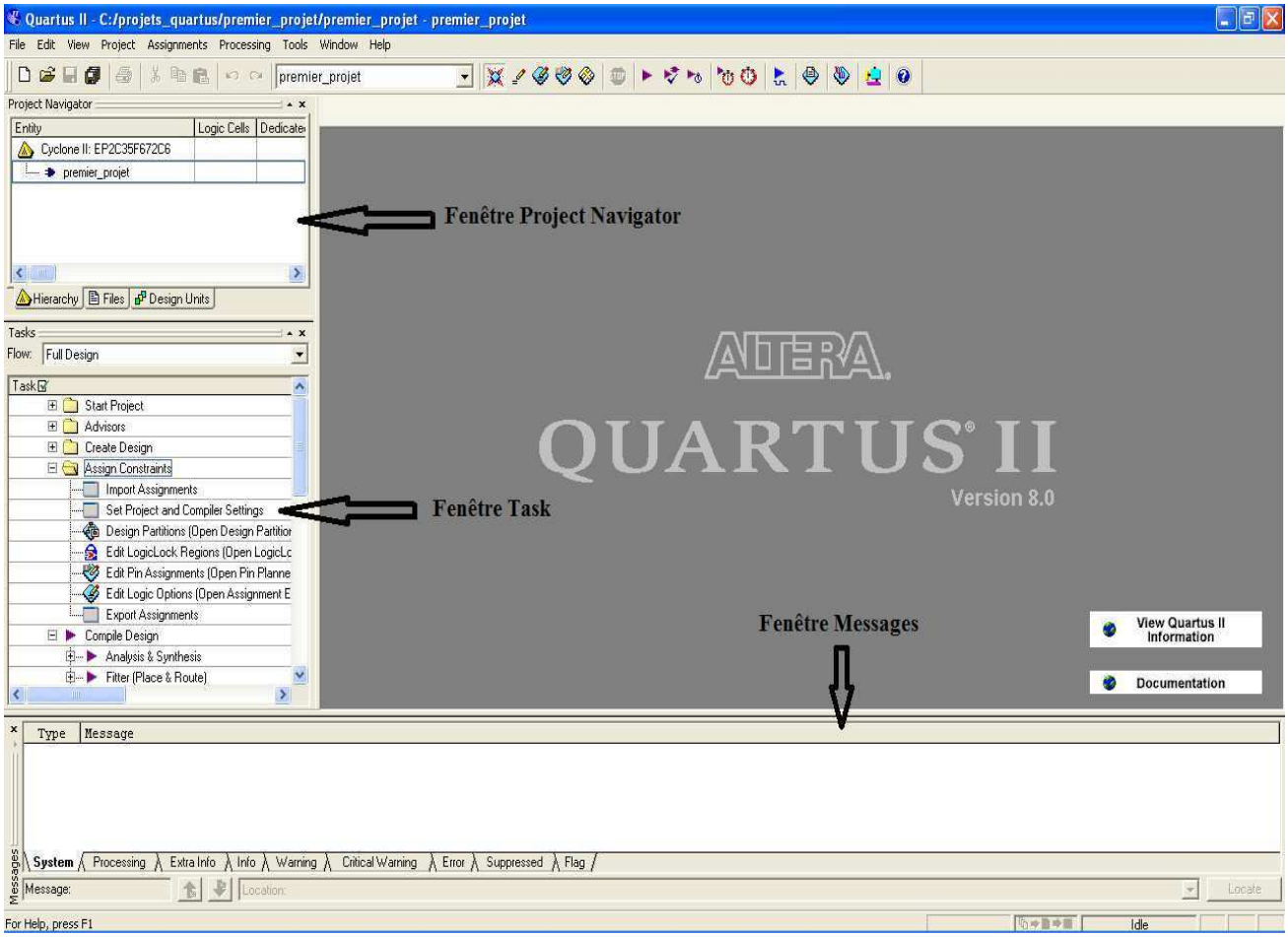

Figure III.1 : L'environnement de développement Quartus II.

L'environnement se découpe en 4 parties, une pour l'éditeur de code qui apparait après avoir créé un nouveau projet, 3 sont visibles après le lancement de Quartus.

- **Une fenêtre Project Navigator :** qui contient toutes les informations relatives au projet dont les détails sur la hiérarchie du projet, les fichiers le constituant, les différentes unités définies
  (entités, architecture, schémas, machines d'états ...).
- **La fenêtre Tasks :** présente les différentes tâches du Design Flow avec la possibilité d'accéder à l'ensemble des processus de traitement de Quartus et les comptes rendus associés.
- **La fenêtre Messages :** permet d'être informé en continu des informations, avertissements et erreurs apparaissant lors de l'exécution des diverses tâches.

❖ Développement sous Quartus II :

1. Création d'un projet sous Quartus :

Après avoir démarrer Quartus , on crée un nouveau projet : File -> New Project..

- La première boite de dialogue qui apparait permet de fixer le nom du projet et son lieu de stockage :

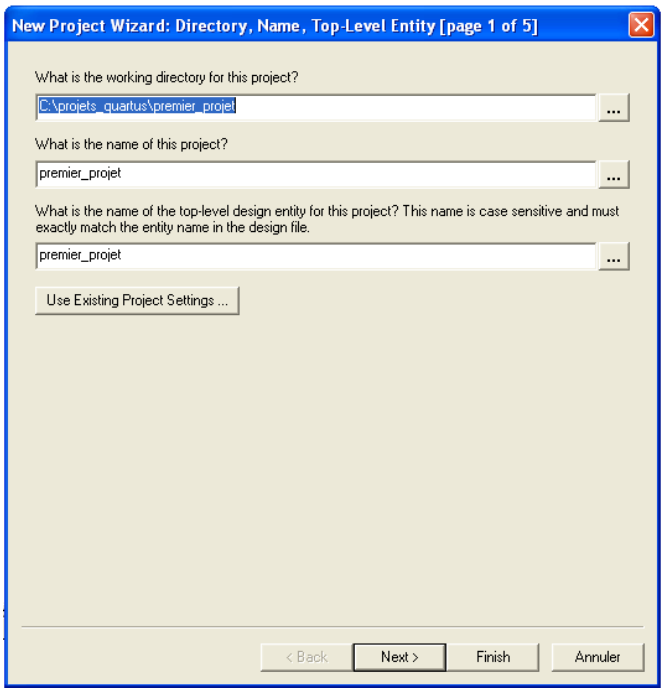

Figure III.2 : Création de projet sous Quartus.

- Dans la seconde, il est possible d'ajouter des fichiers déjà existants au projet. Ne rien ajouter et cliquer sur le bouton Suivant.
- La troisième fenêtre de dialogue permet de choisir le FPGA ou CPLD ciblé.

Une fois choisi, on en termine avec l'assistant de création de projet en cliquant sur Finish.

- **Ajout d'une source VHDL** :

Pour ajouter une nouvelle source VHDL : **File -> New...** et on choisit une source de type VHDL (il est intéressant de remarquer l'étendue des possibilités en matière de source).

- **Sauvegarde d'un projet :**

Après avoir taper le code, la sauvegarde se fait comme suite : **File -> Save As...**puis entrer un nom avec l'extension .vhd).

- **Inclure un modèle :**

Il est possible d'inclure des modèles d'un simple clic dans le codesource en cours d'édition. Cet outil (Template) est accessible via le menu **Edit -> Insert Template**  ou via l'outil dans la palette d'outils de l'éditeur.

2. Vérification de la syntaxe du code :

Quartus propose un outil de contrôle d'erreur pour limiter la présence de coquilles et d'erreurs d'inattention syntaxique dans le code. On le fait comme suite :

Menu Processing -> Analyse current file ou l'outil dans la palette d'outils de l'éditeur.

3. Création d'un stimulus de simulation :

Quand le code VHDL est correct en matière de syntaxe, il est ensuite nécessaire de le tester en lui appliquant différents vecteurs de test, la solution pour réaliser ce test avec Quartus II est l'éditeur graphique de vecteurs de test (VectorWaveform Editor).

- Pour créer un fichier de test graphique, atteindre le menu et faire **File -> New...** et choisir **Vector Waveform File.** utiliser la commande **Save As**... pour enregistrer le fichier de test.

Pour rendre les signaux d'entrées et de sorties accessibles à l'édition dans l'éditeur graphique, il faut réaliser une compilation de Design **: menu Processing -> Start Compilation.**

- Ajouter des signaux d'entrée :
- Pour créer des stimuli sur les entrées, il faut ajouter les signaux d'entrées dans la colonne **Name** par un double clic dans cette même colonne (ou la commande **Insert ->Insert Node or Bus... du menuEdit).** Poursuivre par un clic sur le bouton **Node Finder...** de la boite de dialogue qui apparaît. Dans le second dialogue, sélectionner **Pins :all**dans la boite de liste **Filter** et lancer l'opération de filtrage par un clic sur lebouton **List**.
- Ajouter les signaux d'entrée dans la liste des signaux sélectionnés **(SelectedNodes)** en utilisant le bouton> . Puis valider par **OK** deux fois.
- Donner un stimulus à un signal :
- Sélectionner le signal dans la colonne **Name.**

- Création d'un signal de type horloge pour le stimulus du signal sélectionné:

Menu Edit -> Value ->Clock...(ou cliquer sur l'outil de création d'horloge ).

On remplit le dialogue le dialogue d'horloge pour créer une horloge.

Pour changer la durée de la simulation, utiliser la commande du **menu Edit ->End Time**...

4. Simulation du projet :

Il convient préalablement de définir les paramètres de simulation en accédant à la boite de dialogue correspondante (menu **Assigments -> Settings puis Category Simulator Settings** ou également par la fenêtre de tâches, section **Verify Design ->Simulate Design ->Quartus II simulator -> Edit settings).**

- Choisir une simulation fonctionnelle **(Functional Simulation)** pour le mode de simulation **(Simulation mode).**
- Ajouter le fichier test.vwf pour les paramètres d'entrées de la simulation **(Simulation input).**
- Valider par **OK.**
- Dans la fenêtre des tâches, lancer la simulation par un double clic sur **Quartus II Simulator (Functional).**
- Pour une simulation temporelle, choisir **Timing** pour le mode de simulation.

III.2 Langage de description de matériel VHD (Very High Speed Integrated Circuits Hardware Description Language) [5][6][7] :

Le VHDL est un langage de description de matériel HDL. Il répond à tous les critères établis pour un langage HDL.

Les plus importants points forts de ce langage son :

- **Simulation, Synthèse :**Le VHDL permet d'avoir des lignes de code pouvant être simulés

(Simulation Fonctionnelle) dans le but de vérifier si le code obéit correctement à la fonction souhaitée mais pour parvenir à une maquette réalisable ce n'est pas suffisant. D'où la nécessité de synthétiser le même code. Lors de la synthèse, le compilateur traduit le premier code en un autre équivalent mais à un niveau d'abstraction plus bas. A ce niveau de synthèse, le compilateur traduit le code de haut niveau en portes logiques ceci en fonction du circuit logique programmable qui est ciblé.

- **Portabilité :**
  - **Portable vis-à-vis le circuit logique programmable :** C'est-à-direpeut être implémenté sur n'importe quel circuit logique programmable à condition d'avoir la capacité logique requise.
  - **Portable vis-à-vis du compilateur :** pouvoir passer d'un compilateur à un autre et obtenir un même circuit en fin de processus.
- **Une construction hiérarchique :** Cette approche permet de simplifier la conception.
- **Une description fonctionnelle :** Complémentaire à la précédente. Tout algorithme est la description interne d'un bloc situé quelque part dans la hiérarchie du schéma complet.

III.2.1Structure d'un programme VHDL, le couple Entité, Architecture :

Les concepteurs de VHDL ont adopté l'approche suivante : N'importe quel système est considéré comme une boite noire. Cette boite noire a des entrées et des sorties. Ils ont appelé la boite noire **« Entité ».** N'importe quel système électronique effectue des opérations sur le signal d'entrée pour donner le résultat du traitement en sortie. Ces opérations sont le contenu de la boite noire qui est appelée **« Architecture ».**

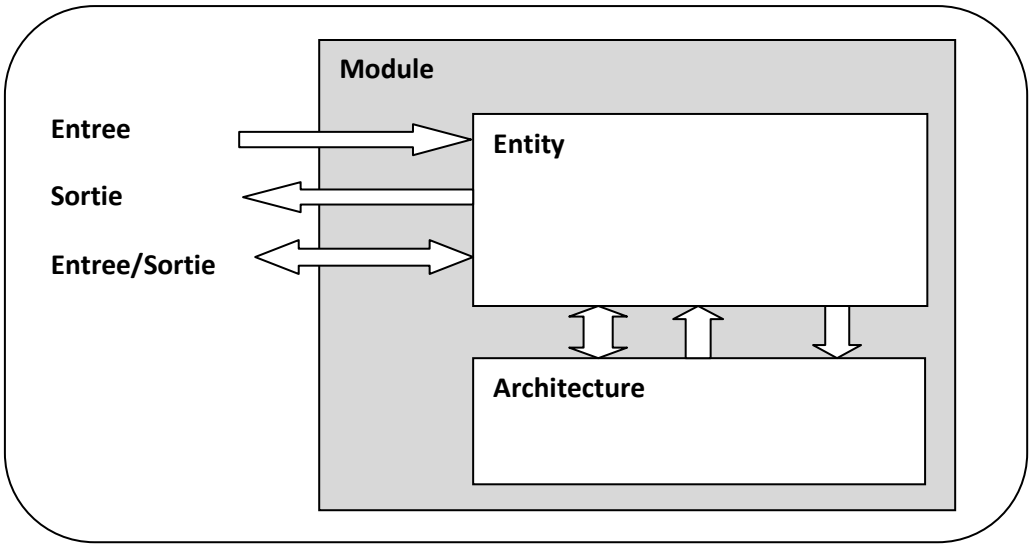

Figure III.2.1: Structure de base d'un module sous VHDL.

- **Entity ou Entité :** Dans la partie déclarative de l'entité, le circuit décrit comment il est vu par l'exterieur. Ceci à travers les entrées, sorties et entrées/sorties.

```
Entity<Entity name>is
Port (
<Signal name> :  <signal direction><data type> );
End <Entityname>;
```

Figure III.2.2: Syntaxe déclarative de l'entité

- **Architecture :** L'architecture décrit le comportement que doit avoir le circuit ou les opérations qu'il doit effectuer. Une architecture doit toujours être attachée à une entité.

```
Architecture< architecture name>OF  <entité name>is
<Define signals and constants>
Begin
End <Architecture name>;
```

Figure III.2.3 : Syntaxe déclarative de l'architecture

C'est dans cette section que le programme est rédigé. Un programme comporte essentiellement les éléments suivants : les signaux internes, opérateurs logiques, les process. La description d'une architecture peut prendre trois formes :

- **Description comportementale :** Spécifie le comportement du composant ou de circuit à réaliser au moyen d'instructions séquentielles ou se forme de flot de données(constituant un process).
- **Description Structurelle :** Les interconnexions des composants préalablement décrits sont énoncées. Cette description est la transcription directe d'un schéma.
- **Description mixte :** Elle regroupe les deux descriptions décrites précédemment.

III.2.2 Éléments de langage VHDL :

- **Le Signal :**le signal est représenté par certains types de données. Il est assigné par un nom et un type de données comme suit : Signal « nom_signal » : « type_donnée ».
- **Les constantes :** Objets qui portent une valeur à l'initialisation.

- **Les variables :** La valeur d'une variable est immédiatement mise à jour lors ce qu'elle est assignée. Une variable est déclarée comme suit :

Variable « nom_variable » : type « expression initiale ».

- **Les types de données :**Le VHDL est un langage très typé. Chaque élément utilisé (signal, variable) doit avoir un type bien défini. Toute opération doit se faire avec des objets de même type.
- **Le process :** Il permet de décrire les instructions parallèles qui utilisent les mêmes signaux. L'énoncé process est habituellement accompagné d'une liste de sensibilité(horloge, bit d'activation ou de sélection..) et il est exécuté lors ce qu'un des signaux de la liste de sensibilité change l'état.
- **Les instructions concurrentes :**Les instructions concurrentes sont décrites directement dans le corps de l'architecture.
- **Les instructions séquentielles :**Les instructions séquentielles sont interne aux processus, aux procédures, et aux fonctions. Exemple **: if-then-else**qui permet de réaliser les boucles conditionnelles
- **Les attributs :**Se sont des propriétés ou des caractéristiques associées à un objet ou un type. Ils sont déclarées comme suit :Nom_var' Nom_attribut ;
- **Les librairies et les paquetages :**Une librairie est constituée d'un ou plusieurs paquetages. Elle sont fournies soit par IEEE en tant que partisan de standard VHDL ou bien crée par le concepteur lui-même. La syntaxe déclarative se fait comme suit :

  LIBRARY IEEE ; pour déclarer la librairie
  USE IEEE.STD_LOGIC_1164.ALL ; pour choisir le paquetage Standard logic 1164.

III.2.3Structure d'un programme sous VHDL:

Un programme écrit sous VHDL obéit à la structure suivante :

- Entête : C'est une partie facultative, elle referme des informations concernant le programmeur, la description de programme en général.
- Déclaration des librairies : En deuxième lieu vient la déclaration des librairies et des paquetages que le programmeur juge nécessaire pour son programme.
- Déclaration d'Entité : En troisième lieu vient la déclaration d'entité avec les signaux à utiliser dans tout le programme avec leur direction.

- Déclaration d'architecture : Juste après l'entité vient la déclaration de l'architecture qui décrit l'entité.

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
Entity<Entity name>is
Port (<list of ports or design input and output>);
End <Entity name>;
Architecture< architecture name>OF  <entité name>is
<in this section define signals and constants>
Signal <signal name> : Data type;
Begin
<concurrent  statements>
<process name>: process (Sensitivity list)
Begin
<Sequential statements >
End;
End <Architecture name>;
```

Figure III.2.3: Structure d'un programme sous VHDL

III.2.4LES TESTBENCHS :

Le langage VHDL est un langage de modélisation et de simulation. Il contient tous les éléments nécessaires à la création de stimuli et surtout à l'exploitation des résultats.

Pour élaborer un TESTBENCH, il faut établir la liste des cas la plus complète que possible. Les stimuli sont les entrées appliquées au programme via le simulateur pour imiter le comportement des vraies entrées. Après exécution de simulateur, les sorties sont observées si elles sont comme prévu par le programme et ceci sans aucun risque sur le matériel (FPGA ...).

III.3 Résultats de simulation :

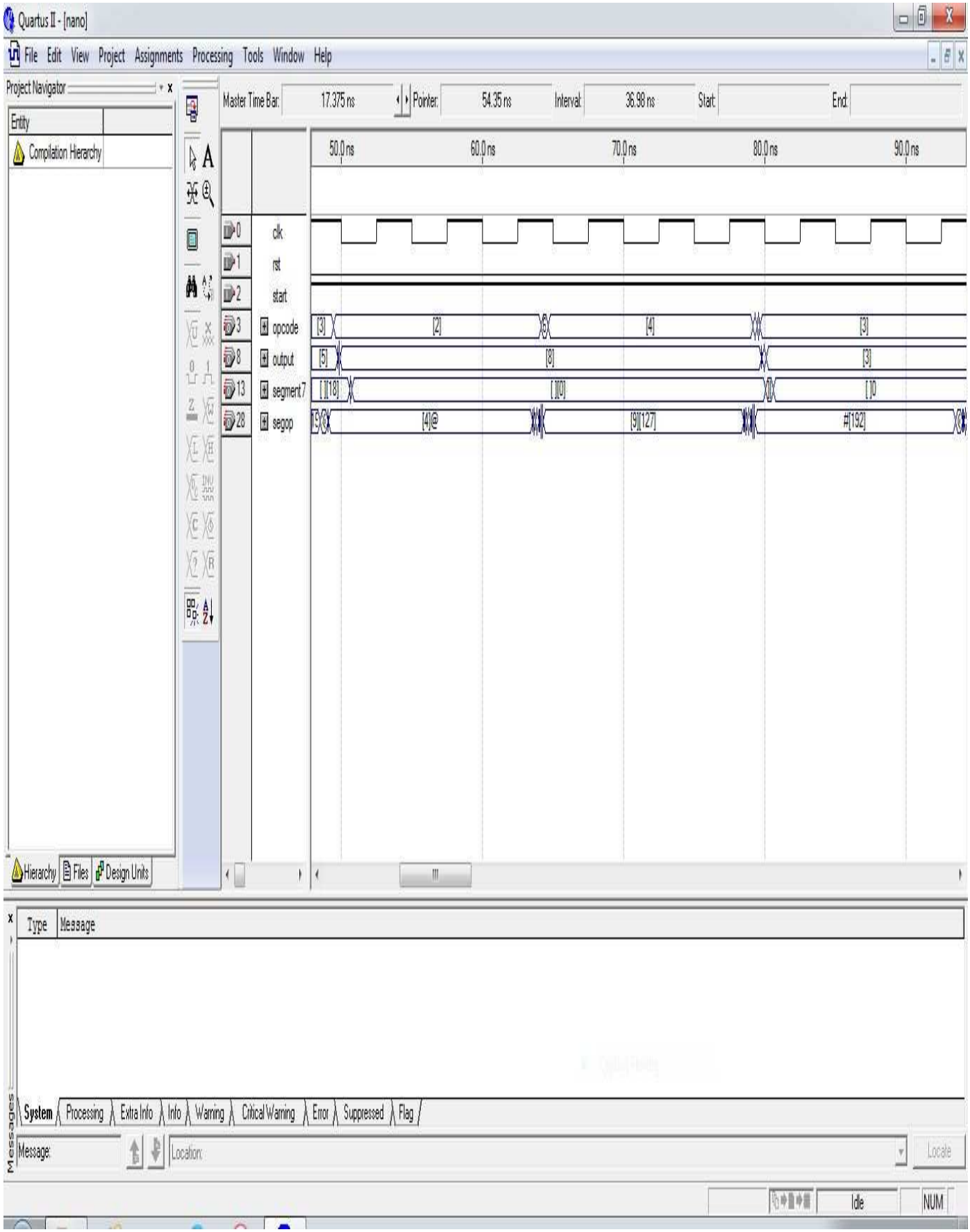

Figure III.3: Résultat de simulation

III.4 Configurer le composant au Programmation :

Pour configurer le composant afin d'obtenir le fonctionnement décrit par le projet, il faut impérativement :

- que la maquette *DE2* soit reliée à l'ordinateur par câble ***USB***.
- qu'elle soit sous tension **(bouton rouge).**
- que le commutateur ***RUN / PROG*** soit positionné sur ***RUN***.

Pour configurer le composant de la maquette DE2 :

Sélectionner l'icône de programmation ou bien Sélectionner **Tools > Programmer.**

1- La fenêtre de **"programmation"** s'ouvre alors.

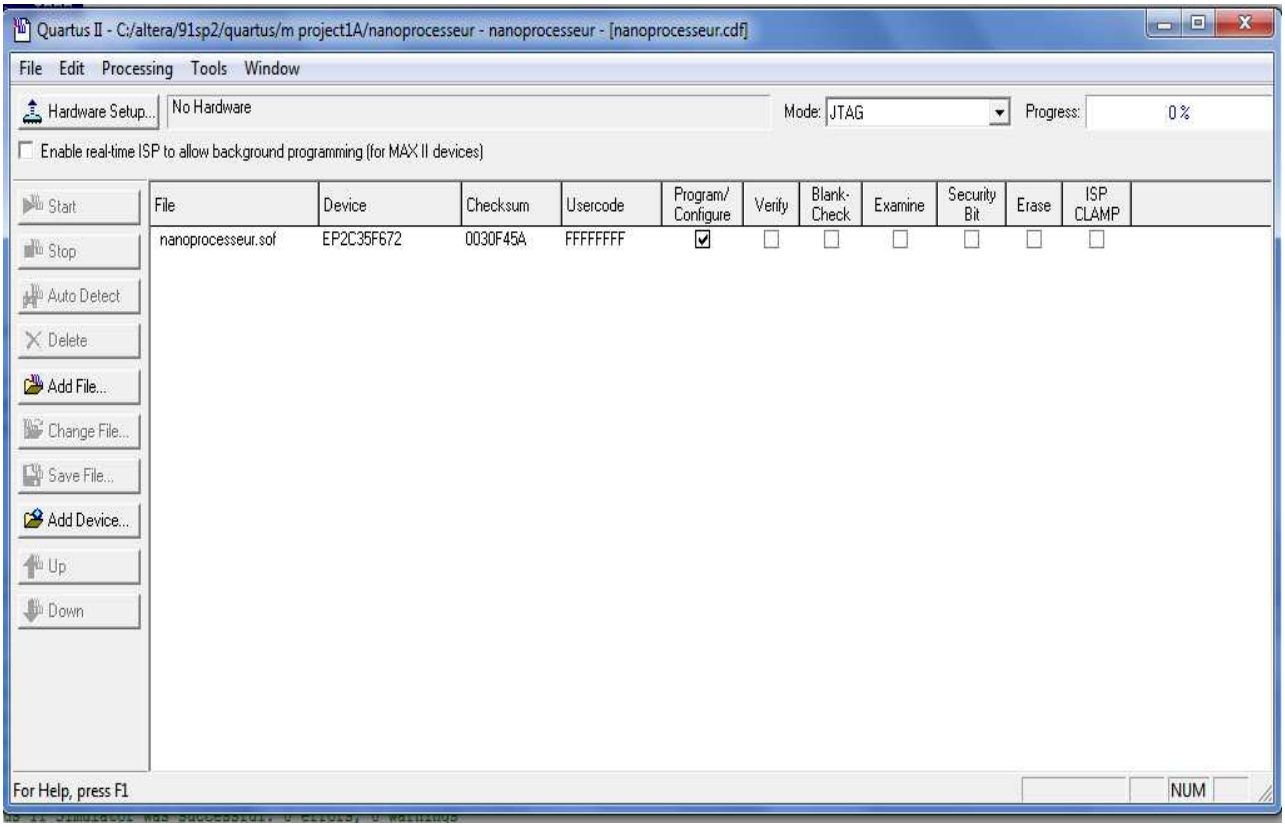

Figure III.4.1 : Fenêtre de "programmation"

2- Cliquer sur **Hardware Setup**

3- Puis dans la nouvelle fenêtre sélectionner **USB-Blaster**à la place de **No Hardware** dans le menu **Currently Selected Harware**.

Figure III.4.2 : Fenêtre Hardware Setup

Fermer cette fenêtre 

4- Enfin, cliquer sur **Start** 

pour lancer la configuration du circuit de la maquette ***DE2***.

III.5 Résultats pratique :

- **Programme exécuté en RAM :**

```
0 =>"00110101",   -- loadi 5
1 =>"00100011",   --add 3
2 =>"01000000",   --store xxxx
3 =>"00110011",   --loadi 3
4 =>"00010001",   --sub 1
5 =>"01000000",    --store xxxx
6 =>"00110001",    --loadi 1
7 =>"01010001",   -- and 1
8 =>"10001010",   -- jump 10
9 =>"00110100",
10=>"00111000",   --load 8
others =>"11111111"
```

Chaque mot mémoire est représenté sur 8 bits dontle code opération(sur 4 bits) et l'opérande (sur 4 bits).

La réalisation pratique s'est fait avec succès sur la carte FPGA.

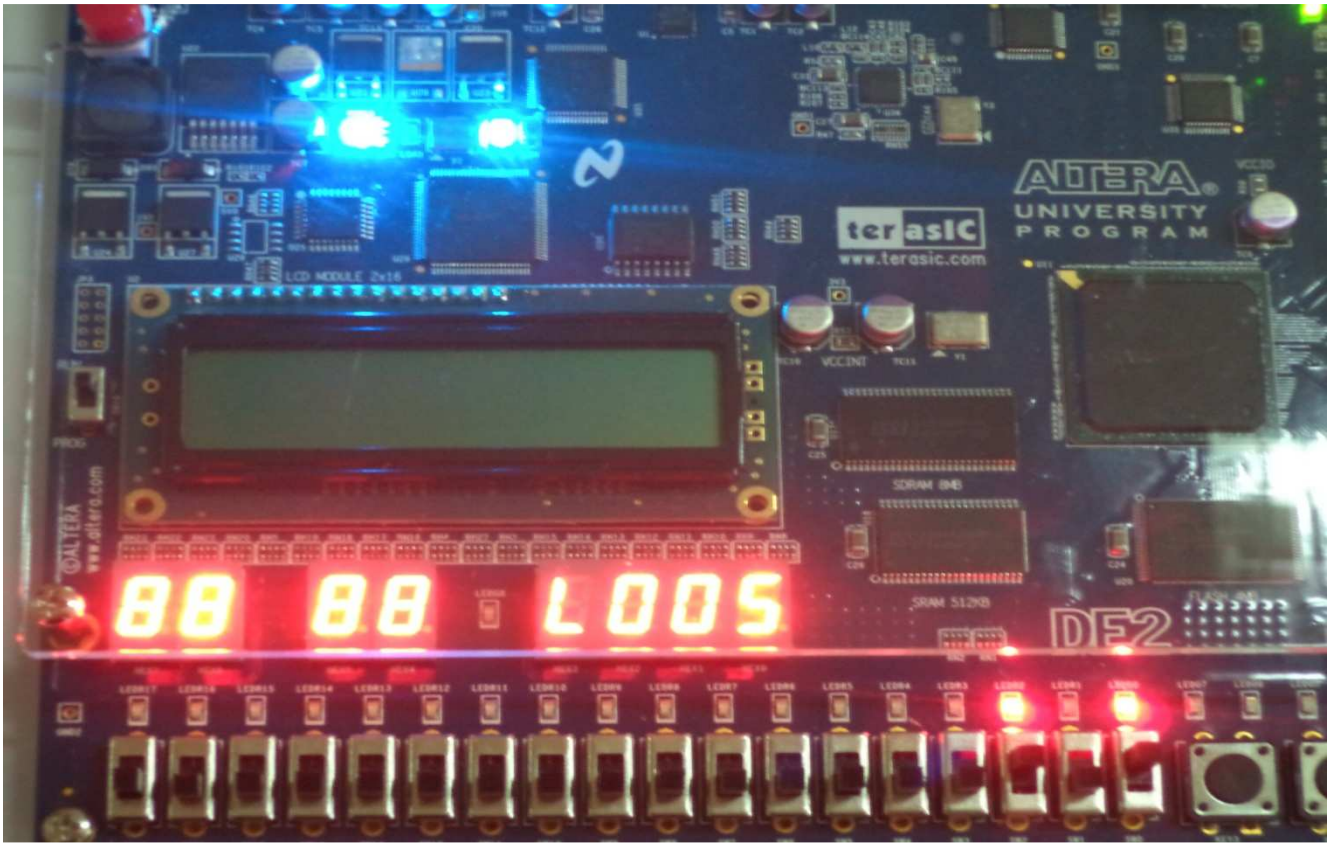

Figure III.5 : Résultat pratique de l'instruction load 5

Conclusion

Dans ce chapitre nous avons présenté l'environnement de développement et le langage VHDL, nous avons aussi simulé notre système sous QuartusII, par la suite nous avons donné une idée sur les différentes étapes à suivre pour réaliser notre système sur une carte FPGA.

Conclusion Générale

Conclusion Générale

Dans ce travail, nous avons étudié et réalisé un processeur simple avec un nombre réduit d'instructions, en poursuivant une méthodologie progressive sur une cibleReconfigurable de type FPGA.

Dans le premier chapitre, nous avons donné un aperçu globale sur les circuits programmables logiques (PAL, CPLD,GAL ….).
Une étude plus approfondies sur l'architecture des circuits FPGAs abordée.
On a ainsi, présenté les différents composants d'un circuit FPGA.

Au deuxième chapitre, une méthodologie de conception d'un microprocesseur est présentée. La méthode a consisté àaméliorer la conception d'une manière progressive. Ainsi à chaque étape les inconvénients et avantages de la conception proposée sont discutés.

Au dernier chapitre, nous avons introduit l'outil de développement utilisé (QUARTUS II) d'Altera et le langage de description matériel VHDL. Ces deux outils sont utilisés avec succès pour la réalisation de notremicroprocesseur .Ainsi, après avoir testé par simulation le fonctionnement de notre application, nous avons programmé notre cible FPGA en utilisant la carte de développement DE2 d'Altera.

Notre travail peut être amélioré et peut constituer des perspectives très intéressantes.

Bibliographie

Bibliographie

[1] Boualem Taous&Bouksil Naima.Etude et réalisation d'une transmission sécurisée de données par le chao sur une carte FPGA. Mémoire de fin d'étude, master automatique 2013/09.

[2] Des machines à états aux processeurs. Cours en Électronique Numérique Intégré.

[3] Généralités sur l'architecture des systèmes à base de microprocesseur. 3ème année licence en commande electrique. Université KasdiMerbah-Ouargla.

[4] Hadj Said Djamal.Implémentation d'une application de tracking sur FPGA. Mémoire de fin d'étude, master ELN 2013/04.

[5] Nachef Toufik. Implementation d'une instrumentation sur un FPGA. Mémoire Magister en électronique Option : Micro-électronique 2011.

[6] Romain Berny. Introduction au langage VHDL pour la synthèse. TP CAO électronique Numérique 2005.

[7] VHDL Tutorial learn by example(site). Esd book:esd.cs.ucr.edu.

Annexe A

Les circuits programmables FPGA offrent et permettent a l'utilisateur l'avantage d'intégrer et d'implémenter des systèmes et fonctions beaucoup plus complexes sur un seul circuit (FPGA).
IL existe des constructeurs joignent des périphériques au circuit FPGA, tels que les ports de communications, des afficheurs, des ports d'extension, et ils offrent des cartes (munies de FPGA+ périphériques) adéquates pour diverses applications, parmi ces carte celle que nous avons utilisé la DE2 de Altera.

A.1 Vue générale sur la carte DE2 :

La figure ci-dessous nous montre la disposition des composants de la carte DE2 et indique l emplacement des connecteurs et des composants.

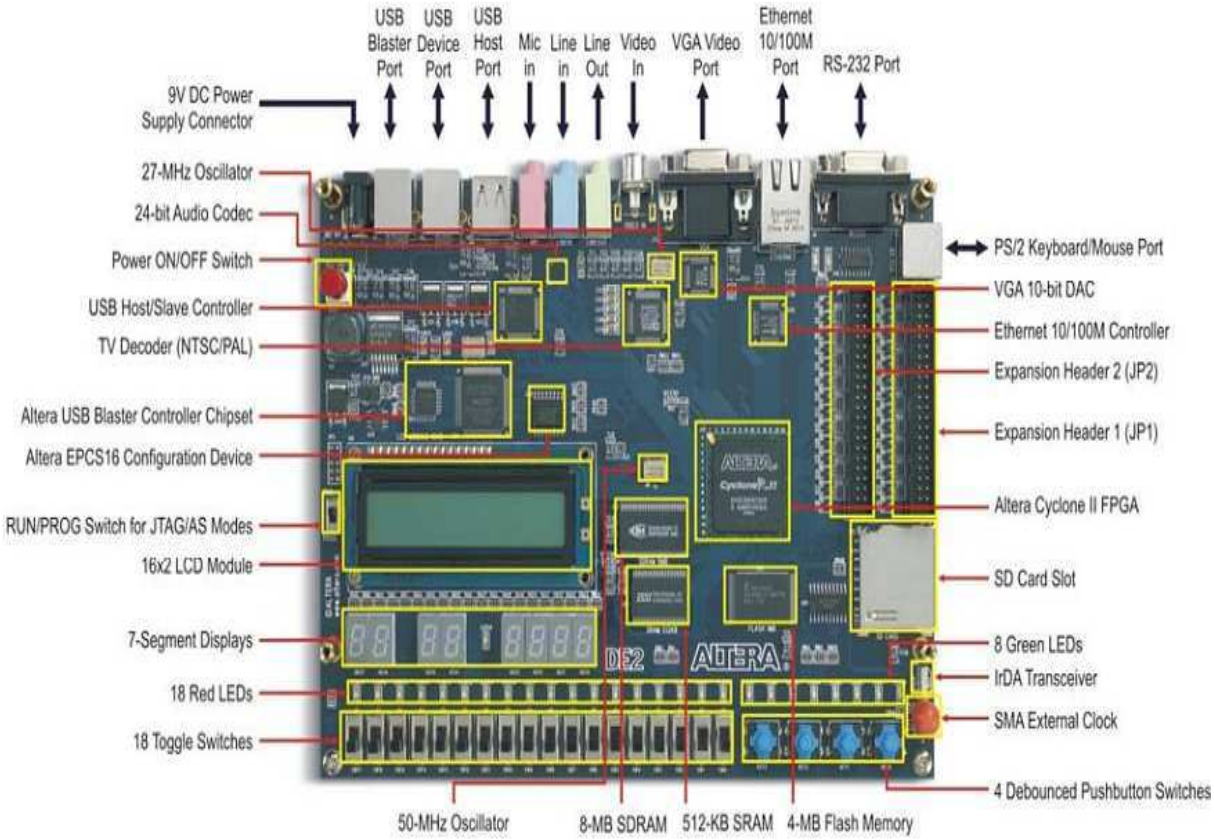

Figure A.1: Aperçu de la carte DE2

La carte DE2 a beaucoup de dispositifs permettent à l'utilisateur de faire de diverses implémentations de circuits, de simples à plusieurs projets multimédias.

Cette carte dispose des éléments suivants :

- Un FPGA Altera Cyclone II 2C35
- Un circuit de mémoire Flash série Altera EPCS16
- Une interface USB-Blaster pour la programmation et le contrôle des API utilisateur, les modes de programmation JTAG et Active Serial (AS) sont supportés
- 512-Kbyte de SRAM
- 8-Mbyte de SDRAM
- Un connecteur SD Card
- 4 boutons poussoirs
- 18 interrupteurs
- 18 LEDs utilisateur rouges
- 9 LEDs utilisateur vertes
- Un oscillateur 50-MHz et un oscillateur 27-MHz comme sources d’horloges
- Un CODEC 24-bit CD-quality avec jacks entrée ligne, sortie ligne et microphone-in.
- Un CNA VGA (10-bit high-speed triple DACs) avec connecteur VGA-out
- Un décodeur TV (NTSC/PAL) avec connecteur TV-in
- Un contrôleur 10/100 Ethernet avec connecteur
- Un contrôleur USB Hôte/Esclave avec connecteurs USB type A et type B
- Un E/R RS-232 avec connecteur 9-broches
- Un connecteur PS/2 souris/clavier
- 2 connecteurs d’extension 40-broches avec diodes de protection.4-Mbyte de mémoire Flash (1 Mbyte sur certaines cartes).

A.2 Schéma bloc de la carte DE2 :

La figure A.2 représente le synoptique de la carte et ses différents blocs constitutifs et leurs emplacements vis-à-vis de circuit FPGA.

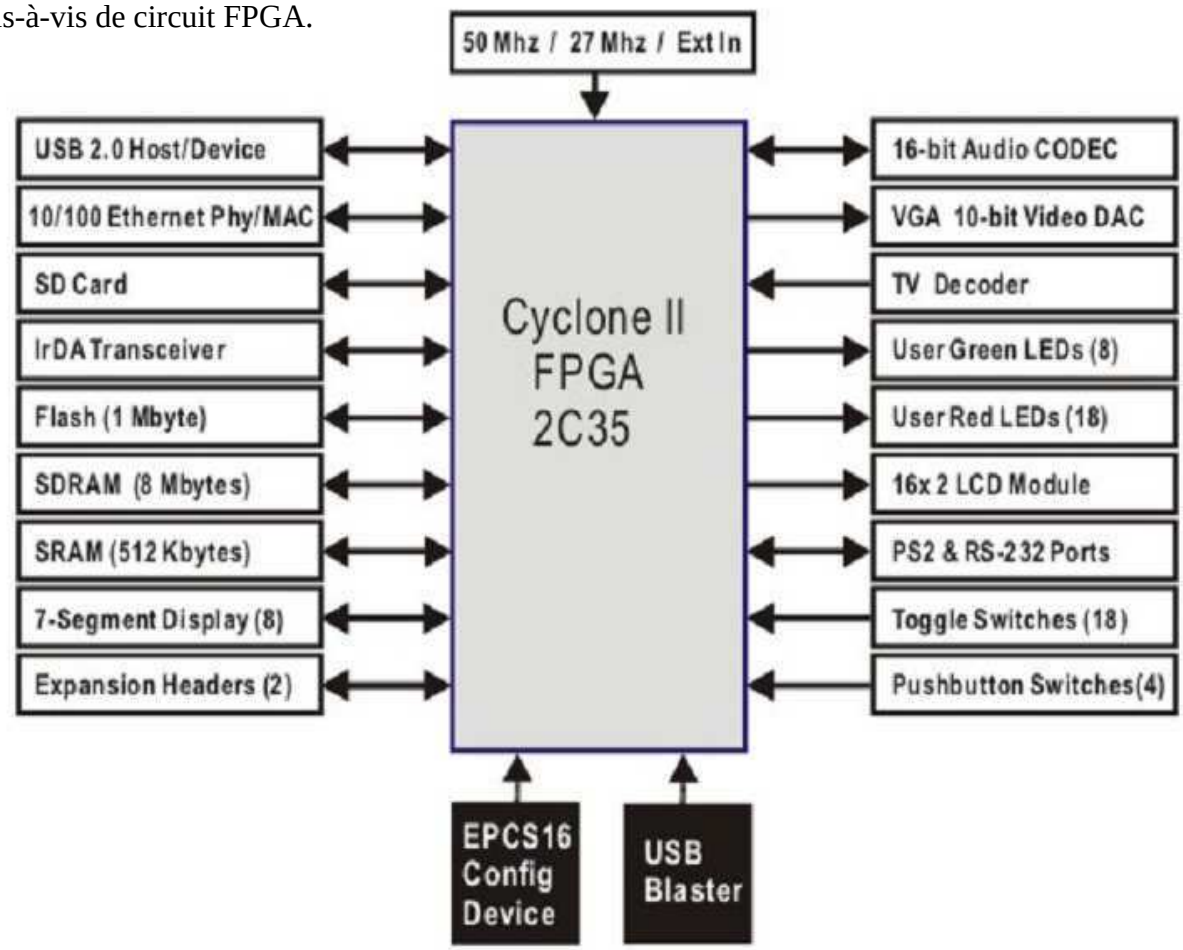

Figure A.2 : Schéma bloc de la carte DE2

A.3 Utilisation de la carte DE2 :

Ce titre donne des indications et instructions nécessaires pour l'utilisation de la carte DE2 et décrit chacun de ses périphériques d'E / S.

A.3.1 Configuration du FPGA Cyclone II

La procédure de ramener le circuit conçu à partir d'un ordinateur à la carte DE2 est décrite dans le tutoriel *Quartus II Introduction.* Ce tutoriel se trouve dans le dossier *DE2_tutorials* sur l' **DE2 CD-ROM Système** et également disponible sur le site d'Altera DE2, pour bien comprendre la procédure l'utilisateur devrais lire le tutoriel d'abord, et consulter le manuel d'utilisation. Nous montrons ci-dessous le principe de fonctionnement et d utilisation de la carte DE2.

Sur la carte on trouve série EEPROM qui enregistre les données de la configuration du Cyclone II FPGA. Ces dernières sont automatiquement chargées à partir de la mémoire EEPROM dans le FPGA. Avec l'utilisation du logiciel Quartus II, il est possible de reprogrammer le FPGA à tout moment, et il est également possible de modifier les donnéesnon volatiles (programme) qui sont stockées dans l'EEPROM.

Les deux types de programmation sont décrits ci-dessous :

- Programmation *JTAG:* Dans ce mode de programmation, nommé d'après les standards IEEE *Joint Test action Groupe,* le train de bits de configuration est chargé directement dans le cyclone II FPGA.

Le FPGA conservera cette configuration tant que la carte est alimentée, et la configuration est perdue des que l'alimentation est coupée.

- programmation *AS* : appelée programmation « Active Serial », Dans cette méthode,le flux de bits de configuration (programme) est chargé dans l'EEPROM « Altera EPCS16 ». Il permet le stockage non-volatile du train de bits, de sorte que l'information est conservée même lorsque l'alimentation de la carte DE2 est éteinte.

Lorsque la carte est alimentée, les données de configuration dans le circuit EPCS16 sont automatiquement chargées dans le FPGA Cyclone II.

Les sections suivantes décrivent les étapes à suivre pour effectuer à la fois la programmation JTAG et AS. Pour les deux méthodes, la carte DE2 est connectée à un ordinateur via un câble USB. Avec cette connexion, la carte sera identifiée par l'ordinateur comme un périphérique « *USB Blaster* Altera ». Procédé ensuite a l'installation du pilote nécessaire de périphérique.

A.3.1.1 programmation du FPGA en mode JTAG

La figure ci-dessous illustre la configuration de la programmation JTAG.

Pour charger un flux de bits dans le Cyclone II FPGA, effectuez les étapes suivantes:

• Assurer que la carte DE2 est alimentée
• Branchez le câble USB fourni au port USB Blaster sur la carte
• configurer le circuit de programmation JTAG en définissant la RUN / PROG commuter (sur la partie gauche de le conseil d'administration) à l' RUN la position.
• Le FPGA peut maintenant être programmé à l'aide de Programmeur Quartus II, le fichier a sélectionner est d'extension «. *sof* »

A.3.1.2 Programmation de l EEPROM EPCS16 en mode AS

La figure ci-dessous illustre la programmation AS mise en place. Pour charger le programme dans l'EEPROM série EPCS16, effectuez les étapes suivantes:

- Assurer que la carte DE2 est alimentée
- Branchez le câble USB fourni au port USB Blaster sur la carte
- configurer le circuit de programmation JTAG en définissant la RUN / PROG commuter (sur la partie gauche de le conseil d'administration) à l'PROG la position.
- Le circuit EPCS16 peut maintenant être programmé à l'aide de Programmeur QuartusII, sélectionnez un fichier de programmation muni de l'extension « . *pof »*
- Une fois l'opération de programmation est terminée, mettre le commutateur RUN / PROG à la position RUN, puis réinitialiser la carte sur le bouton ON/OFF. Cette action provoque la mise d'un nouveau programme dans l'EPCS16.

En plus de son utilisation pour la programmation, le port USB Blaster de la carte DE2 peut également être utilisé pour contrôler certaines fonctions de la carte à distance depuis un ordinateur.

A.3.2 Utilisation des boutons-poussoirs et des Switchs :

La carte DE2 fournit quatre boutons-poussoirs. Chacun des boutons est anti-rebonds en utilisant un trigger de Schmitt, comme c'est indiqué à la figure I.5, les quatre sorties des triggers sont *KEY0, KEY1, KEY2, KEY3,* elles sont reliées directement aux pins de l'FPGA.
Chaque bouton fournit un niveau logique haut de 3,3 volts quand il n'est pas enfoncé, et un niveau logique bas de 0 volt dans le cas contraire.
Étant donné que les boutons poussoirs sont anti-rebond, elles sont appropriées pour une utilisation comme horloge ou réinitialiser les entrées d'un circuit.
La liste des noms des pins du l'FPGA qui sont connectés aux boutons-poussoirs est donnée dans le Tableau suivant :

Signal Name	FPGA Pin No.	Description
KEY0	PIN_G26	Pushbutton[0]
KEY1	PIN_N23	Pushbutton[1]
KEY2	PIN_P23	Pushbutton[2]
KEY3	PIN_W26	Pushbutton[3]

Il ya aussi 18 Switchs sur la carte DE2. Ils ne sont pas anti-rebonds, sont destinés à être utilisés comme des entrées de données d'un circuit. Chacun d'eux est connecté directement à une broche de l'FPGA Cyclone II. Lorsqu'un interrupteur est en position vers le bas, il fournit un niveau logique bas (0 volt) pour le FPGA, et lorsqu'il est dans la position haut, fournit un niveau logique haut (3,3 volts).
La liste des noms des broches du l'FPGA qui sont connectés aux Switchs est donnée dans le Tableau suivant :

Signal Name	FPGA Pin No.	Description
SW0	PIN_N25	Toggle Switch[0]
SW1	PIN_N26	Toggle Switch[1]
SW2	PIN_P25	Toggle Switch[2]
SW3	PIN_AE14	Toggle Switch[3]
SW4	PIN_AF14	Toggle Switch[4]
SW5	PIN_AD13	Toggle Switch[5]
SW6	PIN_AC13	Toggle Switch[6]
SW7	PIN_C13	Toggle Switch[7]
SW8	PIN_B13	Toggle Switch[8]
SW9	PIN_A13	Toggle Switch[9]
SW10	PIN_N1	Toggle Switch[10]
SW11	PIN_P1	Toggle Switch[11]

SW12	PIN_P2	Toggle Switch[12]
SW13	PIN_T7	Toggle Switch[13]
SW14	PIN_U3	Toggle Switch[14]
SW15	PIN_U4	Toggle Switch[15]
SW16	PIN_V1	Toggle Switch[16]
SW17	PIN_V2	Toggle Switch[17]

A.3.3 Utilisation des LEDs :

La carte dispose de 27 LEDs contrôlables par l'utilisateur, dont 18 rouges sont situés au-dessus de la 18 Switchs, et 8 vertes se trouvent au-dessus des boutons-poussoirs, la 9eme est verte située au milieu des afficheurs 7 segments. Chaque LED est connectée directement à une broche sur l'FPGA.
La liste des noms des pins du l'FPGA qui sont connectés LEDs est donnée dans le Tableau suivant :

Signal Name	FPGA Pin No.	Description
LEDR0	PIN_AE23	LED Red[0]
LEDR1	PIN_AF23	LED Red[1]
LEDR2	PIN_AB21	LED Red[2]
LEDR3	PIN_AC22	LED Red[3]
LEDR4	PIN_AD22	LED Red[4]
LEDR5	PIN_AD23	LED Red[5]
LEDR6	PIN_AD21	LED Red[6]
LEDR7	PIN_AC21	LED Red[7]
LEDR8	PIN_AA14	LED Red[8]
LEDR9	PIN_Y13	LED Red[9]
LEDR10	PIN_AA13	LED Red[10]
LEDR11	PIN_AC14	LED Red[11]

LEDR12	PIN_AD15	LED Red[12]
LEDR13	PIN_AE15	LED Red[13]
LEDR14	PIN_AF13	LED Red[14]
LEDR15	PIN_AE13	LED Red[15]
LEDR16	PIN_AE12	LED Red[16]
LEDR17	PIN_AD12	LED Red[17]
LEDG0	PIN_AE22	LED Green[0]
LEDG1	PIN_AF22	LED Green[1]
LEDG2	PIN_W19	LED Green[2]
LEDG3	PIN_V18	LED Green[3]
LEDG4	PIN_U18	LED Green[4]
LEDG5	PIN_U17	LED Green[5]
LEDG6	PIN_AA20	LED Green[6]
LEDG7	PIN_Y18	LED Green[7]
LEDG8	PIN_Y12	LED Green[8]

A.3.4 Utilisation des afficheurs 7 segments :

La carte DE2 a huit afficheurs 7 segments. Ces derniers sont disposés en deux paires et un groupe de quatre, avec l'intention d'afficher des numéros de différentes tailles.

L'application d'un niveau logique bas à un segment amène à éclairer, et l'application d'un niveau logique haut l éteint (les afficheurs sont a anodes communes). Chaque segment d'un afficheur est identifié par un index de 0 à 6.

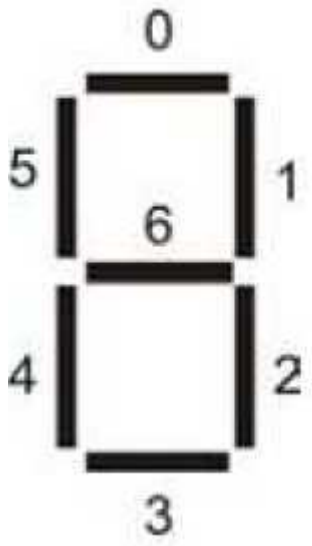

Figure A.3.4 : Position et index de chaque segment d'un afficheur 7 segments

Le Tableau suivant montre les affectations des broches du FPGA vers les afficheurs 7 segments.

Signal Name	FPGA Pin No.	Description
HEX0 0	PIN_AF10	Seven Segment Digit 0[0]
HEX0 1	PIN_AB12	Seven Segment Digit 0[1]
HEX0 2	PIN_AC12	Seven Segment Digit 0[2]
HEX0 3	PIN_AD11	Seven Segment Digit 0[3]
HEX0 4	PIN_AE11	Seven Segment Digit 0[4]
HEX0 5	PIN_V14	Seven Segment Digit 0[5]
HEX0 6	PIN_V13	Seven Segment Digit 0[6]
HEX1 0	PIN_V20	Seven Segment Digit 1[0]
HEX1 1	PIN_V21	Seven Segment Digit 1[1]
HEX1 2	PIN_W21	Seven Segment Digit 1[2]
HEX1 3	PIN_Y22	Seven Segment Digit 1[3]
HEX1 4	PIN_AA24	Seven Segment Digit 1[4]

HEX1 5	PIN_AA23	Seven Segment Digit 1[5]
HEX1 6	PIN_AB24	Seven Segment Digit 1[6]
HEX2 0	PIN_AB23	Seven Segment Digit 2[0]
HEX2 1	PIN_V22	Seven Segment Digit 2[1]
HEX2 2	PIN_AC25	Seven Segment Digit 2[2]
HEX2 3	PIN_AC26	Seven Segment Digit 2[3]
HEX2 4	PIN_AB26	Seven Segment Digit 2[4]
HEX2 5	PIN_AB25	Seven Segment Digit 2[5]
HEX2 6	PIN_Y24	Seven Segment Digit 2[6]
HEX3 0	PIN_Y23	Seven Segment Digit 3[0]
HEX3 1	PIN_AA25	Seven Segment Digit 3[1]
HEX3 2	PIN_AA26	Seven Segment Digit 3[2]
HEX3 3	PIN_Y26	Seven Segment Digit 3[3]
HEX3 4	PIN_Y25	Seven Segment Digit 3[4]
HEX3 5	PIN_U22	Seven Segment Digit 3[5]
HEX3 6	PIN_W24	Seven Segment Digit 3[6]
HEX4 0	PIN_U9	Seven Segment Digit 4[0]
HEX4 1	PIN_U1	Seven Segment Digit 4[1]
HEX4 2	PIN_U2	Seven Segment Digit 4[2]
HEX4 3	PIN_T4	Seven Segment Digit 4[3]
HEX4 4	PIN_R7	Seven Segment Digit 4[4]
HEX4 5	PIN_R6	Seven Segment Digit 4[5]
HEX4 6	PIN_T3	Seven Segment Digit 4[6]
HEX5 0	PIN_T2	Seven Segment Digit 5[0]

HEX5 1	PIN_P6	Seven Segment Digit 5[1]
HEX5 2	PIN_P7	Seven Segment Digit 5[2]
HEX5 3	PIN_T9	Seven Segment Digit 5[3]
HEX5 4	PIN_R5	Seven Segment Digit 5[4]
HEX5 5	PIN_R4	Seven Segment Digit 5[5]
HEX5 6	PIN_R3	Seven Segment Digit 5[6]
HEX6 0	PIN_R2	Seven Segment Digit 6[0]
HEX6 1	PIN_P4	Seven Segment Digit 6[1]
HEX6 2	PIN_P3	Seven Segment Digit 6[2]
HEX6 3	PIN_M2	Seven Segment Digit 6[3]
HEX6 4	PIN_M3	Seven Segment Digit 6[4]
HEX6 5	PIN_M5	Seven Segment Digit 6[5]
HEX6 6	PIN_M4	Seven Segment Digit 6[6]
HEX7 0	PIN_L3	Seven Segment Digit 7[0]
HEX7 1	PIN_L2	Seven Segment Digit 7[1]
HEX7 2	PIN_L9	Seven Segment Digit 7[2]
HEX7 3	PIN_L6	Seven Segment Digit 7[3]
HEX7 4	PIN_L7	Seven Segment Digit 7[4]
HEX7 5	PIN_P9	Seven Segment Digit 7[5]
HEX7 6	PIN_N9	Seven Segment Digit 7[6]