Chapitre 3

Technologie et environnement de développement des FPGA

Le développement des systèmes sur les FPGA est différent du développement de systèmes conventionnels puisqu'il fait appel à la conception physique mais également à la conception logicielle. L'utilisation d'un langage de description de matériel, qui partage certains aspects avec les langages de programmation conventionnels, n'est pas triviale. Ce langage sert à définir physiquement le circuit au niveau d'abstraction choisi par le concepteur. Ainsi les stratégies de développement sont différentes de la conception logicielle qui décrit généralement un traitement séquentiel. La frontière entre le développement matériel et logiciel est encore moins évidente lorsqu'il s'agit du développement de SoC.

En premier lieu, les caractéristiques de base de la technologie FPGA sont présentées à la prochaine section tandis que les SoC sont discutés à la section 3.2. La section 3.3 porte sur les langages de description de matériel mais plus particulièrement sur le *Very-High-Speed Integrated Circuit* (VHSIC) *Hardware Description Language* (VHDL). Finalement la section 3.4 présente les logiciels utilisés lors de ce projet ainsi que la carte de développement FPGA qui a servi pour l'implémentation du simulateur temps réel.

3.1 Technologie FPGA

Les circuits intégrés *Field Programmable Gate Array* contiennent une grande quantité d'éléments de base, qui servent à synthétiser des fonctions logiques ou mathématiques, connectés entre eux par une matrice programmable d'interconnexions. Moins rapide et plus gourmand en puissance et en surface de silicium qu'un circuit intégré dédié (*Application-Specific Integrated Circuit*, ASIC), un FPGA a l'avantage d'être reprogrammable ainsi que d'être moins coûteux en temps et en coût de développement. Cependant, une fois le produit développé, un FPGA sera toujours plus coûteux qu'un ASIC. On s'attarde principalement sur la structure des FPGA *Xilinx* puisque c'est un *Virtex II Pro VP30* de *Xilinx* qui est utilisé pour la réalisation de ce projet.

3.1.1 Structure interne

Les FPGA sont composés, comme tous les circuits intégrés, d'un substrat de silicium ayant plusieurs régions dopées N ou P pour fabriquer les transistors ainsi que de polysilicium pour réaliser la grille des transistors. Actuellement, la résolution varie entre 150 et 90 nm. Par la suite, plusieurs couches de métal, entre sept et neuf et habituellement en cuivre [36, 37 et 39 à 41], constituent les couches supérieures. Ces couches métalliques servent à effectuer les connexions entre les transistors pour former les fonctions plus complexes. Tel qu'illustré à la figure 3.1, selon la compagnie et le modèle de FPGA, il y a des modules embarqués dédiés (*Hard-IP*) tel que des processeurs, des multiplicateurs ou de la mémoire par exemple.

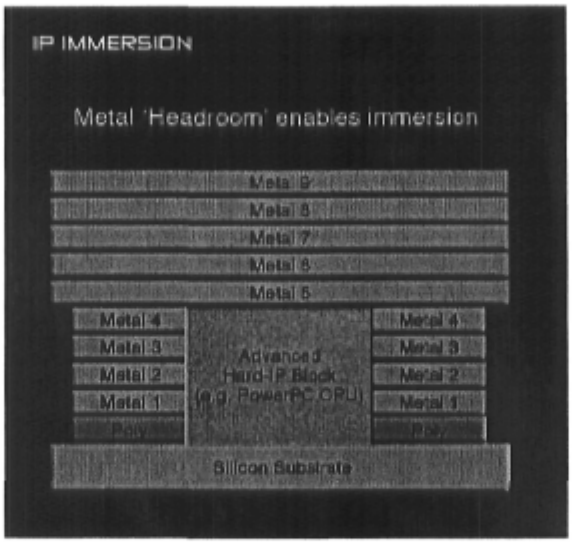

Figure 3.1 : Coupe transversale d'un FPGA *Xilinx* [38].

3.1.2 Ressources logiques

Dérivés des *Complex Programmable Logic Device* (CPLD), les FPGA ont été développés au milieu des années 80. Les FPGA partent du même principe que les CPLD c'est-à-dire des connexions internes programmables, mais pousse le concept plus loin en n'imposant aucune structure de base contrairement au CPLD qui impose une structure « somme de produit ». De plus, les unités de base des FPGA sont plus complexes que celles des CPLD, principalement diverses portes logiques. Un exemple d'unité de base (*Configurable Logic Block*, CLB) de FPGA *Xilinx* est présenté à la figure 3.2. On peut observer qu'une unité de base contient quatre sous éléments nommés tranche agencés en deux colonnes pour la propagation rapide de la retenue lors d'une addition. Ces tranches sont directement reliées à la matrice d'interconnexions pour les connexions longues ainsi qu'à une structure permettant une connexion rapide avec les voisins immédiats de ce CLB. La figure 3.2 (b) illustre la structure interne d'une tranche. Elle contient deux tables de correspondance (*Look-Up Table*, LUT) de 16 bits qui peuvent être utilisées pour implanter une fonction logique booléenne ayant quatre entrées et moins, un registre à décalage ou une mémoire RAM ou ROM. De plus, il y a des ressources logiques, des portes logiques et des multiplexeurs dédiés entre autres, qui permettent de créer des fonctions plus complexes ayant plus d'entrées et qui utilisent plus d'une tranche. Un registre, pouvant être utilisé comme une bascule ou un verrou de différents types, complète chacune des tranches.

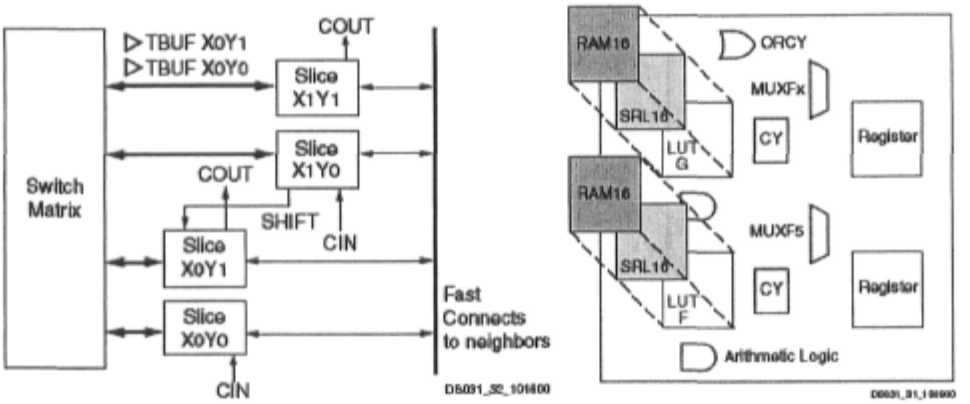

Figure 3.2 (a) *Configurable Logic Block* (CLB) et (b) tranche d'un CLB de la compagnie *Xilinx* [31].

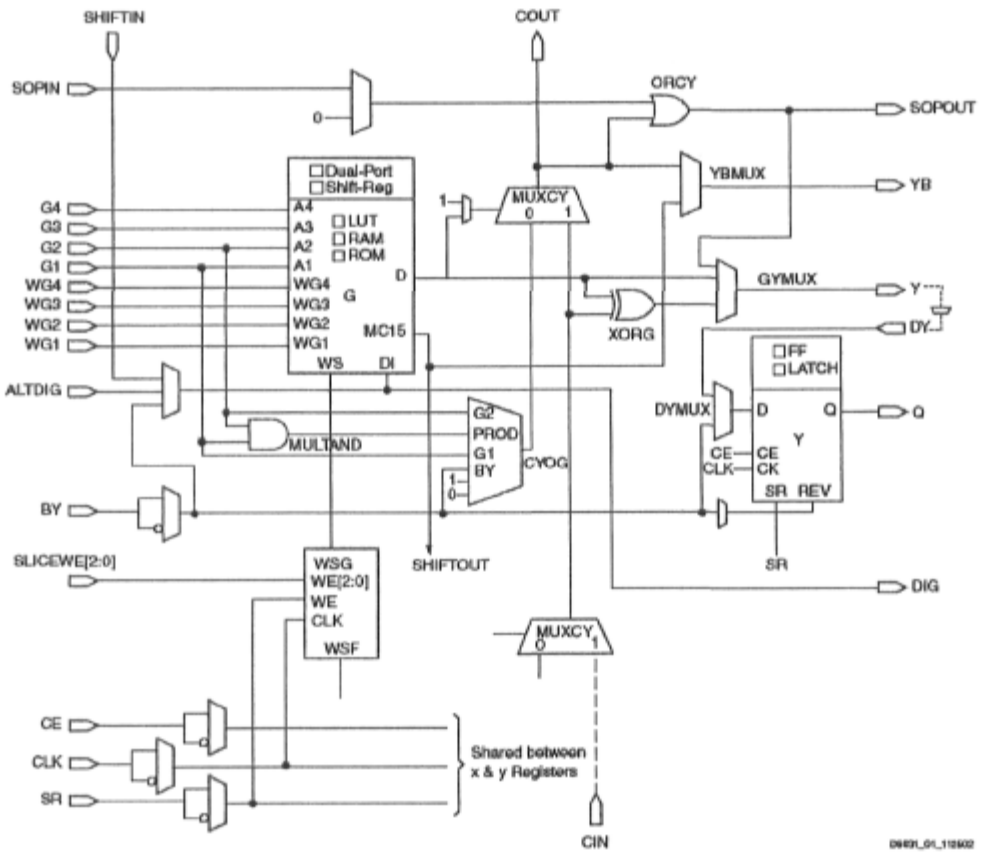

Figure 3.3 Détail de la partie supérieure d'une tranche Xilinx [31].

Plusieurs autres compagnies fabriquent des FPGA et selon les applications visées, la granularité des unités de base variera. Des unités logiques de base plus simples, et donc plus petites, se prêtent mieux à l'optimisation des ressources tandis que des unités de base plus complexes, capables d'implémenter des fonctions plus complexes avec moins de connexions et de routage, rendent l'amélioration des performances temporelles plus aisée.

Xilinx [31] et *Lattice Semiconductor* [33] utilisent des structures à granularité moyenne tandis que *Actel* fabrique des FPGA à granularité plus fine. L'unité logique de base de cette dernière compagnie peut être configurée pour réaliser soit une fonction logique de trois à cinq entrées selon le modèle, un verrou ou une bascule dont le type dépend du modèle de FPGA [35, 37].

Des structures alternatives permettent d'allier ces deux stratégies, mais ce au prix de plus d'espace sur le silicium. Par exemple, la compagnie *Altera* offre avec sa gamme *Stratix II* des unités de base nommées *Adaptative Logic Modules* (ALM) qui permettent différentes configurations internes pour supporter des LUT avec un nombre variable d'entrées [32].

Afin d'améliorer les performances des FPGA pour certaines applications, plusieurs types de ressources dédiées sont embarquées à même la puce de silicium. Généralement ces ressources sont réalisables à partir d'unités logiques de base mais elles en requièrent un très grand nombre et les délais de routage sont alors très élevés. Ainsi les fabricants ont jugé bon d'inclure directement sur la puce de silicium ces éléments. Le type et la quantité d'éléments embarqués dépendent du marché visé. La figure 3.4 en illustre quelques-uns provenant de l'architecture du *Virtex II Pro* de *Xilinx* : les *Input/Output Blocks* (IOB) implémentent les fonctions d'entrées/sorties du FPGA, essentielles pour toutes les applications; les *Digital Clock Managers* (DCM) implémentent toutes les fonctions relatives à l'horloge, que ce soit une multiplication de l'horloge ou la génération d'un signal d'horloge déphasé par exemple; les blocs *SelectRAM* sont des mémoires RAM à double port tandis que le bloc *Multiplier* est un multiplicateur 18 bits par 18 bits dédié, très utile pour les applications de traitement de signal. De plus, sur certains modèles de FPGA, il est même possible d'avoir à même le silicium des processeurs embarqués comme les *PowerPC* (PPC) 405D5 qui se retrouvent sur les FPGA *Virtex II Pro* et *Virtex IV* de *Xilinx*. Les autres compagnies de FPGA utilisent aussi cette stratégie.

3.1.3 Routage

On constate en observant la figure 3.4 que les FPGA *Xilinx* regorgent de ressources d'interconnexions sous forme de matrices d'interconnexions (*Switch Matrix*).

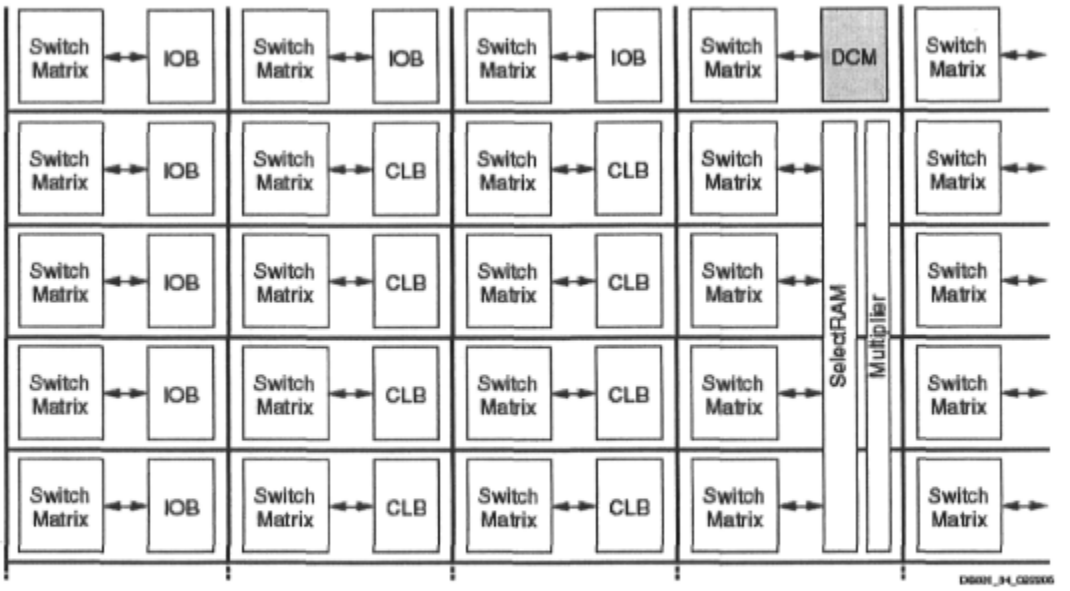

Figure 3.4 Structure interne d'un FPGA *Xilinx* de la famille *Virtex* [31].

Il est intéressant de noter que les ressources de routage sont entrelacées avec les ressources logiques. D'autres compagnies, telle que *Actel*, ont réservé les couches supérieures de leurs puces pour les ressources de routage tandis que les ressources logiques sont situées aux couches inférieures. Les matrices d'interconnexions permettent de connecter les tranches de deux CLB. Les lignes longues (*Long Lines*) s'étendent verticalement et horizontalement de bord en bord du FPGA, permettant une connexion bidirectionnelle entre deux CLB lointains. Les lignes Hex (*Hex Lines*) relient un CLB à son troisième et/ou sixième voisin verticalement et horizontalement. La communication sur ce lien est unidirectionnelle mais le signal peut être lu au point milieu. Similaires aux lignes Hex, les lignes doubles (*Double Lines*) forment un lien entre un CLB et son voisin immédiat et/ou son deuxième voisin. Les connexions directes (*Direct Connections*) permettent de relier un CLB à tous ses voisins immédiats, incluant les voisins en diagonale et finalement les liens *Fast Connects* relient la sortie d'un LUT à l'entrée d'un LUT voisin. Les FPGA *Xilinx* sont munis en plus de ressources de routage dédiées pour l'horloge et pour la propagation de retenue lors d'opérations mathématiques. En ce qui concerne l'horloge, ces ressources dédiées permettent de l'acheminer partout dans la puce avec de très faibles délais et une déformation minimale de la forme d'onde. Pour la propagation de la retenue, toutes les tranches d'une même colonne sont reliées entre elles

par un lien direct afin d'implémenter le plus efficacement possible les fonctions mathématiques de base.

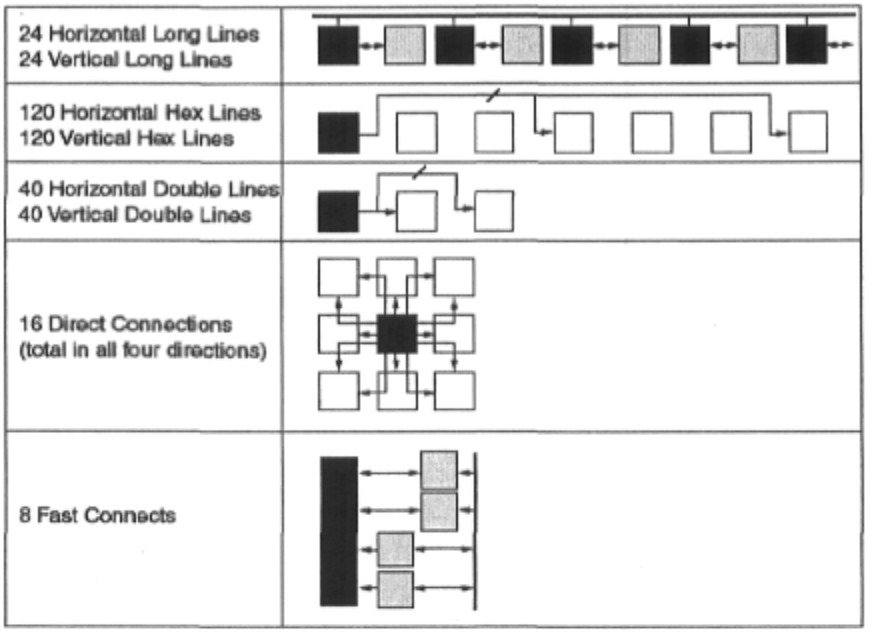

Figure 3.5 Connexions possibles entre deux CLB [31].

Les interconnexions programmables des FPGA *Xilinx* et *Altera* sont basées sur de la mémoire SRAM (*Static Random Access Memory*) qui a comme avantages principaux une bonne rapidité et une capacité infinie de reprogrammation. Une case mémoire est allouée pour chaque connexion et le contenu de cet espace mémoire indique si la connexion à laquelle il se rattache est active (connectée) ou inactive (non connectée). Ce genre de mémoire a la particularité de ne pas nécessiter de rafraîchissement périodique à l'instar de la DRAM (*Dynamic RAM*). Ces deux types de mémoire sont dits volatiles, c'est-à-dire qu'ils conservent les données uniquement s'ils sont alimentés. Le contenu est perdu lorsqu'ils ne sont plus alimentés. Cela implique qu'il est nécessaire de reconfigurer le FPGA à chaque fois que le FPGA est remis sous tension. Pour les designs finaux, le fichier de configuration est généralement contenu dans une mémoire ROM (*Read-Only Memory*) à proximité du FPGA.

Il existe d'autres technologies pour la programmation des FPGA qui offrent des avantages tout aussi intéressants. Par exemple, *Lattice Semiconductor* et *Actel* offrent des FPGA qui enregistrent leur configuration dans de la mémoire *flash* [34, 37], un type de

Electrically Erasable Programmable Read-Only Memory (EEPROM). Puisque la mémoire *flash* est une mémoire non-volatile, le FPGA conserve sa configuration même lorsqu'il n'est pas alimenté. Contrairement aux deux technologies précédentes qui offrent une capacité de programmation infinie, certaines compagnies, *Actel* entre autres, offrent des FPGA qui ne peuvent être programmés qu'une fois. Ces FPGA utilisent une technologie de fusible ou anti-fusible pour stocker leur configuration. Cette capacité très limitée de programmation permet d'immuniser le FPGA aux radiations qui pourraient modifier le contenu d'une mémoire RAM ou ROM, rendant cette technologie intéressante pour les applications spatiales, et rend le piratage de FPGA beaucoup plus difficile, voir impossible [36].

3.2 Technologie SoC

L'augmentation de la densité d'intégration a permis le développement de *System on a Chip* (SoC) qui consiste en un système totalement embarqué sur une puce. Le centre de traitement, la mémoire, les bus de communication et autres unités spécialisées se retrouvent dans la même puce, permettant ainsi une implémentation physique plus compacte. Habituellement, une implémentation SoC est moins coûteuse, moins énergivore et plus fiable qu'un système identique réalisé avec plusieurs puces. Cependant le développement d'un système SoC requiert le développement parallèle des aspects physique et logiciel. Ainsi une certaine expertise dans les deux domaines est nécessaire et si un de ces aspects est négligé, les performances du système en souffriront.

La figure 3.6 illustrent un SoC réalisé dans un FPGA *Virtex II Pro*. On constate rapidement que ce système est très similaire à l'architecture d'un ordinateur personnel : une unité de traitement centrale, des bus de communication, de la mémoire ainsi que plusieurs autres périphériques. Un des points intéressants de cette figure est le bloc *User Logic* dans le coin inférieur gauche. Le concepteur peut réaliser dans ce bloc toutes fonctions désirées afin d'accélérer l'exécution d'une tâche ou tout simplement réduire le fardeau de l'unité de traitement.

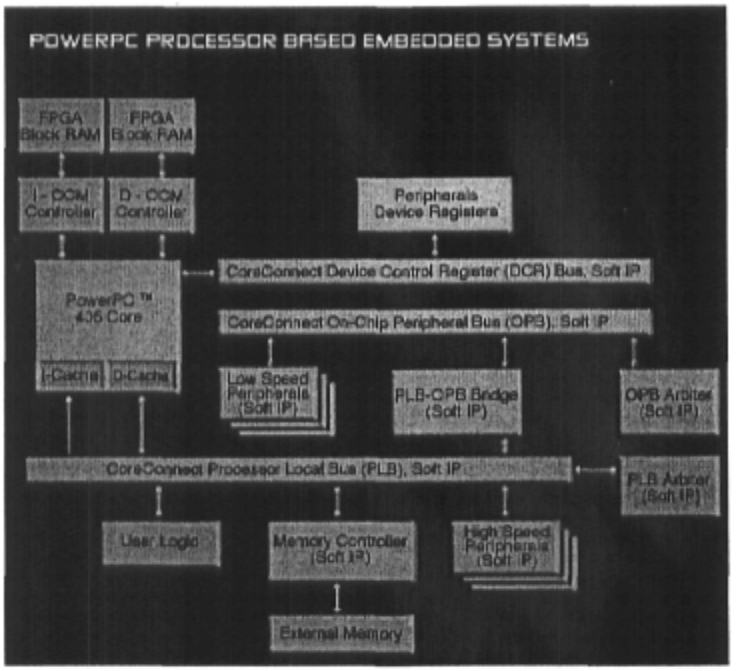

Figure 3.6 : SoC dans un FPGA *Virtex II Pro* de *Xilinx* [42].

3.2.1 Noyaux IP (*Intellectual Property Cores*)

Un autre concept émergeant est le principe des *Intellectual Property* (IP) *cores* ou noyaux IP. À l'instar des langages de programmation, le développement FPGA s'est doté de librairies d'éléments pouvant être utilisés dans plusieurs circuits : une librairie de noyau réalisant les opérations mathématiques de base en virgule flottante ou des noyaux réalisant certaines fonctions mathématiques complexes sont deux exemples de noyaux IP.

Plusieurs compagnies se spécialisent dans le développement de noyaux et de leur distribution. Le coût d'utilisation d'un noyau IP est fonction du type de noyau ainsi que de sa complexité et de ses performances. Les droits d'utilisation de noyaux IP peuvent être astronomiques, plusieurs centaines de milliers de dollars, et peuvent comporter des frais résiduels proportionnels aux ventes. Il existe cependant plusieurs groupes amateurs qui développent et distribuent des noyaux IP gratuitement.

On classifie les noyaux en trois types : les noyaux mous (*soft*), fermes (*firm*) et durs (*hard*). Le premier type consiste en une description de haut niveau réalisant une fonction spécifique. Cette description est indépendante de toute technologie d'implémentation. Ce

type de noyau se prête donc très bien aux modifications et optimisations. Les noyaux IP fermes consistent en une description de niveau d'abstraction moyen, habituellement *Register Transfer Level* (RTL), limitant ainsi les modifications possibles. De plus, ce genre de noyau est généralement optimisé pour une certaine technologie d'implémentation. Finalement le dernier type de noyau, les noyaux durs, consistent en une description de très bas niveau qui est totalement optimisée pour une technologie cible, rendant impossible les modifications.

3.2.2 Flot de conception SoC

Le flot de conception SoC est particulier puisque les aspects matériel et logiciel d'un projet sont traités parallèlement contrairement à la conception usuelle de systèmes comparables. Le développement de ces deux aspects est étroitement lié, ce qui permet une grande optimisation. La figure 3.7 représente le flot de conception SoC.

La branche de gauche représente le développement matériel utilisant un langage de description de matériel. Tous les noyaux IP sont rassemblés pour former le système. Il arrive souvent qu'une ou plusieurs fonctions ne soient pas disponibles sous forme de noyau IP, il faut donc développer des noyaux réalisant ces fonctions. De plus, il est très fréquent que l'interface de certains noyaux IP utilisés ne corresponde pas exactement à l'interface désirée, surtout s'ils proviennent de plusieurs sources différentes. Il faut donc réaliser une « colle logique » permettant d'interfacer tous les noyaux ensemble pour former le système. Il arrive parfois que la réalisation de cette « colle logique » demande des efforts considérables qui rivalisent avec ceux nécessaires au développement maison des noyaux fautifs. Une fois la description du système complétée, il reste à le synthétiser et à réaliser les étapes menant à l'implémentation physique. Ces dernières étapes seront présentées à la section 3.3.3.

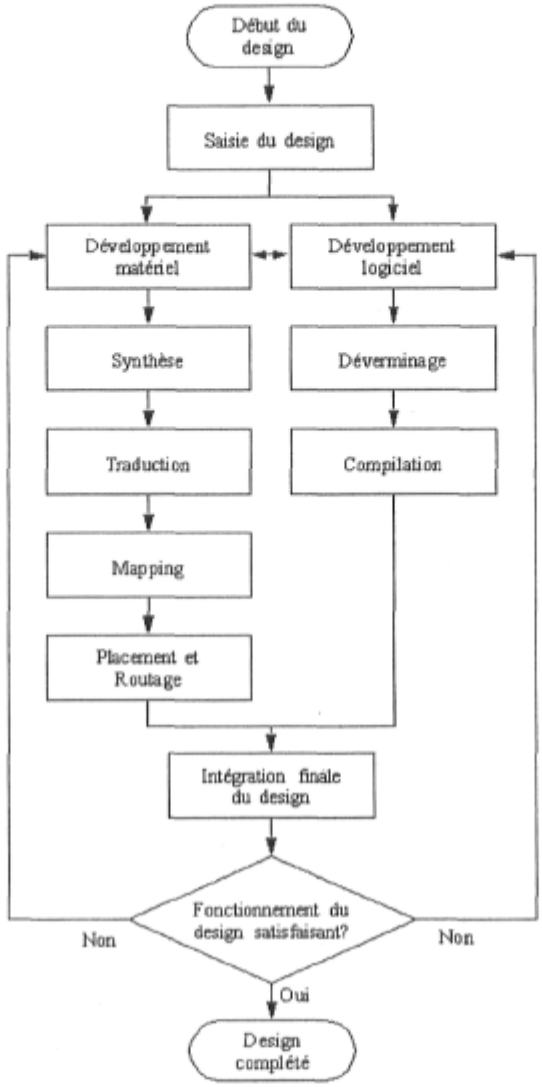

Figure 3.7 Flot de conception SoC simplifié.

La branche de droite représente le développement des programmes nécessaires au fonctionnement de l'unité de traitement contenu dans le SoC, généralement un processeur ou un microcontrôleur. Un langage de programmation conventionnel est utilisé et son niveau dépend de la nature de l'unité de traitement. De nombreux outils de simulation sont disponibles dans les suites de logiciel de développement SoC afin de faciliter la conception logicielle et physique. À noter que ce schéma n'illustre pas les retours aux étapes de développement suite aux diverses simulations effectuées.

3.3 Langage de description de matériel VHDL

Pour la conception sur FPGA, les deux langages de description les plus utilisés sont le Verilog et le VHDL. Tous deux font l'objet de standards IEEE afin d'uniformiser ces langages autant pour les concepteurs que les manufacturiers. Le langage utilisé dans le cadre de ce projet est le VHDL et il sera présenté au cours de cette section.

3.3.1 Historique

Développé à la demande du département de la défense des États-Unis au début des années 80, le VHDL avait comme but premier la documentation des ASIC utilisés dans les projets militaires. Parallèlement, l'utilisation du VHDL uniformisa les langages de description de matériel utilisés par les fournisseurs, facilitant l'intégration de composantes d'origines diverses, et diminua la sensibilité des projets à la technologie cible puisque le VHDL permet une certaine abstraction de cette dernière. Cela implique donc que les designs peuvent être réutilisés malgré un changement dans les technologies d'implémentation.

Les simulateurs de VHDL ont été développés en premier, permettant ainsi de lire un document VHDL et de valider le comportement du circuit décrit. Par la suite les outils de synthèse ont été développés. Ces derniers interprètent un circuit décrit en VHDL et synthétisent l'implémentation physique de ce dernier. Cependant ce n'est pas la totalité du langage VHDL qui peut être synthétisé, il reste encore du travail à faire en ce qui concerne les outils de synthèse afin d'augmenter le sous-ensemble du VHDL qui est synthétisable.

La première version du VHDL a vu le jour en 1987 sous le sigle *IEEE Standard 1076-1987*. La seconde version du standard voit le jour en 1993. Elle rajoute à la première version une syntaxe plus claire et plus détaillée, une plus grande flexibilité dans l'identification des éléments par un jeu de caractères étendu ainsi que plusieurs opérateurs. Plusieurs ajouts ont été créés afin de répondre à plusieurs problèmes ou nécessités de conception comme l'ajout de logique à états multiples (*Multi-Valued Logic*, MVL) dans le standard IEEE 1164, une extension pour l'ensemble synthétisable

(standard IEEE 1076.3) et un ajout pour la temporisation des circuits (standard IEEE 1076.4) entre autres.

3.3.2 Description du VHDL

Le VHDL, qui possède une syntaxe similaire à celle d'Ada, permet la description à trois niveaux d'abstraction : le niveau comportemental, le plus élevé, décrit l'algorithme réalisé par le circuit; le niveau des flots de données décrit comment les données sont traitées et où elles sont stockées et le niveau structural définit exactement la structure du circuit à l'aide d'éléments de base tels des portes logiques ou même des transistors [51].

Ce langage de description est très typé, c'est-à-dire qu'il impose beaucoup de restrictions sur comment les variables et signaux d'un certain type interagissent avec des variables et signaux d'autres types. Plusieurs types de base sont définis pour traiter les circuits logiques (*std_logic*, *std_logic_vector*, *bit*, *bit_vector*, etc.) et d'autres ont été définis pour traiter les opérations arithmétiques (*unsigned*, *signed*, *real*, *int*, etc.).

Tout circuit décrit en VHDL est constitué d'une entité et d'une architecture. L'entité définit l'interface du circuit avec l'environnement extérieur. On y définit les ports d'entrées et de sorties ainsi que les variables de paramétrisation. Pour chaque entité, il faut définir au moins une architecture. Cette dernière explicite les fonctionnalités du circuit en fonction des entrées et des sorties définies dans l'entité. Tel que mentionné précédemment, le circuit peut être défini au niveau comportemental, au niveau des flots de données ou structurellement. Il est possible d'allier plusieurs styles dans une description. Il est aussi possible de définir plusieurs architectures pour une entité mais il faut alors spécifier quelle architecture il faut utiliser pour chaque instanciation de l'entité. Pour décrire les fonctionnalités d'un circuit, le VHDL offre une syntaxe parallèle, séquentielle ainsi que des moyens pour décrire les interconnexions (*netlist*). Puisque le VHDL décrit une implémentation physique, une grande partie du langage est dédié à la description de phénomènes concurrents. La syntaxe séquentielle est similaire aux langages de programmation conventionnels mais contrairement à ces derniers, le VHDL traite des signaux. Il est donc important de définir les connexions relatives à ces signaux

et leur « direction ». À l'intérieure d'une architecture, il est possible d'instancier des composantes pour réaliser la description du circuit. Les composantes sont des entités qui ont été préalablement définies, qui sont accessibles via des librairies, et qui sont connectées dans l'architecture présentement définie afin de réaliser une certaine fonction, à l'instar de composantes discrètes dans un circuit physique.

Le VHDL comporte plusieurs autres fonctionnalités qui ne sont pas synthétisables mais qui sont bien utiles pour la validation de circuit et la gestion de projets d'envergure. Par exemple, il est possible de définir une entité sans port d'entrées/sorties qui sert uniquement comme banc d'essai pour un circuit. Dans l'architecture de cette entité, le circuit à tester est instancié et une séquence de stimuli est définie afin de valider son fonctionnement. Plusieurs outils ont été développés pour fournir une interface graphique pour les bancs d'essai ou pour directement simuler le circuit sans la création d'un banc d'essai.

Présentement la tendance pour les outils VHDL est de fournir une interface graphique afin de réaliser le circuit en plaçant des blocs représentant certaines fonctionnalités ou structures disponibles sur le FPGA cible, à la *Simulink* de *MATLAB*. Une fois le circuit complété, le logiciel génère les fichiers VHDL nécessaires.

3.3.3 Flot de conception FPGA

Pour passer d'un circuit décrit en VHDL à l'implémentation sur un FPGA il faut effectuer la traduction (*translation*), le mappage et finalement le placement et le routage. La première étape, la traduction, consiste à aplanir la hiérarchie du circuit, c'est-à-dire remplacer toutes les entités par leur architecture et construire une liste d'interconnexion (*netlist*) globale. Par la suite, l'étape de mappage remplace les fonctions logiques par des primitives de la technologie cible qui implémentent physiquement ces fonctions. Finalement, une fois que le circuit est représenté en éléments de base de la technologie cible, les outils automatisés déterminent exactement quels éléments sont utilisés et comment ils sont connectés afin de réaliser le circuit. Cette dernière étape est généralement itérative et elle cherche à optimiser le placement et le routage selon des

contraintes temporelles ou des contraintes de ressource au choix du concepteur. Le placement et le routage peuvent être très longs à effectuer si le circuit utilise une très grande partie du FPGA et/ou s'il y a des contraintes temporelles sévères. Toutes ces opérations sont généralement automatisées mais il est généralement possible de les effectuer manuellement afin d'optimiser le circuit. Une fois toutes ces étapes complétées, un fichier de configuration est créé et peut être ensuite téléchargé dans le FPGA.

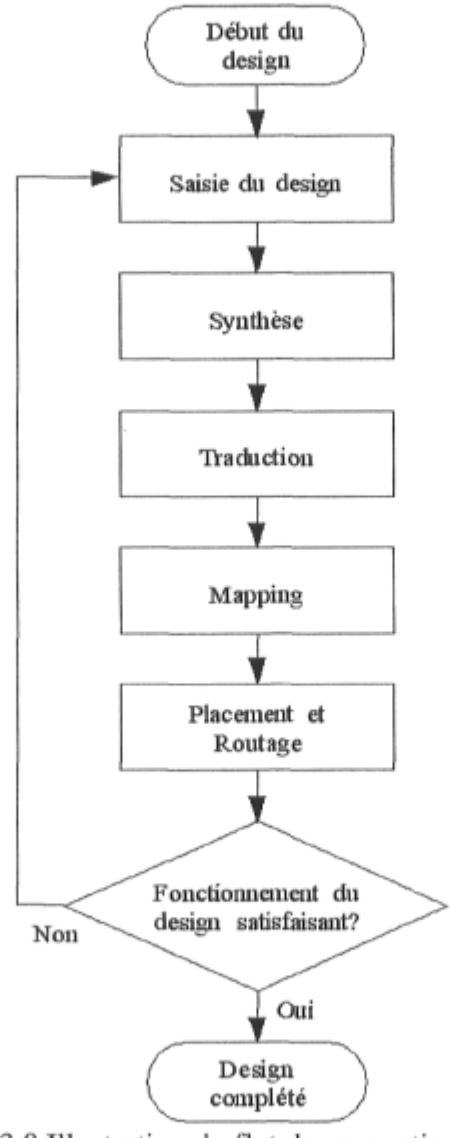

Figure 3.8 Illustration du flot de conception FPGA.

3.4 Plateforme de développement utilisée

Dans le cadre de ce projet, une carte de développement *Amirix AP130* est utilisée. Cette carte, dont le contenu est illustré à la figure 3.9, est dotée d'un FPGA *Xilinx Virtex II Pro VP30* ainsi que de plusieurs périphériques de communication.

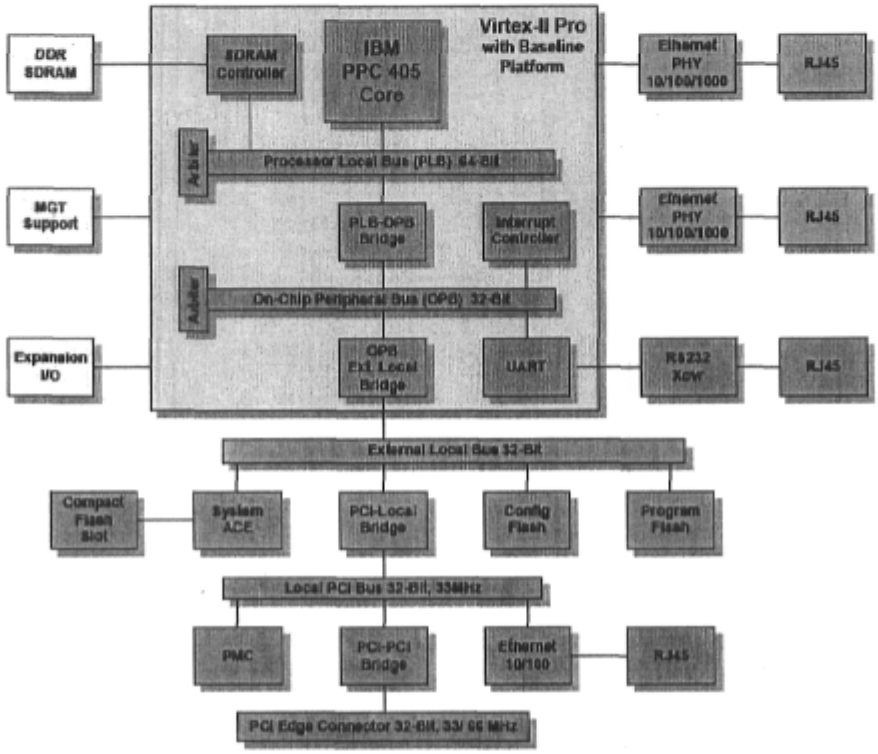

Figure 3.9 Schéma de la carte de développement *Amirix AP130* ainsi que du SoC dans le FPGA *Xilinx* [43].

3.4.1 *Virtex II Pro VP30*

Le *Virtex II Pro VP30* fait partie de la deuxième génération de FPGA plateforme de *Xilinx*. Ces FPGA sont conçus pour le développement de SoC. Réalisé avec une technologie CMOS 130 nm opérant à 1.5 V et ayant 9 couches de cuivre [44], le *Virtex II Pro VP30* offrent une quantité de ressources respectable comme on peut le constater en observant le tableau 3.1

Tableau 3.1 : Caractéristiques du *Xilinx Virtex II Pro VP30 -6 FF896* [44].

Blocs *RocketIO*	8
Blocs *PowerPC*	2 (f_{max} = 350 MHz)
Tranches	13696
Multiplicateurs 18x18 bits	136
Bloc *SelectRAM+*	136 (2448 kb)
Digital Clock Managers	8
Pattes d'entrées/sorties	556

Les blocs *RocketIO* servent à implémenter des liens de communication très rapide avec un taux binaire pouvant aller jusqu'à 3.125 Gb/s. Un bloc *PowerPC* (PPC) est une implantation à 130 nm d'un *IBM PowerPC 405D4*, qui porte le numéro *405D5*, pouvant être cadencé jusqu'à 350 MHz sur les versions -6. Ce processeur de 32 bits ne dispose que d'une unité d'arithmétique en nombre entier et utilise un jeu d'instructions réduit (*Reduced Instruction Set Computer*, RISC). Toutes les opérations en virgule flottante sont émulées par le compilateur. Les tranches correspondent exactement à celle présentée précédemment. Les multiplicateurs 18x18 sont des ressources dédiées qui sont en mesure d'effectuer la multiplication de deux opérandes d'au plus 18 bits à un rythme de 300 MHz. Évidemment plusieurs multiplicateurs peuvent être agencés afin d'accommoder des opérandes de taille supérieure. Un bloc *SelectRAM+* correspond à 18 kb de mémoire RAM à double port. Ces blocs sont répartis dans tout le FPGA. Les *Digital Clock Managers* permettent un routage efficace de l'horloge, limitant les délais et la déformation de ce signal, ainsi que la génération de signaux d'horloge dérivés de l'horloge externe. Ils peuvent effectuer une multiplication ou une division de l'horloge externe ou ils peuvent générer un signal d'horloge avec une phase différente. Finalement, le nombre de pattes d'entrées/sorties représente le nombre de pattes disponibles pour l'implémentation du circuit.

3.4.2 Logiciels de développement

Pour le développement du noyau de simulation temps réel dans le cadre de ce projet, la suite de logiciel *Integrated Software Environment* (ISE) *Foundation 7.1i* a été utilisée. Cette suite contient tous les logiciels nécessaires au développement et à la validation de circuits FPGA. Pour le développement du SoC ainsi que la programmation du PPC, la

suite *Xilinx Platform Studio* (XPS), qui fait partie du *Embedded Development Kit* (EDK) 6.3, a été utilisée. Le diagramme du flot de conception SoC est repris ici pour illustrer le champ d'action de ces logiciels. Chacun de ces logiciels possèdent aussi divers outils afin de simuler et de valider le design en cours.

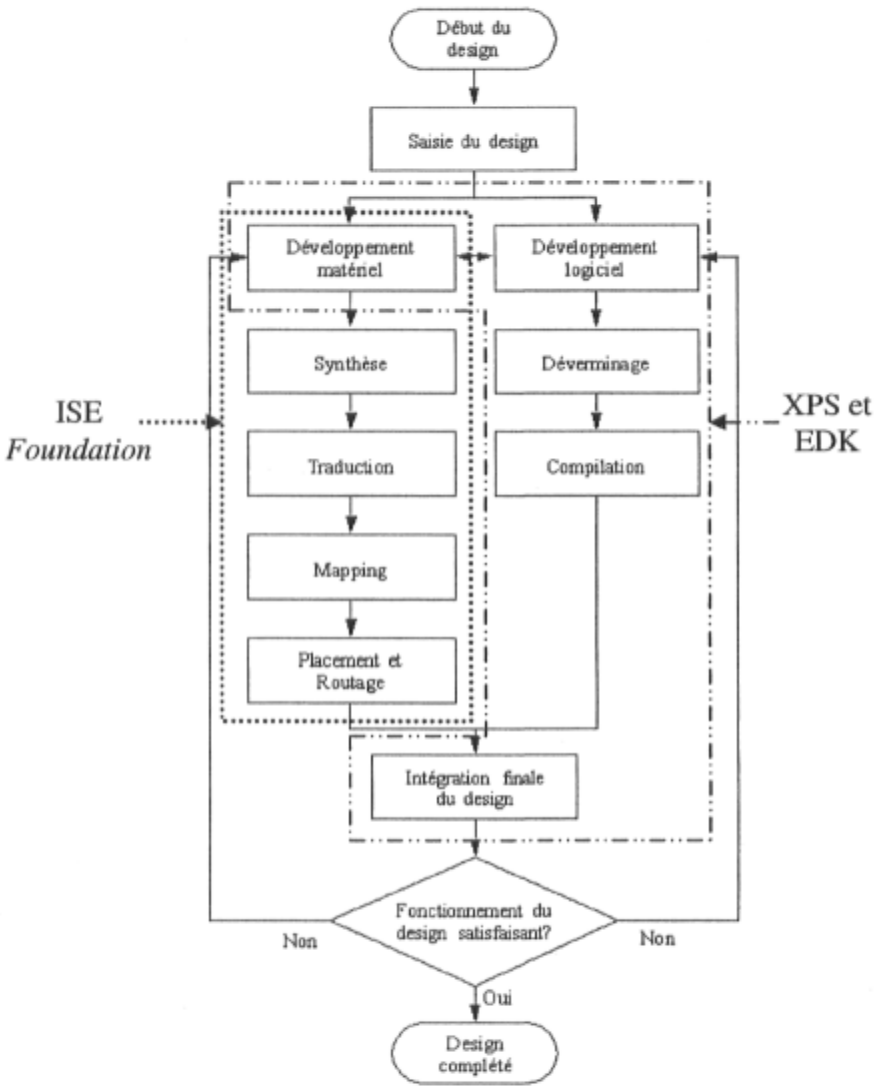

Figure 3.10 Logiciels utilisés au cours de ce projet et leurs fonctions.

100% COURS
MCours.com
Cours et mémoires gratuits
mcourscom@gmail.com