MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF-1-
UFAS (ALGERIE)

MEMOIRE

Présenté à la faculté de Technologie

Département d'Electronique

Pour l'obtention du Diplôme de

MAGISTER

Option : Contrôle

Par :
Mr. Moncef KHITAS

THEME

Implantation d'application de traitement de signal sur système mono-puce reconfigurable 'SoPc'

Soutenu le : / /2014 devant la commission d'examen :

Mr le Pr.F.DJAHLI	PROF – UFAS	**Président**
Mr le Dr.H.CHEMALI	MCCA – UFAS	**Examinateur**
Mr le Dr. N.BOUKEZZOULA	MCCA – UFAS	**Examinateur**
Mr le Dr.L.ZIET	MCCA – UFAS	**Rapporteur**

Dédicace

A la mémoire de mon chère Père…

Remerciement

بسم الله الرحمن الرحيم

الحمد لله الذي علّم الإنسان بالقلم...علّم الإنسان ما لم يعلم

Je tiens à remercier en premier lieu mon encadreur Mr le **Dr.L.ZIET**, son soutien scientifique et psychique était de valeur et essentiel tout au long de ce mémoire.

Je remercie les honorables membres du jury en citant le président du jury Mr. le **Pr.F.DJAHLI** et les examinateurs Mr. le **Dr.H.CHEMALI** et Mr. Le **Dr. N.BOUKEZZOULA**.

Tous ceux qui m'ont aidé, soutenu moralement et qui m'ont encouragé à poursuivre le travail de ce mémoire je les remercie chaleureusement.

Un grand merci aux membres de ma famille pour leur amour et leur soutien, en particulier mes grands-parents, mon frère et mes deux sœurs dont je suis très fière et ma chère épouse.

Merci à ma chère mère ***'FATIMA'***, elle mérite tout le respect pour tous ce qu'elle a faits pour moi et pour ce qu'elle fait chaque jour Que Dieu fait.

و آخر دعوانا...أن الحمد لله رب العالمين

Sommaire

Sommaire

Liste des figures

Chapitre 1

Chapitre 2

Liste des tables

Liste des abbreviations

ADC Analog to Digital convertor

ASIC Application Specific Integrated Circuits

DAC Digital to Analog Converter

DSP Digital Signal Processing

FF Flip-flop

FFs Flip-flops

FIR Finite Impulse Response

GPP General Purpose processor

IO Input Output

LUT Look up Table

MAC Multiply and accumulate

MSB Most Significant Bit

MSI Medium Scale Integration

MUX Multiplexer

RTL Register Transfer Level

SET Single-edge Triggered

VHDL Very High Speed Integrated Circuit Description Language

VLSI Very Large Scale Integration

Introduction générale

Pendant les dix dernières années, plusieurs architectures combinant des processeurs et/ou des circuits reconfigurables (FPGA) ont été proposées pour accélérer l'exécution d'applications de plus en plus complexes. Les systèmes dédiés au traitement du signal et de l'image utilisent actuellement soit des processeurs à usage général ou dédiés, soit des solutions câblées intégrant des circuits spécifiques de type ASIC ou la combinaison des deux. Cependant, la conjugaison de la complexité croissante des algorithmes et d'un grand volume de données à traiter, le tout avec des contraintes temps réel fortes, réclame des performances que les processeurs ne peuvent fournir. La mise en parallèle de processeurs peut apporter la rapidité et la souplesse de programmation, mais au détriment du coût et de la complexité de mise en œuvre. De plus, l'utilisation des ASICs qui offrent une plus grande vitesse de traitement souffre d'une certaine rigidité et d'un cycle de développement onéreux. En effet, ajouter une nouvelle fonctionnalité à un système câblé nécessitera sûrement une nouvelle conception d'un ou plusieurs ASICs faisant accroître les coûts. Offrant de meilleures performances par rapport aux architectures programmables et donnant plus de flexibilité par rapport aux solutions câblées, les architectures reconfigurables représentent une réponse intermédiaire appropriée, Elles utilisent des composants logiques reconfigurables qui permettent à l'utilisateur de modifier l'architecture après fabrication de façon logicielle, contrairement aux ASICs dont les algorithmes sont câblés dans le silicium.

Le terme reconfiguration désigne l'opération qui vise à implanter dans le ou les composants une nouvelle fonctionnalité sans modification de l'architecture matérielle du système.

Dans cette optique, les FPGAs SRAM, par leur reconfigurabilité rapide, insitu et illimitée, offrent des perspectives riches dans le domaine du traitement d'image bas niveau (TIBN). La tâche qui nous incombe, entre dans ce contexte et consiste à exploiter les performances des circuits FPGAs en particuliers leurs version récentes (Altera Cyclone-II à cyclone-IV) pour montrer l'efficacité d'implémenter des applications liées au traitement de signal.

A travers ce travail, nous avons exploité les performances des circuits FPGAs pour montrer l'efficacité d'y implémenter des applications liées au traitement de

signal et cela en grande parti grâce la tendance au parallélisme dans le calcul permis par les FPGAs.

Dans ce mémoire, nous avons commencé par donner des notions sur les FPGA et leurs performances au chapitre 1, avant de présenter dans le chapitre 2 les différents outils de CAO accompagnant ces plateformes d'implémentations (Quartus, modelSim, SOPC-Builder).

Dans le chapitre 3, nous avons rappelé les notions de filtrage numérique en insistant plus sur la catégorie des filtres à réponse finie (FIR).

Dans le chapitre 4, nous avons fait la synthèse des éléments constituant la conception des Filtres FIR, nous avons montré comment faire l'implémentation en utilisant un langage de description matérielle et cela par la présentation des résultats niveau d'abstraction RTL pour la synthèse tout en élaborant des modèles de test pour la simulation.

Chapitre 1

Introduction aux FPGAs

1.1. Introduction

Au niveau le plus élevé, un FPGA est un circuit en silicium reprogrammable. A l'aide de blocs logiques préconstruits et de ressources de routage programmables, On peut configurer ce circuit afin de mettre en œuvre des fonctionnalités matérielles personnalisées, sans avoir jamais besoin d'utiliser une maquette ou un fer à souder. Il nous suffit de développer des tâches de traitement numérique par logiciel et de les compiler sous forme de fichier de configuration ou de flux de bits contenant des informations sur la manière dont les composants doivent être reliés. En outre, les FPGAs sont totalement reconfigurables et peuvent adopter instantanément une nouvelle « personnalité » si on recompile une nouvelle configuration de circuits. Jusqu'à présent, seuls des ingénieurs particulièrement expérimentés en matière de conception de matériel numérique pouvaient utiliser la technologie FPGA. Toutefois, la généralisation des outils de conception de haut niveau est en train de modifier les règles de la programmation de l'FPGA, grâce à de nouvelles technologies permettant de convertir des diagrammes graphiques ou même du code C ANSI 'American National Standards Institute' en circuits matériels numériques.

Si les FPGAs rencontrent un tel succès dans tous les secteurs, c'est parce qu'ils réunissent le meilleur des ASIC et des systèmes basés processeur. Ainsi, ils offrent un cadencement par matériel qui leur assure vitesse et fiabilité, mais sont plus rentables que les ASIC personnalisés. Les circuits logiques reprogrammables jouissent également de la même souplesse d'exécution logicielle qu'un système basé processeur, mais ils ne sont pas limités par le nombre de cœurs de traitement disponibles. Contrairement aux processeurs, les FPGAs sont vraiment parallèles par nature, de sorte que plusieurs opérations de traitement différentes ne se trouvent pas en concurrence pour l'utilisation des ressources. Chaque tâche de traitement indépendante est affectée à une section spécifique du circuit, et peut donc s'exécuter en toute autonomie sans dépendre aucunement des autres blocs logiques. En conséquence, on peut accroître le volume de traitement effectué sans que les performances d'une partie de l'application n'en soient affectées pour autant.

1.2. Les circuits FPGAs

Les FPGAs font partie de la famille de composants électriquement programmables. Initialement constitués de réseaux de matrices de portes élémentaires ET et OU (Program Array Logic ou PAL), ces circuits programmables sont devenus au milieu des années 90 des circuits plus complexes grâce à l'intégration de ressources spécifiques. Les circuits FPGA sont constitués de trois parties : une matrice d'éléments logiques configurables CLB (Configurable Logic Bloc), des blocs d'entrées/sorties et d'un réseau d'interconnexions programmable.

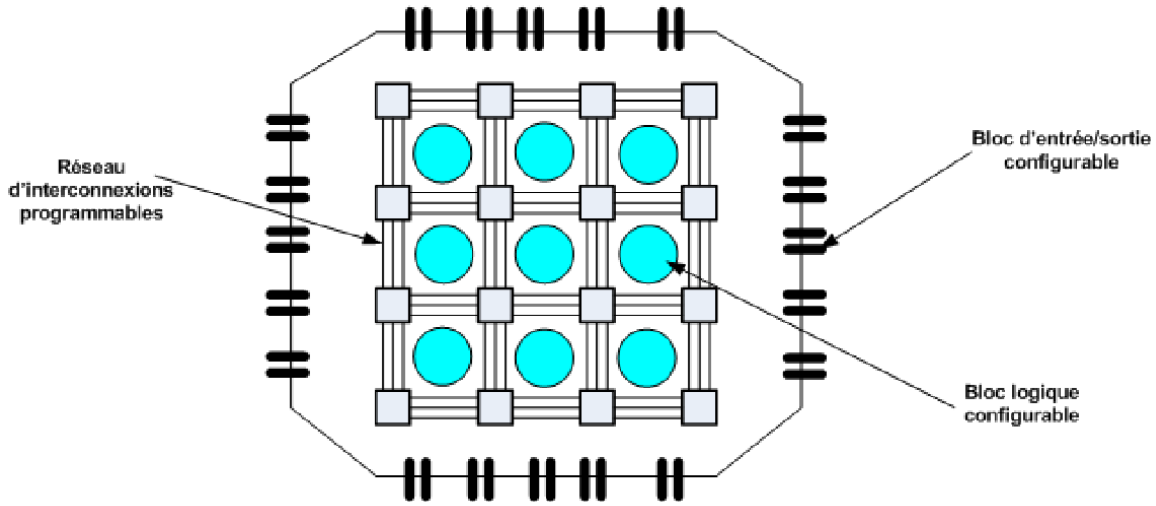


Figure 1.1 ***L'architecture générique d'un circuit FPGA***

Comparés à d'autres circuits programmables, les circuits FPGA possèdent de meilleures performances au niveau des communications inter macro-cellules grâce à un réseau d'interconnexions plus performant. En plus du niveau d'intégration, la différence entre les circuits CPLD et FPGA, par exemple, réside dans la maîtrise du temps de propagation dans les couches logiques du circuit. Ce temps est prédictif dans les circuits CPLD car les chemins parcourus par les signaux sont connus alors que dans les circuits FPGA, ce temps dépend de l'organisation et de la distance entre les macro-cellules interconnectées.

D'autre part, les circuits FPGAs peuvent contenir suffisamment de mémoire qui peut être configurée pour héberger un cœur de processeur ou un DSP afin d'exécuter une

application logicielle. Les éléments de base constituant un circuit FPGA sont constitués d'une partie combinatoire et d'une partie séquentielle.

- La partie combinatoire est essentiellement utilisée pour réaliser des fonctions arithmétiques et logiques de complexité variable. Les Fabricants de FPGA proposent plusieurs approches pour implanter les architectures développées sur les ressources matérielles disponibles. Il est en effet possible d'utiliser plusieurs méthodes de synthèse dont les principales sont : la synthèse de fonctions à 4 ou 5 variables avec des portes classiques ET, OU et NON, la synthèse de fonctions à l'aide de multiplexeurs, la synthèse de fonctions combinatoires à l'aide de mémoires vives. Dans ce dernier cas, on dit aussi réalisation de fonctions logique par LUT (Look-Up Table) signifiant table de réalisation (ou d'observation).
- La partie séquentielle comporte en règle générale une ou deux bascules de type D.

Les familles de FPGAs les plus réputées sont ceux de XLINX et ALTERA.

1.3. Principaux caractéristiques de la technologie FPGA [5]

1.3.1. Performances

Comme ils tirent parti du parallélisme matériel, les FPGAs offrent une puissance de calcul supérieure à celle des processeurs de signaux numériques (DSP), car ils s'affranchissent du modèle d'exécution séquentielle et exécutent plus d'opérations par cycle d'horloge. BDTI, une importante société d'analyse et de "benchmarking", a publié des études montrant que les FPGAs peuvent offrir une puissance de traitement par dollar plusieurs fois supérieure à celle d'une solution DSP dans certaines applications. Contrôler les entrées et sorties (E/S) au niveau matériel permet d'obtenir des temps de réponse plus courts ainsi que des fonctionnalités spécifiques, qui répondent mieux aux besoins de l'application.

1.3.2. Temps de mise sur le marché

Face à des préoccupations croissantes concernant les temps de mise sur le marché, la technologie FPGA représente une solution souple offrant des capacités de prototypage rapide. Ainsi, on peut tester une idée ou un concept, puis le vérifier sur du matériel sans avoir à passer par le long processus de fabrication d'un ASIC personnalisé. Par la suite,

on peut apporter les éventuelles modifications nécessaires à notre FPGA, en quelques heures au lieu de quelques semaines. Le matériel actuellement commercialisé propose également différents types d'E/S déjà connectées à un circuit FPGA programmable par l'utilisateur. La multiplication des outils logiciels de haut niveau disponibles sur le marché permet de réduire le temps d'apprentissage. Ces outils comprennent souvent des cœurs de propriété intellectuelle (fonctions précompilées) utiles pour le contrôle avancé et le traitement de signaux.

1.3.3. Coût

Les coûts d'ingénierie non récurrents (NRE) des ASIC personnalisés sont bien supérieurs à ceux des solutions matérielles basées sur du FPGA. L'important investissement de départ que requièrent les ASIC se justifie largement pour les OEM. Cependant, la plupart des utilisateurs finaux ont besoin de matériels personnalisés pour quelques dizaines ou quelques centaines de systèmes en développement. Par nature, les circuits programmables n'impliquent ni coût de fabrication, ni longs délais d'assemblage. Les besoins de la plupart des systèmes évoluent avec le temps ; or la modification progressive d'un FPGA représente un coût négligeable comparé à la dépense considérable qu'exige la re-conception d'un ASIC.

1.3.4. Fiabilité

Tandis que les outils logiciels fournissent l'environnement de programmation, les circuits FPGAs sont une véritable implémentation matérielle de l'exécution logicielle. Les systèmes basés processeur comprennent souvent plusieurs couches d'abstraction, pour aider à la planification des tâches et à la répartition des ressources entre les différents processus. La couche de driver contrôle les ressources matérielles et le système d'exploitation gère la mémoire et la bande passante du processeur. Sur chaque cœur de processeur, une seule instruction peut s'exécuter à la fois ; c'est pourquoi les systèmes basés processeur risquent toujours de voir des tâches prioritaires entrer en conflit. Les FPGA, qui n'utilisent pas de système d'exploitation, minimisent les problèmes de fiabilité car ils assurent une exécution véritablement parallèle et un matériel déterministe dédié à chaque tâche.

1.3.5. Maintenance à long terme

Les circuits FPGAs sont évolutifs et nous épargnent donc la dépense de temps et d'argent qu'implique la re-conception des ASIC. Les spécifications des protocoles de communication numériques, par exemple, évoluent avec le temps. Or les interfaces basées sur ASIC peuvent poser des problèmes de maintenance et de compatibilité. Comme ils sont reconfigurables, les circuits FPGAs sont capables de s'adapter aux modifications éventuellement nécessaires. A mesure qu'un produit ou qu'un système évolue, on peut y intégrer des améliorations fonctionnelles sans perdre de temps à reconcevoir le matériel ou à modifier l'implantation du circuit.

1.4. Choix du FPGA

Lorsque on étudie les spécifications d'un circuit FPGA, souvenons-nous qu'ils sont souvent constitués de différents éléments : des blocs logiques configurables, comme des slices (tranches) ou des cellules logiques ; de la logique fixe, comme des multiplicateurs ; des ressources de mémoire, comme des blocs de RAM intégrés. Bien que les FPGAs contiennent bien d'autres composants, ceux mentionnés sont généralement les plus déterminants lorsqu'il s'agit de choisir et de comparer des FPGAs pour une application particulière. Le tableau de comparaison ci-dessous montre les spécifications de différentes familles de FPGAs de Xilinx. Le nombre de portes est un moyen habituel de comparer la taille des circuits FPGAs par rapport à la technologie ASIC ; cependant ce critère ne donne pas vraiment le nombre de composants qui constituent un FPGA. C'est notamment pourquoi Xilinx n'a pas précisé le nombre de portes pour sa nouvelle famille Virtex-5.

	Virtex-II 1000	Virtex-II 3000	Spartan-3 1000	Spartan-3 2000	Virtex-5 LX30	Virtex-5 LX50	Virtex-5 LX85	Virtex-5 LX110
Nombre de portes	1 million	3 millions	1 million	2 millions	-----	-----	-----	-----
Bascules Flip-Flop	10 240	28 672	15 360	40 960	19 200	28 800	51 840	69 120
LUT	10 240	28 672	15 360	40 960	19 200	28 800	51 840	69 120
Multipli-cateurs	40	96	24	40	32	48	48	64
Blocs de RAM (kbits)	720	1 728	432	720	1 152	1 728	3 456	4 608

Table 1.1 Spécifications de différentes familles de FPGAs de Xilinx

Actuellement Deux tendances fortes se confirment autour du niveau de performance que doivent permettre les ressources matérielles des circuits reconfigurables. Les deux catégories se distinguent principalement par leur granularité qui représente la taille des cellules logiques élémentaires que l'utilisateur peut manipuler. On parle de technologie à grain fin pour l'une, et à grain épais pour l'autre. Les circuits FPGAs sont classés dans la première catégorie. Ils sont constitués d'une structure matricielle de blocs logiques et d'un réseau d'interconnexions qui peuvent être exploités pour construire des unités de traitement à l'échelle du bit. La majorité des projets de recherche, utilisent des composants FPGAs du commerce pour concevoir des systèmes numériques en manipulant des portes logiques et des registres (bascules). Les FPGAs sont constitués en général de tables de codage ou LUT (Look-Up Table) connectées entre elles par un réseau de routage hiérarchisé et reconfigurable, Chaque LUT associée à un ou plusieurs éléments de mémorisation ou FF (Flip-Flop), permet la réalisation d'une à deux fonctions logiques avec un nombre d'entrées variable, de quatre à six selon les familles et les fabricants. Les FPGAs reconfigurables les plus répandus utilisent un modèle de configuration basé sur des SRAM programmables. La deuxième catégorie, les architectures reconfigurables à grain épais, est plus récente. Elles intègrent des unités de calcul complètes comme des multiplieurs et des cœurs de DSP de taille plus grande que les blocs logiques des FPGAs. La largeur des mots que les opérateurs manipulent est souvent de plusieurs bits. La reconfiguration de ce type d'architectures peut se réduire à définir les communications entre les unités en programmant les chemins de données. En raison de la distribution spatiale des ressources de calcul, les FPGAs offrent une bonne densité fonctionnelle par unité de temps. Ainsi, ils autorisent une puissance de calcul élevée en exploitant un parallélisme massif inhérent à leur structure. Le concepteur a la possibilité de contrôler les opérations à l'échelle du bit, ce qui permet une meilleure optimisation de l'architecture pour une application donnée. Un autre avantage important est la maturité atteinte par les outils de conception qui, conjuguée avec la caractéristique intrinsèque des FPGAs à être reconfigurés, ont fait leur popularité notamment dans le prototypage rapide. Les FPGAs par la régularité de leur structure et les ressources logiques intégrées nécessaires à la reconfiguration accroissent la complexité des composants, ainsi que la consommation. De plus, la finesse des cellules élémentaires

pénalise la rapidité des opérateurs élaborés en exploitant des ressources réparties sur plusieurs blocs. Certaines familles de FPGA limitent l'effet de cette contrainte en câblant dans le silicium des multiplieurs (Virtex de Xilinx) et des cœurs de DSP (Stratix d'Altera). La démarche qui consiste à intégrer toujours plus d'opérateurs spécialisés aboutit à une consommation élevée et à une complexité accrue des méthodes de conception conduisant à une augmentation du temps de placement/routage (ou P&R) indispensable pour mapper les ressources dans le circuit cible. Les architectures reconfigurables à grain épais sont destinées à des applications manipulant des données à l'échelle d'un mot (8, 16 ou 32 bits). Du fait que les cellules logiques élémentaires sont optimisées pour effectuer des opérations sur des données de grande largeur, leurs performances sur ce type de traitement dépassent celles des structures à grain fin. Les opérations essentielles telles que les additions, les multiplications et les opérations d'accumulations étant pour l'essentiel optimisées dans le silicium, la quantité de données nécessaires à la reconfiguration est réduite. Un cycle de compilation allégé et un temps de reconfiguration réduit font de ces circuits un choix judicieux pour construire des architectures reconfigurables dynamiquement. Toutefois, la taille des blocs logiques étant définie à la fabrication, la structure du circuit ne peut optimiser la réalisation d'opérateurs ayant des tailles autres que celles qui sont prévues. Il est notamment très coûteux d'implanter des algorithmes manipulant des bits ou des machines à états finis. Nous nous intéressons plus particulièrement aux FPGA-SRAM. Le choix de cette technologie est dicté par plusieurs impératifs:

- simplicité, facilité de mise au point et rapidité de prototypage
- disponibilité des composants dans le commerce qui, malgré leurs limitations, suffisent pour valider un concept
- disponibilité d'outils logiciels performants et de langages de description évolués.

1.5. Notion du temps réel

Généralement, ce terme exprime un mode de fonctionnement dans le processus d’acquisition et de traitement de données. Une première définition, plus générale, consiste qu’un système est dit temps réel lorsque l'information après acquisition et

traitement reste encore pertinente. Cela veut dire que dans le cas d'une information arrivant de façon périodique, les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information. Un temps maximum d'exécution est garanti et non pas un temps moyen. Nous pouvons dire que le temps réel représente la disponibilité immédiate (à temps) d'informations pour l'utilisateur.

Une autre définition, spécialement pour les systèmes numériques, considère qu'un système temps réel est une association logiciel/matériel où le logiciel permet, entre autres, une gestion adéquate des ressources matérielles en vue de remplir certaines tâches ou fonctions dans des limites temporelles bien précises.

1.5.1. Les avantages du FPGA pour le traitement d'images en temps réel [8]

La majeure partie des traitements d'images de bas niveau peut être séparée en deux types de tâches. Le premier type de tâches est assimilable à des opérations à coefficient fixe qui sont effectuées identiquement sur chaque pixel de l'image. Le deuxième type regroupe des traitements utilisant le voisinage immédiat de chaque pixel de l'image (convolution). Dans ce cas, le résultat généré pour chaque pixel est lié à une fenêtre de pixels centrés autour de celui-ci. Ces opérations prouvent qu'il y a un degré élevé de répétition du traitement à travers l'image entière. Ce type de traitement se prête à une implantation basée sur un modèle en pipeline sur un FPGA qui peut exécuter les mêmes opérations mathématiques sur un flot de données. Les FPGAs fournissent un grand nombre de blocs logiques où chaque bloc contient plusieurs bascules et LUTs capables de mettre en application de nombreuses fonctions logiques. En outre, des ressources dédiées pour la multiplication et le stockage mémoire sont disponibles et permettent d'améliorer l'exécution et les performances. De telles ressources matérielles permettent l'implantation des traitements d'images aux débits très élevés, avec des taux atteignant plusieurs centaines de MHz. Grâce au mode pipeline, ces tâches peuvent être directement exécutées sur le flot de données provenant de la caméra sans incorporer de délai de traitement supplémentaire. Les possibles "goulots d'étranglements" du flot de données, qui limitent les performances, peuvent être ainsi réduits et dans certains cas supprimés. Ainsi, les tâches plus complexes telles que la convolution peuvent être implantées avec succès sur les FPGAs. Le procédé global de convolution est une

multiplication de matrice et exige plusieurs multiplications afin d'être exécutée pour chaque pixel. Le nombre exact de multiplieurs nécessaires dépend de la taille de la fenêtre utilisée pour la convolution. Pour une fenêtre 3x3, 9 multiplieurs sont nécessaires, et pour une fenêtre 5x5, 25 multiplieurs sont requis, etc. Les FPGAs possèdent un grand nombre de multiplieurs. Par exemple, avec le Virtex-II-1000 (un million de portes logiques), 40 multiplieurs sont disponibles et dans la version 8000 (huit millions de portes), on augmente ce nombre à 168. La complexité croissante des systèmes pour lesquels la réalisation résulte de l'association d'une partie matérielle et d'une partie logicielle, la diversité des choix technologiques et les contraintes de coûts et délais de plus en plus sévères nécessitent l'utilisation de nouvelles méthodologies et outils logiciels associés pour diminuer leur durée de conception et accroître leur qualité. Le co-design est l'une de ces méthodologies.

1.6. Le CO-DESIGN

1.6.1. Préambule

Dans la méthodologie traditionnelle de conception de systèmes, les conceptions des parties logicielles et matérielles sont développées de manière indépendante et ce dès les premières étapes. Ces étapes demeurent indépendantes et ont peu d'interaction. Cette approche restreint les possibilités d'exploration de différentes solutions où certaines fonctionnalités qui permettraient de migrer du logiciel vers le matériel ou vice-versa. Or les dernières avancées technologiques ont permis des développements de composants spécifiques (ASICs) pour un coût et dans un temps raisonnables . Ceci suggère une méthodologie de conception plus souple, où le logiciel et le matériel peuvent être conçus en interaction. Une telle approche doit permettre la réalisation de systèmes dédiés à des applications spécifiques - traitement du signal, télécommunications, où le choix d'implantation matériel/logiciel peut évoluer en fonction des besoins - certaines fonctions réalisées en logiciel au départ pouvant être intégrées au matériel par la suite - ce qui rend possible l'évolution des produits en fonction de la technologie (matérielle/logicielle) et par là même une arrivée plus rapide sur le marché. Deux problèmes clés sont à aborder dans la co-spécification d'une part la spécification du système d'autre part le partage matériel/logiciel. Si l'on veut pouvoir spécifier sans a priori matériel ou logiciel, en prenant

en compte le système dans son ensemble l'utilisation de spécifications formelles s'impose. L'étape de répartition logiciel/matériel consiste à déterminer les parties du système qui seront réalisées en matériel et celles qui seront réalisées avec du logiciel. A partir des spécifications, formelles ou informelles, ou d'un algorithme décrivant le comportement du système, il faut trouver la meilleure réalisation possible en terme de surface, de coût, et de respect des contraintes temporelles qui satisfont les spécifications initiales. Cette répartition est en général faite par le concepteur en fonction de certains critères et de certaines contraintes (vitesse, complexité, . . .).

1.6.2. Partitionnement Logiciel/Matériel

Dans une stratégie de conception conjointe, la co-spécification a pour but de permettre au concepteur d'effectuer la spécification d'un système sans se soucier du découpage sur des architectures hétérogènes logicielles et matérielles. Cette spécification abstraite (sans choix d'implantation) est ensuite raffinée (choix d'algorithmes particuliers, choix des représentations des structures de données, . . .) pour aboutir à une spécification dans laquelle il est possible d'identifier les fonctionnalités à réaliser. Ces fonctionnalités de base coïncident le plus souvent avec les objets de plus bas niveau spécifiés dans la spécification la plus raffinée. L'étape suivante de partitionnement logiciel/matériel consiste à rechercher pour chaque fonctionnalité une réalisation soit logicielle soit matérielle. L'objectif du partitionnement est de trouver le meilleur compromis entre les parties logicielles et matérielles en fonction de leurs interactions et des contraintes dictées notamment par des critères de performances et de réutilisation.

1.6.3. Caractéristiques du matériel et du logiciel

Le tableau suivant résume ce qui caractérise les composants logiciels et matériels d'un point de vue performance, flexibilité et temps de conception. Aux deux extrêmes, on trouve d'un côté le processeur standard offrant la souplesse de la programmation et de l'autre l'ASIC permettant des temps de traitement optimaux. Entre les deux, apparaissent des solutions intermédiaires telles qu'un processeur « brut » (sans interface d'entrées/sorties ou de mémoire associées) et un processeur dédié (DSP par exemple) offrant des compromis souplesse-performance satisfaisants pour une large classe d'applications.

	Processeur Standard	Cœur de Processeur	ASIP	ASIC
Performance	Moyenne	Moyenne	Elevée	La plus élevée
Puissance consommée	Elevée	Moyenne	Faible	La plus faible
Flexibilité	Moyenne	Elevée	Elevée	Faible
Temps de conception	Faible	Moyen	Le plus élevée	Elevée

Table 1.2 Compromis suivant différentes approches de conception

Lors de la conception de systèmes hétérogènes, le matériel est plutôt dédié aux parties nécessitant des performances élevées. Par contre, le logiciel est préféré pour des raisons de coût, de facilité d'évolution, de temps de développement plus court et aussi parce qu'il permet des modifications tardives dans la spécification. Lorsque l'on regarde l'évolution d'un produit industriel (par opposition aux considérations techniques ci-dessus), nous devons préférer une conception logicielle d'un produit dans le début de son cycle de vie pour répondre plus facilement aux évolutions et améliorations de ce produit. Finalement, une réalisation plutôt matérielle à sa maturité permettra une meilleure fiabilité, de meilleures performances et un coût de production plus faible.

1.6.4. Problématique du partitionnement

Le problème du partitionnement se pose dès que nous disposons de plus d'un composant pour réaliser une tâche. Le problème est de déterminer « qui » réalise chaque tâche (ou fonctionnalité). C'est un des problèmes classique de l'allocation des ressources. Or, les contraintes liées au partage des ressources dépendent de l'ordonnancement des tâches. En effet, le parallélisme d'action dépend du nombre de ressources disponibles sur l'architecture. Autrement dit, pour répondre correctement au « qui » il faut répondre au « quand » et réciproquement. Il est donc nécessaire d'aborder la problématique dans son ensemble. Précisément, pour des systèmes embarqués orientés traitement de données, la problématique est pour chaque tâche :

- De déterminer sa date d'exécution (Ordonnancement),
- De choisir une réalisation logicielle ou matérielle (Partitionnement),
- De définir le nombre et le type de ressources pour implanter le système (Allocation).

L'ordonnancement, le partitionnement et l'allocation sont trois problèmes interdépendants. L'idéal serait de les réaliser simultanément. Cependant, pour échapper à la trop grande complexité du problème général, il faut réaliser des approximations guidées par des choix locaux. Ainsi dans la majorité des cas, l'allocation, le partitionnement et l'ordonnancement sont réalisés séparément. Pour renseigner le concepteur sur la qualité des solutions trouvées, on utilise des estimateurs (estimations des performances, des coûts, . . .) afin de prédire les résultats de la conception sans aller jusqu'à la réalisation du système. Toutes ces considérations amènent une nouvelle façon d'envisager la conception de systèmes où le partitionnement logiciel/matériel devient de plus en plus stratégique.

1.6.5. Les classes de systèmes concernées par le co-design

Le développement des systèmes électroniques composés d'une partie matérielle et d'une partie logicielle nécessite une compétence technique dans au moins 3 domaines, qui sont l'électronique analogique, l'électronique numérique et l'informatique. La nature spécifique du traitement à effectuer et le couplage du système avec son environnement nécessitent aussi des compétences complémentaires: en traitement de l'information (signal, image, parole...), en électronique de puissance lorsque l'environnement utilise des courants forts, en réseaux et télécommunications, etc.
Une analyse des types de systèmes conduit à la classification suivante:

- les systèmes typiquement électroniques qui impliquent essentiellement le développement de matériel. Ils ne font donc pas directement partie du domaine de l'activité de co-design même si l'apparition de composants matériels programmables, des langages de description de matériel (VHDL, Verilog) et des outils de synthèse tend à rendre de plus en plus floue la frontière entre matériel et logiciel.
- Les systèmes interactifs concernés par l'exploitation d'interfaces pour le dialogue homme-machine. Ces systèmes sont constitués principalement de logiciels (système d'information exploitant une base de données par exemple). Ils ne font donc pas partie du champ d'application de l'activité de co-design.
- Les systèmes de communications dominés par le transfert d'informations. Ces systèmes reçoivent, transforment et émettent des flots de messages. Ils sont

représentatifs des systèmes pour lesquels les réalisations autrefois à dominante matérielle ont progressivement évoluées vers un partage matériel/logiciel avec un accroissement de la partie logicielle car les microprocesseurs sont devenus de plus en plus puissants. Mais, une partie matérielle est pour ce type de systèmes indispensable pour respecter les débits élevés des protocoles de communications.

- Les systèmes de traitement concernés principalement par les traitements de toutes formes d'informations: signal, image, parole... Ces systèmes sont caractérisés par des techniques de conception orientées flot de données. Tout comme pour les systèmes de communications, les réalisations de ces systèmes étaient autrefois surtout matérielles et basées sur une partie opérative et une partie contrôle. Puis peu à peu, l'arrivée de microprocesseurs spécifiques (DSP) de plus en plus performants a donné une part de plus en plus importante au logiciel. Aujourd'hui, le développement de ces systèmes est bien maîtrisé et repose sur l'utilisation de méthodes et outils mais limités uniquement à leur domaine d'application (approche algorithmique, synthèse haut niveau,…).
- Les systèmes de contrôle/commande dominés par des problèmes de suivi et de commande pour des applications incluant des procédés physiques en tous genres à piloter. Ils font partie d'un ensemble (système+environnement) composé d'actionneurs qu'ils contrôlent en fonction des événements perçus par des capteurs.

1.6.6. Méthodologie pour le co-design

Le processus de co-design est représenté par la ***figure 1.1***.L'activité de co-design est intégrée dans la démarche d'ingénierie système. Les spécifications de la partie relevant du co-design résultent de l'approche système qui consiste en la spécification, la conception fonctionnelle et le partitionnement au niveau système. A partir de ces spécifications, le concepteur doit définir l'architecture matérielle constituée de processeurs matériels (ASICs, FPGA, composants standards) et de processeurs logiciels (MPU, ASIP, DSP) et l'allocation des éléments de la solution fonctionnelle sur cette architecture: c'est la problématique communément appelée partitionnement matériel/logiciel. Le terme allocation correspond à la sélection d'une architecture et le terme partitionnement à la répartition des éléments de la description fonctionnelle sur l'architecture.

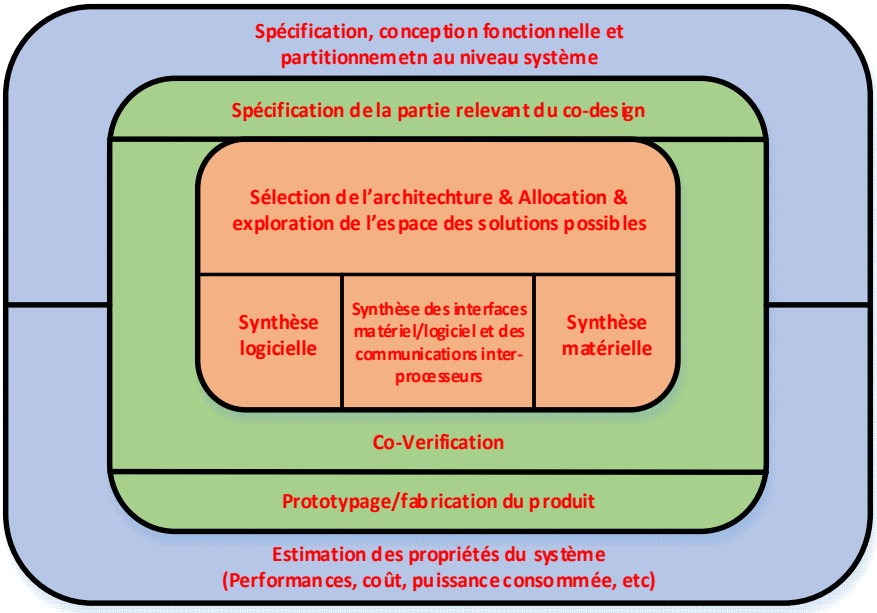


Figure 1.2 ***Représentation du processus de co-design***

Un balayage rapide de l'espace des solutions conduit à sélectionner la solution la plus appropriée. Ce parcours de l'espace des solutions possibles peut être automatique ou interactif et repose toujours sur une estimation des propriétés résultantes du partitionnement choisi. Les propriétés retenues sont généralement des contraintes de coût et de performances statiques (taille, puissance consommée) ou dynamiques (temps de réponse, charge du processeur). Une fois le partitionnement matériel/logiciel terminé, les descriptions fonctionnelles sont transformées en code machine pour une implantation en logiciel et en un ensemble de portes logiques et de bistables pour une implantation en matériel. C'est le rôle de la phase de co-synthèse qui concerne la synthèse des parties matérielles et logicielles et la synthèse des interfaces matériel/logiciel. Avant et après synthèse, une vérification fonctionnelle détaillée (co-verification) est effectuée généralement par une technique de co-simulation. Un prototype ou directement le produit est le résultat en sortie de l'activité de co-design et s'intègre alors dans le système global.

Plusieurs aspects de la conception conjointe des systèmes matériels et logiciels sont actuellement étudiés:

- La co-spécification qui concerne les étapes de spécification et de conception fonctionnelle et pour lesquelles beaucoup de formalismes existent. Ces formalismes

(modèles et langages) servent à décrire principalement l'aspect fonctionnel. Les spécifications non-fonctionnelles telles que les performances, la fiabilité, la sécurité, le coût, sont beaucoup plus délicates à formaliser et pourtant elles sont essentielles pour décider d'un partitionnement approprié et justifié. A ce stade, les différences de culture et de jargon entre le monde du génie logiciel et du génie matériel ont nécessité des efforts d'uniformisation de la terminologie.

- Le partitionnement matériel/logiciel qui concerne l'étape de conception architecturale. Dans le cas général, l'architecture matérielle peut être quelconque, c'est-à-dire hétérogène et distribuée. Plus l'architecture est particularisée, plus les concepts et les outils associés sont actuellement avancés. Souvent, l'architecture générique cible est choisie. Dans un ensemble de travaux on retrouve très souvent l'emploi d'une architecture du type maître/esclave utilisant un microprocesseur conventionnel ou un microcontrôleur comme maître associé à un ou plusieurs ASICs comme esclaves. Il résulte que la solution est fortement dépendante de l'architecture et des composants standards utilisés pour la réalisation.
- La co-simulation qui concerne l'étape de conception architecturale pour la co-simulation fonctionnelle et l'étape de réalisation pour la co-vérification détaillée. La co-simulation est une simulation de la description mixte matériel-logiciel résultant d'un partitionnement. La difficulté de la co-simulation est liée à la différence de modèles et/ou de niveaux d'abstraction des représentations du logiciel et du matériel. Les solutions de co-simulation reposent sur l'emploi d'un langage de représentation unique et exécutable ou pour une simulation hétérogène sur l'utilisation d'un mécanisme de communications entre différents simulateurs.
- La co-synthèse (étape de réalisation) qui se charge de transformer les descriptions fonctionnelles en descriptions directement implantables sur les processeurs matériels et logiciels de l'architecture cible. Considérés séparément, le développement des constituants logiciels et matériels (synthèse logique et haut niveau) sont aujourd'hui bien maîtrisés. Les outils de synthèse logique sont très robustes. La synthèse logicielle souffre cependant de deux problèmes: le manque d'outil de compilation efficace pour des microprocesseurs spécifiques, et le manque d'outil de génération de code exécutable pour une description multitâches.

1.6.7. Technique Pour L'évaluation De Performance [5]

Rappelons que la procédure itérative à suivre par les concepteurs se décompose en 3 phases:

- Modélisation incrémentale du système ou de l'application.
- Evaluation des performances souhaitées.
- Visualisation des résultats, interprétations et prise de décision.

La troisième phase conduit généralement à un retour sur la première phase pour modifier, améliorer, poursuivre la décomposition du modèle, ou à un retour sur la deuxième phase pour modifier la sélection des résultats demandés ou des paramètres de l'évaluation. A noter que la deuxième phase se doit d'être la plus transparente et la plus efficace possible pour l'utilisateur, l'objectif de ce dernier étant de permettre d'itérer rapidement entre les phases 3 et 1.

Le choix d'une méthode spécifique pour l'évaluation passe par une analyse des possibilités. D'une manière générale, l'observation des propriétés d'un système peut résulter de 3 techniques différentes:

- L'évaluation analytique (ou mathématique).
- La simulation.
- L'observation et la mesure.

L'évaluation analytique est appropriée pour un modèle dont tous les éléments sont décrits d'une manière analytique, et si en plus il existe une méthode globale directe ou itérative permettant de déduire les propriétés globales du modèle à partir de ses constituants. L'observation et la mesure ne s'appliquent que sur un système réel que l'on place en fonctionnement. Ceci suppose que toute la conception soit achevée et qu'un prototype existe. La simulation est une technique appropriée pour un modèle dynamique. Ainsi, c'est la seule technique utilisable durant la spécification et la conception des systèmes matériels/logiciels. Deux techniques de simulation sont possibles: l'utilisation d'un simulateur spécifique pour le modèle ou la transcription du modèle en un langage pour lequel un simulateur existe.

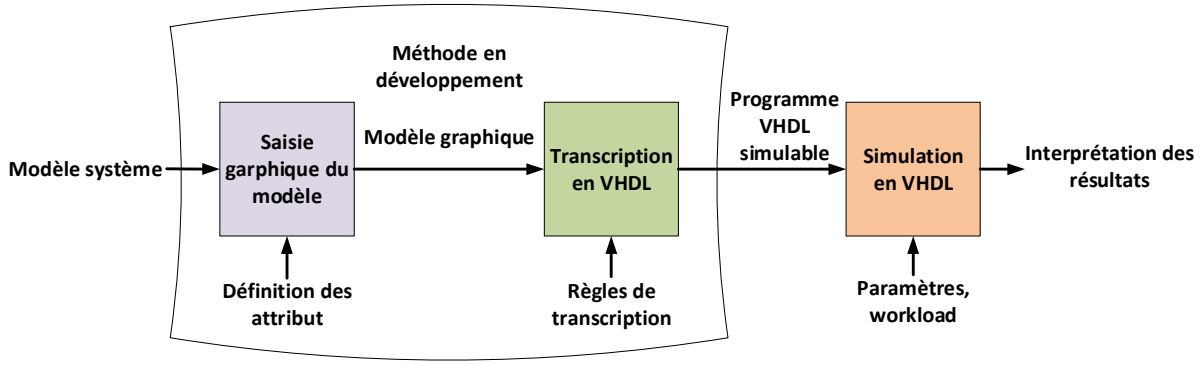


Figure 1.3 ***Démarche et outil pour l'évaluation des performances***

Le modèle du système à simuler est saisi par un outil graphique. Les attributs sont aussi ajoutés. Cette description sauvée sous forme textuelle est ensuite transcrite en un programme VHDL. Le programme VHDL est instrumenté puis simulé pour obtenir les observations souhaitées. La transcription du modèle concerne le modèle de structure (fonctionnel et exécutif) et le modèle de comportement.

1.6.8. CHOIX DU LANGAGE VHDL

A première vue, le choix du langage VHDL conduit à disposer d'une excellente plate-forme pour l'évaluation des performances aussi bien pour la simulation que pour l'extraction et l'analyse des résultats. Pour la simulation, les propriétés et certaines constructions du langage VHDL autorisent ou facilitent la simulation du modèle de performance, à savoir:

- Parallélisme inhérent. La notion de process permet de simuler des tâches concurrentes dont le comportement de chacune est purement séquentiel. VHDL possède donc une propriété qui est aussi intéressante pour le co-design. En effet, une simulation VHDL peut très bien permettre d'étudier un partitionnement logiciel/ matériel donné. Les descriptions concurrentes représentent alors les différents processeurs matériels et logiciels et pour chaque processeur logiciel la partie logicielle est représentée par une description séquentielle. Pour la description logicielle, il existe dans le domaine public des outils de conversion de VHDL vers C et réciproquement.
- Instanciation multiple (instruction Generate) qui permet la création multiple de composants, de blocs ou de process. Pour l'extraction de résultats, on peut facilement définir une librairie de composants permettant d'extraire par exemple le taux d'occupation

d'une ressource ou le débit sur un port de communication. L'utilisation des possibilités de "debugging" et d'analyse du simulateur utilisé (sortie sous forme de chronogrammes) permet aussi de visualiser en détail des résultats de simulation.

L'emploi de VHDL offre un certain nombre d'avantages tels que:

- la *portabilité*. C'est un langage standard et normalement indépendant de tout vendeur. Les modèles générés en VHDL peuvent s'échanger facilement. L'emploi d'un langage unique élimine aussi tout risque d'erreurs ou d'approximations de transcription.
- la *lisibilité* et la *compréhensibilité*. L'emploi de VHDL pour l'évaluation des performances est intéressant pour avoir un meilleur impact sur la communauté des concepteurs d'architectures matérielles habitués à utiliser VHDL. A cet égard, il est indispensable que le code VHDL généré automatiquement à partir du modèle de performance soit le plus lisible et compréhensible possible.
- la *complétude*. La vérification de la complétude et de la consistance du modèle généré est assurée automatiquement par le compilateur/simulateur VHDL utilisé.
- la *traçabilité*. VHDL permet de décrire plusieurs niveaux d'abstraction du système, plusieurs architectures pour un même composant et peut être utilisé durant tout le cycle de développement d'un système.
- la *disponibilité des outils*. Il y a aujourd'hui de nombreux outils basés sur VHDL disponibles sur le marché. L'utilisation conjointe de ces différents outils permet de couvrir la plupart des phases de conception de la spécification à la réalisation. Sous certaines conditions, des outils de synthèse haut-niveau permettent même de transformer une description VHDL comportementale en une description VHDL au niveau RTL. Cette description résultante est ensuite transformée par synthèse logique en une netlist de portes et de bistables pour une implantation dans un circuit.

Les propriétés intrinsèques du langage VHDL satisfont aussi certains critères de qualité:

- *modèle hiérarchique*. Les notions d'entité, de block et de composant permettent de décomposer la description du système. On peut également définir plusieurs solutions d'architecture pour une même entité et choisir l'architecture retenue uniquement au moment de son instanciation.

La notion de composant est très importante car elle permet la réutilisation de modèle :

- *modèle paramétrable*. VHDL permet l'utilisation de paramètres génériques.
- *modèle d'interdépendance*. Très proche syntaxiquement du langage ADA, VHDL se distingue des langages de programmation par le concept de signal.

1.6.9. Justification d'emplois de partie logiciel 'IP type Nios d'Altera' [7]

Comme expliqué précédemment, le choix du langage VHDL se justifie par sa standardisation, sa portabilité, la disponibilité des nombreux outils sur le marché et ses propriétés intrinsèques (modèle hiérarchique, paramétrable, parallélisme inhérent, instanciation multiple, etc.). Cependant, comme le langage VHDL ne dispose pas de mécanisme de suspension de process, il faut utiliser explicitement un composant ordonnanceur et une procédure spécifique de gestion des temps d'attente. Le manque de généricité pour la déclaration des types et les constructions telles que l'attente conditionnelle, l'achèvement forcé d'activité et certains cas de l'instanciation multiple posent des difficultés de transcription. Pour ces cas, la complexité de la solution d'implantation en VHDL se paie malheureusement au niveau de la génération de code et de la lisibilité et de l'efficacité du code VHDL produit. Malgré le niveau d'abstraction du modèle sous VHDL, la simulation des programmes VHDL obtenus à partir des modèles de traitement d'image en temps-réel et du système de gestion des résultats permet de constater la nécessité de ressources importantes (puissance de calcul et mémoire) et une durée de simulation un peu trop longue pour trouver rapidement la solution optimale recherchée. Ces constatations justifient pleinement l'intérêt de l'utilisation d'une partie logiciel embarqué sur un IP type processeur virtuel (Ex : Nios d'Altera) qui allège le processus de traitement.

1.7. Résumé

Dans ce chapitre nous avons introduit les circuits FPGAs avec leurs principales caractéristiques et les critères relatifs à leurs utilisations, ensuite nous avons introduit les avantages de l'utilisation des FPGAs pour le traitement d'images en temps réel, et en fin en a énuméré différents notions relatives aux Co-Design.

Chapitre 2

OUTILS DE CONCEPTION ET DE SIMULATION POUR FPGA

2.1. Introduction

Un concepteur possède actuellement d'énormes solutions afin d'implanter ses applications. L'intégrateur peut choisir entre l'utilisation d'une architecture programmable figée utilisant un microcalculateur avec son jeu d'instruction, ou une philosophie faisant appel à une architecture matérielle dédiée pour cible ASIC (Application Specific Integrated Circuit) ou FPGA (Field-Programmable Gate Array) selon des contraintes de réalisation.

Les architectures programmables offrent une grande flexibilité d'utilisation et de programmation. Toutefois, elles sont rarement utilisées seules lors de la conception de systèmes embarqués dédiés au traitement d'image et du signale, à cause de leurs piètres performances (consommation d'énergie élevée et/ou bien puissance de calcul faible).

Les architectures matérielles dédiées implantés sous la forme de circuits ASIC offrent des performances optimales. Les circuits ASIC sont des circuits intégrés non programmables conçus pour des besoins spécifiques. Afin de minimiser les délais incompatibles avec la contrainte de temps de conception, des technologies reprogrammables sont apparues. Ces dernières offrent des capacités d'intégration permettant l'intégration de nombreux systèmes.

En effet, les circuits FPGA (Field Programmable Gate Array) ou réseau pré-diffusés, possèdent des capacités d'intégration très importantes : jusqu'à 2 million de LUTs (Look-up Table) à 6 entrées pour les circuits pour certain constructeur. Cependant leurs performances (fréquence d'utilisation, consommation d'énergie) sont en retrait face à des implantations sur technologie ASIC. De plus, les circuits ASIC permettent de réduire les coûts de production sur de larges séries et leur fiabilité est supérieure à celle des circuits reprogrammables.

L'évolution de la conception des circuits numériques s'oriente toujours vers la mise en place de systèmes plus performants permettant ainsi d'intégrer des applications toujours plus complexes. C'est le cas des applications multimédia.

Dans ce contexte, ce chapitre détaille spécifiquement des notions sur la suite logicielle se rattachant à la procédure d'implantation FPGA pour la conception d'applications matérielles sur puce programmable ''SOPC'' (System On Programmable Chip).

2.2. Technologie Altera [9] [10]

2.2.1. Quartus II

2.2.1.1. Présentation

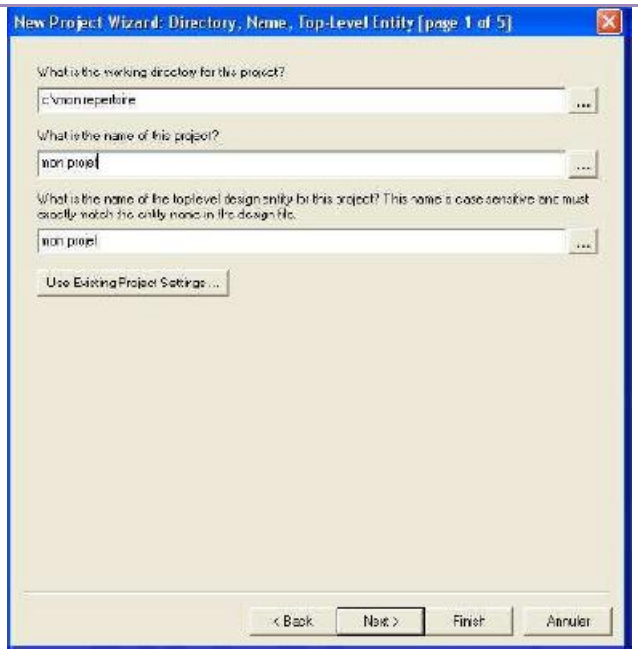
Quartus est un logiciel proposé par la société Altéra, permettant la gestion complète d'un flot de conception FPGA. Ce logiciel permet de faire une saisie graphique ou une saisie texte (description VHDL) d'en réaliser une simulation, une synthèse et une implémentation sur cible reprogrammable.

En ce qui suit un tutorial des principales fonctionnalités nécessaires à la création d'un projet, son analyse et en fin son implémentation sur l'FPGA. Ce tutorial est construit avec des tables explicatives contenant des captures d'écran ainsi que des instructions à suivre.

Figure 2.1 Fenêtre du logiciel Quartus II version 7.2 d'Altera

2.2.1.2. Création d'un nouveau projet

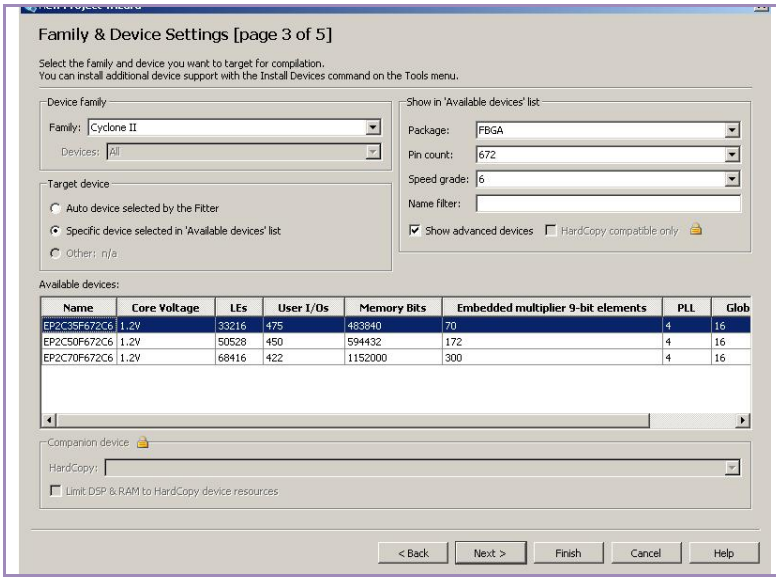
Pour créé un nouveau projet cliquer sur **File** puis sur **New Project Wizard** en fin sur **OK**



Choisir l'emplacement du répertoire ou seront stockés tous les fichiers du projet.
Choisir le nom de votre projet.
Choisir le nom de l'entité maître du projet (niveau le plus haut dans le design).
Conseil : il faut créer un répertoire par projet. Ne pas le créer dans le dossier :
c :\altera\72\quartus mais dans un répertoire de travail.

Figure 2.2 ***Création d'un nouveau projet sous Quartus II***

Cliquer sur **Next** puis quand la fenêtre **Add Files** apparaît recliquer sur **Next**.



Choisir la famille du composant programmable ainsi que le circuit cible.

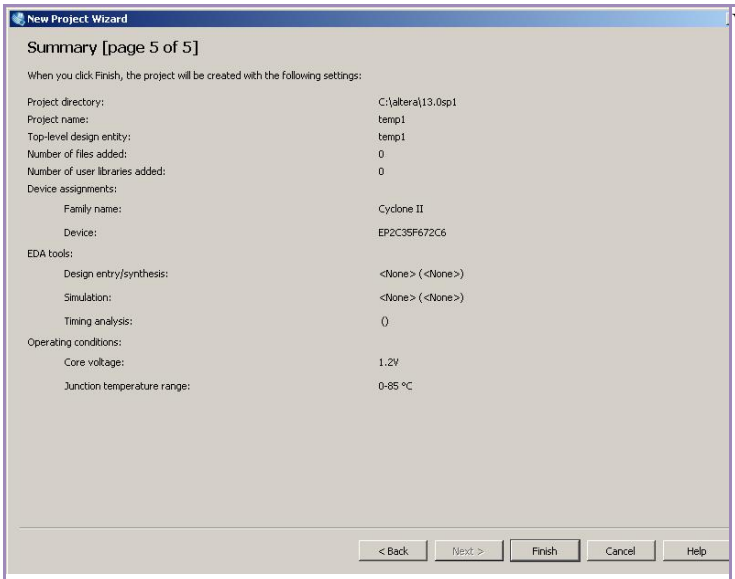
- **Family :**

Choisir Cyclone II "Carte Altéra DE2".

- **Avalaible device :** sélectionner **EP2C35F672C6 "Carte Altéra DE2".**

Figure 2.3 ***Configuration du choix du circuit cible a implémenter***

Après avoir valider les choix précédents en cliquant sur **Next**, la fenêtre **EDA Tool Settings** apparaît. Cliquer encore une fois sur **Next** ce qui fait apparaitre une fenêtre récapitulative :



Vous pouvez valider les choix par **Finish** ou bien faire **Back** pour des modifications éventuelles. Dans le navigateur de Projet, un onglet avec le type de composant et l'entité maître apparaît :

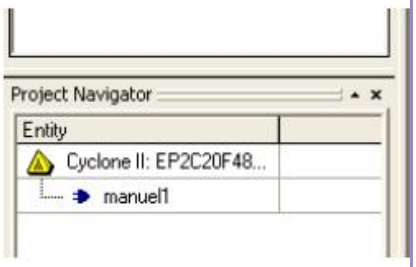


Figure 2.4 ***validation de la configuration et affichage du sommaire du projet***

2.2.1.3. Saisie d'un projet &Création d'un schéma

Pour créé un schéma relatif à un projet cliquer sur **File** puis sur **New**

Pour créer un schéma relatif à un projet cliquer sur **File** puis sur **New** ensuite sélectionner **Block diagram / Schematic file** et en fin cliquer sur **OK**

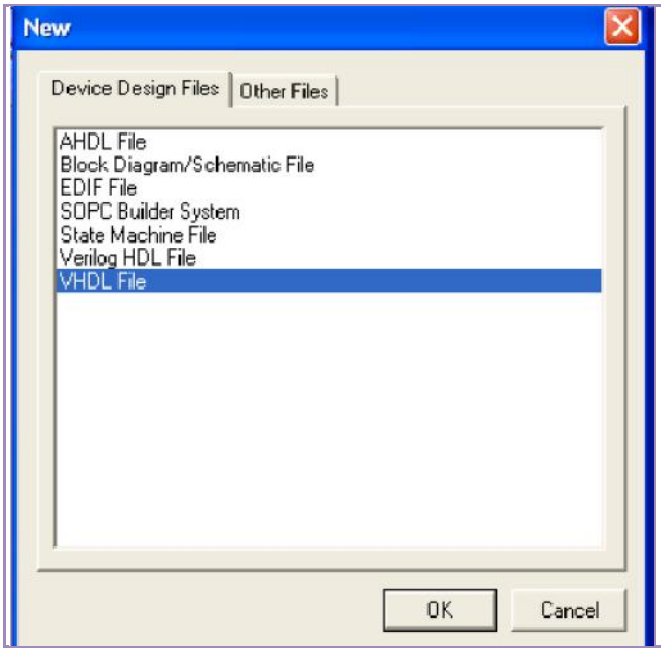
Figure 2.5 ***Saisi du projet et création du schéma***

Pour sauvegarder le schéma choisir **File** ensuite en clique sur **Save as** pour choisir l'emplacement de l'enregistrement ensuite le nom du fichier.

Conseil : Une feuille blanche se crée intitulée Block1.bdf. On prendra soin de sauver cette feuille sous le nom de l'entité maître, car c'est maintenant cette feuille de saisie graphique qui a la hiérarchie la plus haute dans le projet.

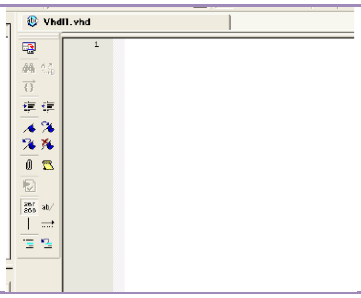
Figure 2.6 ***Utilisation de la boite à outils***

2.2.1.4. Création d'un fichier VHDL



Pour créer un fichier de type **VHDL** cliquer sur **File** puis cliquer sur **New** ensuite sélectionner **VHDL File** et en fin valider en cliquant sur **OK**

Figure 2.7 ***Creation d'un fichier VHDL sous Quartus II***

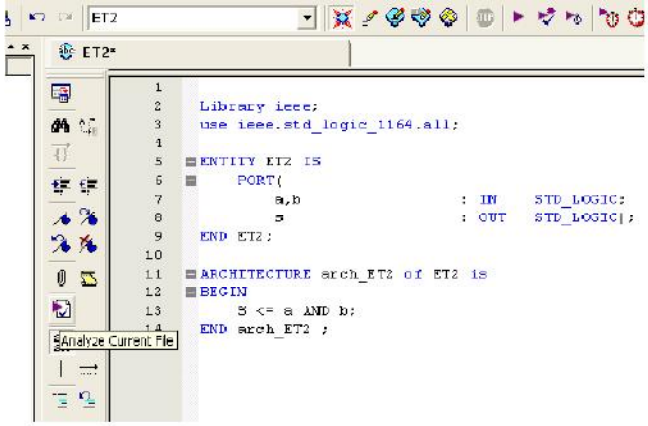


Un petit éditeur de texte apparaît après avoir validé sur **OK** et c'est sur cet éditeur ou en écrira notre code en **VHDL**

Figure 2.8 Editeur de code VHDL sous Quartus II

Une fois le code VHDL saisie, il convient de le sauver (**File** puis **Save As**) puis d'en vérifier la syntaxe.

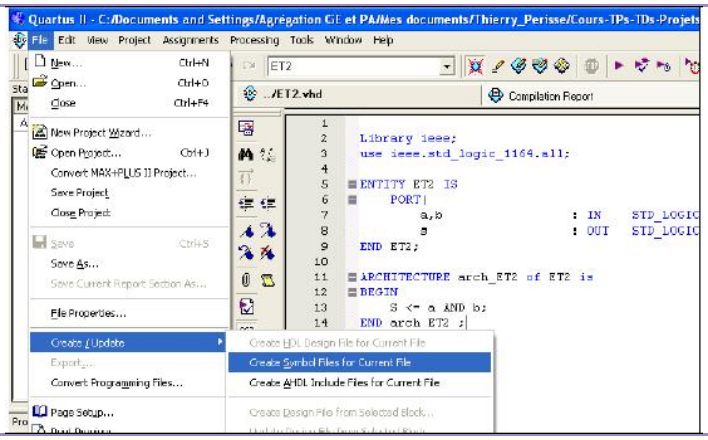
Conseil : Il est important de sauver le fichier sous le même nom que l'entité. Cela évite des intersections d'entité entre fichiers.



Lorsque l'édition du fichier est terminée et qu'il est sauvegardé, vérifier la bonne syntaxe de la description en cliquant sur **Analyse Current File.** Corriger les éventuelles erreurs.

Figure 2.9 Sauvegarde du script VHDL et analyse des erreurs

2.2.1.5. Création d'un symbole



Une fois que le fichier est **OK**, on peut alors créer un symbole graphique qui nous permettra d'insérer le composant dans la feuille graphique initiale.

Figure 2.10 Création d'un symbole équivalent aux instructions du script VHDL

2.2.1.6. Compilation

Durant la compilation, Quartus va réaliser 4 étapes :

a- La transformation des descriptions graphiques et textuelles en un schéma électronique à base de portes et de registres : c'est la ***synthèse logique***.

b- L'étape de Fitting (ajustement) consiste à voir comment les différentes portes et registres (produit par la synthèse logique) peuvent être placés en fonction des ressources matérielles du circuit cible (EP2C35F672C6) : c'est la ***synthèse physique***.

c- L'assemblage consiste à produire les fichiers permettant la programmation du circuit. Ce sont des fichiers au format Programmer Object Files (.pof), SRAM Object Files (.sof), Hexadécimal (Intel-Format) Ouput Files (.hexout), Tabular Text Files (.ttf), et Raw Binary Files (.rbf).

d- L'analyse temporelle permet d'évaluer les temps de propagation entre les portes et le long des chemins choisis lors du fitting.

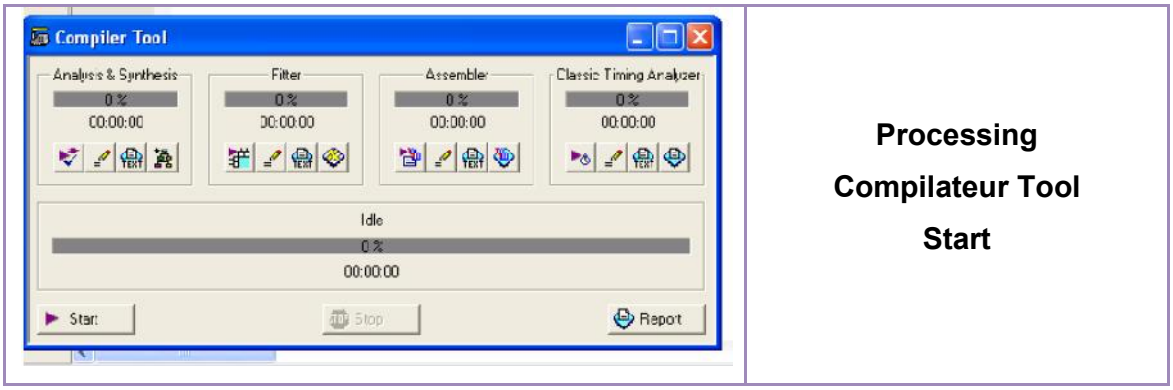


Figure 2.11 ***l'outil "Compilateur" en vue de sa mise en marche***

Normalement, il ne doit pas y avoir d'erreur. Si ce n'est pas le cas, on vérifie dans la zone **Processing** (en bas où s'affichent les messages) la source du problème.

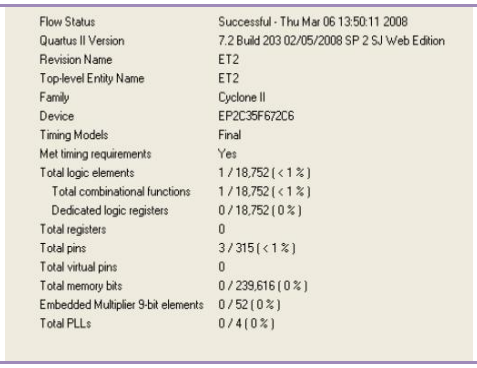
	Cliquer sur **Report** Multitude d'information : - Pourcentage - d'occupation - Temps de propagation

Figure 2.12 ***Capture d'écran du rapport d'information***

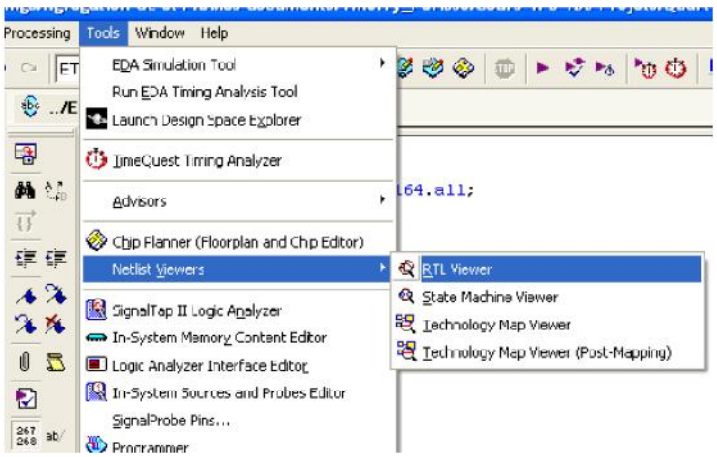
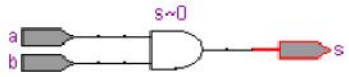
	**RTL** :Register Transfer Logic Visualisation de la Synthèse logique :  On peut voir ici comment un fichier texte *.vhd contenant le code VHDL a été transformé en portes et bascules.

Figure 2.13 ***Visualisation de la synthèse logic***

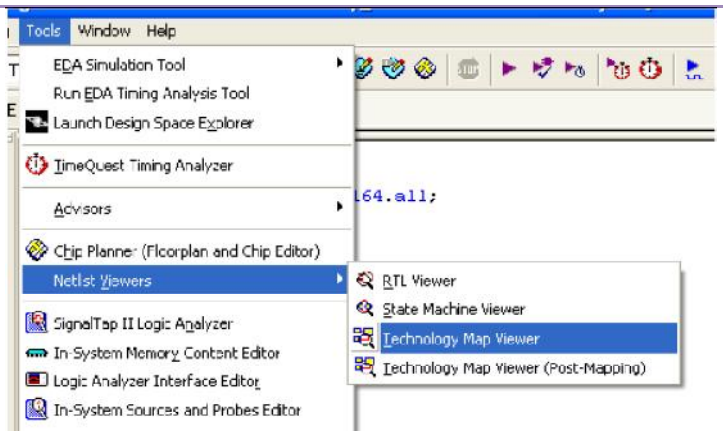
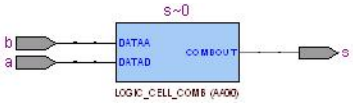
	Visualisation de la synthèse physique :  On retrouve les instances placées dans le circuit et repérées par leurs références.

Figure 2.14 ***Visualisation de la synthèse physique***

2.2.1.7. Simulation d'un circuit

- La partie du circuit à simuler doit être munie de pins d'entrée/sortie.
- Elle doit aussi se trouver au niveau le plus élevé de la hiérarchie. Si ce n'est pas le cas, pour l'y mettre : dans le « **Project Navigator** », cliquer avec le bouton droit de la souris sur le nom du fichier, puis sur **Set as Top-Level Entity**.
- Il faut également vérifier qu'il n'y ait pas d'erreur dans le circuit en cliquant sur **Processing** puis sur **Start**, et enfin sur **Start Analysis & Elaboration**.
- Le circuit étant prêt, il faut maintenant créer le fichier contenant les informations sur les signaux à appliquer sur les entrées du composant et la liste des signaux que l'on veut analyser.

Cliquer sur **File** puis sur **New**

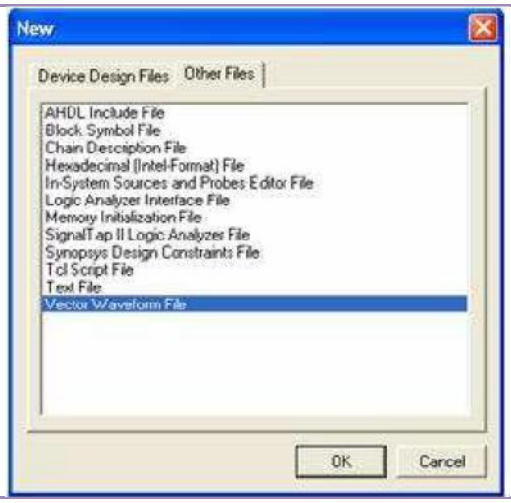
	Sélectionner l'onglet **« Other Files »** et cliquer sur **Vector Waveform File**.

Figure 2.15 ***Création du fichier des vecteurs de test***

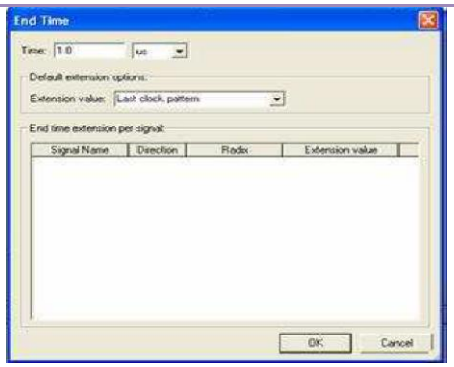
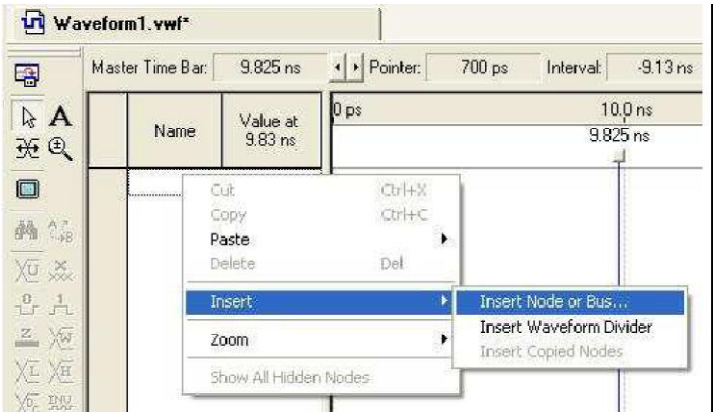
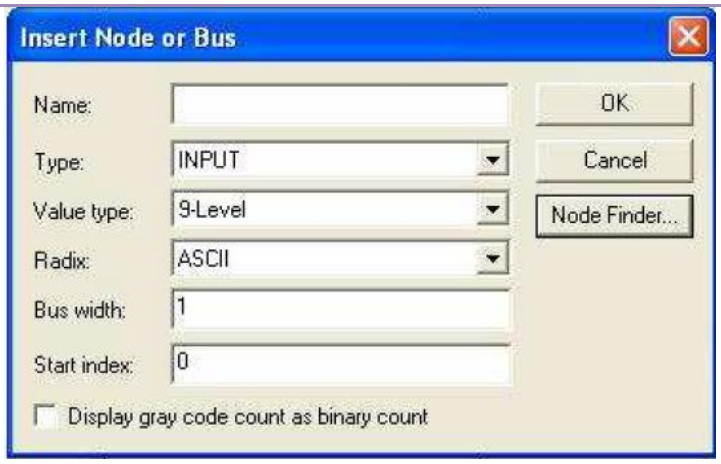
	Par défaut, la durée de simulation est de **1 µs**. Pour la modifier, cliquer sur **Edit**, puis **EndTime**. Une fois la durée modifiée, cliquer sur **OK**.

Figure 2.16 ***Choix des durées des vecteurs de test***

Sauvegarder le fichier sous son nom définitif avec son extension (**.vwf)** en cliquant sur **File** puis **Save As**.



Dans la fenêtre « **Name** », cliquer avec le bouton droit de la souris, puis sélectionner **Insert** et cliquer sur **Insert Node or Bus**.



Dans la boite de dialogue qui s'ouvre, cliquer sur **Node Finder**

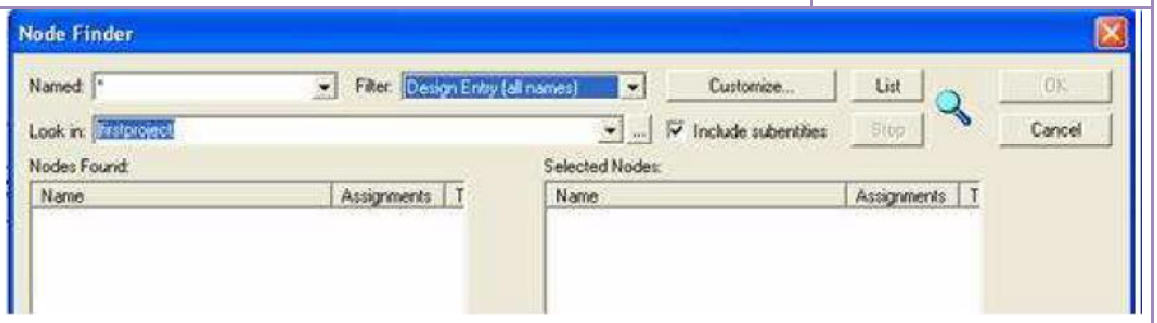


Figure 2.17 ***Insertion des stimuli afin de les générer***

Dans la catégorie « **Filter** », choisir **all names**. Cliquer ensuite sur le bouton **List**. Ajouter les signaux souhaités dans la fenêtre **Selected Nodes**, en cliquant sur > Cliquer sur **OK** pour fermer les différentes fenêtres et revenir à l'éditeur de signaux.

Afin de simuler le design, il convient de lui injecter des stimuli. Lorsque ces stimuli sont générés à partir d'un fichier on dit que l'on utilise un **fichier de Bench**.

Cliquer avec le bouton droit de la souris sur le nom d'un signal, sélectionner **Value**, puis choisir la valeur du signal dans le menu.

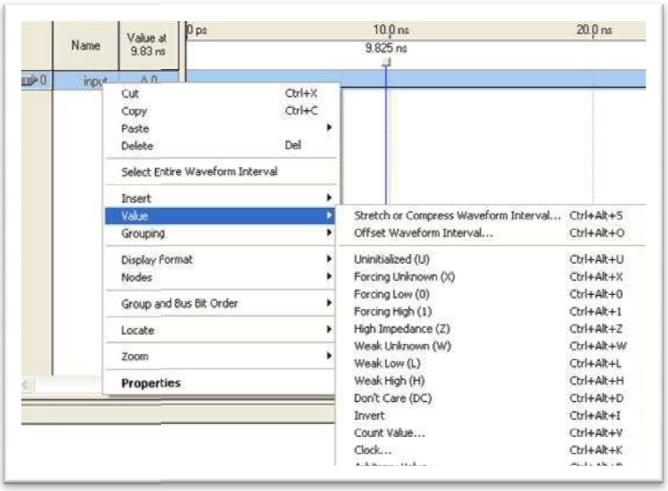
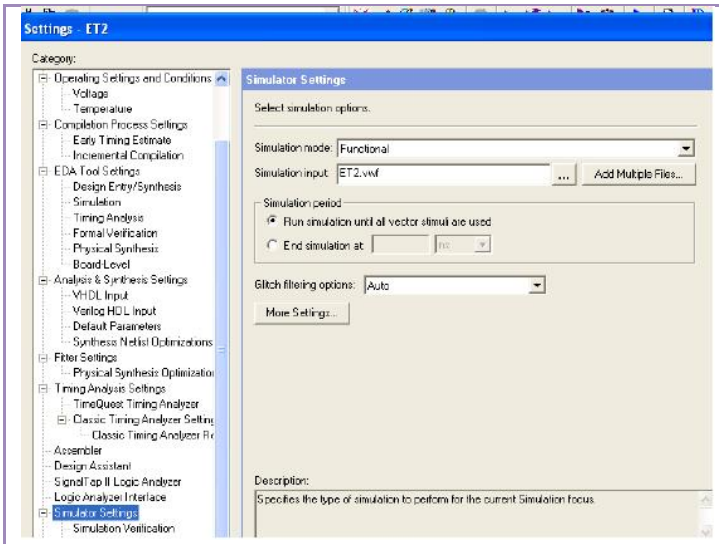


Figure 2.18 menu du choix de la valeur du signal

Il est possible d'effectuer la même opération sur une partie seulement d'un signal en sélectionnant une zone dans la partie « chronogramme ». Il faut pour cela maintenir le bouton gauche de la souris appuyé en déplaçant le curseur. Lorsque tous les signaux d'entrées sont définis, sauvegarder le fichier.

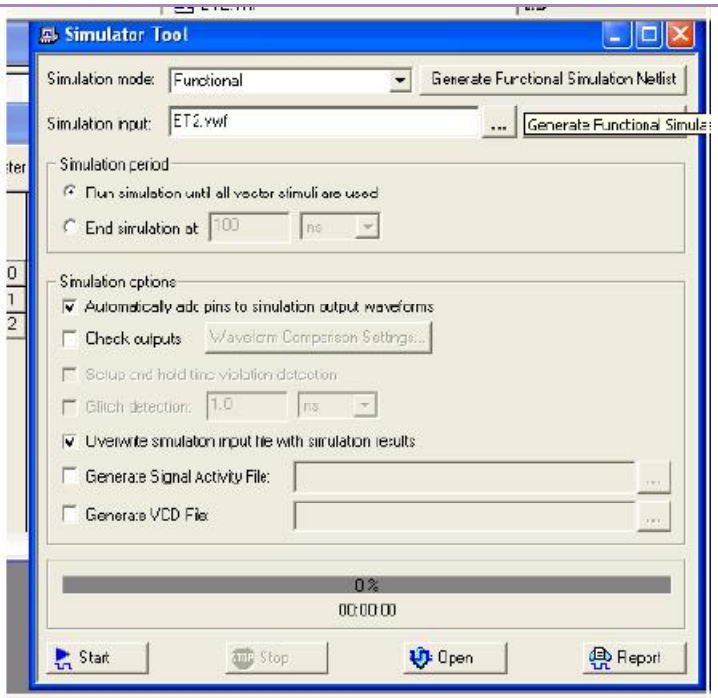
2.2.1.8. Simulation Fonctionnelle



Cliquer sur **Assignments**, puis sur **Settings**. Sélectionner **Simulator Settings** et entrer les paramètres suivants : **Simulation mode : Functional Simulation input** : entrer le nom du fichier .vwf que vous avez créé.

Cliquer sur **OK**.

Figure 2.19 Configuration du mode de simulation



Cliquer sur **Processing** puis sur **Simulator Tool**

Sélectionner **Functional** puis cliquer sur **Generate Functional Simulation Netlist**

A présent tout est prêt pour effectuer la simulation.

Cliquer sur **Processing** puis sur **Start Simulation** ou cliquer sur  .

Figure 2.20 ***Finalisation et démarrage de la simulation***

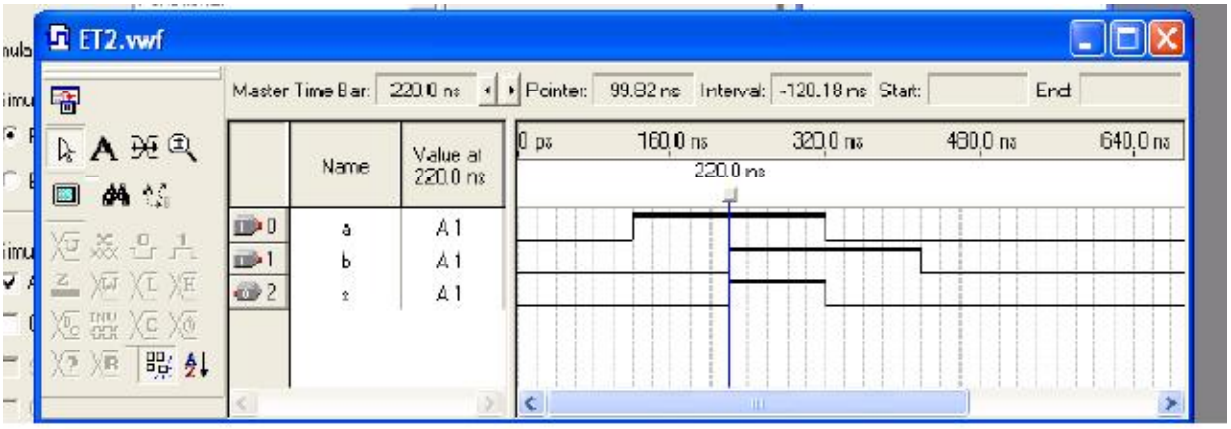
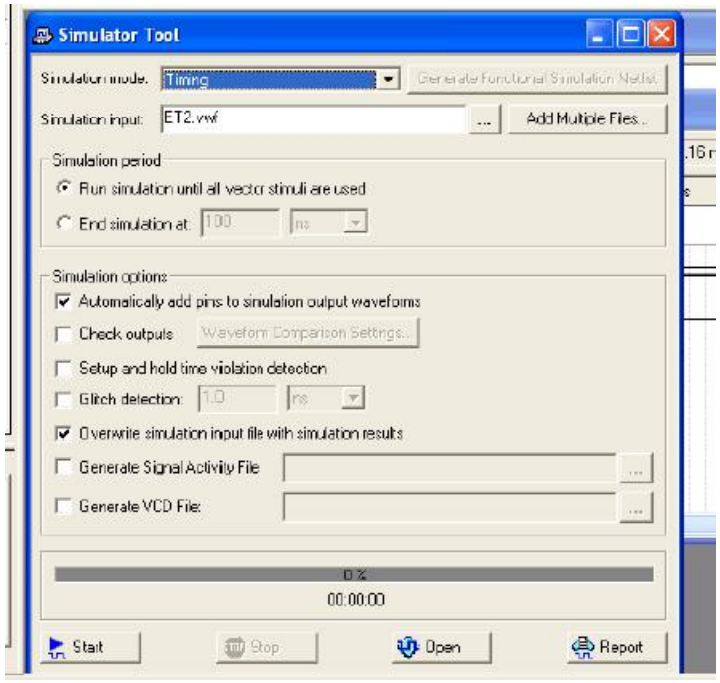


Figure 2.21 ***Chronogramme de la simulation fonctionnelle***

2.2.1.9. Simulation Temporelle



Cliquer sur **Processing** puis sur **SimulatorTool**.

Sélectionner **Timing**.

A présent tout est prêt pour effectuer la simulation.

Cliquer sur **Start** puis **OK**. Il est maintenant possible de voir le résultat en cliquant sur **Report**.

Autres possibilités :

Cliquer sur **Processing** puis sur **StartSimulation** ou cliquer sur  .

Figure 2.22 ***Configuration et démarrage de la simulation temporelle***

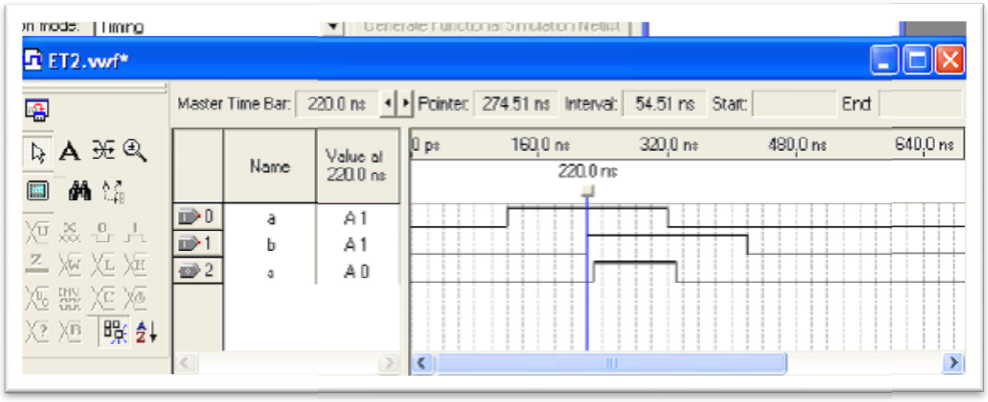


Figure 2.23 ***Chronogramme de la simulation temporelle***

Avant la programmation du circuit il faut assigner les pins d'entrées et de sorties du design aux broches du circuit physique.

2.2.1.10. Affectation des entrées et des sorties

Cliquer sur **Assignements** puis sur **pins**

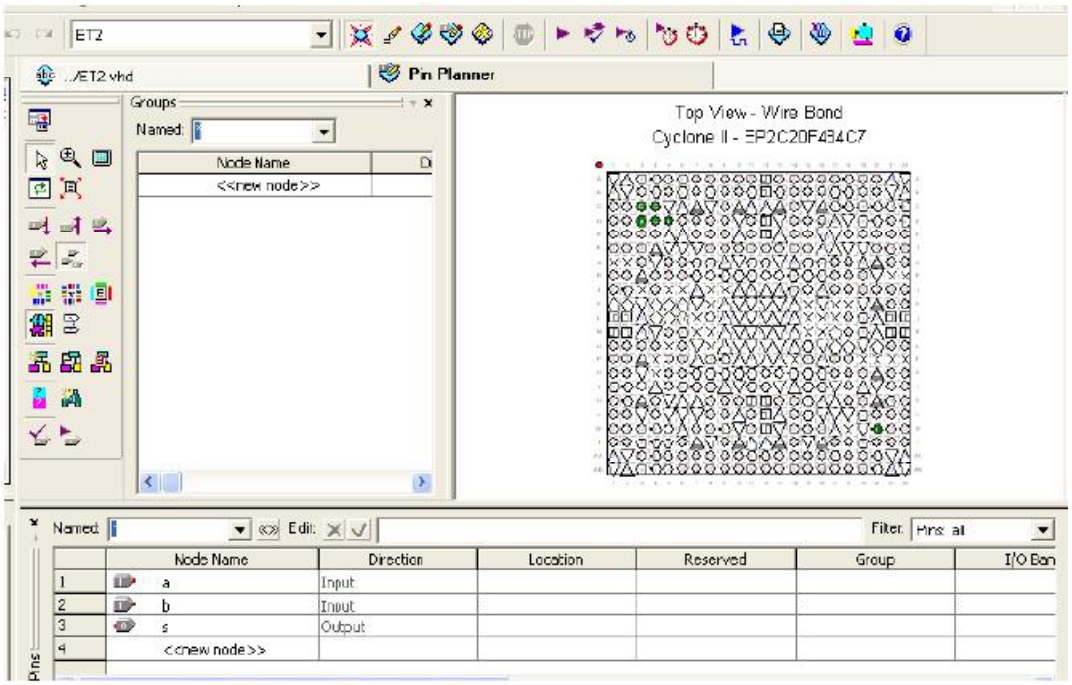
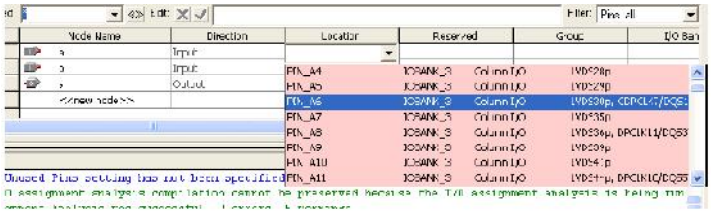


Figure 2.24 Assignements des pins avec Quartus



On double clique sur la colonne **location** au niveau du pin voulu de manière à faire apparaître un menu déroulant où sont répertoriées les broches disponibles du circuit.

Figure 2.25 Localisation des broches disponible du circuit

La liste des broches utilisables pour le FPGA et sortant sur les connecteurs est donnée dans le manuel de la carte DE2 d'Altéra. Ne pas oublier de compiler avant la programmation.

2.2.1.11. Programmation du circuit

La programmation du circuit se fait via le protocole JTAG. Pour cela, vérifier que la connexion entre le PC et la carte DE1 via le module USB-Blaster est opérationnelle. Si tout est **ok** lancer le programmateur : Cliquer sur **Tools** puis sur **Programmer.**

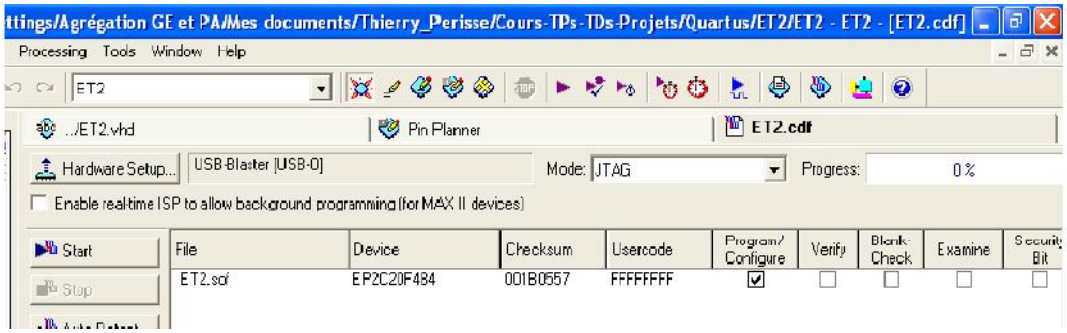


Figure 2.26 étape pré-programmation avec Quartus

Vérifier que le fichier avec l'extension **.sof** est bien là (sélectionner le) et que la case **Program/Configure** est cochée, puis cliquer sur **Start**.

2.2.2. SOPC Builder

Le SOPC Builder permet, entre autres, de concevoir des microcontrôleurs spécifiques à une application. Ces microcontrôleurs comportent donc une partie processeur à laquelle on associe des périphériques (PIO, Timers, UART, USB, composants propriétaires, …) et de la mémoire. Cette dernière peut-être embarquée dans le FPGA (on parle alors de RAM/ROM On Chip) ou à l'extérieur du composant FPGA. La partie microprocesseur proprement dite est le NIOS2 de ALTERA, processeur de 32 bits qui se décline en trois versions : économique, standard, rapide. La version économique, la moins puissante, utilise le moins de ressources du FPGA. Bien sûr il est possible d'intégrer d'autres types de processeurs pour peu qu'on dispose de leurs modèles (VHDL, Verilog, …). La création d'une application SOPC comprend les étapes suivantes :

- Création du composant matériel (processeur + périphériques) dans l'environnement Quartus II
- Téléchargement dans le composant FPGA (configuration) (environnement Quartus II)
- Création du logiciel dans l'environnement NIOS2IDE et téléchargement dans le FPGA.

2.2.2.1. Intégration d'un composant propriétaire

Lors de la création du système, il se peut que certaines ressources ne soient pas parmi celles proposées par le constructeur. Il devient nécessaire de :

- Intégrer un composant d'un fournisseur tiers

- Créer son propre composant (sa mise en œuvre dans le SOPC Builder).

2.2.3. L'environnement Qsys d'Altera

L'outil Qsys utiliser on conjonction avec Quartus II permet de créé facilement le système basé sur le processeur Nios II, et ceci en sélectionnant les unités fonctionnelles et spécifier leurs paramètres.

Pour implémenter le système nous devons instantier les éléments suivants :

• le processeur Nios II

• On-chip memory, qui consiste en la mémoire bloque dans la puce FPGA

• deux interfaces parallèles I/O

• l'interface de communication avec le PC hôte le JTAG UART

2.2.4. ModelSim d'Altera : [11]

Le logiciel ModelSim SE permet la simulation temporelle au niveau RT(transfert de registre) ou au niveau porte, à partir des langages VHDL ou Verilog.

2.3. Technologie XILINX

Les différentes architectures de FPGA Xilinx sont Spartan-II, Spartan-IIE, Virtex-E et Virtex-II. Leurs caractéristiques principales sont:

- Complexités allant de 15000 à plus de 8 millions de portes.

- Faible consommation.

- Grande souplesse d'utilisation des entrées-sorties avec adaptation d'impédance (Virtex-II) et configuration en mode différentiel (Spartan-IIE, Virtex-E et Virtex-II).

- Fonctions mémoire (distribuée et blocs de Ram).

- Dispositifs de gestion des horloges (DLL et DCM).

- Multiplieurs câblés (Virtex-II). Et bien d'autres possibilités permettant d'optimiser à la fois la performance et la densité des fonctions logiques et/ou arithmétiques.

La figure ci après montre une organisation générale des FPGA Xilinx, l'essentiel des fonctions logiques sera implantée en CLB (Configurable Logic Bloc), ces blocs configurables sont constitués de «slices» qui veut dire un ensemble deux bascules D, et une logique combinatoire et arithmétique composée essentiellement de LUT et/ou des fonctions de mémoire distribuée ou de registre à décalage.

Suivant les familles, un CLB peut être constitué de deux ou quatre slices, c'est-à dire qu'il peut comprendre jusqu'à huit bascules D, et, par exemple, 128 bits de mémoire ou des fonctions arithmétiques et/ou logiques.

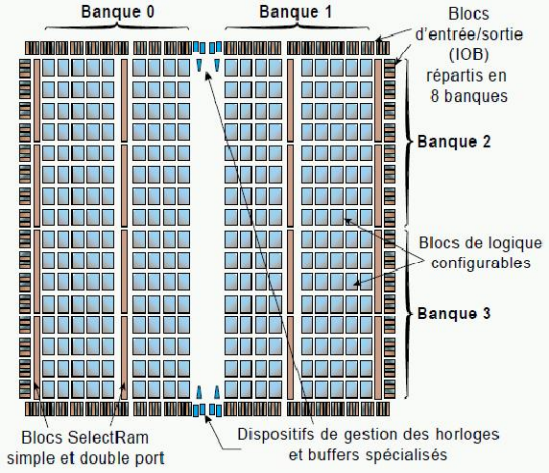


Figure 2.27 Organisation des FPGA de Xilinx

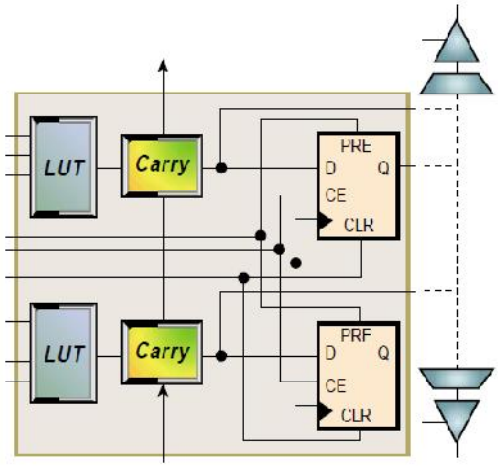


Figure 2.28 Organisation des FPGA de Xilinx avec la structure d'un slice

La logique combinatoire est implantée grâce aux LUT (look-up tables) contenues dans chaque slice. Ces LUT peuvent également être configurés comme éléments de mémoire synchrone, simple ou double-port de 16 bits, ou encore comme registres à décalage de 16 bits. Autrement dit, les LUT sont des mémoires dont le contenu est initialisé lors de la configuration du FPGA. De ce fait, elle nous permet d'en disposer en mode «élément mémoire» dans chacun des slices

La technologie Xilinx offre trois types de LUT, celles utilisant des simples multiplexeurs pour la génération de fonction, celles utilisant en plus des composants décodeurs et en fin celles conçu à base de simples mémoires débitant sur multiplexeurs ***'figure 2.29'***

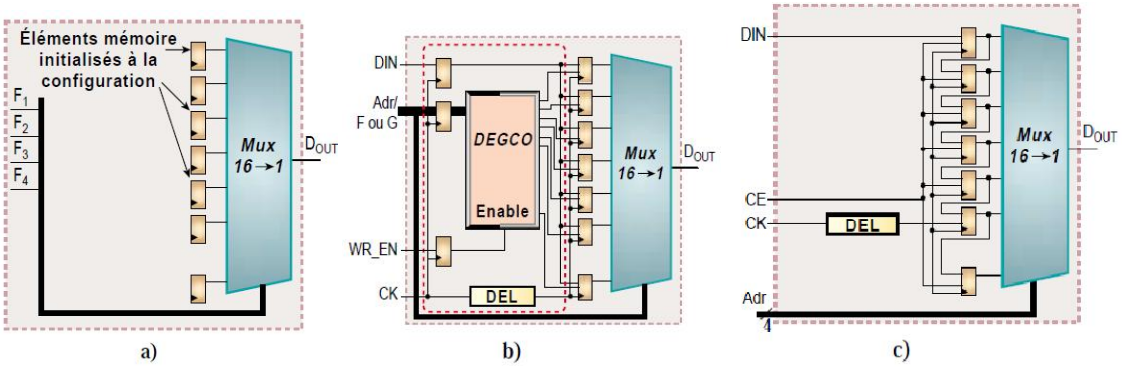


Figure 2.29 Modes de configuration des LUT

Il est à noter que Xilinx met à la disposition un outil de développement "ISE Web Pack" semblable a celui d'Altera qui va permettre de décrire le circuit logique qu'on souhaite réaliser, d'en faire une ou plusieurs simulations, enfin de programmer le FPGA.

2.4. Résumé

Dans ce chapitre nous nous somme explicitement concentré sur les outils de conception, de simulation, de synthèse et d'implantation des FPGAs de type ALTERA et de sa suite logiciel QUARTUS, SOPC Builder, Qsys et ModelSim. En fin nous avons fait allusion au produit XILINX qui est donnée comme Alternatif.

Chapitre 3

NOTION DE FILTRAGE DU SIGNAL

3.1. Introduction

Une des opérations de base dans le traitement numérique du signal ou image est le filtrage. Cette opération est largement utilisée dans de nombreux appareils électroniques, son rôle est d'annuler une partie du signal redondant, éliminer les parties non désirées du signal, par exemple un bruit aléatoire, ou extraire des parties utiles du signal, telles que les composantes se trouvant à l'intérieur d'une certaine plage de fréquences. Par exemple, dans le cas des installations téléphoniques, il n'est pas nécessaire de transmettre tous les échantillons du signal et de se contenter de ceux se trouvant dans la bande de fréquences (bande 400 à 3400 Hz). Par conséquent, dans cette situation, toutes les fréquences au-dessus et en dessous de cette bande seront filtrés.

Figure3.1 structure générale d'un filtre.

Il existe deux types de filtres : Analogique et numériques.

3.2. Filtres analogiques

Un filtre analogique utilise des circuits électroniques analogiques constitués de composants tels que des résistances et des condensateurs pour produire l'effet de filtrage requis. Ces circuits de filtrage sont largement utilisés dans des opérations d'élimination, de réduction du bruit, et de mise en forme de signal. Ces filtres présentent des avantages et inconvénients:

3.2.1. Avantage

- Méthodes simples et consolidées
- Réalisations simples et rapides.

3.2.2. Inconvénients

- Non stable, très sensible à la température
- Cout de réalisation élevé

3.3. Filtres numériques [3] [4]

Le filtre numérique utilise un processeur digital pour effectuer des calculs numériques sur les valeurs échantillonnées du signal. Le processeur peut faire partie d'un ordinateur à usage général, tel qu'un PC, ou une puce DSP spécialisé (Digital Signal Processor). Les filtres numériques présentent beaucoup d'avantages et peu d'inconvénients.

3.3.1. Avantages du filtre numérique

- Programmable
- Conception facile, mis en œuvre et testé sur ordinateur
- Grande stabilité en temps et en température
- Traitement précis des signaux a fréquence basse
- Polyvalent

3.3.2. Inconvénients du filtre numérique

- Complexité de réalisation

Dans le domaine temporel, ces filtres numériques sont décrits par un modèle mathématique régissant à partir d'équations aux différences:

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) - \sum_{i=1}^{N} b_i y(n-i) \qquad (3.1)$$

où, *y(n)* représente la sortie en cours du filtre, les *y(n-i)* sont sorties précédentes du filtres, *x(n-i)* sont les entrées en cours et précédentes du filtrage , les a_i sont les poids de pondération ou coefficients de pondération des entrées et *bi* sont les poids de pondération ou coefficients de pondération des sorties.

Les fonctions de transfert plus précisément leurs versions transformée en Z modélisent le comportent du filtre dans le domaine fréquentiel.

$$G(z) = \frac{B(z)}{A(z)} = \frac{b(0) + b(1)z^{-1} + \cdots + b(Q)z^{-Q}}{1 + a(1)z^{-1} + \cdots + a(P)z^{-P}} = b(0)\frac{\prod_{i=1}^{Q}(1 - z_i z^{-1})}{\prod_{i=1}^{P}(1 - P_i z^{-1})} \quad (3.2)$$

Il ya deux types de base de filtres numériques :

- Filtre à Réponse Impulsionnelle Infinie "Infinite Impulse Response" (IIR)
- Filtre à Réponse Impulsionnelle Finie " Finite Impulse Response" (FIR)

3.3.3. Les filtres IIRs

Un filtre RII, est un filtre numérique défini par l'équation suivante :

$$y(n) = \sum_{l=0}^{N} a_l . y(n-l) + \sum_{k=0}^{M} b_k . x(n-k) \quad (3.3)$$

Cette équation fait intervenir non seulement les échantillons retardés du signal d'entrée mais également les échantillons retardés du signal de sortie. On parle également de filtre récursif.

Il existe différentes structures permettant d'implémenter les filtres RII sur un calculateur. En voici deux exemples :

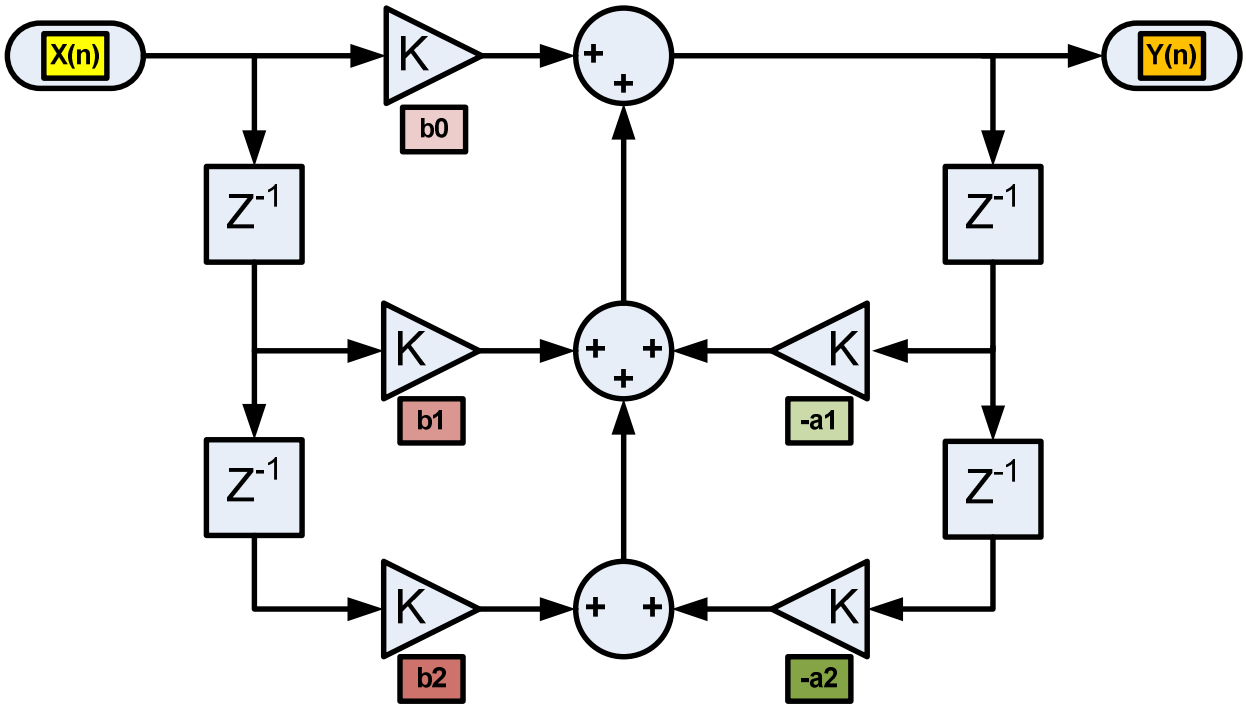


Figure 3.2 ***Structure directe canonique de type 1 du filtre FIR***

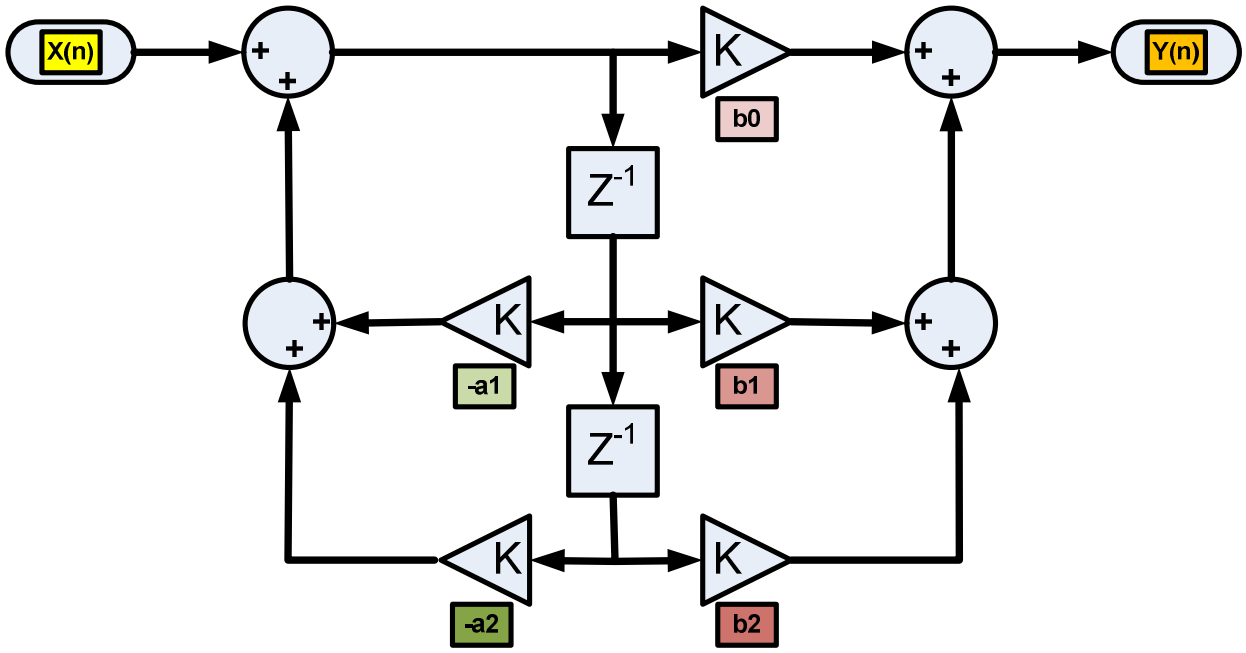


Figure 3.3 ***Structure directe canonique de type 2 du filtre FIR***

3.3.4. Les filtres FIR [12] [13]

Les filtres FIR sont les premiers filtres à être utilisés en traitement de signal, ils sont appelés ainsi à cause de la non existence du signal de rétroaction.

Le signal d'entrée est lié au signal de sortie par les équations **aux** différences suivante:

$$y[n] = x[n]b[0] + x[n-1]b[1] + x[n-2]b[2] + \cdots + x[n-o+1]b[o-1] \quad (3.4)$$

Sa fonction de transfert est donnée par :

$$H(z) = \sum_{n=0}^{o-1} b[n].z^{-n} \quad (3.5)$$

La structure directe d'un filtre FIR est donnée par la figure suivante. Cette forme est habituellement la plus utilisée dans l'implémentation en raison ses meilleurs paramètres.

La structure suivante "***Figure 3.4"*** utilise plusieurs additionneurs et multiplieurs, d'autres structures utilisent uniquement un seule multiplieur et un seul additionneur comme dans le cas de notre conception.

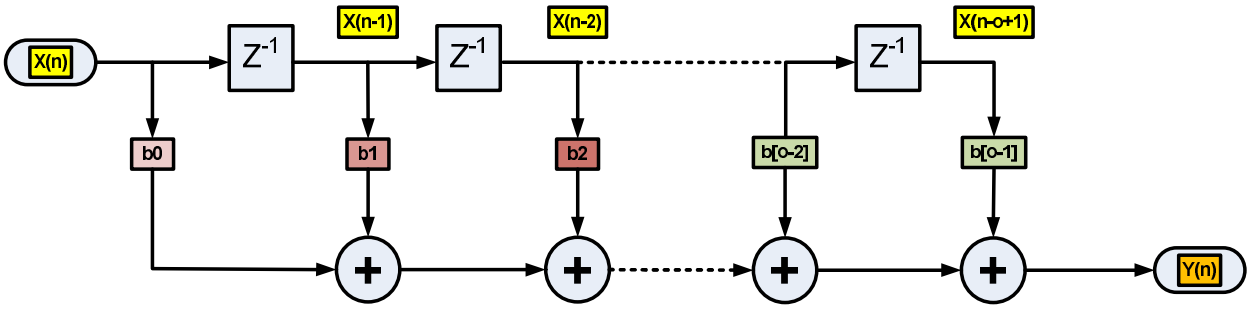


Figure 3.4 La structure direct du filtre FIR

3.3.4.1. Caractéristiques des filtres numériques FIR

Les termes suivants sont utilisés pour caractériser les performances et le comportement des filtres FIR :

Coefficients du filtre: c'est un ensemble de constantes, appelé aussi poids, utilisé pour multiplier ou pondérer les valeurs retardés de l'échantillon. Pour un filtre FIR, les coefficients de filtre sont, par définition, la réponse impulsionnelle du filtre.

Réponse impulsionnelle: Temps de réponse de la séquence du filtre lorsque l'entrée est une impulsion. Une impulsion est un échantillon unique de valeur égale à l'unité précédée et suivie par des valeurs nulles (zéro échantillons évalués). Pour un filtre FIR sa réponse impulsionnelle est l'ensemble des coefficients filtre.

Tap : Le nombre d'étage N du filtre FIR, généralement N, nous informe sur le type du FIR et surtout, sur quantité de mémoire nécessaire, et le nombre d'étapes de calculs.

Mac : Dans un contexte de filtrage utilisant les FIRs, un "MAC" est l'opération de multiplication d'un coefficient par l'échantillon de données retardé correspondant et en accumulant le résultat. Il ya généralement un MAC par étage.

3.3.4.2. Avantages des filtres FIR

- Simples à concevoir
- Muni d'entrée-sortie bornée(BIBO) stable
- Filtre FIR peut être assuré d'avoir à phase linéaire. Ceci est une propriété souhaitable pour de nombreuses applications telles que la musique et le traitement vidéo

- Les Filtres FIR ont également une faible sensibilité pour filtrer les erreurs
- Coefficient de quantification. C'est une propriété importante pour avoir la mise en œuvre d'un filtre sur un processeur DSP ou sur un circuit intégré.

3.3.4.3. Applications du monde réel de filtres FIR

Quelques applications populaires pour les filtres FIR sont énumérées ci-dessous:

- annulation d'écho
- télécommunications
- La communication de données
- Les communications sans fil
- Ghosting annulation
-HDTV
- DTV
- le traitement de la vidéo
- La synthèse vocale
- Synthèse de forme d'onde
- filtrage
- modems haut débit
- ADSL

3.3.4.4. Conclusion :

Les filtres FIR sont d'une synthèse relativement simple par rapport aux performances du filtre désiré. Ils possèdent une phase linéaire. L'inconvénient majeur étant, pour une précision donnée, un nombre de coefficients élevé nécessitant un temps de calcul élevé donc une fréquence d'échantillonnage assez faible.

3.4. Format des données utilisées dans les structures d'implémentations

Du point de vue arithmétique, l'évolution du niveau d'intégration des FPGA s'est accompagnée de l'apparition progressive d'opérateurs rapides intégrés à ces circuits. Ainsi, si les premiers FPGA ne comportaient que des cellules logiques, les générations suivantes pouvaient déjà faire des additions rapides. Désormais, la plupart des FPGA intègre aussi de petits multiplieurs et même des opérateurs de multiplication-

accumulation. Cependant, les composants arithmétiques embarqués dans ces circuits sont dédiés au calcul entier et non réels. Il y a donc nécessité de développement d'opérateur pour des calculs nécessitant de la précision.

Il fut ainsi développé des bibliothèques d'opérateurs destinés à l'arithmétique des nombres réels sur FPGA en particulier ceux utilisant la manipulation de nombre sous forme de virgule fixe ou flottante surtout dans le domaine de traitement d'images et de signaux.

Quel que soit le système de représentation adopté, le choix de la précision des valeurs manipulées joue un rôle prépondérant sur le coût et les performances du circuit. Ainsi, intuitivement, plus la précision requise sera grande et plus les opérateurs seront gros et lents. À l'inverse, un circuit conçu pour être petit et rapide sera enclin à faire des calculs entachés d'importantes erreurs.

3.4.1. Représentation en virgule fixe

La virgule fixe, format très proche des entiers, offre des opérateurs petits et rapides. Elle n'est néanmoins pas capable de supporter des nombres ayant une dynamique trop importante, ce qui restreint son utilisation essentiellement aux applications de traitement du signal.

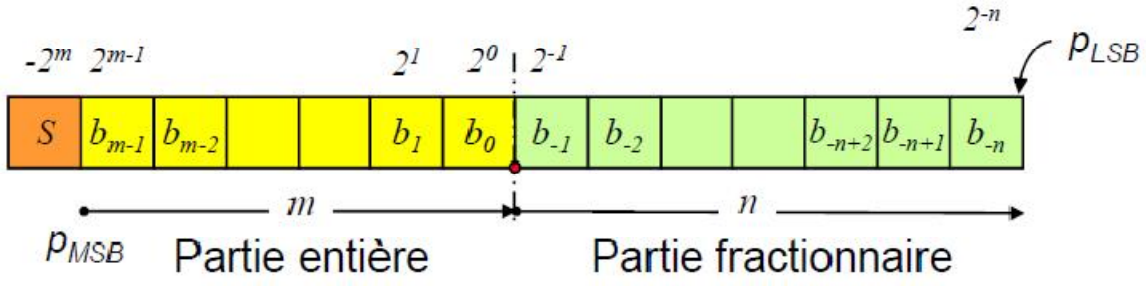


Figure 3.5 ***Représentation en virgule fixe***

$$x = -2^m S + \sum_{i=-n}^{m-1} b_i 2^i \quad (3.6)$$

• *m* : distance (en nombre de bits) entre la position du bit le plus significatif p_{MSB} et la position de la virgule p_v.

• *n* : distance entre la position de la virgule p_V et la position du bit le moins significatif p_{LSB}

- Précision du codage "Pas de quantification":$q = 2^{-n}$

- Domaine de définition du codage : D= [-2^m, $2^m - 2^{-n}$]

•**Addition** *a+b*
• *Choix d'un format identique*
• *Alignement de la virgule*
• *Extension de bits*

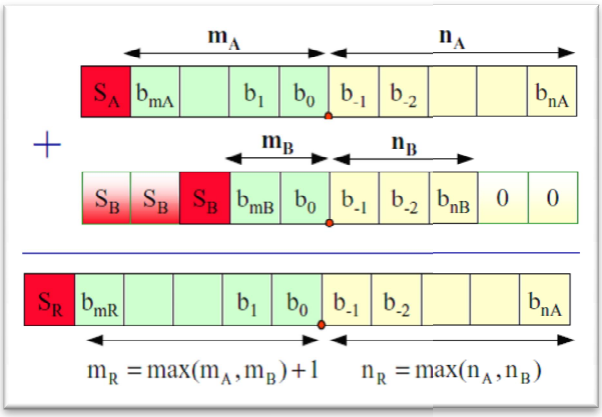


Figure 3.6 Addition de nombres sous format virgule fixe

• **Multiplication** *a* × *b*
• Représentation identique
  ✓ Doublement du bit de signe

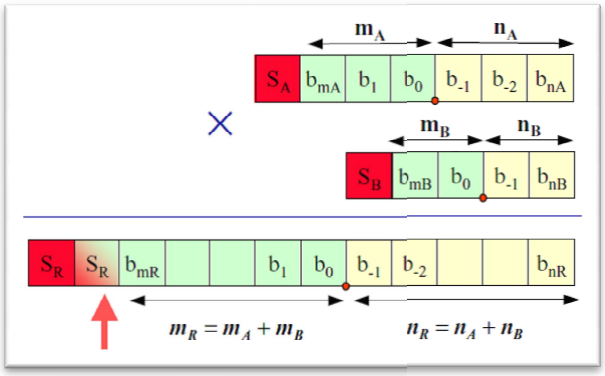


Figure 3.7 Multiplication de nombres sous format virgule fixe

3.4.2. Représentation en virgule flottante

Cette représentation permet de manipuler une gamme de valeurs et d'ordres de grandeur bien plus large, mais en contrepartie les opérateurs pour ce système sont relativement plus complexes que leurs équivalents en virgule fixe. Cependant, l'intérêt d'un tel système est que la plupart des algorithmes numériques en logiciel utilisent ce format, et leur portage sur circuit ne requiert aucune connaissance particulière en arithmétique.

Les nombes en virgule flottante sont représenter par la relation : $x = (-1)^S\, M\, 2^E$

- Exposant : *E*
- Mantisse : *M*
- Signe : *S*

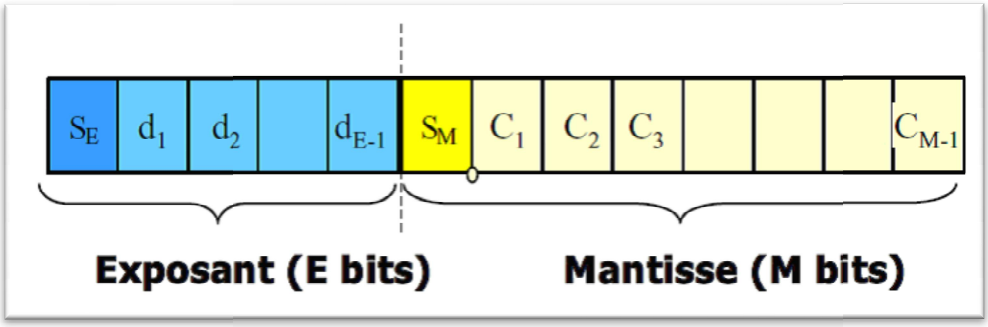


***Figure 3.8** Représentation de la virgule flottante norme IEEE 745*

$$x = 2^u(-1)^{S_E}\left(\frac{1}{2} + \sum_{i=1}^{M-1} C_i\, 2^{i-1}\right) \text{avec} u = (-1)^{S_E}\sum_{i=1}^{E-1} d_i 2^i \quad (3.7)$$

La norme IEEE 745 standardise plusieurs formats de représentation des nombres en virgule flottante dont la simple précision codée sur 32 bits et la double précision codée sur 64 bits. Au nombre de bits utilisés prés le schéma de codage est identique pour tous les formats.

	Simple précision			Double précision		
	Taille	Bits	Intervalle	Taille	Bits	Intervalle
Registre	32			64		
Signe *S*	1	x_{31}	[0 ;1]	1	X_{63}	[0 ;1]
Exposant *E*	8	$x_{30..}\, x_{23}$	[-126 ;+127]	11	$X_{62..}\, x_{52}$	[-1022 ;+1023]
Mantisse *M*	23	$x_{22..}\, x_0$	[0.0 ;1.0]	52	$X_{51..}\, x_0$	[0.0 ;1.0]

***Table 3.1** résumé de la norme IEEE 745*

3.4.2.1. Conversion d'un nombre en Virgule Flottante Double Précision (VFDP)

Exemple : convertir -532^{10} en format (VFDP)

Comme le nombre est négatif, le bit de signe est 1.

$532^{10}=214^{16}=1000010100^{2}=1,000010100x2^{9}$.

L'exposant biaisé est obtenu en ajoutant 1023. C'est donc 1023+9=1032.

Or, $1032^{10}=408^{16}=100000001000^{2}$ sur 11 bits.

La partie fractionnaire de la mantisse normalisée est ,000010100.

On a donc le résultat :

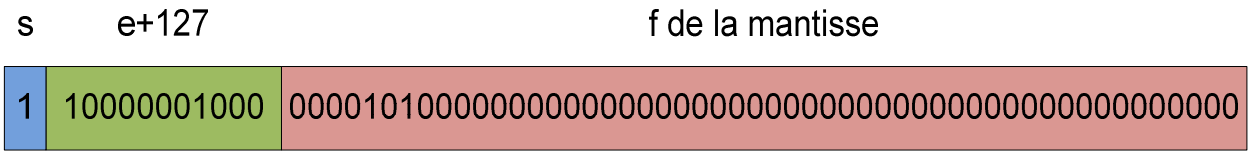


Figure 3.9 ***Représentation binaire de conversion d'un nombre en (VFDP)***

On peut écrire ce résultat sous la forme abrégée suivante :

C080A00000000000.

3.4.2.2. Conversion d'un nombre en Virgule Flottante Simple Précision (VFSP)

Exemple : convertir 432_{10} en format (VFSP)

a. Convertir en binaire par n'importe quelle méthode.$432_{10} = 110110000_{2}$

b. Multiplier par 2^{0}, ce qui ne change rien puisque c'est égal à 1 :

$$432_{10} = 110110000x2^{0}$$

c. Déplacer la virgule à gauche jusqu'au premier 1 et ajuster l'exposant en conséquence : $43210=1,10110000x2^{8}$

d. Ajouter 127 à l'exposant, et convertir en binaire : $8+127=135=10000111_{2}$

e. Inscrire le signe du nombre dans le premier bit du format IEEE. Dans le cas présent, le signe est +, donc écrire 0.

f. Placer l'exposant biaisé dans le champ exposant du format IEEE.

g. Placer la partie fractionnaire de la mantisse dans le champ de droite et compléter avec des 0.

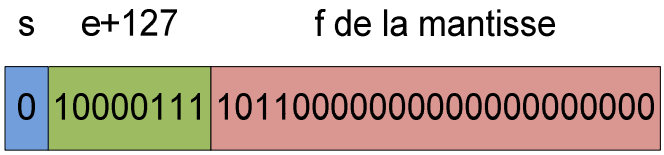


Figure 3.10 ***Représentation binaire de conversion d'un nombre en (VFSP)***

3.4.2.3. Addition de deux nombres en Virgule Flottante Simple Précision (A-VFSP)

Exemple : On additionne A=40E9999A et B=BE99999A en virgule flottante IEEE.

On a pour A :

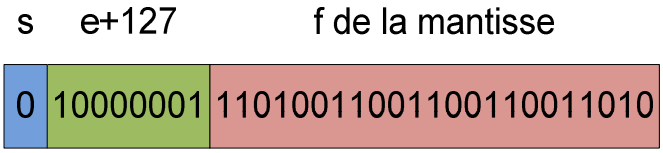


Figure 3.11 Représentation binaire du nombre A en (VFSP)

e+127=129, donc e=2, et la mantisse est 1,11010011001100110011010.

Donc A=1,11010011001100110011010x2^2

On a pour B :

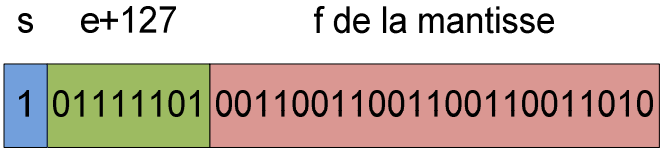


Figure 3.12 Représentation binaire du nombre B en (VFSP)

E+127=125, donc e=-2, et la mantisse est -1,00110011001100110011010.

Donc B=-1,0.000100110011001100110011010x2^2

On effectue A+B (ici il s'agit d'une soustraction puisque B est négatif) :

$$\begin{array}{ll} 1,1101\ 0011\ 0011\ 0011\ 0011\ 010 & x2^2 \\ -0,0001\ 0011\ 0011\ 0011\ 0011\ 001\ 1010 & x2^2 \\ \hline 1,1100\ 0000\ 0000\ 0000\ 0000\ 000 & x2^2 \end{array}$$

On n'additionne pas les quatre derniers bits de B, puisqu'ils sont hors des 23 bits significatifs.

On replace le résultat dans le format IEEE :

E+127=129=10000001

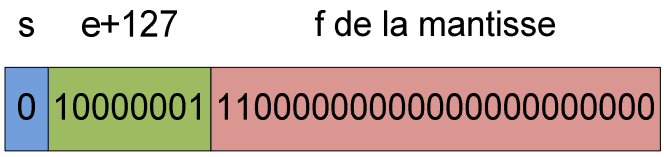


Figure 3.13 Représentation binaire du résultat de l'addition de A et B en (VFSP)

3.4.2.4. Multiplication de deux nombres en Virgule Flottante Simple Précision (M-VFSP)

Exemple : On multiplie A=40A00000 et B=C0E00000virgule flottante IEEE.

On a pour A :

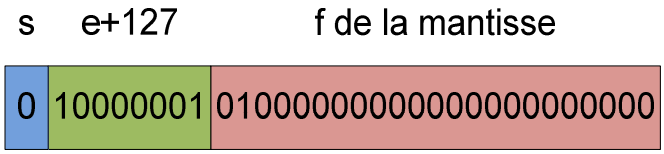


Figure 3.14 Représentation binaire du nombre A en (VFSP)

Donc $A=1,01x2^2$

On a pour B :

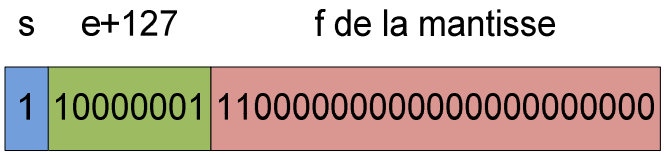


Figure 3.15 Représentation binaire du nombre B en (VFSP)

Donc $B=-1,01x2^2$

On effectue le produit :

$$\begin{array}{rl} 1,01 & x2^2 \\ -1,11 & x2^2 \\ \hline 101 & \\ 101 & \\ 101 & \\ \hline -10,0011 & x2^4 \end{array}$$

On normalise le résultat pour le mettre sous la forme 1,f et on obtient :

$AxB=-1,00011x2^5$

On replace ce résultat dans le format IEEE :

Exposant+127=5+127=132=10000100

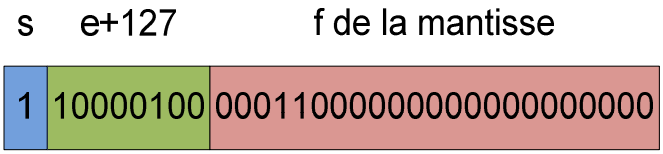


Figure 3.16 ***Représentation IEEE de la multiplication de A et B en (VFSP)***

3.5. Résumé

Dans ce chapitre nous avons introduit des notions sur le filtrage du signal et nous avons retenu le filtre numérique de type FIR après avoir énumérer ses qualités et ses applications. Pour des raisons de précision et de normalisation toutes les valeurs sont prisent en virgule flottante norme IEEE 745.

Chapitre 4

IMPLANTATION MATERIELLE DU FILTRE NUMERIQUE "FIR"

4.1. Introduction

Nous nous plaçons dans la situation d'une conception de système embarqué qui doit implémenter une application comprenant une étape de filtrage numérique. Il s'agit d'un filtre à Réponse Impulsionnelle Finie (**FIR**) à 16 étages. Ce filtre est appliqué à un tableau de donnée (input [ ]) et produit en résultat un tableau de résultats (output [ ]).

La conception en question devra utiliser des outils performants de développement de synthèse et de simulation. Dans ce chapitre nous allons décrire la structure générale et spécifique des différents modules utilisés dans la conception.

4.2. Structure et model du filtre

Le paragraphe suivant va se concentrer uniquement sur la partie "accélérateur matériel" qui assure la partie traitement et filtrage numérique.

Comme nous l'avons déjà indiqué, il s'agit de faire l'implantation d'un coprocesseur destiné à accomplir la fonction filtrage numérique FIR L'équation ou modèle mathématique du filtre est donnée par:

$$y(n) = \sum_{i=0}^{N-1} x(n-i).c(i) \quad (4.1)$$

Où ***n*** est le numéro d'échantillon et ***N*** est l'ordre du filtre (pris dans notre cas égal à 16).

Dans la mise en œuvre appropriée, les coefficients et les échantillons du filtre seront codés en virgule flottante sur 32 bits.

A cause de l'opération de multiplication et pour éviter de s'affranchir des risques de débordements de calculs, nous veillerons à réaliser les accumulations successives sur 40 bits.

L'algorithme software est très coûteux en temps de calcul, aussi nous allons chercher à réaliser ce traitement sous la forme d'une architecture matérielle spécialisée sur une structure implantée sur une plate-forme reconfigurable à base d'un FPGA ALTERA CYCLONEII. Cette architecture sera décomposée en une unité de traitement et une unité de contrôle réalisée sous la forme de machine à état (FSM). L'objectif à

long terme est d'en faire un co-processeur pour le processeur NIOS II tel qu'illustrée par la ***"figure 4.1"***.

Il est à noter que nous avons utilisé le langage VHDL (*VHSIC Hardware Description Language*) comme langage de description des différentes unités, Quartus II et SOPC-Builder comme outil de développement et synthèse.

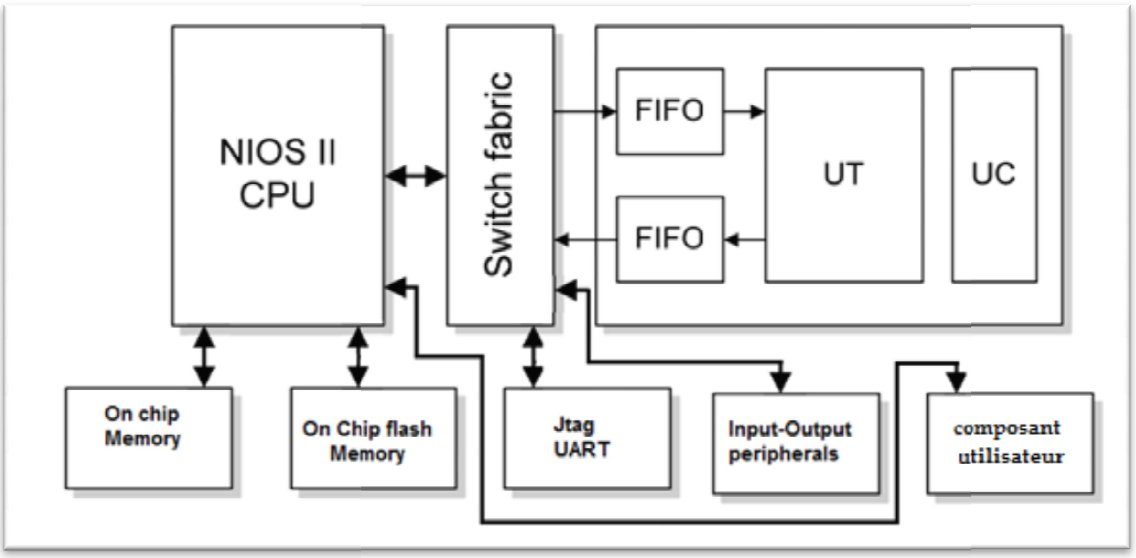


Figure 4.1 Structure bloc générale du système embarqué

La conception de ces deux unités se base sur la structure algorithmique suivante:

Début

1. ***Initialisation matérielle*** **reset matériel**
2. ***X[adx]=read_fifo();*** ** lecture des échantillons**
3. ***y=0; i=0;*** **initialisation interne des registres**
4. ***while(i<16) faire:***

   y=y+X[adx]*C[i];

   adx=(adx+1) mod[16] **boucle des itérations**

   inc(i)

Fin tant que

5. ***adx=(adx-1)mod[16];***
6. ***write_fifo(y);***

Fin

Figure 4.2 diagramme exécutif fonctionnel

4.3. Eléments constructifs de l'implantation

D'après la structure de la *"figure 4.1"*, la partie concernée par notre travail consiste en trois modules essentiels: un module interfaçage entre le système **NIOS** et l'application filtrage assurée par deux composants de même type et jouant le rôle de régulateur de flux: c'est l'interfaçage par **FIFO**.

Les deux autres modules concernent la partie chemin de données assuré par l'unité de traitement **UT** et le séquenceur logique déterminé à partir de l'unité de contrôle **UC**.

4.3.1. Interfaçage par FIFO

Le composant qui réalise le traitement FIR est connecté au reste du système à travers une interface à base de FIFO matérielles. Ce type d'interface est très utilisé pour des traitements qui opèrent sur des flux de données en entrée, et produisent un flux de données en sortie. Une FIFO matérielle se présente en général comme un composant disposant de deux ports indépendants : un pour la lecture et l'autre pour l'écriture comme décrit ci-dessous :

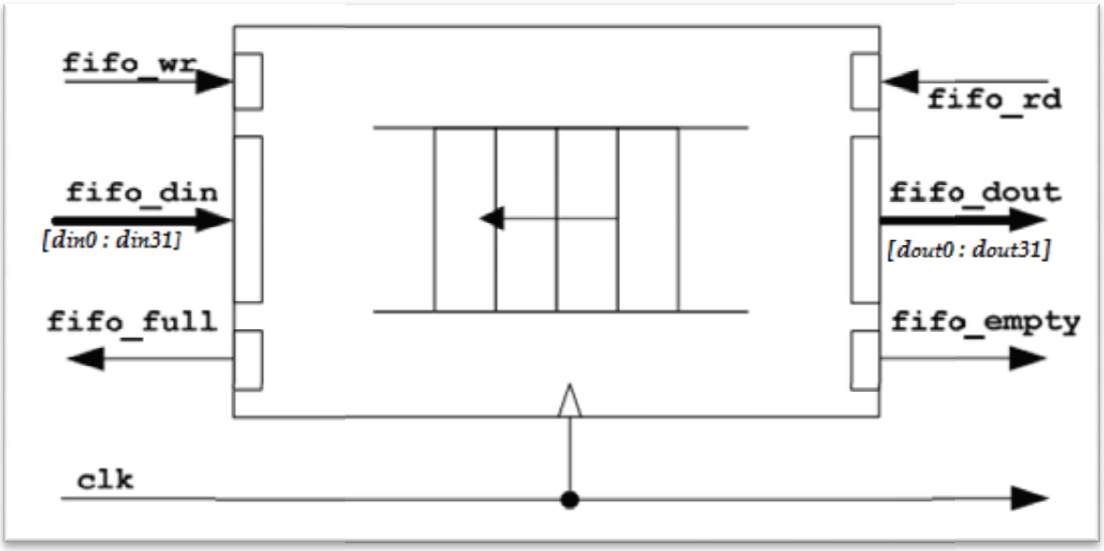


Figure 4.3 ***Principe fonctionnel d'une FIFO***

La FIFO présente deux modes essentiels d'accès: le mode d'accès en lecture et le mode d'accès en écriture.

4.3.1.1. Accès en lecture dans une FIFO

L'interface coté lecture est formée par trois principaux signaux :

- **fifo_read** : c'est un signal de commande de lecture actif à **'1'** qui est pris en compte sur un front montant d'horloge. Si la commande de lecture est activé au cours du cycle n, la donnée demandée est disponible sur le port de donnée **fifo_dout** au cycle n+1.
- **fifo_empty** : est à '1' si la FIFO ne contient aucune donnée, à **'0'** sinon. Une demande de lecture sur une FIFO vide a pour effet de corrompre le contenu de la FIFO.
- **fifo_dout[15:0]** : c'est le port de donnée de lecture de la FIFO. Il est mis à jour sur un front montant d'horloge (lorsque **fifo_read='1'**).

Un chronogramme résumant le fonctionnement d'une interface FIFO en lecture est donné ci-dessous

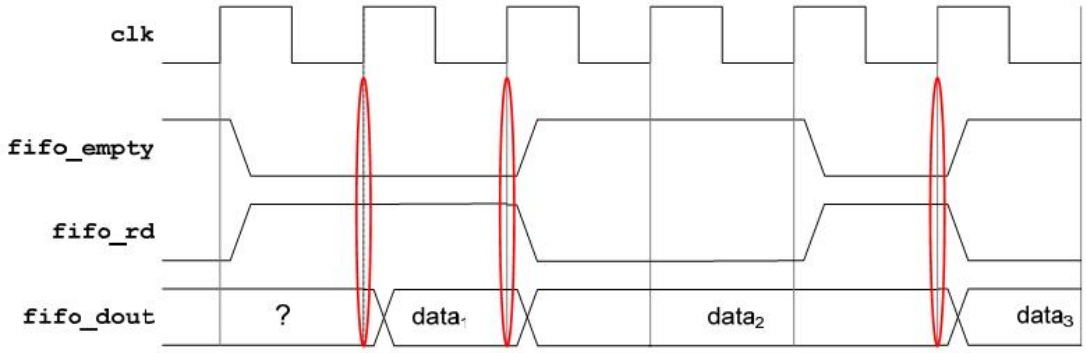


Figure 4.4 ***Chronogramme de lecture pour une FIFO***

4.3.1.2. Accès en écriture dans une FIFO

Quant à l'interface coté écriture, elle est aussi formée par trois signaux :

- **fifo_write** : c'est un signal de commande d'écriture actif à **'1'** qui est pris en compte sur un front montant d'horloge. Si la commande d'écriture est activée au cours du cycle n, la donnée présente sur le port **fifo_din** est stockée dans la FIFO sur le front montant d'horloge.
- **fifo_full** : est à **'1'** si la FIFO est pleine, et à **'0'** sinon. Une demande d'écriture sur une FIFO plein a pour effet de corrompre le contenu de la FIFO.

- **fifo_din[15:0] :** c'est le port de donnée d'écriture de la FIFO. Une donnée sur ce port est stockée dans la FIFO sur un front montant d'horloge lorsque **fifo_write='1'**.

  Un chronogramme résumant le fonctionnement d'une interface FIFO en écriture est donné ci-dessous

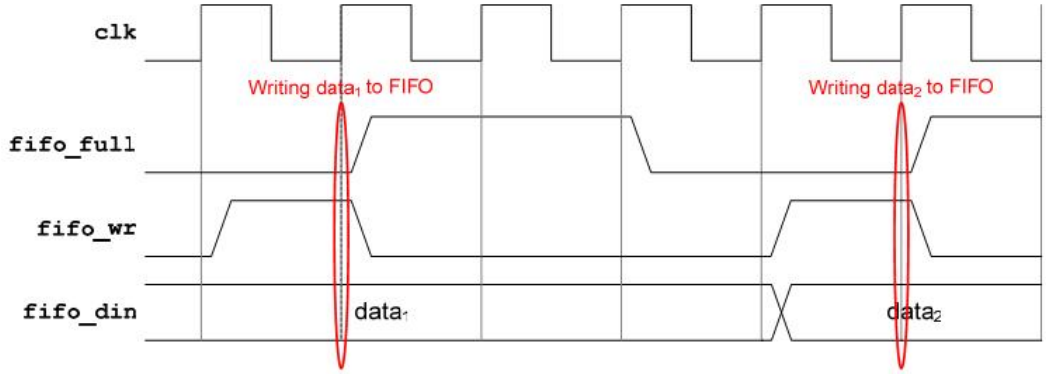


Figure 4.5 ***Chronogramme d'écriture pour une FIFO***

4.3.1.3. Accès concurrent en lecture et en écriture

Les FIFO sont généralement mises en œuvre à l'aide de mémoires « double ports » qui permettent de faire une lecture et une écriture à des adresses différentes pendant un même cycle. Cela autorise un fonctionnement concurrent de la lecture et de l'écriture (plus prosaïquement on peut écrire une donnée dans la FIFO et lire une donnée de la FIFO en même temps sans problème).

Le comportement de fonctionnement de l'interface FIFO a été décrit en langage de description matérielle VHDL (FIFO.vhd), puis synthétisé par Quartus afin d'être instancié comme composant dans de futures utilisations.

4.3.2. L'unité de traitement

Cette unité accomplit les deux opérations de base permettant de mener à bien la fonction filtrage FIR. Elle est composée de plusieurs sous modules représentés par la "***figure 4.6".***

La fonction itérative est réalisée grâce à deux opérations: un produit suivi d'une addition accumulée ou multiplication accumulée **MAC**. Ces opérations sont modélisées par deux composants **additionneur** et **multiplicateur** virgule flottante et instanciés

dans la description de l'unité de traitement UT. Les résultats sont validés et contrôlés de façon comportementale par un process (instruction concurrente se manifestant sous forme d'une série d'instructions séquentielles) en langage VHDL.

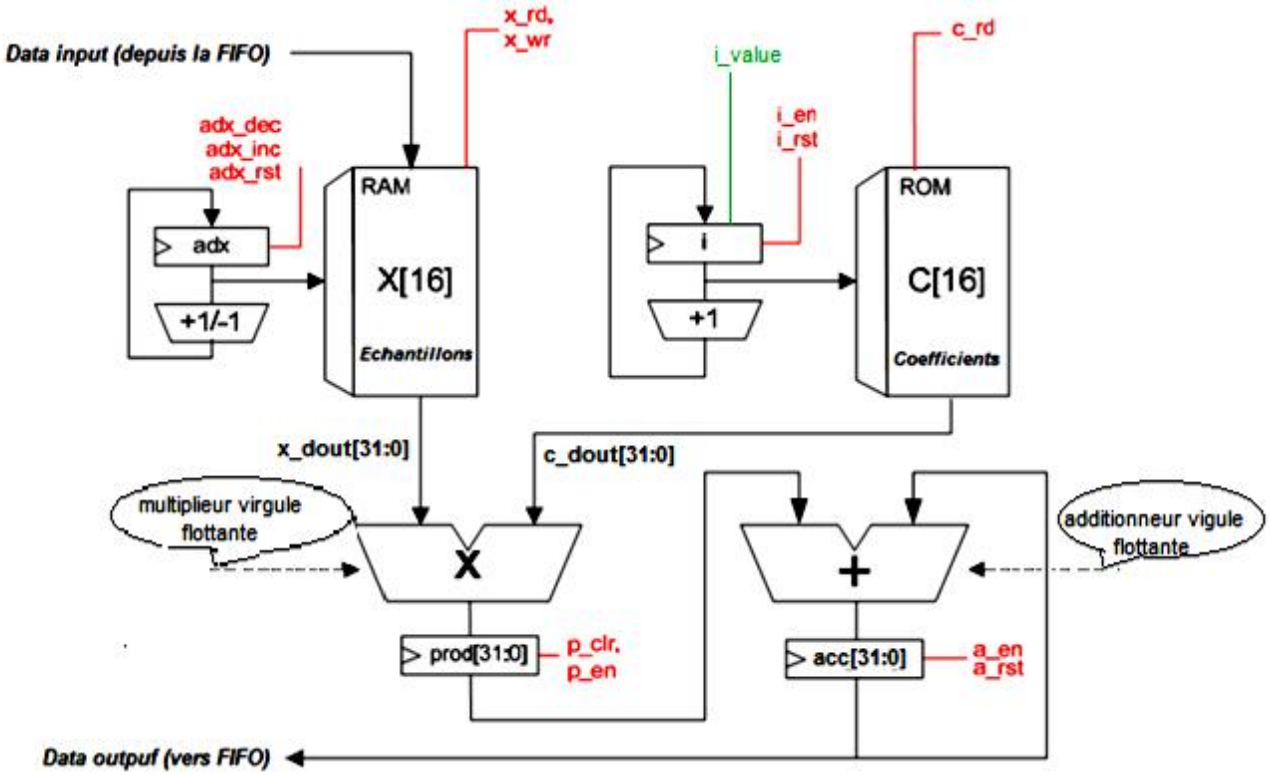


Figure 4.6 architecture structurelle de l'unité de traitement UT

4.3.3. Unités de calcul

4.3.3.1. Unité de multiplication virgule flottante

La multiplication de deux nombres en virgule flottante est effectuée en trois grandes étapes:

- Multiplication des mantisses.
- Addition des exposants.
- Détermination du signe.

$$A = (-1)^{sA} * m_A * 2^{eA}$$
$$B = (-1)^{sB} * m_B * 2^{eB}$$
$$P = A * B$$
$$P = ((-1)^{sA} * m_A * 2^{eA}) * ((-1)^{sB} * m_B * 2^{eB})$$
$$P = (-1)^{(sA+sB)} * (m_A * m_B) * 2^{(eA+eB)}$$

$$m_P = m_A + m_B$$
$$e_P = e_A + e_B$$
$$s_P = s_A + s_B$$

On remarque que ces trois opérations sont indépendantes les unes des autres et qu'elles se prêtent donc à une exécution en parallèle. C'est ce qui donne plus de force à favoriser une implantation matérielle.

La synthèse matérielle de cet opérateur a été réalisée en utilisant le langage haut niveau VHDL, sous QUARTUS II.

Le résultat de la synthèse dans un FPGA 35CF672 niveau RTL a donné la structure représentée par la "***figure 4.7***".

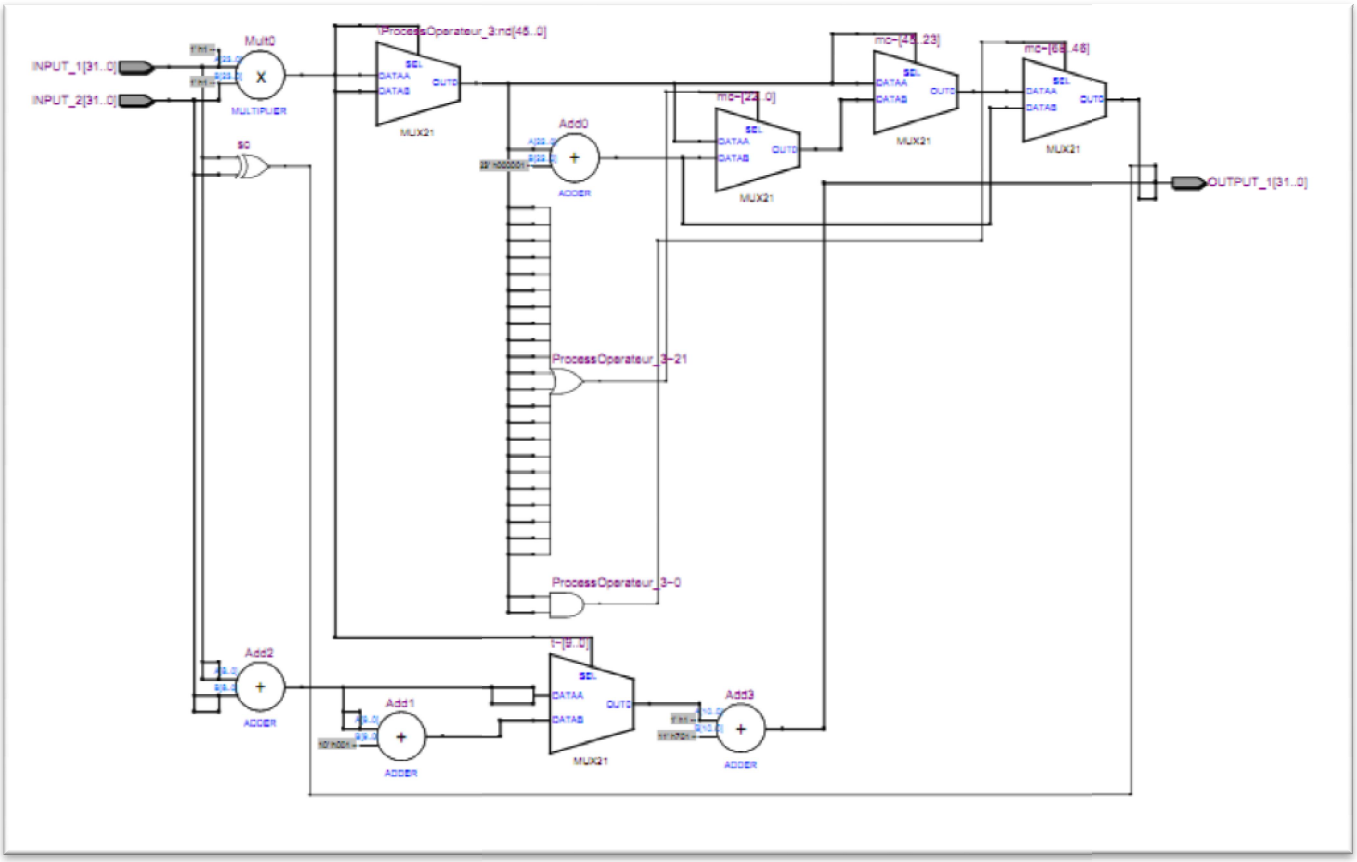
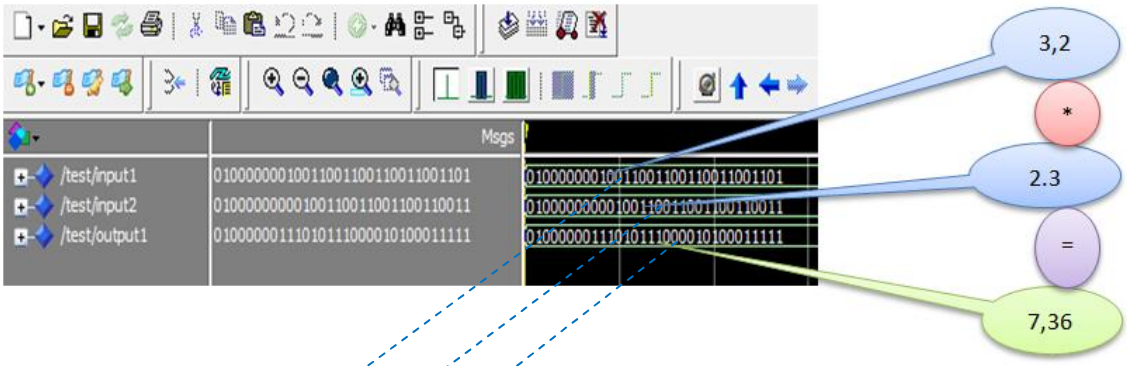


Figure 4.7 Structure du multiplieur 32 bits en virgule flottante niveau RTL

Le fonctionnement de ce multiplieur est validé par la conception d'une unité banc test ou unité de test, décrite elle aussi toujours par une description en langage VHDL utilisant **ModelSim.**

La "*figure 4.8"* montre les états des différents bits de sortie selon des échantillons de bits d'entrées représentant le multiplicateur et le multiplicande.

Figure 4.8 Résultats de la simulation du multiplieur.



Interprétation:

Nous allons vérifier une opération de multiplication réelle par exemple :

$$« \ 3.2 * 2.3 = 7.36 »$$

La représentation des nombres en virgule flottant 32 bit est commet suivant :

3.2= « 01000000010011001100110011001101».

2.3= « 01000000000100110011001100110011».

7.36= « 01000000111010111000010100011111».

La simulation de premier opération donne (3.2 *2.3=7.36)

4.3.3.2. Unité additionneur binaire Virgile flottant

Le processus de l'addition est plus complexe comparé à celui de la multiplication, il est basé sur quatre étapes :

- Alignement des mantisses si les exposants de A et B sont différents.
- Addition ou soustraction des mantisses alignées.
- Normalisation de la mantisse de la somme S si elle n'est pas normalisée.
- Arrondi de la mantisse de la somme S.

Nous nous contentons dans cette partie de vous donner seulement les résultats trouvés au niveau structure RTL, et simulation avec une petite interprétation. Concernant la structure niveau RTL, et puisque l'additionneur se prétend plus complexe grâce à son aspect parallèle, les résultats sont donnés en différents niveaux "Le niveau RTL1 block, et les niveaux RTL de chaque étage".

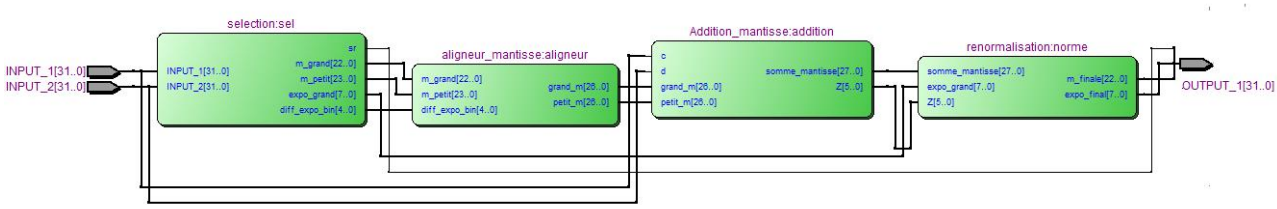


Figure 4.9 ***représentation RTL bloc d'un additionneur virgule flottante***

La structure du module de Sélection **Sel est** donnée par le schéma RTL suivant:

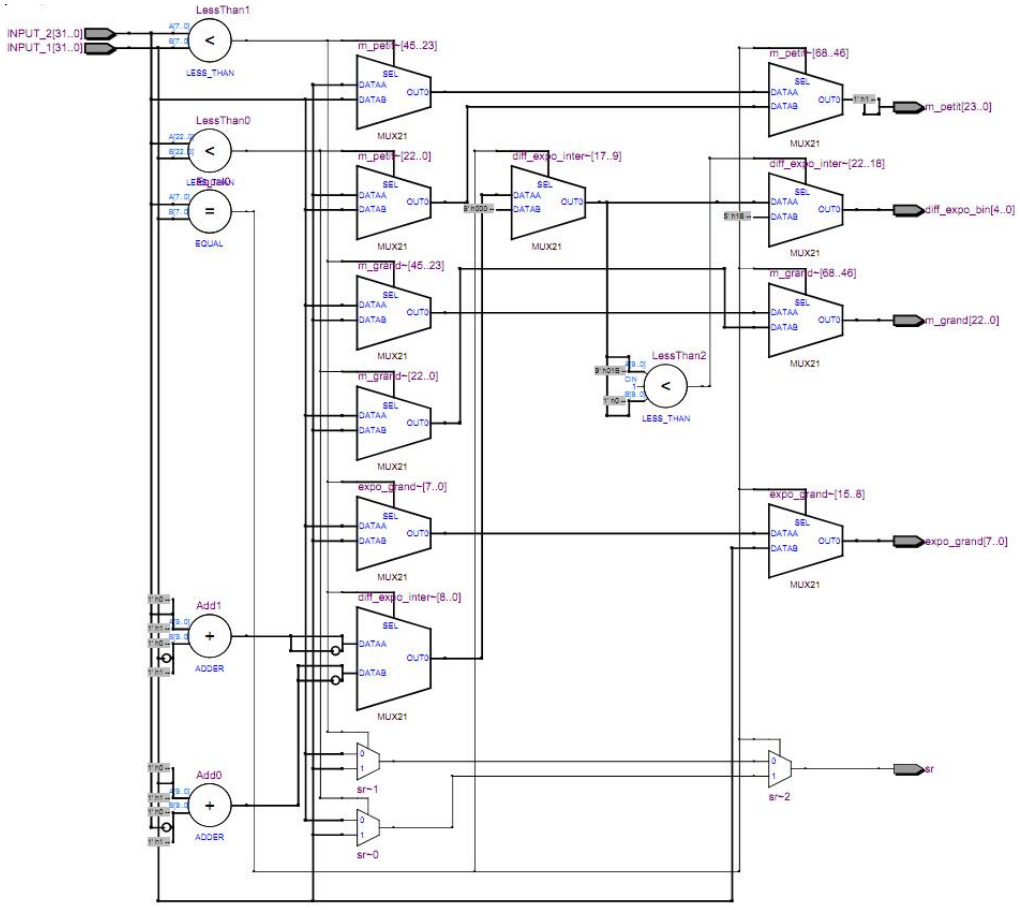


Figure 4.10 ***Résultât de la synthèse du module Sélection Sel***

Quand au résultat de synthèse niveau RTL de l'aligneur de mantisse se présente par la figure suivante:

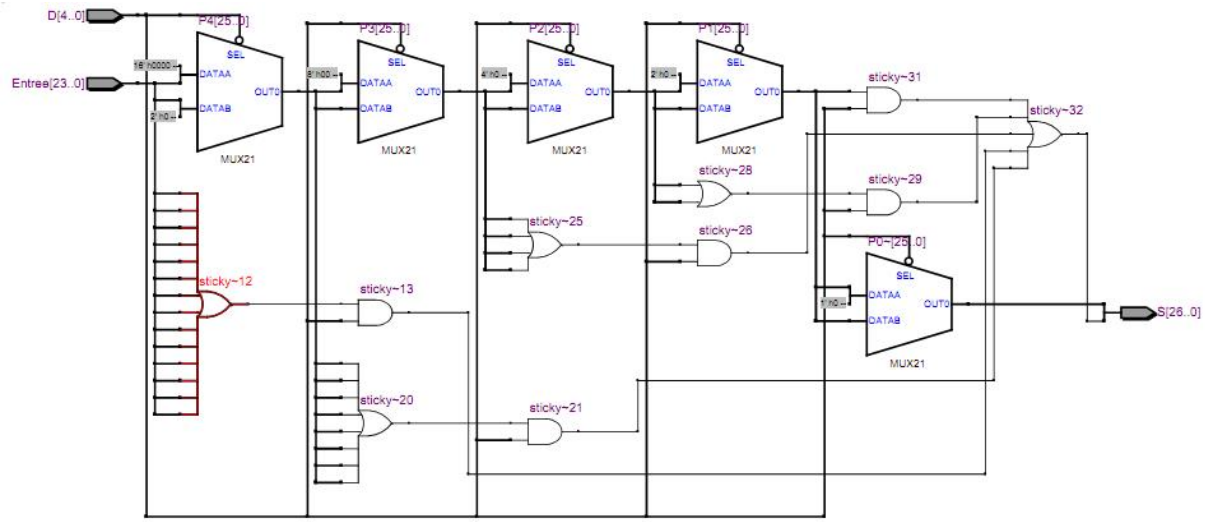


Figure 4.11 ***Structure RTL de l'aligneur de mantisse***

Les deux figures suivantes représentent respectivement les résultats de synthèse niveau RTL obtenus pour l'étage d'additionneur de mantisse et le module de normalisation.

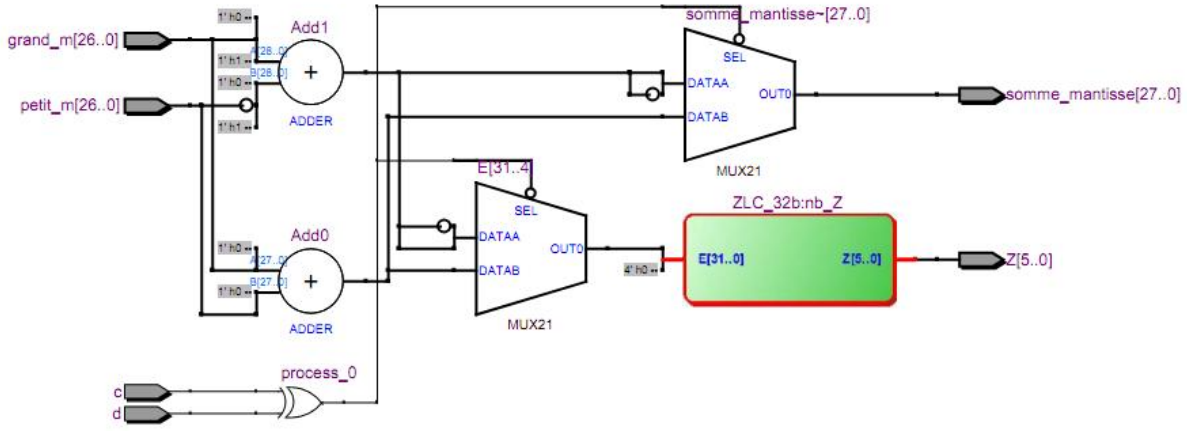


Figure 4.12 ***Additionneur de mantisse niveau RTL***

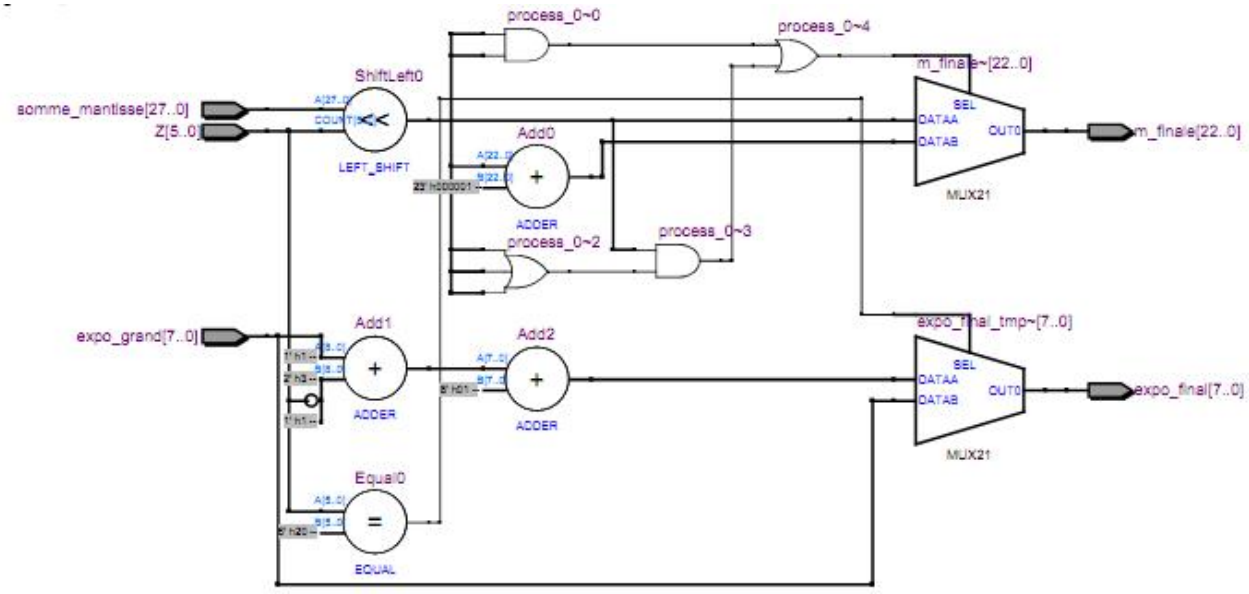


Figure 4.13 ***Synthèse RTL de l'étage de normalisation***

Une fois synthétisé le fonctionnement de l'additionneur est soumis à un test fonctionnel en élaborant une unité de test développée en langage VHDL.

Les résultats de simulation sont conformes et sont représenté à travers un exemple par la figure suivante:

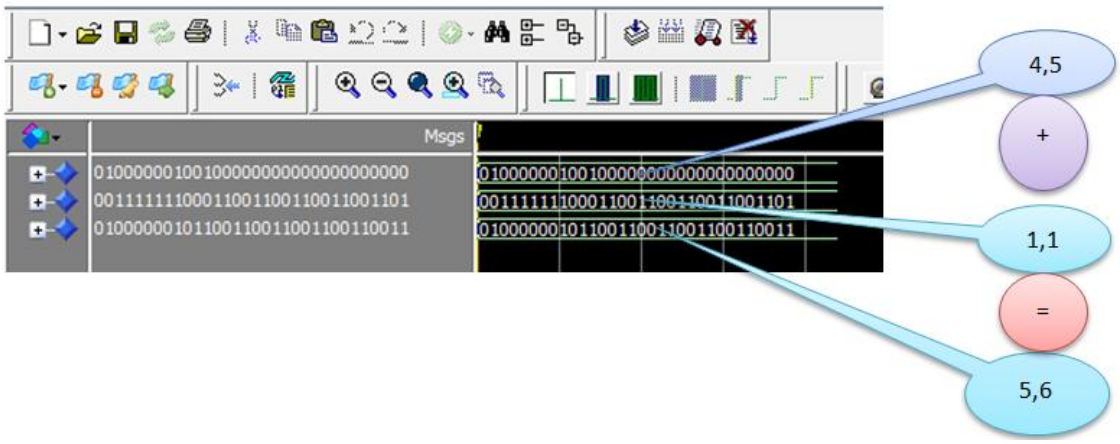


Figure *4.14* ***Résultats de la simulation MODELSIM***

4.3.3.3. Interprétation:

Nous allons vérifier maintenant le fonctionnement de notre additionneur par un exemple réel choisi comme suit: « $4.5 + 1.1 = 5.6$ »

La représentation virgule flottante de 4.5= "01000000100100000000000000000000".

La représentation virgule flottante de1.1="00111111000110011001100110011001101".

Le résultat 5.6= "01000000101100110011001100110011". ***Test Confirmé***

Les contrôles de l'évolution des adresses mémoire ROM et RAM permettant de pointer les adresses des échantillons et coefficients sont aussi modélisés par des process directement dans la description de l'unité de traitement UT.

La RAM et la ROM sont deux composants utilisés par cette unité, ces composants sont préalablement modélisés en VHDL puis instancié dans la description.

Il est à noter qu'à chaque fois qu'il faudra utiliser un composant pour être instancier dans une conception, ce composant devra être déjà modélisé préalablement et synthétisé d'avance. C'est le cas aussi de nos fameux composants qui réalisent la multiplication et l'addition vus dans le chapitre précédent.

4.3.4. Utilisation de composants mémoires

Le chemin de données du composant utilise des mémoires pour stocker les coefficients et les échantillons à traiter. Nous utiliserons ici des composants mémoire prédéfinis, disponible dans la librairie de composants du FPGA que nous utiliserons par la suite (en l'occurrence un circuit Altera).

4.3.4.1. Lecture sur une RAM/ROM synchrone

Dans le cas d'un RAM/ROM synchrone, la prise en compte d'une commande de lecture ne se fait que sur les fronts montant d'horloge. En pratique, si la commande d'autorisation de lecture (**rd_en**='1') et l'adresse sont présentes au cycle *n*, la donnée sera disponible en sortie de la mémoire au cycle *n+1*. Un exemple de fonctionnement d'une RAM synchrone est donné par la ***"figure 4.7"***.

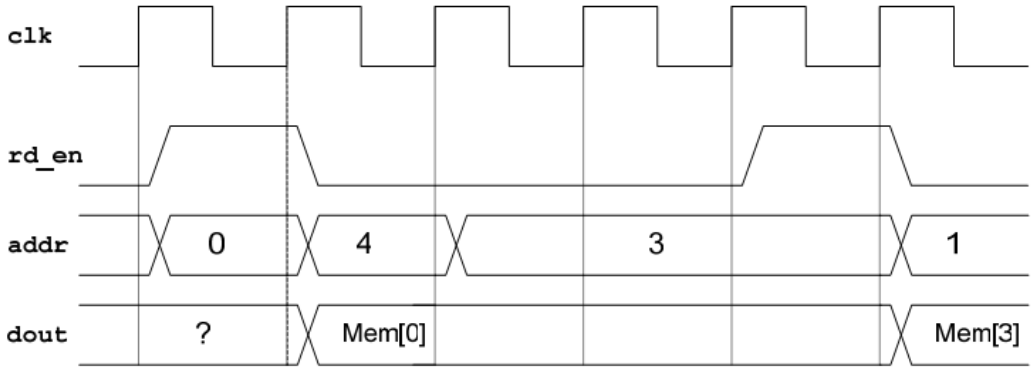


Figure 4.15 ***Chronogramme de lecture dans une RAM/ROM synchrone***

4.3.4.2. Ecriture dans une RAM synchrone

Dans le cas d'un RAM synchrone, la prise en compte d'une commande d'écriture se fait également sur un front montant d'horloge. En pratique, si **wr_en='1'**, et on a la donnée et l'adresse présentes au cycle *n*, la donnée sera considérée comme écrite à partir du cycle n+1. Un exemple de fonctionnement d'une RAM synchrone en écriture est donné ***"Figure 4.16"***.

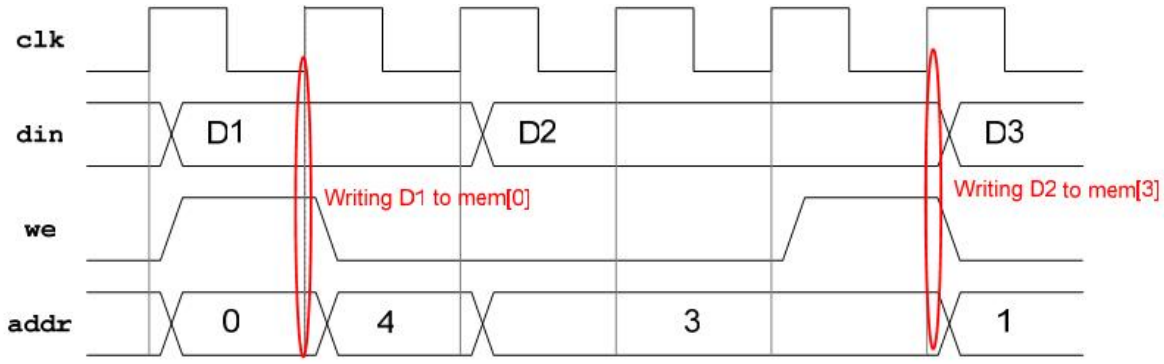


Figure 4.16 ***Chronogramme d'écriture dans une RAM synchrone.***

4.3.5. L'unité de contrôle ou séquenceur RIF

Cette unité génère tous les signaux de contrôle approprié pour l'unité de traitement et calcul son schémas block est donné par la ***"figure 4.17"***, alors que son modèle est régit grâce à une machine d'état FSM de la ***"figure 4.18"***:

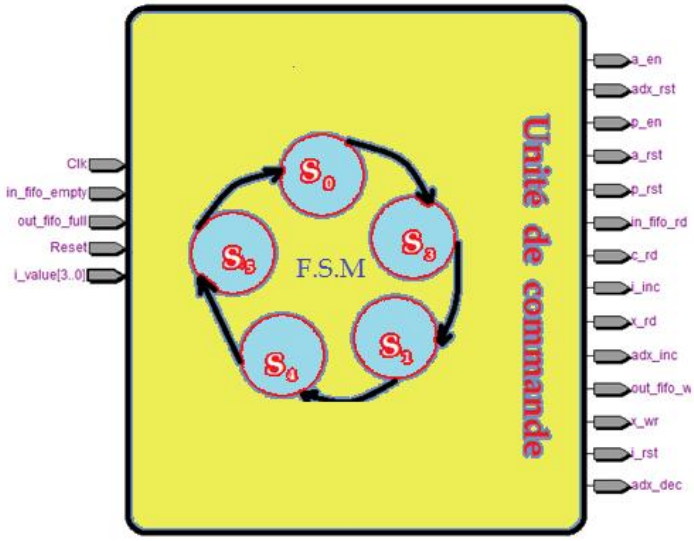


Figure 4.17 ***Entité de l'unité de commande***

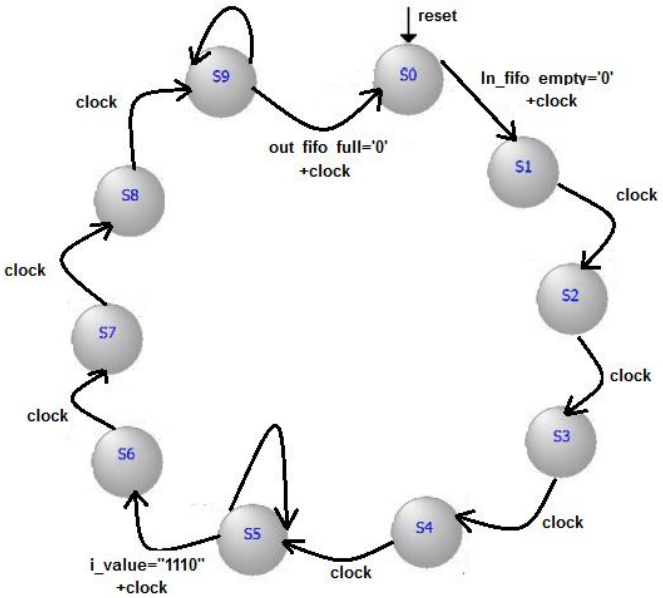


Figure 4.18 ***Modèle FSM de l'unité de contrôle d'un filtre FIR***

La structure niveau RTL donné après synthèse de cette machine à état sous Quartus II est donnée par la ***"figure 4.19"***, nous remarquons qu'il s'agit d'une machine

de Moore étant donné que les sorties sont belle et bien en fonction des entrées et des états actuels de la machine

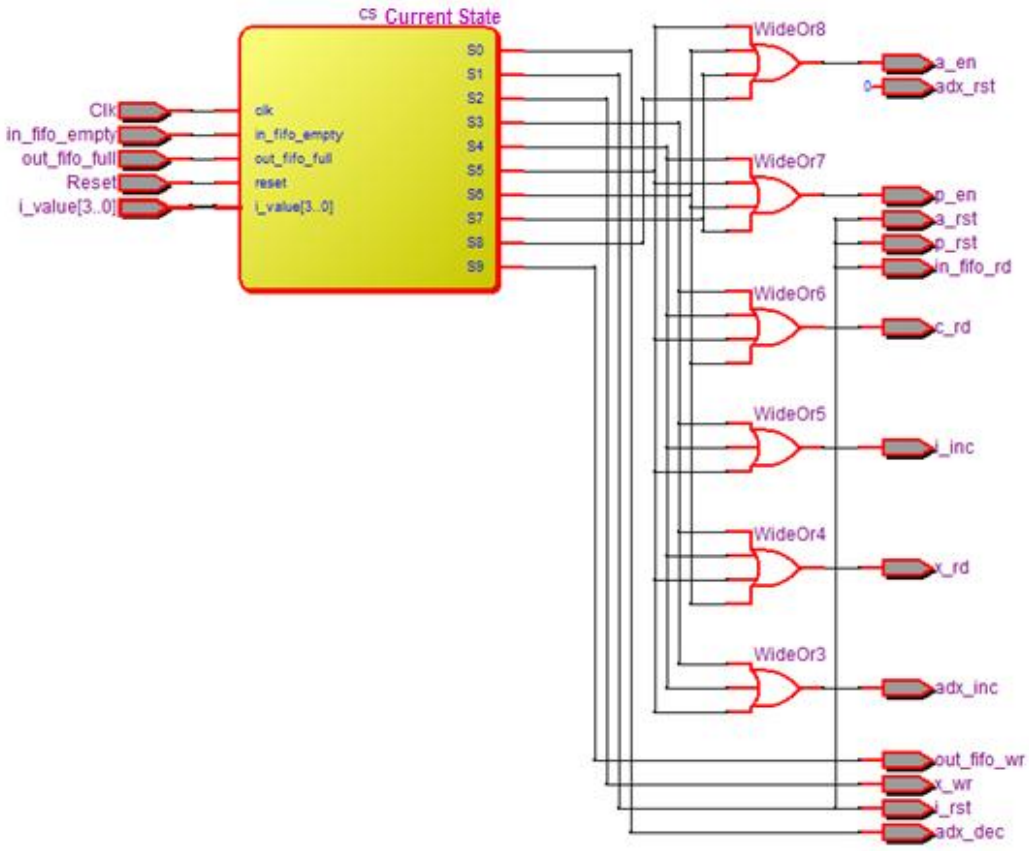


Figure 4.19 ***Synthèse RTL de la machine représentant l'unité de commande.***

4.4. Résumé

Dans ce dernier chapitre nous avons présentait les diverses parties structurelles de l'implantation du filtre **FIR** retenu pour le traitement du signal numérique.

Conclusion générale

En vue de la conception de systèmes embarqués nécessitant une étape de filtrage numérique, nous avons implémenté une application comportant une itération d'opérations de multiplication accumulation qui généralement, dans des systèmes classiques (software) s'exécute en plusieurs cycles de temps et donc handicape les cas temps réel.

L'application choisie concerne une implémentation d'un filtre numérique dans une plateforme reconfigurable (FPGA).

Ce filtre est de type réponse impulsionnelle finie d'une profondeur de calcul étalée sur 16 étages utilisant une précision fine grâce à un décodage de données réalisé sur un format 32 bits, virgule flottante.

L'architecture de ce filtre a était décomposée en une unité chemin de donnée, unité de traitement et une unité de contrôle réalisée sous la forme de machine à état (FSM).

Le langage VHDL, langage de description hardware a été utilisé pour la modélisation et la synthèse des différentes unités et modules (IPs), sur un environnement de développement utilisant la suite ALTERA QUARTUS-II, MODELSIM et SOPC-Builder.

Le StarterKit DE2 cyclone II 35F672C6 nous a permis de faire le prototypage et les tests intermédiaires réels des unités de conception réalisées.

L'assemblage des différents modules en un composant unique IP, contrôlé par un processeur softcore NIOSII, fera une idée géniale, pour choisir des opérations a partir d'une variété d'applications rassemblées dans une bibliothèque (DCT, la FFT, DFT, RII, PWM…) hardware pour des projets sophistiqués de traitement d'image.

Bibliographie

[1] Master Electronique et Télécommunications Synthèse de filtres (Cours), 2006-07.

[2] S.Mirsaki, A.Hosangadi and R.Kastner, FPGA High speed FIR filter using add and shift Method, 2006.

[3] Mourice B, Masson, "traitement numérique d'un signal théorie est pratique".

[4] BS2EL cours Physique appliquée, filtrage numérique.

[5] Messerli; Etienne "manuel VHDL", haute école d'ingénieur et de gestion du canton, France, Septembre 2007.

[6] Khare, Rakhi Thakur and Kavita, High Speed FPGA Implementation of FIR Filter for DSP, February 2013.

[7] ALTERA corporation. DE2 Development and Education Board, 2007.

[8] Pedroni; Volnei A. Circuit Design with VHDL, MIT Press , Cambridge Massachusetts, London 2004.

[9] Cooperation Altera QuartusII, Scupling Reference Manual, December 2009 WWW.altera.com.

[10] Corporation Altera , CycloneII FPGA Starter Development Board Reference Manual Document, October 2006.

[11] Getting Started Started with Quartus II Simulation using the ModelSim-Altera, Software user guide, June 2011 www.Altera.com.

[12] K.N.Macpherson, low FPGA area multiplier parallel FIR filters, dec 2004.

ملخص

من خلال هذا العمل، استغللنا أداء شرائح (FPGA) على وجه الخصوص إصدار (Altera Cyclone-II) لإظهار فعاليتهم لتنفيذ التطبيقات المتعلقة بمعالجة الإشارات "مرشح FIR" وهذا من خلال التوازي في الحساب الممكن أداؤه بواسطة شرائح (FPGA).

Abstract

Through this work, we have exploited the performance of FPGAs in particular their recent release (Altera Cyclone-II) to show the effectiveness of them to implement applications related to signal processing "FIR filter" this in large part through the parallelism in the computation allowed by FPGAs

Résumé

A travers ce travail, nous avons exploité les performances des circuits FPGAs en particuliers leurs version récentes (Altera Cyclone-II) pour montrer l'efficacité d'y implémenter des applications liées au traitement de signal "filtre de type FIR" et cela en grande parti grâce à la tendance au parallélisme dans le calcul permis par les FPGAs.