République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane MIRA de Bejaia

Faculté des Sciences Exactes

Département Informatique

Mémoire de fin de cycle

En vue de l'obtention du diplôme de Master en informatique

Option **:** ***Réseaux et Systèmes Distribués***

Présenté par : BOUZEROURA Rachida

Thème

Conception et intégration des systèmes à processeur complet SoC avec VHDL

Promoteurs:

M^me^ ALOUI .S

M^er^ BOUZEROURA. A/G

Devant le jury:

Président: M^er^ ABBACHE. B

Examinatrice: M^me^ METIDJI. R

Examinatrice: M^me^ BEN KHELLAT. Z

2011/2012

Remerciements

Tout d'abord, j'aimerai remercier ***Dieu*** le tout puissant, de nous avoir donné le courage, la patience et la volonté afin de réaliser ce travail.

Je suis reconnaissante à M^me^ ***ALOUI Soraya*** d'avoir accepter l'encadrement de ce mémoire, pour tous ses conseils avisés, ces encouragements, jusqu'aux dernières minutes et la très grande liberté qu'elle m'a accordée dans l'orientation de mes travaux de recherches, merci pour la confiance, la disponibilité et les nombreuses relectures , vraiment avec vous j'ai eu un réel plaisir à travailler.

J'exprime tout particulièrement ma reconnaissance à M^me^ ***MEZAH Samia*** pour la disponibilité et l'aide qu'elle m'a offert, ainsi que M^r^ ***BOUZEROURA A/ Ghani***, pour le soutien qu'il a apporté dans le Co-encadrement du présent mémoire.

Je tiens à remercier très chaleureusement les membres de jury, qui ont acceptés de juger ce travail.

Enfin, je remercie toutes les personnes qui me sont chères et qui tiennent une grande place dans mon cœur, en commençant par ma famille : mes parents d'abord: ma mère ***Gagou*** pour son soutien pour toujours (que dieu la garde parmi ses bien aimés nchallah), mon père qui m'a transmis son goût pour la science, ***Nora*** pour tous ses sacrifices ainsi que mes sœurs, mes frères, leurs petites familles et tout le reste de la famille pour leurs encouragements.

Je voudrais aussi remercier mes amies, et amis, ceux qui m'entourent et qui ont une place importante dans ma vie. Et enfin toute la promotion Master 2 RSYD 2012.

A maman qui nous a quittés

Sans dire au revoir....

A mon père que dieu le guérisse

Aux anges : Sarah, Rayan et Nihad.

Table des matières

Chapitre IV : Simulation des résultats sous XILINX ISE

Liste des figures

LISTE DES ABRÉVIATIONS

ABEL: Advanced Boolean Expression Language.

AHDL: Altera **H**ardware **D**escription **L**angage.

ASIC: **A**pplication **S**pecific **I**ntegrated **C**ircuit.

CAO: **C**onception **A**ssistée par **O**rdinateur.

CPLD : Complexe **P**rogrammable **L**ogique **D**evice.

DOD : **D**épartement de la **D**éfense des États-Unis.

DSP: Digital Signal Processor.

EDCA: **E**lectronic and **E**lectrical **C**omputer-**A**ided **D**esign

FPGA: **F**ieled **P**rogrammable **G**ate **A**rray.

IEEE: Institut of **E**lectrical and **E**lectronics **E**ngineers.

IP: **I**ntellectual **P**roperties.

NoC: **N**etwork **o**n **C**hip.

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

SoPC: **S**ystem **o**n **P**rogrammable **C**hip.

VHDL: **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**angage

XML: Extensible **M**arkup **L**angage**.**

Introduction générale

Les avancées technologiques de ces dernières années continues à stimuler l'évolution fulgurante des circuits logiques programmables. Nous disposons actuellement de circuits avec plusieurs millions de portes logiques. Ces évolutions permettent d'offrir des circuits logiques de plus en plus performants.

Le domaine d'utilisation de ces circuits ne se limite plus à des petites applications périphériques mais s'élargi jusqu'à l'intégration de système à processeur complet (SoPC) « System on Programmable Chip » (système sur puce reprogrammable).

Pour répondre aux différents besoins notamment la conception des systèmes de commande, le recours aux outils de Conception Assistées par Ordinateur (CAO) est plus qu'indispensable. L'avantage des méthodes de CAO est de faire une conception matérielle et logicielle simultanément afin de réduire le temps de développement et d'augmenter la fiabilité par le test des prototypes virtuels avant la réalisation sur un circuit intégré qui sera le lieu d'implantation de la solution finale **[53].**

La complexité des systèmes à réaliser nécessite l'utilisation de nouvelles méthodes de conception. L'association de ces méthodes, avec un langage de description de haut niveau, apporte la souplesse nécessaire à une conception rapide, efficace, fiable et évolutive. Le langage VHDL (VHSIC Hardware Description Language) fourni un très bon moyen de description associé à une méthodologie adaptée, il permet de réaliser les projets les plus ambitieux.

Les outils de CAO servent à passer directement d'une description fonctionnelle en VHDL à un schéma en porte logique. Ces outils ont révolutionnés la conception des circuits numériques tels que les ASIC ou FPGA **[53].**

Ce travail a pour objectif de découvrir le langage VHDL et son utilité en présentant deux exemples d'application dans deux domaines différents : l'électronique et l'informatique, pour atteindre cet objectif nous avons organisé notre travail comme suit:

- ✓ Le premier chapitre décrit brièvement l'état de l'art des langages de programmation, comment et pourquoi on a pensé un nouveau langage qui est le VHDL ;
- ✓ Le second chapitre est consacré à la description du langage VHDL par la définition des structures de ce langage ainsi une comparaison aux autres langages de programmation existants ;

- ✓ Le troisième chapitre concerne la présentation des deux exemples d'application et leurs implémentations VHDL ;
- ✓ Dans le dernier chapitre on valide nos solutions par une simulation VHDL des résultats qui seront présentés et commentés sous XILINX ISE ;
- ✓ Enfin, nous terminerons par une conclusion et quelques perspectives.

Introduction

Les premiers langages de programmation de programmation sur ordinateur datent des années 1950**[55]**, avec le développement des ressources informatiques, les langages informatiques sont omniprésents, qu'on le sache ou pas. Il nous a semblé utile de fournir un peu de culture et de recul pour envisager les 50 prochaines années **[59]**.

Dans ce chapitre on mettra l'accent sur les langages de description de matériel, sur l'évolution obligatoire de ces langages liée à la technologie ainsi que la situation complexe présente aujourd'hui due notamment à l'industrie.

I.1 Art des langages de description matériel HDL

Les langages de programmation généraux ont connu une évolution laborieuse et étrange depuis 1946. Les principales catégories de langages sont les langages fonctionnels et procéduraux (impératifs), et les langages logiques **[10]**, **[24], [46]**, **[52]**. Un langage est dit fonctionnel, au sens mathématique du mot fonction, si chaque opération est indépendante du contexte, et si le résultat d'une fonction dépend exclusivement de ses arguments : *Haskell, Lisp*, sont des langages fonctionnels de genre différents destinés à l'intelligence artificielle, *Prolog* est un langage logique. Pratiquement tous les langages, fonctionnels ou impératifs sont maintenant orientés objets et utilisent des classes décrivant des objets réels ou purement informatiques **[55]**.

On distingue quatre générations de langages de programmation plus au moins deux nouvelles tendances :

1) Langages machine ;
2) Langages symboliques et Autocode ;
3) Langages indépendants du matériel, comme *Basic, C, Cobol, java*...
4) Langages conçus pour décrire le problème, comme Simula et autres langages à objets.
5) Les langages à programmation logique prétendent représenter la cinquième génération, mais leur utilisation est marginale. La cinquième génération pourrait être celle des langages Internet, donc fonctionnant sur toute machine et compilés en code virtuel (intermédiaire).
6) Les langages "*Markup*" inspirés de *XML* sont la dernière tendance, ils intègrent le code et les données sous une forme extensible, et qui fonctionnent sur le web.

I.2 L'apport de l'informatique dans l'électronique

Le langage machine est la suite de bits qui est interprétée par le processeur d'un ordinateur exécutant un programme informatique **[54]**. Il est composé d'instructions et de données à traiter codées en binaire. Pour que le langage machine soit représenté sous une forme lisible par un humain on utilise un langage de bas niveau qui est le langage assembleur **[41]**, **[55]**.
Chaque processeur possède son propre langage machine, dont un code machine qui ne peut s'exécuter que sur la machine pour laquelle il a été préparé, donc un langage spécifique à chaque processeur.
Un processeur construit en un seul circuit intégré est un microprocesseur **[52]**, **[55]** Les circuits intégrés ont permis la miniaturisation et la standardisation des processeurs qui ont conduit à leur diffusion dans la vie moderne bien au-delà des usages des machines programmables dédiées.

I.2.1 Technologies de fabrication des circuits intégrés

Un système électronique fait appel à plusieurs fonctions de l'électronique et comporte de façon générale des circuits analogiques et/ou des circuits numériques. Jusqu'au milieu des années 60, les fonctions de l'électronique étaient réalisées à l'aide de composants discrets exemple des tubes à vide, puis après l'invention du transistor en 1948 qui a été la première étape de la révolution apportée par la micro-électronique .La deuxième étape a été l'invention du circuit intégré **[05]**.

Un circuit intégré est un circuit électronique réalisant une fonction et comportant plusieurs transistors fabriqués de façon collective. La fabrication du circuit intégré est précédée d'une phase de conception durant laquelle s'élaborent les plans du circuit sur la base de ses spécifications fonctionnelles. Le développement de méthodes et d'outils de CAO a permis d'accélérer le cycle de conception et de rendre les circuits plus performants **[42], [50]**.

Ces outils utilisent en principe un langage de programmation spécial: langage de description de matériel (HDL pour *hardware description language* en anglais) qui est un langage informatique permettant la description d'un circuit électronique. Celui-ci peut décrire les fonctions réalisées par le circuit (description comportementale) ou les portes logiques utilisées (AND, XOR…) par le circuit (description structurelle).Il est possible d'observer le fonctionnement d'un circuit électronique modélisé dans un langage de description grâce à la simulation.

A la différence d'un langage de programmations logicielles, la syntaxe et la sémantique d'un HDL incluent des notations explicites pour exprimer le temps et le parallélisme qui sont les attributs principaux du matériel **[55]**.

La figure suivante illustre l'intervention de la programmation informatique au niveau du domaine électronique.

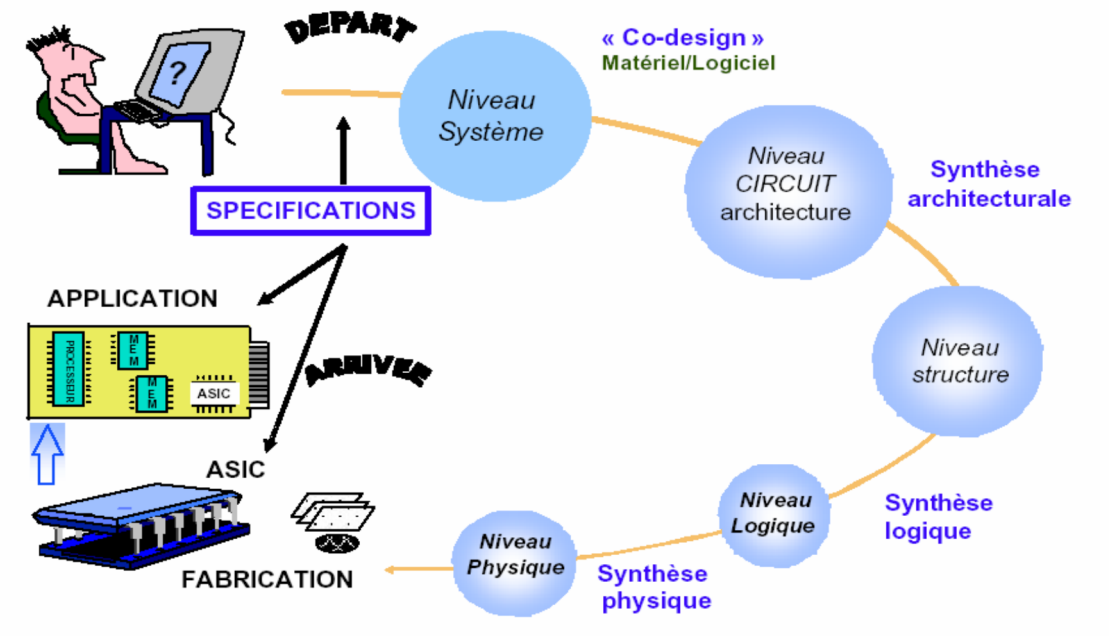


Figure I.01. Conception d'un circuit intégré à l'aide des outils de CAO.

La réalisation d'un circuit intégré est faite à bases des outils de CAO suivant plusieurs étapes. Commençant par une description comportementale (niveau système) pour spécifier le fonctionnement du système. En suite une description des composants qui forment sa structure (niveau structure, logique) et finissant par une implémentation matérielle du système.

I.3 Historique du HDL [53], [55]

I.3.1 **Dessin au micron :** première époque

La conception de circuits intégrés était un métier manuel. On dessinait les composants (transistors) à la main, sur un papier spécial (Mylar) avec des crayons de couleur. C'est ce qu'on appelle le *dessin au micron*.

Une telle technique limitait naturellement la complexité des dispositifs conçus. La complexité était également limitée par les performances de la technologie disponible chez les fondeurs. A cette

époque le concepteur manipulait des objets élémentaires qui étaient des surfaces rectangulaires de couleur (un transistor était constitué de l'assemblage d'une dizaine de ces rectangles).

Exemple : 25 rectangles pour l'inverseur CMOS ci-dessous.

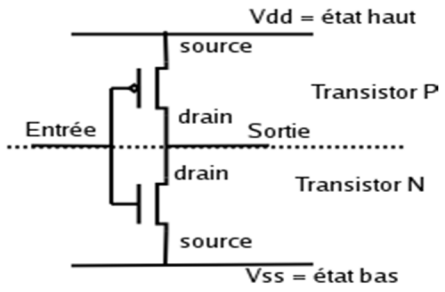


Figure I.02. Inverseur CMOS.

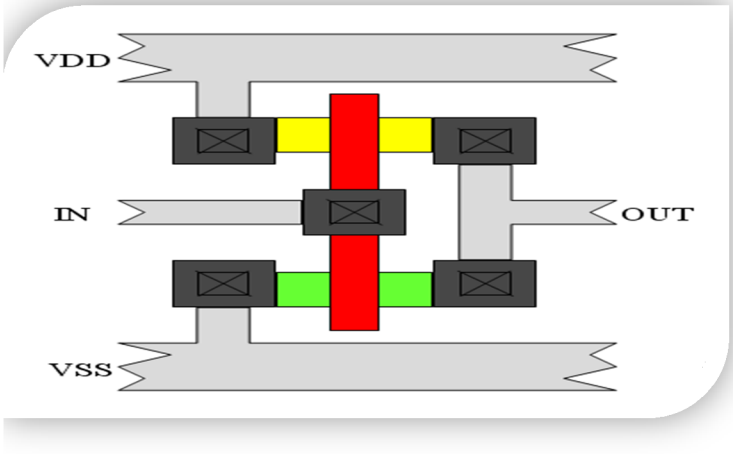


Figure I.03. Dessin au micron d'un inverseur CMOS.

I.3.2 Les langages de description: deuxième époque

Lorsque la technologie a évolué suffisamment pour offrir des ordinateurs puissants et des technologies permettant d'intégrer un grand nombre de transistors sur la même puce il a fallu envisager de nouvelles méthodes de travail. Les langages de description de matériel (HDL) ont ainsi fait leur apparition. Ils avaient pour but de modéliser**,** donc de simuler, mais aussi de concevoir. Des outils informatiques (placeurs-routeurs) ont fait leur apparition, permettant de traduire automatiquement une description textuelle en dessins de transistors (dessins au micron).

Les principes de cette évolution-révolution sont assez simples :

- Les langages sont structurels**,** c'est à dire qu'ils décrivent la structure de l'ensemble par assemblage de composants élémentaires.
- Il existe des bibliothèques de composants élémentaires. Ces bibliothèques contiennent, pour chaque composant élémentaire, un dessin au micron, destiné au placement-routage, et une représentation de la fonction réalisée, destinée au simulateur.

- Les outils informatiques (simulateurs, placeurs-routeurs) utilisent la description textuelle structurelle de l'ensemble et le contenu des bibliothèques.

Le concepteur aborde les problèmes à un niveau d'abstraction plus élevé. Il manipule des objets élémentaires de l'ordre de la dizaine de transistors : les portes logiques.

Un exemple de description textuelle est présenté ci-dessous :

```
bloc BK(IO, I1, I2 : in;
        SO, S1 : out)

  noeud N;

  {
  inv(IO, N);
  nand(I1, I2, S1);
  and(N, S1, SO);
  }
```

Figure I.04 Exemple de description textuelle.

I.3.3 Les schémas: troisième époque

Une pseudo-révolution modifie encore le métier de concepteur de circuits intégrés : l'apparition des interfaces graphiques et donc des éditeurs de schémas. La description textuelle est remplacée par une description schématique :

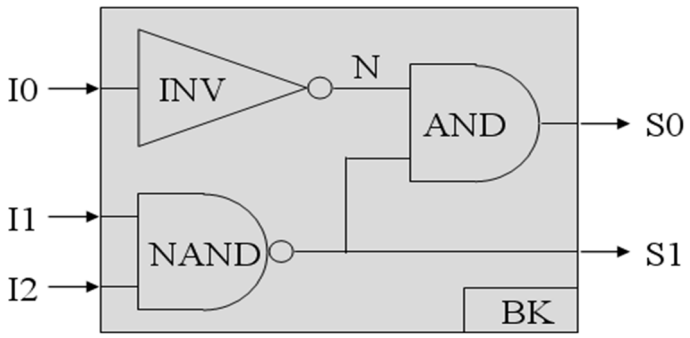


Figure I.05 Description schématique du même circuit.

La seule réelle modification est d'ordre ergonomique : il est en général beaucoup plus facile de lire et de comprendre un schéma qu'une description textuelle.

Les deux exemples ci-dessus montrent l'équivalence entre les deux types de représentation. On imagine comment passer de la forme schématique à la forme textuelle. On comprend aisément pourquoi les informations supplémentaires contenues dans la première (symboles des portes, position relative et orientation des portes, taille et couleur des objets, polices de caractère utilisées, etc.) ne sont utiles qu'au seul concepteur humain. Les outils informatiques n'ont pas l'usage.

Certaines propriétés intéressantes des langages textuels de description de matériel méritent d'être notées :

- Une description textuelle est plus facilement transportable d'un outil de CAO à l'autre, d'un concepteur à l'autre, d'une entreprise à l'autre. La portabilité est accrue.
- Il est beaucoup plus facile de modifier une description textuelle qu'un schéma. La maintenabilité est plus grande.
- On peut aisément archiver un module sous la forme de sa description textuelle afin de le réutiliser dans un autre projet (éventuellement après modification). C'est plus difficile avec d'autres représentations pour lesquelles le coût de mise à jour est plus élevé.

I.3.4 L'abstraction fonctionnelle : quatrième époque

Les langages fonctionnels (ou comportementaux) de description de matériel, grâce aux nouvelles possibilités de description à un niveau d'abstraction plus élevé, ont répondu à des besoins fondamentaux des concepteurs de circuits intégrés :

- La réduction des temps de conception.
- L'accélération des simulations qui devenaient prohibitives avec l'accroissement de complexité des dispositifs.
- La normalisation des échanges : le monde de l'électronique a souffert pendant longtemps de la multiplicité des langages, des formats, des outils de CAO utilisés. L'émergence des langages fonctionnels de description de matériel a été l'occasion de remédier à ce problème.
- L'anticipation : grâce aux modèles HDL il est possible de concevoir un système alors que ses composants ne sont pas encore disponibles.
- La fiabilité : les langages HDL sont conçus pour limiter en principe les risques d'erreur.
- La portabilité : les langages normalisés sont très largement portables.
- La maintenabilité et la réutilisabilité : les modifications et adaptations sont rendues plus simples donc moins risquées et moins coûteuses.

La complexité des circuits conçus augmente toujours. On conçoit aujourd'hui des circuits composés de l'assemblage de plusieurs millions de portes. On a donc tiré encore une fois parti de l'augmentation de puissance des ordinateurs et des progrès réalisés en informatique pour imaginer des outils logiciels encore plus puissants : les *synthétiseurs logiques* **[55].**

Pour travailler à un niveau d'abstraction encore plus élevé, de manipuler des objets élémentaires encore plus grands, il faut permettre au concepteur de décrire le "quoi" au lieu du "comment". Il doit pouvoir s'affranchir de la connaissance des primitives disponibles. Le dispositif à modéliser ou à concevoir sera désormais représenté par sa fonction et non plus par sa structure. C'est le synthétiseur logique qui déterminera la structure automatiquement à partir de la fonction. Bien sûr, les possibilités de description structurelle existent toujours mais on leur associe d'autres possibilités afin d'augmenter l'efficacité de l'ensemble **[17]**, **[55]**.

```
bloc BK(I0, I1, I2 : in;
        S0, S1 : out)

  {
  S0 = non(I0 ou (I1 et I2));
  S1 = non(I1) ou non(I2);
  }
```

Figure I.06 Description fonctionnelle (comportementale) du même circuit.

Les langages fonctionnels (on dit aussi comportementaux) de description de matériel possèdent des avantages certains sur les langages structurels en terme de portabilité, maintenabilité et versatilité **[55]**:

- Une description textuelle fonctionnelle, donc ne faisant pas référence à une quelconque bibliothèque de composants, est plus facilement transportable d'un outil de CAO à l'autre, d'un concepteur à l'autre, d'une entreprise à l'autre. La portabilité est accrue.
- Du fait de sa plus grande compacité et de sa meilleure lisibilité il est plus facile de modifier une description textuelle de fonctions qu'un texte décrivant la structure. La maintenabilité est plus grande.
- Les coûts de mise à jour ou d'adaptation à de nouveaux besoins sont réduits. La réutilisabilité d'un module archivé est donc plus grande.

I.4 Définition complète de HDL

HDL ou langage de description de matériel est un langage de programmation pour la description formelle de circuits électroniques. Il peut décrire l'opération du circuit, sa conception et son organisation, et essais pour vérifier son opération au moyen de simulation **[55]**.

HDLs sont des expressions basées par texte standard de la structure spatiale et temporelle et comportement des systèmes électroniques. Contrairement à un l langage de programmation, La syntaxe de HDL et la sémantique incluent les notations explicites pour exprimer le temps et la simultanéité, qui sont les attributs primaires du matériel **[39], [38]**.

HDLs sont employés pour écrire des caractéristiques exécutables d'un certain morceau de matériel. Un programme de simulation, conçu pour mettre en application la sémantique fondamentale des rapports de langue, couplée à simuler le progrès du temps, procure au concepteur de matériel de modeler un morceau de matériel avant qu'il soit créé physiquement **[58]**.

Concevoir un système dans HDL prend généralement beaucoup plus du temps qu'écrivant un programme de logiciel pour faire la même chose. En conséquence, il y a eu beaucoup de travail effectué sur la conversion automatique du code de C dans HDL, mais ceci n'a pas atteint un à niveau élevé du succès commercial **[55]**.

I.5 Exemples et usages des langages de description de matériel

Actuellement les langages de description sont purement numérique et de différents degrés d'abstraction :

- Advanced Boolean Expression Language (ABEL) orienté bas niveau ;
- AHDL (Altera HDL) langage propriétaire essentiellement structurel proche d'ABEL;
- SystemC utilisant le C++ et qui permet de modéliser les interactions logiciel/matériel.
- Les langages mixtes, qui sont souvent des extensions des précédents. Ils permettent la modélisation des systèmes à l'aide d'équations différentielles.

- Verilog qui mélange la description structurelle et algorithmique ;
- VHDL légèrement plus abstrait que Verilog qui est inspiré du langage ADA ;
- Verilog-AMS : est un dérivé de Verilog il définit le comportement des systèmes analogique et de signal-mixte (Analog and Mixed-Signal : AMS) ;
- VHDL-AMS Il comprend des extensions analogiques et des signaux mixtes;
- Modelica langage de modélisation généraliste orienté objet.

Ainsi que les deux les plus employés couramment et les variétés bien-soutenues de HDL utilisées dans l'industrie:

Il est important de comprendre la nature des différentes utilisations que l'on peut faire des langages fonctionnels de description de matériel. Les plus importantes sont :

I.5.1 Simulation

En tout premier lieu vient la modélisation**.** Elle n'a pas pour fonction de décrire la structure de l'objet mais son comportement. Celui qui écrit le modèle peut recourir à la description structurelle pour se simplifier la tâche en découpant le problème en sous-problèmes plus simples à modéliser, mais le découpage ainsi obtenu n'a aucune raison à priori de correspondre au partitionnement qui sera effectivement utilisé lors de la conception de l'objet. Le but de la modélisation, c'est la simulation **[60].** Ce qui compte avant tout, c'est :

- la fidélité : un modèle se doit d'être aussi précis que possible dans son champ d'application;
- l'efficacité : le modèle doit pouvoir être simulé le plus rapidement possible et doit être portable, réutilisable et facile à maintenir.

Pour simuler un modèle, il faut disposer du modèle, bien sûr, mais aussi de stimuli, c'est à dire de la description des signaux d'entrée du modèle au cours du temps **[12]**. Jadis, avec les langages structurels, il n'était pas possible de décrire les modèles et les stimuli d'entrée à partir du même langage. Il fallait donc utiliser plusieurs langages, plusieurs outils et c'était autant de risques d'erreurs et de coût portés à la portabilité.

Grâce aux possibilités de description fonctionnelle des nouveaux langages, on utilise le même formalisme pour ces deux opérations pourtant bien différentes. La description de stimuli est une autre utilisation importante de ces langages.

Le résultat d'une simulation n'est pas toujours évident à vérifier. La seule inspection visuelle ne suffit pas dans les cas complexes où le nombre de données à analyser est grand, ou encore lorsque les calculs à effectuer pour valider un résultat sont trop complexes. Là encore, les langages fonctionnels de description de matériel permettent de simplifier cette tâche en analysant automatiquement les résultats en cours de simulation.

Un environnement de simulation complet comprend donc un *générateur de stimuli***,** un *modèle de l'objet à simuler* et un *vérificateur automatique des résultats*. Ces trois composantes sont modélisées à l'aide du même langage **[41]**. Le gain en simplicité de mise en œuvre et en portabilité de l'ensemble est considérable.

I.5.2 Synthèse

Les langages fonctionnels de description de matériel servent aussi à concevoir. Il ne s'agit plus de modéliser en vue de la simulation, mais de décrire les objets qui seront véritablement fabriqués **[61]**.

La totalité du langage ne peut plus être utilisée. Il est nécessaire de prendre en considération les limites des outils logiciels qui assureront la traduction du code en portes logiques (synthétiseurs) puis la traduction de la description structurelle en dessin au micron (placeurs-routeurs) **[60]**.

Si les considérations de vitesse d'exécution en simulation existent toujours (la description sera simulée avant d'alimenter le synthétiseur, afin de vérifier que la fonction décrite est bien la fonction désirée) elles ne sont plus prioritaires. Ce qui compte le plus désormais, c'est l'efficacité du code au sens du synthétiseur. En effet, pour que ce dernier produise la description structurelle la plus économique possible il faut que le concepteur ait tenu compte de ses limites et de ses spécificités qui ne vont souvent pas dans le sens de l'efficacité en simulation.

Conclusion

Les langages et outils de simulation (présentés dans ce chapitre) initialement développés pour la modélisation et la simulation, mais également utilisés par la suite pour la synthèse automatique, pour valider le fonctionnement d'un système électronique à des niveaux de plus en plus éloignés des technologies de fabrication. Parmi ces langages, on a choisi le VHDL qui sera détaillé dans le chapitre suivant.

Introduction

Le langage VHDL a été commandé par le Département de la Défense des États-Unis (DOD) **[56]**. Le VHDL est le fruit du besoin de normalisation des langages de description de matériel**.** Auparavant, chaque fournisseur de CAO proposait son propre langage de modélisation (GHDL chez GENRAD, BLM ou M chez Mentor Graphics etc...) mais aussi un autre langage pour la synthèse et encore un autre pour le test. Au début des années 80, (DOD) confiait le soin à Intermetrics, IBM et Texas Instruments de mettre au point ce langage **[11].** Il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui **[02]**.

Dans cette partie on introduira le VHDL comme langage de programmation pour le travail visé, par une étude globale de ce langage.

II.1 Définition complète du VHDL

Very High Speed Integrated Circuit, **H**ardware **D**escription **L**anguage

- VHDL : est un langage de description matériel destiné à représenter le comportement ainsi que l'architecture d'un système électronique numérique **[12]**, **[60]**. L'intérêt d'une telle description réside dans son caractère exécutable : une spécification décrite en VHDL peut être vérifiée par simulation, avant que la conception détaillée ne soit terminée **[26], [49].**
- Le VHDL est un langage portable **[02]**, c'est à dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit **[46],** qui va trouver place dans le cycle de conception du niveau spécification au niveau porte logique, mais aussi lors de la génération des vecteurs de test. La description VHDL est inséparable de la simulation de type événementielle **[39].**

II.2 Historique:

Le Département de la Défense des Etats-Unis a initié puis dirigé le projet "Very High Speed Integrated Circuit" (VHSIC). Ce projet avait pour but de formaliser la description des circuits intégrés développés pour le DoD dans un langage commun. L'intérêt primaire était de définir, au travers du langage, une spécification complète et non ambiguë du circuit à développer indépendante de la technologie employée et des outils de CAO.

- 1980: Début du projet, financé par le DOD **[39],** Initiative VHSIC pour son compte; langage inspiré de ADA ;
- 1982: Contrat pour Intermetrics, IBM et Texas;

- 1985: Version 7.2 dans le domaine public ;
- 1987: Standard IEEE 1076 (VHDL-87). La version initiale de VHDL, incluait un large éventail de types de données, numériques (entiers, réels), logiques (bits, booléens), caractères, temps, plus les tableaux de bits et chaînes de caractères **[47].**
- 1993: Nouvelle version du standard (VHDL-93) **[55]**, IEEE 1076.2 intègre les std_logic MVL9 (multivalued logic, nine values) **[26]** ;
- 1994 : IEEE 1076.3 (Numeric Standard) pour la synthèse ;
- 1995 : IEEE 1076.4 : VITAL initiative (VHDL Initiative Toward ASIC Libraries) pour la génération de modèles de timing des composants synthétisés ;
- 1999 : IEEE 1076.1 (1999), ou VHDL-AMS (VHDL-Analog and Mixed Systems) créé par L'IEEE Design Automation Standards Committee (DASC) afin de répondre aux différents problèmes de l'électronique **[11] , [55]** ;
- 2001: Nouvelle version du standard (VHDL-2001) ;
- 2008:Nouvelle version du standard (VHDL-2008) **[12].**

II.3 Domaine d'application

Circuits logiques : Circuits composés d'éléments effectuant des opérations logiques fondamentales **[09]**. Ces circuits sont à la base des systèmes numériques utilisés en :

- Informatique : développement d'applications industrielles ;
- Télécommunications : Cartes d'applications télécommunication **[40]** ;
- Robotique : mobile d'assistance au handicap ;
- Médecine : reconstruction d'images en temps réel pour un appareil d'imagerie médicale ;
- Electronique grand public : Afficheur électronique, console de jeux… etc **[30]**.
- Militaire/aérospatial **[28]** ;
- Contrôle industriel : les systèmes de commande **[03], [45]** ;
- Automobile : un système de contrôle de vitesse réactif (Cruise control) **[57]**;
- Communications de données (réseaux) : des routeurs pour des réseaux à puce **[04], [22]**;
- Stockage de données : stockage des données dans la mémoire ROM ;
- Téléphones mobiles : nouveau microprocesseur telecom VHDL pour Téléphones portables **[23]** ;
- Combinés multimédias ;
- Microprocesseurs **[20]**;
- Processeurs graphiques : cartes graphiques de micro-ordinateurs **[61]**.

II.4 Pourquoi le VHDL ?

Le schéma structurel que l'on utilise depuis si longtemps et si souvent n'est en fait qu'un outil de description graphique. Aujourd'hui, l'électronique numérique est de plus en plus présente et tend bien souvent à remplacer les structures analogiques utilisées jusqu'à présent. Ainsi, l'ampleur des fonctions numériques à réaliser nous impose l'utilisation d'un autre outil de description. Il est en effet plus aisé de décrire un compteur ou un additionneur 64 bits en utilisant l'outil de description VHDL plutôt qu'un schéma **[38]**.

Le deuxième point fort du VHDL est d'être "un langage de description de haut niveau". D'autres types de langage de description, comme l'ABEL par exemple, ne possèdent pas cette caractérisation. En fait, un langage est dit de haut niveau lorsqu'il fait l'abstraction la plus possible de l'objet auquel ou pour lequel il est écrit. Dans le cas du langage VHDL, il ne fait jamais référence au composant ou à la structure pour lesquels on l'utilise **[05]**.

Ainsi, il apparaît deux notions très importantes :

- **Portabilité des descriptions VHDL** : possibilité de cibler une description VHDL dans le composant ou la structure que l'on souhaite en utilisant l'outil que l'on veut (en supposant que la description en question puisse s'intégrer dans le composant choisi et que l'outil utilisé possède une entrée VHDL) ;
- **Conception de haut niveau** : une conception qui ne suit plus la démarche descendante habituelle (du cahier des charges jusqu'à la réalisation et le calcul des structures finales) mais qui se "limite" à une description comportementale directement issue des spécifications techniques du produit que l'on souhaite obtenir **[38].**

Pour implanter nos systèmes (produits), le choix du langage VHDL semble incontournable, quelque soit la cible que nous choisirons pour notre circuit final. Notre système sera alors réutilisable, flexible, extensible et permettra de modéliser la chaîne de communication **[42]**.

Le VHDL permet des descriptions pour la :

- Simulation: description logique et temporelle. Pour la modélisation (comportementale ou non) et le « testbench » ;
- Synthèse: description logique, à finalité d'implantation de circuits on doit respecter des règles d'écriture (types de signaux, instructions) plus strictes.

II.5 Objectif du langage VHDL

On peut donner une idée de la place occupée par VHDL en formulant une analogie entre les outils de développement pour les structures logique câblée et logique programmée :

Logique câblée	Logique programmée
Schéma structurel	Langage machine
Description par équations logiques	Langage assembleur
VHDL	Langage structuré

Tableau II.1. Les logiques câblée et programmée.

VHDL répond à deux objectifs :

- ✓ A l'origine le langage a été créé pour obtenir un modèle de simulation permettant de valider une solution avant de réaliser le composant ;
- ✓ Actuellement il est aussi utilisé pour la synthèse particulièrement pour la mise en œuvre des circuits programmables de type FPGA ou CPLD **[07]**.

II.6 Avantages et limites du VHDL

II.6.1 Avantages :

- ❖ L'utilisation du VHDL n'est pas limitée à l'électronique ;
- ❖ Adaptation efficacement structuré des processus électronique : conception du haut vers le bas (de la fonction spécifiée au circuit relatif) ;
- ❖ Structure hiérarchique de base de données de conception (implémentation, synthèse, simulation, implantation sur FPGA) ;
- ❖ Base de données mélangée de conception, c.à.d. schéma de circuit, IP (Intellectual Property) et VHDL ;
- ❖ Le langage de programmation VHDL inclut la langue « construction concourante » idéalement convenues à la description de matériel;
- ❖ Permet l'exécution des conceptions accrues de complexité ;
- ❖ Fournit des conceptions plus de haute qualité ;
- ❖ Documentation efficace de conception: fournit des spécifications détaillées qui peuvent être exécutées, un entretien plus simple que des méthodes schématiques de capture pour un code bien commenté en VHDL;
- ❖ Permet l'exploration des architectures alternatives de système dans le cycle de conception en décrivant le système à un niveau élevé - modèle abstrait de HDL- (sans besoin de conception détaillée au niveau porte logique) ;

- Permet la détection de problème depuis la description fonctionnelle du comportement au niveau élevé de l'abstraction (sans synchronisation complexe, problèmes liés à gâtes/transistors) ;
- Description de matériel indépendante de technologie:
  - Le choix de technologie est laissé jusqu' à l'étape de synthèse
  - Les outils de synthèse automatisent une grande partie du détail de technologie décisions (synchronisation, secteur, conduisant force, composant choix) ;
- Permet au concepteur de se concentrer sur la fonction de système.
- Les nouveaux outils d'ECAD peuvent fournir la « prochaine génération » de réalisations d'une base de données de conception VHDL existante.
- Des modèles VHDL peuvent être facilement réutilisés et adaptés.

II.6.2 Les limites actuelles

La norme qui définit la syntaxe et les possibilités offertes par le langage de description VHDL est très ouverte. En effet il est possible de créer une description VHDL de systèmes numériques non réalisable, tout au moins, dans l'état actuel des choses. Il est par exemple possible de spécifier les temps de propagations et de transitions des signaux d'une fonction logique, c'est- à-dire créer une description VHDL du système que l'on souhaite obtenir en imposant des temps précis de propagation et de transition. Or les outils actuels de synthèses logiques sont incapables de réaliser une fonction avec de telles contraintes. Seuls des modèles théoriques de simulations peuvent être créés en utilisant toutes les possibilités du langage **[38]**.

II.6 Comparaison

La structure VHDL a été tirée du langage ADA qui est un langage de création des logiciels, VHDL est destiné à décrire, simuler et synthétiser des circuits **[07]**. Malgré la diversité des langages HDL, ils répondent tous aux objectifs fixés par les grandes lignes des langages HDL. Actuellement, les plus utilisés dans l'industrie sont : VHDL et Verilog.

En comparaison entre VHDL et Verilog, ont été conçus chacun en donnant plus d'importance à un aspect plus tôt qu'un autre d'une description matérielle. En effet, VHDL se base plus sur le fait d'obtenir le circuit le plus optimisé que possible donc le moins encombrant dans le souci d'embarquer plus de fonctionnalités dans un circuit logique programmable. Ce qui n'est pas sans conséquence car le code se voit plus long que celui rédigé en Verilog pour une même

fonction. Par contre le langage Verilog cherche à réduire le temps de mise en place d'un code opérationnel donc le plus cours possible mais en réduisant les dimensions du code la conséquence, le circuit synthétisé consomme plus de ressources **[13]**.

II.8 Structure d'un programme VHDL

La structure d'un programme en VHDL est présentée comme suit **[49]** :

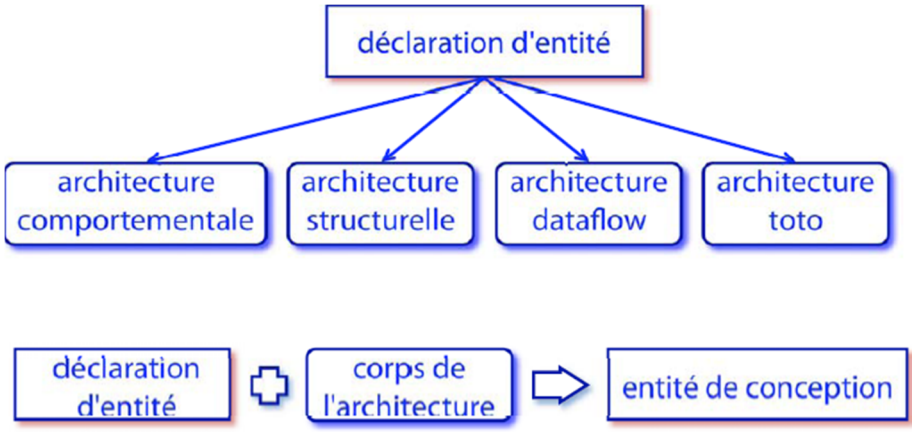


Figure II.01 Structure d'un programme en VHDL.

II.8 .1 Description structurelle

L'approche structurelle permet de décrire le fonctionnement d'un circuit en fonction de ses composants physiques. La description du fonctionnement des composants physiques eux-mêmes doit alors exister dans une bibliothèque prédéfinie ou définie par le programmeur **[27], [31]**. Pour illustrer en quoi consiste la description structurelle d'un circuit, considérons le sommateur complet dont le circuit logique est donné par la figure suivante **[29]** :

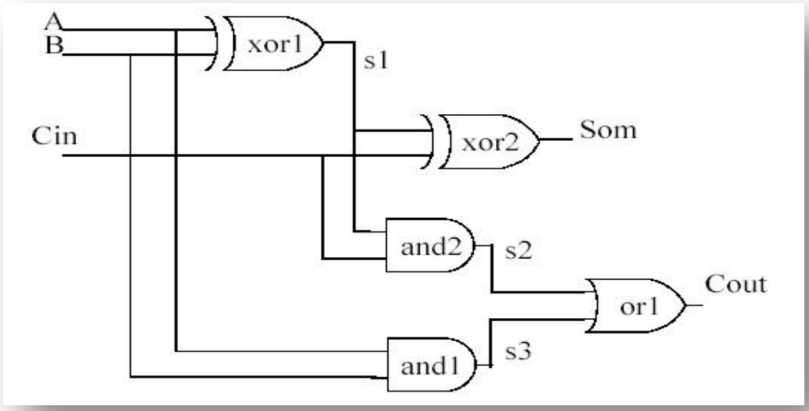


Figure II.02 Schéma logique d'un sommateur.

La description structurelle de ce sommateur est présentée à la figure II.03.Les mots clés réservés sont en gras tandis que les étiquettes ou variables définies par le programmeur sont en lettres minuscules. Tel qu'il a été mentionné, toute modélisation doit commencer par une définition d'entité. Cette définition débute par le mot clé **entity** suivie du nom du circuit modélisé ("sommateur") et contient la définition de ses signaux d'entrées et de sorties. Une initialisation de ces signaux est également possible à ce stade-ci, comme c'est le cas pour le signal "cin" qui est initialisé à "0". La définition d'architecture suit celle de l'entité et commence avec le mot-clé **architecture**, suivi du nom de l'architecture ("struct1"). Entre la définition de l'architecture et sa description au niveau des portes logiques se situent la déclaration des signaux internes (mot-clé **signal**) et des composants (mot-clé **component**) faisant partie du circuit du sommateur **[49]**.

définition de l'entité	**entity** sommateur **is** **port**(A,B : **in** qsim-state; C,in : **in** qsim_state:='0'; Som : **out** qsim_state); **end** sommateur;
définition de l'architecture signaux internes	**architecture** struct1 **of** sommateur **is** **signal** s1, s2, s3: qsim_state;
définition des composantes utilisées dans la modélisation structurelle du sommateur	**component** xor **port**(x1, x2 :in qsim_state; xo1: out qsim_state); **end component**; **component** and **port**(x1, x2 :in qsim_state; xo1: out qsim_state); **end component**; **component** or **port**(x1, x2 :in qsim_state; xo1: out qsim_state); **end component**;
description structurelle du sommateur	**begin** xor1 : xor **port map**(A, B, s1); xor2 : xor **port map**(s1, Cin, Som); and1 : and **port map**(A, B, s3); and2 : and **port map**(s1, Cin, s2); or1 : or **port map**(s2, s3, Cout); **end** struct1;

Figure II.03. Code VHDL structurel du sommateur en question.

On peut distinguer trois grandes étapes générales lors de la réalisation d'une architecture structurelle:

1) Identifier tous les signaux qui permettrons de relier les composantes entre elles (les fils sur le circuit) et les déclarer (**signal**);
2) Recenser tous les composants utilisés (les puces) et les définir par la syntaxe :

component <nom de la composante>

port (...)

end component;

3) Affectation des liaisons du schéma aux broches des composantes (les fils entre chaque portes) avec le mot réservé **port map(…).**

II.8.2 Description comportementale de type algorithmique

Le type de description comportementale se subdivise, d'après le style de programmation, en deux sous catégories: algorithmique et flux de données **[27]**. Ces deux styles ne sont toutefois pas mutuellement exclusifs et une combinaison des deux est parfois appelée style mixte **[49]**.

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

entity multiplex is
port(a,b,sel : in std_logic;
        y : out std_logic);
end multiplex;

architecture comportemental of multiplex is
begin
process(sel,a,b) -- la liste de sensibilité
        begin
                if sel = '0' then -- si sel vaut 0 alors
                        y <= a;
                elsif sel = '1' then -- si sel vaut 1 alors
                        y<= b;
                else
                        y <= 'x'; -- si sel n'est pas à 0 ou à 1, y vaut X (état indéterminé)
                end if;
        end process;
end comportemental;
```

Figure II.04. Exemple de description comportementale algorithmique.

La description illustrée par la figure II.04 est un exemple classique de description comportementale de type algorithmique. Lorsqu'on fait la description VHDL en style algorithmique, on ne se base pas sur un schéma logique, c'est une approche de "boîte noire". Le programmeur décrit le comportement des entrées et des sorties de la composante (son fonctionnement) sans avoir à se soucier du design intérieur de celle-ci.

L'intérieur de la puce (la boite noire) créée sera déterminé lors de la synthèse, par l'outil de synthèse. Le modèle de la figure II.04 est un multiplexeur à deux entrées implémenté selon le style comportemental algorithmique. Comme toujours la description commence par la définition de l'entité "multiplex". Ensuite on décrit le comportement de notre multiplexeur de façon

algorithmique. Le style comportemental algorithmique est en fait un programme VHDL qui utilise des instructions de contrôle comme if...then...else, case...when, etc..., comme dans la plupart des langages de programmation.

Une structure importante dans une architecture de type comportemental et qui est utilisée est la boucle process. Cette boucle peut avoir une liste de sensibilité ou non. Ainsi, l'instruction process (sel,a,b) indique que les instructions contenues entre le début et la fin de la boucle process ne sont à exécuter que dans le cas où l'un des trois signaux "sel", "a" et "b" subit une transition d'état. Une boucle process sans liste de sensibilité est en fait une boucle sans fin.

II.8.3 Description comportementale de type flux de données

Le type de description flux de données est illustré à la figure II.06 où le fonctionnement d'un comparateur simple est décrit. Ce comparateur est schématisé par la figure II. 05.

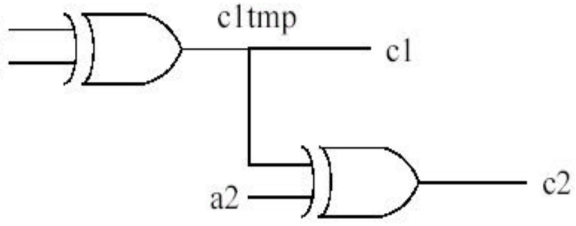


Figure II.05 Schéma logique du design de la figure II.06.

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

entity compar2 is
port (a1, b1, a2 : in std_logic := '0';
          c1 : out std_logic;
          c2 : out std_logic);
end compar2;

architecture behavior of compar2 is
signal c1tmp : std_logic;
begin
        c1tmp <= a1 XOR b1;
        c2 <= c1tmp XOR a2;
        c1 <= c1tmp;
end behavior;
```

Figure II.06 Exemple de description comportementale « Flux de données ».

Le flux de données est caractérisé par une assignation séquentielle ou concurrentielle de valeurs aux signaux de sortie du circuit **[27]**. On décrit donc en fait la relation logique entre les différents nœuds du circuit (entrées, sorties et nœuds internes). Le programme VHDL que l'on fait montre la façon dont les données ou signaux se propagent dans le circuit.

Remarque : Contrairement à la réalisation des programmes, quand on écrit une description VHDL, il faut toujours penser à la synthèse. Le but n'est pas d'optimiser le code et de réduire le temps d'exécution, mais bien de réaliser un circuit réduit et rapide.

II.8 .4 Description comportementale de type TOTO

La première ligne du code présenté par la figure II.07, permet la déclaration de l'entité BLA, TOTO et BLA sont des identificateurs. On fera référence au couple dans le langage et dans les outils par la syntaxe BLA (TOTO). Il peut donc y avoir plusieurs architecture par entité ce qui est une des forces du VHDL.

```
architecture TOTO of BLA is …
   [declarations]
begin
   instructions
end [architecture] [TOTO];
```

Figure II.07 Schéma de l'architecture TOTO.

Conclusion

Dans cette partie, on a définit le langage VHDL vue son objectif et son utilité, nous avons aussi cité quelques avantages par rapport aux autres langages de programmation pour motiver notre choix, afin de pouvoir exploiter tous ces connaissances pour réaliser nos systèmes qui seront présentés dans le chapitre suivant.

Introduction

L'application du VHDL est largement utilisée suivant des différentes méthodologies, ce qui a permis aux différents domaines d'évaluer très rapidement, en exploitant des nouvelles technologies, la manipulation des spécifications et les fonctionnalités des systèmes selon les besoins du marché. Dans ce chapitre, nous allons présentés deux exemple d'application du langage VHDL pour deux domaines distincts : le premier dans le domaine électronique et l'autre concerne les réseaux informatiques.

III.1 Méthodologie de conception utilisée

Il existe trois méthodes à suivre dans une conception soft/hard : Top-Down, Bottom-Up, et Meet-In-The-Middle.

III.1.1 Conception descendante (Top-Down) :

C'est la méthodologie utilisée pour ce projet, en effet elle est la plus employée dans le cadre commercial le plus courant, en effet un client veut un circuit spécifique, un ASIC (*Application Specific IC*) pour une application, par exemple un DSP spécifique pour un téléphone. Il commande la chose à une maison de conception (*Design-House*) qui est en générale une structure moyenne ou un petit département d'une grosse entreprise, travaillant sur CAO. La conception se fait alors des spécifications vers le circuit, la validation étant sous le contrôle du client. Les styles employés vont du niveau système au niveau flot de données. Dans ce cadre il faut un accord tripartite entre le client, la maison de conception et le fondeur : le *Sing-off* (figure III.01). Ce contrat est signé entre un vendeur de CAO qui garantit la qualité de ses outils, le fondeur qui garantit que la puce (le circuit) qui sortira sera conforme aux simulations de l'outil de CAO et le client qui paie **[14]**.

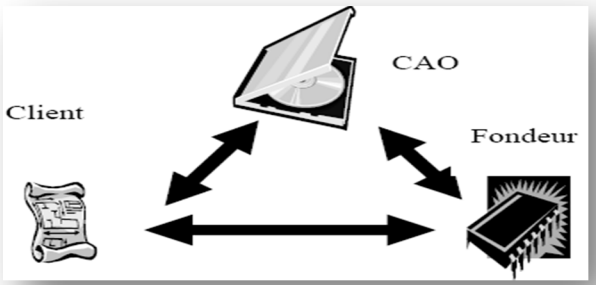


Figure III.01 Le Sing –off.

III.1.2 Conception montante (Bottom-Up)

C'est le cas de figure où un fournisseur cherche à faire un produit « d'étagère » ; le meilleur processeur possible est : un circuit spécialisé générique (exemple : contrôleur d'écran ou de disque). Le développent se fait en interne avec des spécifications souples que l'on peut remettre en question plus facilement que si l'on avait un contrat à amender. C'est le marché qui valide ensuite. La technologie peut ou ne peut pas être un paramètre important de la question **[14]**.

III.1.3 Conception montante – descendante (Meet-In-The-Middle)

Un fondeur doit développer pour chaque technologie une bibliothèque d'éléments de moyen ou bas-niveau, ou extrêment génériques, sur laquelle les outils de CAO et en particulier les outils de synthèse vont replier les circuits conçus par les maisons de conception **[14]**.

- **Les circuits programmables par l'utilisateur**

Un circuit programmable est constitué d'opérateurs logiques élémentaires (portes, multiplexeurs simples et bascules), dont une partie des interconnexions sont modifiables par l'utilisateur **[54]**. Ce dernier définit la fonction à réaliser par un schéma, des équations logiques, un programme dans un langage de description ou une combinaison des méthodes précédentes. La programmation d'un circuit fait appel à un outil logiciel, un compilateur et un simulateur **[08]**. La complexité d'un circuit programmable est généralement exprimée en nombre de portes équivalentes, en nombre de bascules disponibles et en nombre de cellules d'entrées sorties (reliées à une broche du circuit) **[18]**.

III.2 Phase de développement:

Le développement de notre application suivra une démarche particulièrement différente au développement habituel des applications informatique sous java, Delphi ... etc, en effet, un projet VHDL est très souvent décomposé en sous-structures, appelées modules ou composants **[17], [40],[53].** Comme on pourrait le faire sur un circuit imprimé, il s'agit d'instancier différents composants et de les relier entre eux par des fils (signaux en VHDL) pour réaliser le système complet. Chacun de ces modules est alors décrit dans un fichier source indépendant. Chacun de ces fichiers sources peut alors être testé indépendamment en lui associant un fichier de simulation (ou testbench). Enfin des fichiers de contraintes permettent de faire le lien entre la description logicielle et la structure matérielle de la cible et de son environnement (position des entrées, des sorties...) **[17]**. Alors, nous allons suivre les étapes décrites à la figure III.02:

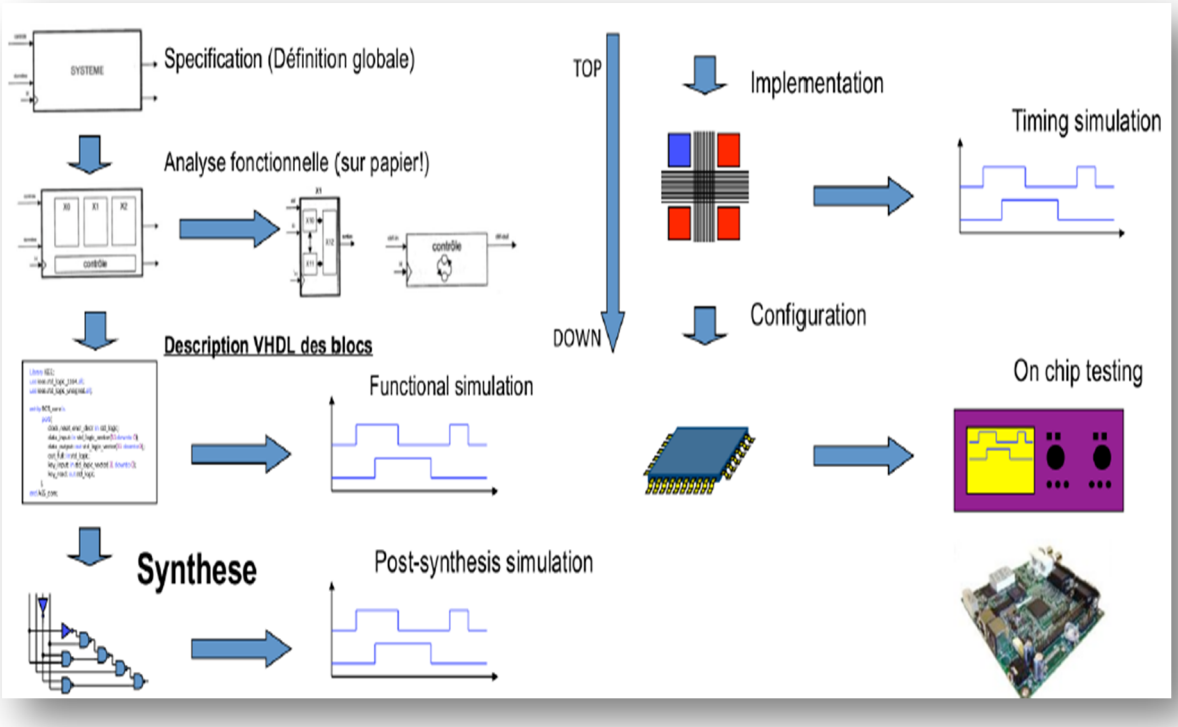


Figure III.02 Les étapes de la phase de développement.

III.2.1 Spécification: Dans cette étape on définit globalement le système à concevoir, à savoir les entrées, les sorties, et les entrées sorties, ainsi que leurs types.

III.2.2 Analyse fonctionnelle (sur papier) : Cette étape consiste à spécifier les fonctionnalités du système et les décrire sur papier par exemple : blocs, organigramme...etc.

III.2.3 Description VHDL : C'est l'implémentation du l'algorithme descriptif des fonctionnalités du système à concevoir et lancer la *simulation fonctionnelle* qui permet de savoir si le fonctionnement du système est celui décrit.

III.2.4 Synthèse: C'est la visualisation de circuit réalisé et lancer la *simulation post-synthèse* qui permet, après synthèse, de vérifier à nouveau le fonctionnement du système.

III.2.5 Implémentation : Une fois que toutes les étapes de «compilation» sont réussies, un fichier (nom_projet.bit) est maintenant généré et placé dans le répertoire du projet et est prêt à être chargé sur le FPGA de la planchette de développement. Dans cette étape on lance aussi une simulation dit *simulation temporelle* qui va permettre, après le routage, de valider les contraintes temporelles associées au placement des composants.

III.2.6 Configuration: La configuration est la dernière étape dans le processus de conception en particulier du projet, il s'agit du transfert de la solution vers un support physique FPGA. La configuration permet de charger le fichier de description VHDL (nom_projet.bit) sur un support FPGA (migration de la solution vers la cible hardware) choisi convenable (cette étape est réalisée au laboratoire) **[09], [10], [16].**

III.3 Présentation des deux exemples d'application

On présente dans ce qui suit les deux exemples d'utilisation du langage VHDL, selon le domaine d'application :

- Conception d'un modèle d'une gestion par batterie pour une gestion optimisée d'énergie électrique pour un système multi sources, dans le cadre des travaux développés pour les énergies renouvelables ;
- Conception et implémentation d'un routeur tolérant aux fautes (détection et correction des erreurs), pour un réseau sur puce.

III.3.1 Conception d'un modèle d'une gestion par batterie pour un système multi sources

Ce travail s'inscrit dans le cadre de développement des applications utilisant un système de production d'énergie hybride autonome à partir des panneaux photovoltaïques, génératrices éoliennes et de groupe électrogène **[24]**. Nous n'étudierons pas ici l'architecture interne des différentes technologies utilisées en électronique numérique, pour le concepteur de système numérique un circuit apparaît comme une « boite noire » **[30]** dont le fonctionnement est entièrement défini par ses caractéristiques externes, tant statiques (volts et milliampères) que dynamiques (nanosecondes et mégahertz) **[05]**.

La modélisation et l'approche par simulation numérique permettra l'étude du système globale et son optimisation par : la définition et le dimensionnement des différentes sources d'énergie, la gestion du stockage et des transferts d'énergie et la gestion de la distribution d'énergie vers les récepteurs **[15]** ...etc.

III.3.1 .1 Objectif

L'idée est de développer un circuit pour une gestion optimisée d'énergie électrique pour un système multi sources **[28],** avec VHDL et implémentation sur FPGA XILINX d'une carte

d'acquisition de manière à s'adapter à tous types de matériels. La conception du code VHDL doit rester générique et configurable par l'utilisateur sur le PC sans reprogrammer le FPGA.

Le but de ce projet est de fournir un FPGA modulable pour que chaque I/O du FPGA puisse réaliser la fonction souhaitée au moment de sa programmation. Il faut donc au départ fournir une programmation de chaque I/O, c'est-à-dire définir si chaque I/O est une entrée, une sortie ou une horloge, c'est la configuration du FPGA fournie par le logiciel PC (l'utilisateur). Chaque I/O peut se schématiser comme suit :

Figure III.03 Vue générale des I/O configurable sur FPGA par PC.

III.3.1.2 Le cahier des charges

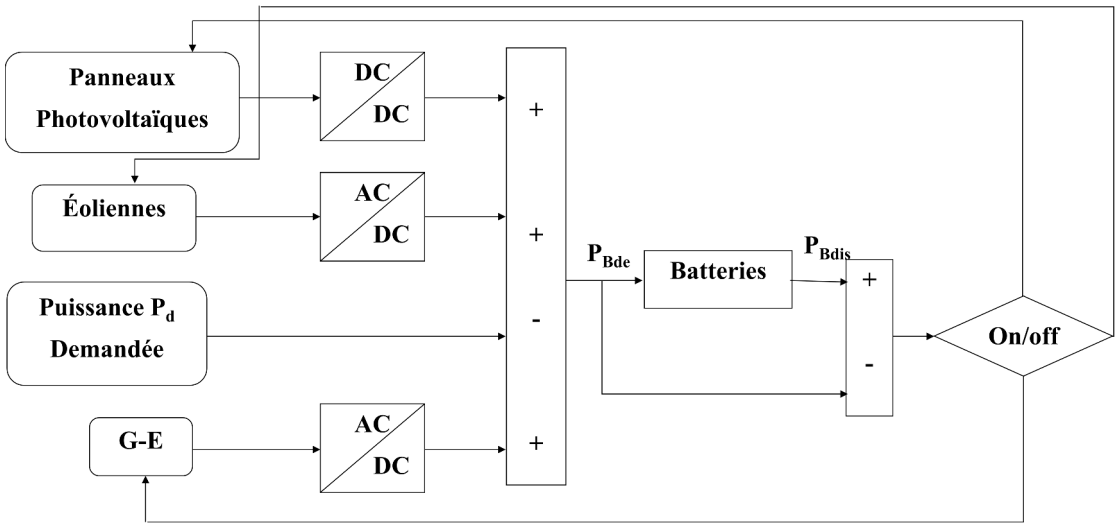


Figure III.04 Schéma de gestion de l'énergie électrique pour un système multi sources.

Les modules solaire et éolien permettent de déterminer l'énergie que peuvent fournir les systèmes de production pour des conditions météo et des technologies données. Ces dernières sont fournies à l'outil d'optimisation avec les données de consommation, les caractéristiques du réseau, du dispositif de stockage et de l'onduleur ainsi que leurs performances énergétiques et leurs fonctions coûts. L'outil d'optimisation nous donne alors le dimensionnement des différents

éléments du système ainsi que la stratégie optimale des transferts d'énergie. Autre avantage de ce montage PV- éolienne - GE // batterie / charge en parallèle : la batterie jouera le rôle de régulateur de tension pour alimenter la charge. La batterie impose la tension du montage parallèle et stabilisera ainsi la tension fournie à la charge, ce qui est un avantage **[01]**.

On prend une partie de ce système qui est la batterie comme un exemple d'application VHDL. La gestion des batteries est assurée indépendamment des trois commandes citées plus haut et repose sur deux critères fondamentaux : protection contre les décharges profondes et charge optimale **[15].** Nous adoptons une régulation dite « on-off » qui est représentée par 0/1 pour la gestion des batteries **[01]**.

Cette gestion basée sur la tension des batteries déconnecté ou reconnecte celles-ci suivant quatre seuils :

- **V4** : seuil haut de coupure, en anglais HVD pour « High Voltage Disconnect »: ce seuil correspond a la tension de la batterie lorsqu'elle est complètement rechargée ; on arrête alors la charge : « Off ».
- **V3** : seuil haut de reconnexion : il permet d'introduire une hystérésis afin d'éviter les phénomènes de pompage du régulateur ; la recharge ne reprend que lorsque la tension de la batterie est inferieure a V3.
- **V2** : seuil bas de reconnexion : il introduit une deuxième commande permettant d'éviter les effets de pompage lorsque la batterie est en état de décharge. En effet, lorsque la batterie atteint sa tension minimale de fonctionnement (tension fixée suivant la profondeur de décharge voulue) et qu'une recharge s'en suit, la reconnexion de la charge n'est possible que lorsque la tension de la batterie est supérieure à V2.
- **V1** : seuil bas de coupure ; lorsque la tension de la batterie atteint le seuil bas de coupure, on considère qu'elle est complètement déchargée, il faut donc la déconnecter de la charge **[24]**.

Pour le rôle que joue la batterie dans l'optimisation d'énergie pour le système multi sources, ainsi que les différentes interactions entre la batterie, les entrées et les sorties du système, on propose le schéma suivant :

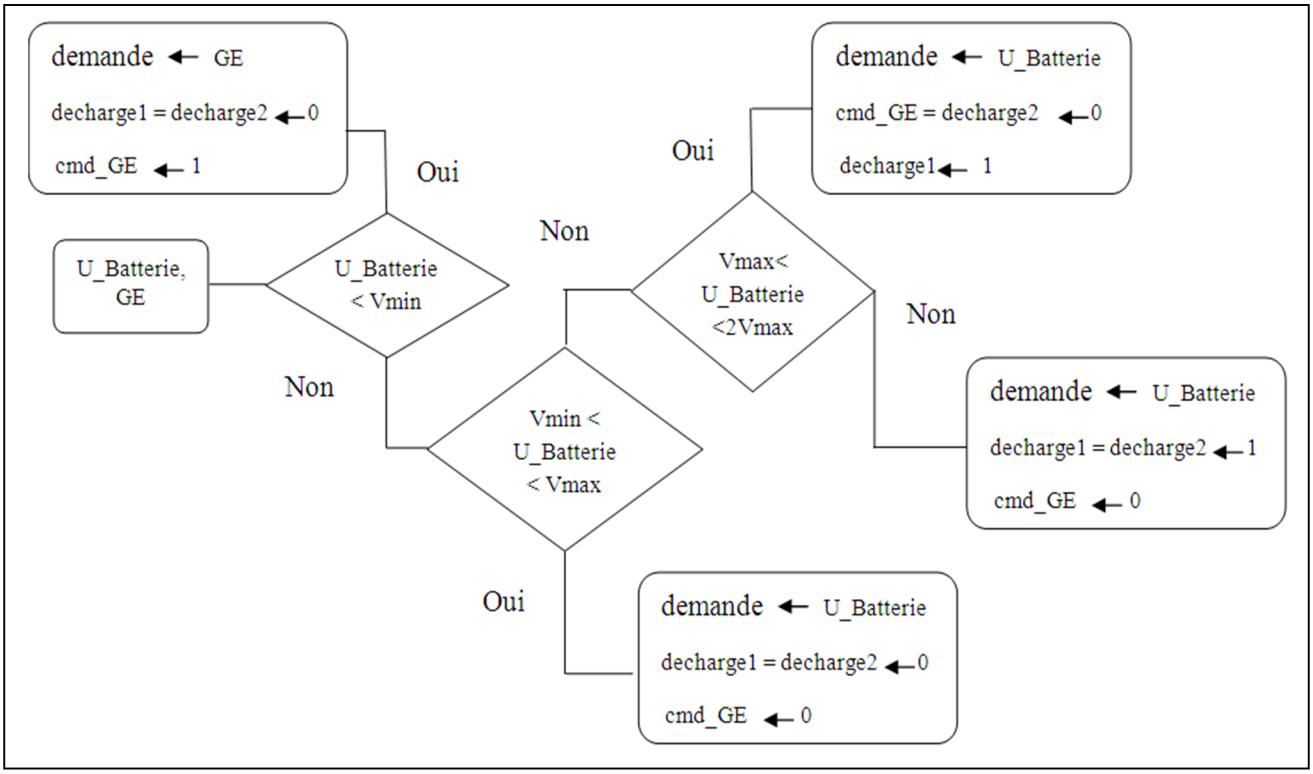


Figure III.05 Principe de fonctionnement de la batterie.

Indication

Selon le schéma et par correspondance avec les quatre seuils cités ci-avant, le modèle de batterie fonctionne comme suit :

- Dans le de **V1** : l'alimentation sera fournie par une autre source (groupe électrogène *GE*) que les unités de batterie *(U_Batterie)* qui sont déchargées, et ne peuvent pas satisfaire la demande de consommation car elle est inferieur à la limite inferieure du chargement *(Vmin),* à cet effet les deux décharges : *décharge1* et *décharge2* sont désactivées (off).
- **V2** : la demande de consommation sera fournie par *U_Batterie* qui est entre *Vmin* et la limite supérieure, et on désactive : *GE, décharge1* et *décharge2*.
- **V3** : là on aura *U_Batterie* dépasse *Vmax*, l'alimentation sera toujours fournie par *U_Batterie* mais avec déclanchement d'une seule décharge *decharge1* et on désactive *decharge2* et *GE*.
- **V4** : dans ce cas on aura *U_Batterie* dépasse deux fois *Vmax,* de même que pour **V3** mais il faut activer la *decharge2*.

III.3.2 Conception d'un routeur tolérant aux fautes pour un NoC

Durant cette dernière décennie, l'évolution technologique conduit à l'augmentation de la densité d'intégration et de la fréquence de fonctionnement des circuits intégrés **[23].** Ceci permet aujourd'hui d'assembler sur une même puce un système numérique complet, appelé système sur puce ou SoC (System-on-Chip). Ces systèmes font de plus en plus partie de notre quotidien et se retrouvent dans des domaines comme la télécommunication, l'automobile, la télévision numérique …etc. par conséquent ils sont aussi de plus en plus susceptibles à être défaillants. C'est pourquoi la notion de tolérance aux fautes des données dans les SoCs est de plus en plus préoccupante pour les concepteurs des circuits intégrés, et également fait l'objet de nombreux thèmes de recherche **[31], [33]**.

III.3.2.1 Objectif : Conception d'un routeur *NoC* tolérants aux fautes

Un routeur est composé de buffers d'entrée, de sortie ou d'entrée-sortie dont le rôle est d'accepter temporairement les paquets à transmettre de la part du *PE* qui lui est associé ou de ses routeurs voisins dans le but de faire transiter vers une direction des paquets que ne lui sont pas destinés. Pour cela, un routeur dispose généralement d'un module *crossbar* utilisé pour l'aiguillage des paquets des ports d'entrée vers les ports de sorties du routeur selon l'adresse de destination contenue souvent dans le premier paquet (*paquet d'en-tête - header packet*).

Afin d'éviter les collisions des paquets arrivant simultanément sur les entrées d'un même routeur et ayant pour destination la même sortie du routeur, ce dernier intègre un module de contrôle ou d'arbitrage (*control logic*). En résumé, un message constitué de plusieurs paquets de données traverse un ou plusieurs routeurs à travers les canaux d'interconnexion avant d'arriver à un *PE* destinataire final.

III.3.2.2 Etat de l'art des réseaux sur puce

1) Réseaux sur puce

Les réseaux sur puce (NoC) apparaissent comme une alternative pour connecter des modules intégrés sur une même puce vis-à-vis d'une approche d'interconnexion basée sur une structure de communication de type bus partagé ou hiérarchique **[25]**. Ils représentent une solution de transposition et de réduction d'échelle (au niveau d'un même substrat) des concepts des grands réseaux de communications (*large-scale networking*) vers le domaine des systèmes sur puce embarqués (*SoC, MPSoC*). En effet il doit répondre à des contraintes de performance et de coût liés

à la complexité et l'augmentation croissante de modules ou d'IPs (Intellectual Properties) interconnectés **[27]**. Actuellement un tel réseau de communication sur puce met en œuvre des transmissions de données par paquets vers les nœuds interconnectés au réseau correspondant aux modules ou IPs intégrés au système (processeurs, mémoires, contrôleurs de périphériques reliés, etc.). Cette transmission est réalisée à travers des routeurs (constituant le réseau) en mettant en œuvre des règles d'aiguillage et de routage des paquets de données dans le réseau, (les *NoC* reposent sur une paquetisation de données pour transporter les données d'une source vers une cible à travers des routeurs et de canaux d'interconnexion (*interconnection link)*) **[26]**.

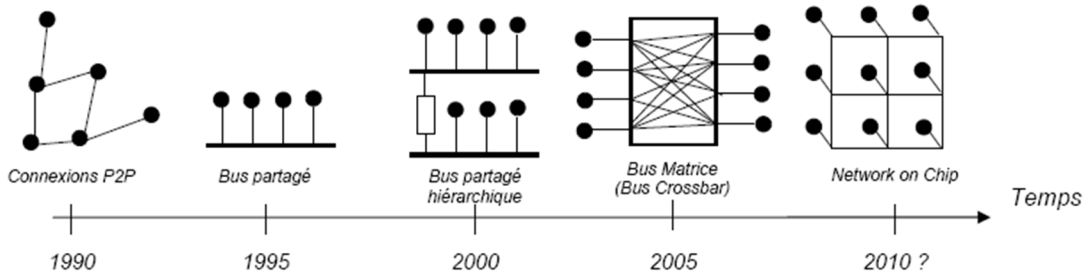


Figure III.06 Evolution des interconnectes dans les Systèmes sur Puce.

2) Architecture d'un NoC (Network on Chip)

NoC est un réseau d'interconnexion relie des routeurs ou Switchs, auquel sont connectées les IP **[20].** Une architecture contenant des processeurs, des blocs IP, des composants de mémorisations et des interfaces E/S comme le montre la figure III.07. Les NoCs permettent une extensibilité et flexibilité de l'architecture, une variabilité des données transférées à haut débit.

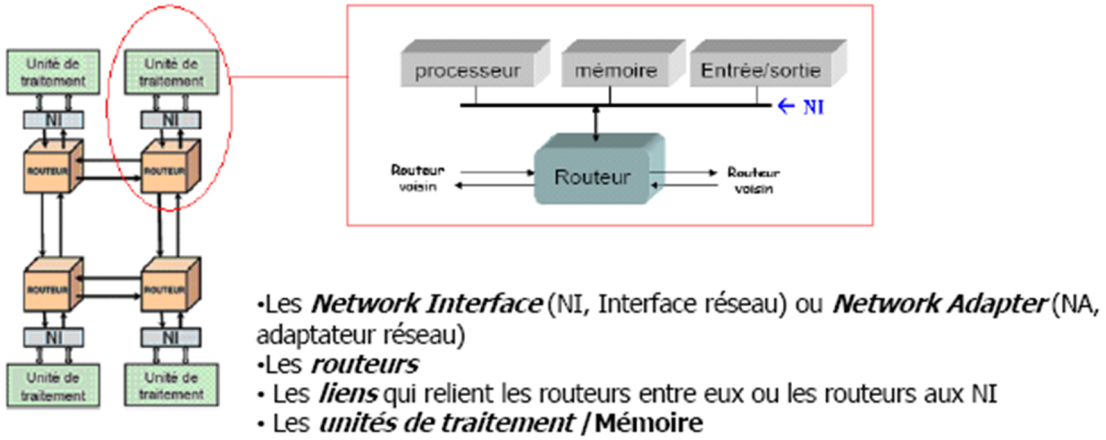


Figure III.07 Architecture d'un NoC.

3) Les réseaux sur puces selon le modèle de référence *OSI :*

Puisque le concept de NoC est emprunté aux réseaux informatiques, il est possible d'employer les mêmes protocoles de communication que ceux utilisés dans les réseaux informatiques. Dans la conception de NoC, il faut simplifier ces protocoles afin d'obtenir le meilleur compromis entre la performance et le coût d'implémentation. Différentes propositions dans **[19]**, **[23]**, **[26]** se concentrent sur l'adaptation de la structure du modèle de référence OSI, dont les sept couches sont présentées dans la figure III.08, pour le protocole de NoC.

Dans la plupart des cas seules les quatre couches basses du modèle OSI sont implémentées dans les NoC. Les couches supérieures sont en général plus spécifiques à l'application et prises en charge par les ressources de calcul (IP, CPU) connectées au réseau (implémentées au niveau système).

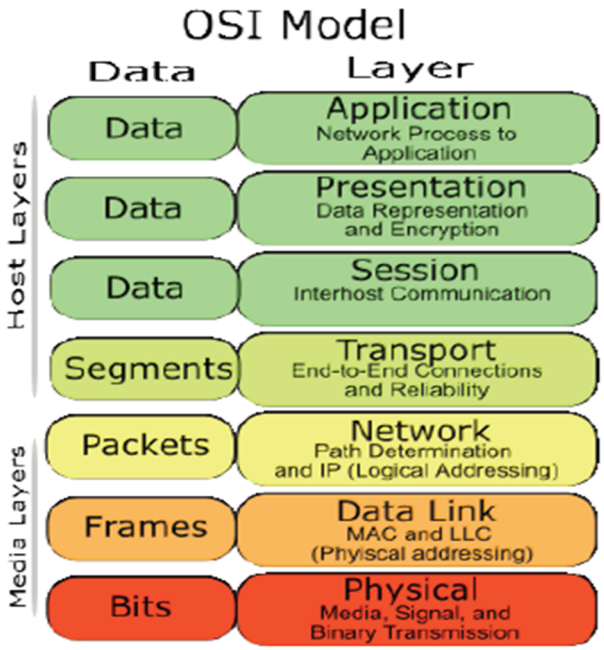


Figure III.08 Le modèle OSI.

Dans le reste du paragraphe, nous présentons rapidement les fonctionnalités des quatre couches implémentées dans les NoC :

✎ ***Couche physique*** : correspond au niveau le plus bas dans le modèle OSI. Elle décrit les conditions matérielles du transfert de données : la tension et les temps caractéristiques des signaux, le nombre et la longueur des fils d'interconnexion entre les routeurs. Elle est le support de transmission de l'information au niveau bit ou mot entre deux routeurs.

Couche liaison de données (Data Link) : établit une connexion logique entre les routeurs ou entre un routeur et l'interface de réseau attachée. Elle définit la façon de transmettre les informations entre deux routeurs et assure la fiabilité de communication sur les liens physiques. Donc, elle peut inclure des mécanismes de synchronisation, de contrôle de flux, et de correction d'erreur.

Couche réseau (Network) : concerne l'´echange des informations au niveau du paquet dans le réseau. Elle définit comment envoyer un paquet à travers le réseau d'un expéditeur à un destinataire. Elle prend en charge les mécanismes de commutation et les algorithmes de routage.

Couche Transport : est le niveau d'échange des informations entre deux unités des données. Elle découpe les blocs de données de la couche session en paquets de taille raisonnable pour le réseau. Elle prend en charge le contrôle de flux, un ensemble de mécanismes pour éviter la surcharge du réseau et régler le trafic. Les trois techniques suivantes sont souvent utilisées :

- *Technique de fenêtre* : une fenêtre glissante détermine que certains paquets peuvent circuler dans le réseau pendant une période prédéfinie ;
- *Technique de contrôle de débit* : le débit d'envoi de chaque émetteur est contrôlé et limité ;
- *Technique de crédit* : l'´emetteur doit avoir reçu des crédits en provenance de la destination avant d'envoyer un paquet de données. Le destinataire envoie des crédits à l'´emetteur en fonction de ses capacités de stockage de nouveaux paquets. A chaque paquet envoyé, l'´emetteur doit baisser son compteur de crédits.

4) Topologies des réseaux sur puce

La topologie d'un réseau définit comment les routeurs sont interconnectés entre eux en utilisant les liens de réseau. Elle spécifie l'organisation physique du réseau et elle est donc souvent modélisée par un graphe. Comme pour les réseaux informatiques, de nombreuses topologies sont envisageables pour construire un NoC. La figure III.09 introduit sélectivement les topologies les plus couramment utilisées dans la conception des NoC. Mais la topologie prédominante est la topologie de maillage simple à deux dimensions (2D-mesh). Les raisons sont ses avantages tels que la facilité d'implémentation dans les technologies actuelles, la simplicité des stratégies de routage, et l'évolutivité du réseau **[25]**.

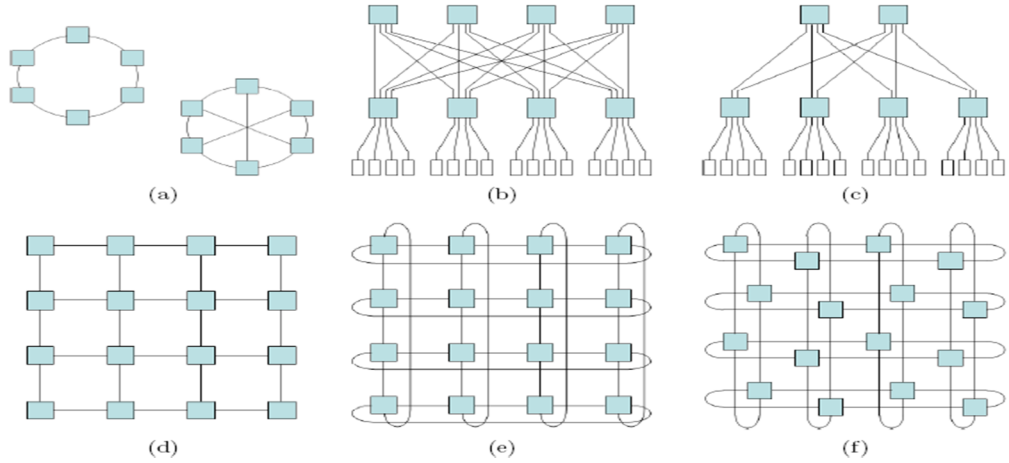


Figure III.09 Topologies usuelles de réseaux sur puce.

(a) Réseau en anneau avec ou sans corde ; **(b)** Réseau en arbre élargi ;

(c) Réseau en arbre élargi en papillon ; **(d)** Réseau à maillage simple à deux dimensions ;

(e) Réseau maillé en tore à deux dimensions ; **(f)** Réseau maillé en tore replié à deux dimensions.

5) Avantages et inconvénients

➲ **Les avantages des réseaux sur puce :**

- ❖ Interconnexions : flexibles, extensibles, grand débit cumulé ;
- ❖ Pas d'arbitrage central ;
- ❖ Deux qualités de service : BE, GT.

➲ **Les inconvénients des réseaux sur puce :**

- ❖ Latence (fonction du nombre de routeurs traversés) ;
- ❖ Besoin de règles pour garantir le trafic (contexte GT) ;
- ❖ Risque de contention ou de deadlock ;
- ❖ Coût matériel plus important comparé à une approche bus ;
- ❖ Complexité de mise en œuvre accrue.

6) Pourquoi le NoC devient incontournable ?

- ❖ Complexité grandissante des schémas d'interconnexion entre les UT (Unité de Traitement) ;

- Nécessité de proposer une qualité de service pour les communications pour répondre aux besoins actuels et futurs des SoC : fexibilité, évolutivité, scalabilité.

III.3.2.3 Cahier de charge : application VHDL

Un exemple d'une structure *NoC* de type maillé *2D* (*mesh*) est présentée à la figure III.10, Elle est composée de plusieurs éléments de calcul *PE* (*Processing Element*) connectés les uns aux autres via les routeurs et les fils d'interconnexion d'une taille fixe. Un élément de calcul *PE* (souvent appelé un nœud), peut être soit un *processeur*, une fonction spécifique de calcul (*IP*), soit un bloc de mémorisation ou une combinaison de tous ces composants reliés ensemble. Un *PE* est généralement connecté à un routeur via un module d'interface *NI* (*Network Interface*) dont la fonction principale est la mise en paquets de toutes les données générées et à transmettre par un *PE* vers un autre *PE* à travers le réseau de communication.

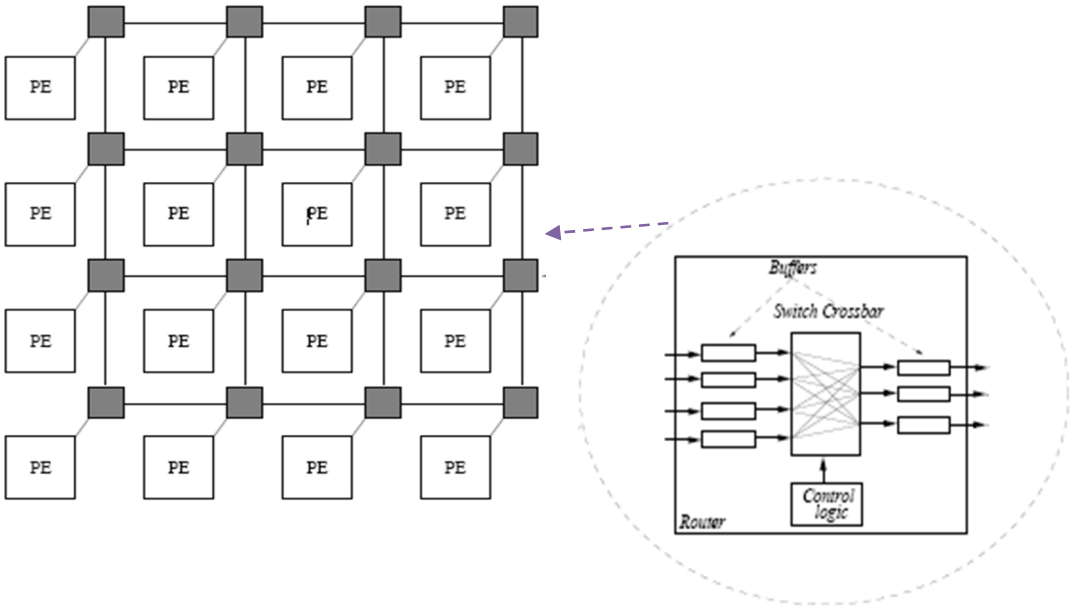


Figure III.10 Structure du NoCs de type Mesh.

A) Description du Model *VHDL* comportemental des routeurs interconnectés selon une structure MESH 4x4

Il s'agit d'une description d'un réseau de topologie maillée de taille *4x4* dont l'architecture structurelle d'un routeur est détaillée en Figure III.11. Chaque nœud de routage dispose de quatre directions (*North*, *South*, *East* et *West*) et des interconnexions unidirectionnelles permettant l'envoi et la réception simultanée de paquets de données issus du réseau.

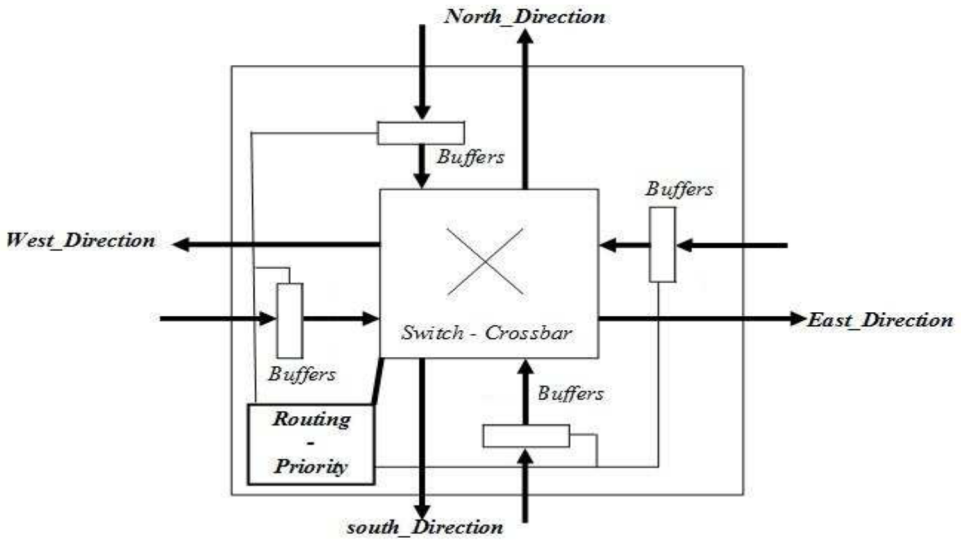


Figure III.11 Architecture d'un routeur *NoC.*

La structure des paquets de données à transmettre dans le réseau, les règles de contrôle et de transmission des paquets à travers le model du réseau proposé décrits ci-dessous sont :

- ***Structure des messages*** *:* Afin de faciliter les règles d'échanges des paquets de données entre les routeurs, les paquets sont composés d'un seul *flit (*mot de données de taille fixe paramétrable). Le réseau étant constitué de 16 nœuds selon un acheminement des paquets de données sur les axes *X* et *Y* du réseau, un paquet de données est alors constitué de quatre bits d'adressages (deux bits pour la position du nœud selon l'axe *X* et deux bits pour la position selon l'axe *Y*). Un paquet contient quatre bits de données. La figure III.12 illustre la structure d'un paquet de données. La taille générale du message est paramétrable via une déclaration générique afin de permettre l'intégration de données de contrôle en vue de l'intégration de concepts de tolérance aux fautes sur les données circulant dans le réseau

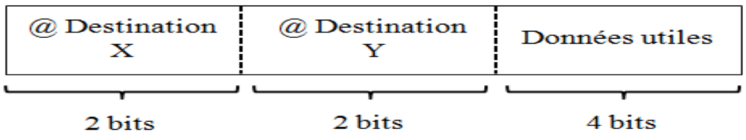


Figure III.12 Structuration d'un paquet de données du réseau *NoC –Mesh 4x4*.

- ***Technique d'aiguillage*** *:* La technique d'aiguillage des messages entre les routeurs s'effectue par commutation de paquet (*packets switching*) de type *Store and Forward* **[28].** Cela signifie qu'un paquet ne peut pas être transféré vers un autre routeur tant que ce dernier ne peut le recevoir dans son intégralité.

- ***Contrôle de flux*** *:* Le contrôle de flux des données entre les routeurs est de type *ACK/NACK* **[28].** Plus précisément, une copie de la donnée à transmettre est gardée dans le routeur émetteur (buffer local) jusqu'à ce que le routeur destinataire valide la réception de la donnée. Dans le cas d'une validation positive (*ACK*), la copie est supprimée. Dans le cas contraire (*NACK*) la donnée est retransmise.
- ***Algorithme de routage et technique d'arbitrage*** *:* L'algorithme de routage initialement implanté dans les nœuds du réseau est de type *XY* déterministe **[27]** signifiant que les paquets de données sont dirigés vers le *PE (nœud)* destinateur à travers les routeurs du réseau selon d'abord l'axe des abscisses *X* du réseau, puis selon son axe des ordonnées *Y***.** La technique d'arbitrage est celle d'une priorité à droite **[27].** Lorsqu'un routeur reçoit simultanément plusieurs paquets, le paquet le plus à droite est routé en priorité. Si quatre paquets arrivent simultanément dans un routeur, c'est une direction prioritaire préalablement définit par l'utilisateur qui est routé en premier **[32]**. Dans notre cas d'étude, la direction *EAST* est définie comme prioritaire dans ce cas de figure.

La modélisation de l'algorithme de routage *X_Y* est réalisée en *VHDL* selon l'algorithme suivant:

If X_routeur < X_destination, direction paquets = Direction_Est ;

If X_routeur > X_destination, direction paquets = Direction_Ouest ;

If X_routeur=X_destination et Y_router>Y_destination, direction paquets = Direction_Sud ;

If X_routeur=X_destination et Y_router <Y_destination, direction paquets = Direction_Nord ;

If X_routeur =X_destination et Y_router =Y_destination, direction paquets = Local_PE.

A partir d'un fichier contenant la description comportementale *VHDL* d'une structure *NoC* de topologie maillée (*Mesh* 4x4) et d'algorithme de routage *X-Y*, on propose de modifier l'architecture interne des routeurs afin d'y intégrer des modules de détection et de correction d'erreurs de données. Les solutions apportées sont globalement libres mais doivent valider en termes de qualité les détections de fautes par simulation et évaluer le coût en termes de ressources logiques et de performance induites par les modifications architecturales des routeurs du réseau.

B) Conception et implantation d'un routeur tolérant aux fautes

A partir d'un réseau supposé émettant des erreurs lors des échanges de paquets de données entre les deux IPs interconnectés, on modifie la structure des routeurs afin de les adapter pour une

sûreté de fonctionnement. La fiabilité du réseau proposée consiste à détecter et corriger les erreurs de transmission à travers une nouvelle conception des routeurs mettant en œuvre des techniques de détection et de correction d'erreurs et des solutions algorithmiques de contournement des nœuds défaillants (proposition et implantation d'algorithme de routage adaptatif au sein des blocs de routage des nœuds du réseau).

C) Détection et correction d'erreurs dans une structure *MESH 4x4*

Le travail consiste à détecter les erreurs présentent dans le NoC et de les corriger. Il existe plusieurs algorithme de détection d'erreurs parmi aux : le codage binaire de Golay, codage de Golay étendu, et le codage de *Hamming.* On se voit proposer une solution simple à mettre en œuvre basée sur l'implantation d'un code correcteur de *Hamming* et d'une parité :

L'ajout de 4 bits de *Hamming* ainsi qu'un bit de parité, permet de détecter jusqu'à deux erreurs et de corriger une erreur. Ce bloc de détection et de correction est décrit en *VHDL* et intégré à l'ensemble des routeurs constituants le réseau. De même, des blocs de codage de *Hamming* sont également implantés dans les blocs *IPs* de transmission et de réception de paquets de données. Les spécifications de conception du principe du codage et décodage d'*Hamming + 1 bit de parité* sont décrites ci-dessous.

Pour un message initial codé sur 8 bits [D1 D2 D3 D4 D5 D6 D7 D8], 4 bits d'*Hamming* (P1, P2, P3, P4) et un bit de parité (P5) sont générés. Le calcul de ces parités repose sur l'hypothèse d'une parité paire entre les bits du massage ***Di*** et les bits de parité ***Pi*** telle que:

P1 = D1⊕ D2⊕ D4⊕ D5⊕ D7; P2 = D1 ⊕D3 ⊕ D4⊕ D6⊕ D11 ;

P3 = D2 ⊕ D3⊕ D4 ⊕ D8, P4 = D5 ⊕D6 ⊕ D7 ⊕ D8 ;

P5 = P1⊕ P2 ⊕ D1 ⊕ P3 ⊕ D2 ⊕D3 ⊕ D4 ⊕ P4 ⊕D5⊕ D6⊕ D7⊕ D8.

Le principe du décodeur de *Hamming + 1 bit de parité* est réalisé à la réception du message. Les bits de vérification ***Vi*** sont calculés de la même manière que pour le codage des bits de parité ***Pi***. Un bit de parité globale ***P*** est également calculé :

V1 = P1⊕ D1⊕ D2⊕ D4⊕ D5⊕ D7, V2 = P2⊕ D1 ⊕D3 ⊕ D4⊕ D6⊕ D11 ;

V3 = P3 ⊕D2 ⊕ D3⊕ D4 ⊕ D8, V4 = P4 ⊕ D5 ⊕D6 ⊕ D7 ⊕ D8

P = P5 ⊕ P1⊕ P2 ⊕ D1 ⊕ P3 ⊕ D2 ⊕D3 ⊕ D4 ⊕ P4 ⊕D5⊕ D6⊕ D7⊕ D8.

A partir d'une analyse des bits de vérification et de la parité globale, des détections et corrections d'erreurs peuvent être réalisées en considérant le mot binaire **V** = V4 V3 V2 V1. Quatre cas de figure se présentent alors :

- Si **V** = 0000 et **P** = 0 : aucune erreur est détectée;
- Si **V** ≠ 0 et **P** = 1 : une seule erreur pouvant être corrigée est détectée. Le codage ***V*** donne la position du bit erroné à inverser pour correction (Par exemple ***V*** = *0110* signifie inversion du digit à la 6ème position) ;
- Si **V** ≠ 0 et **P** = 0 : deux erreurs sont détectées mais ne peuvent être corrigées ;
- Si **V**= 0000 et **P** = 1 : une erreur est présente sur le bit de parité ***P.***

Dans le cas d'une erreur détectée, le paquet est corrigé et la transmission est acquittée. Dans le cas d'une détection de deux d'erreurs, deux solutions sont alors envisagées :

- Le routeur est déclaré définitivement fautif. Il est isolé du reste du réseau en activant de façon permanente ses connexions d'indisponibilité aux routeurs voisins.
- Une mémorisation des syndromes de *Hamming* (valeurs ***Vi*** et ***P***) et une demande de retransmission (*NACK*) est mise en œuvre. Si lors de cette seconde transmission du message, les mêmes syndromes sont obtenus, alors une erreur permanente est considérée et le routeur est déclaré fautif.

D) Conception VHDL d'un bloc de routage adaptatif : Modification de la logique de routage des routeurs

Lorsqu'un ou plusieurs nœuds ou routeurs (zone) sont déclarés fautifs, une solution consiste à mettre en œuvre un algorithme adaptatif de routage des paquets de données. L'objectif est le contournement de la zone fautive afin de maintenir la qualité de service du réseau tout en fiabilisant le réseau. On modifie alors la logique de routage des routeurs afin d'obtenir des acheminements adaptatifs des paquets de données. La solution qu'on propose est d'ajouter des interconnexions supplémentaires entre les routeurs indiquant un état fautif ou non. Si un routeur a ses indications *d'état fautif* activées à ses routeurs voisins, sa position est contournée selon des règles de routage à élaborer (voir exemple en figure III.13).

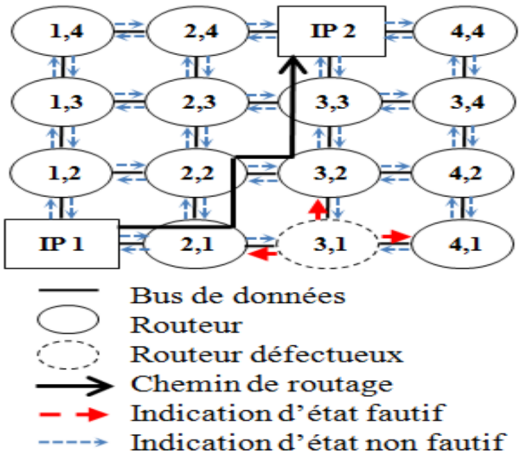


Figure III.13 Illustration d'une solution d'algorithme adaptatif.

Conclusion

Dans ce chapitre on a proposé une conception et implémentation VHDL de deux applications différentes: un modèle de gestion par batterie pour un système multi source comme premier exemple, pour le deuxième exemple concerne la conception d'un *NoC* tolérant aux fautes à travers la détection et la correction d'erreurs de transmission de paquets de données circulant dans un tel réseau. La conception et l'intégration sur carte FPGA d'éléments assurant une sûreté de fonctionnement à un système de communication sur puce ou un système de commande permet une prise de conscience par les concepteurs des phases de développement, méthodologies utilisées et de conception microélectronique de systèmes embarqués fiables.

Introduction

Dans cette partie on présentera les étapes suivies pour concevoir nos systèmes à savoir: la batterie dans le cadre d'une gestion optimisée d'énergie électrique pour un système multi sources et le routeur tolérant aux fautes pour un réseau sur puce. Ces systèmes seront vérifiés avant l'implémentation finale sur FPGA, à l'aide du logiciel XILINX ISE.

IV.1 Présentation de l'environnement de développement « XILINX ISE (Integrated Software Environment) » :

C'est un logiciel de programmation des produits XILINX (CPLD, FPGA, Spartan et Virtex…) téléchargeable gratuitement sur le site Internet de XILINX (www.xilinx.com).Le logiciel XILINX ISE est un environnement de développement qui possède différent outils de CAO. Les sociétés spécialisées en CAO fournissent des environnements logiciels spécialisés. Tous les fabricants de FPGA proposent des outils de CAO pour configurer leurs circuits.

Figure IV.01 Icône de XINLINX ISE 13.2_1.

L'offre logicielle dans le domaine de conception des circuits numériques est très variée parmi ces environnement, nous allons exploiter au cours de ce travail XILINX ISE qui est un logiciel multitâche de création et de gestion de projets CAO, possédant dans son soft deux manières différentes permettant la création des systèmes ou circuits numériques: textuelle ou graphique en vue d'une intégration dans un circuit logique programmable (CPLD ou FPGA).

XILINX ISE intègre différents outils permettant de passer à travers le flot de conception d'un numérique. Il dispose de :

- Un éditeur de textes, de schémas et diagramme d'états ;
- D'un compilateur VHDL et Verilog ;
- D'un simulateur ;
- D'outils pour la gestion des contraintes temporelles ;
- D'outils pour la synthèse ;
- D'outils pour la vérification ;

➢ D'outils pour l'implémentation sur FPGA et CPLD;

L'éditeur de texte ou entrée graphique est pour faire introduire la description dans le logiciel ou textuelle. La simulation du système est faite pour vérifier la validité du code avant-synthèse, après-synthèse et même après placement_routage. Les deux étapes synthèse et routage succéderont par la suite où la synthèse consiste à faire la transcription de la description d'une forme texte vers une autre graphique à base de portes logiques et pour la deuxième étape nommée routage, n'est qu'une adaptation du circuit logique synthétisé sur les ressources disponibles dans le circuit FPGA ciblé.

En double cliquant sur l'icône de XILINX présentée en figure IV.01, on aura la interface d'acceuil de XILINX ISE, qui permet d'accéder aux différents outils de conception que fournie ce logiciel.

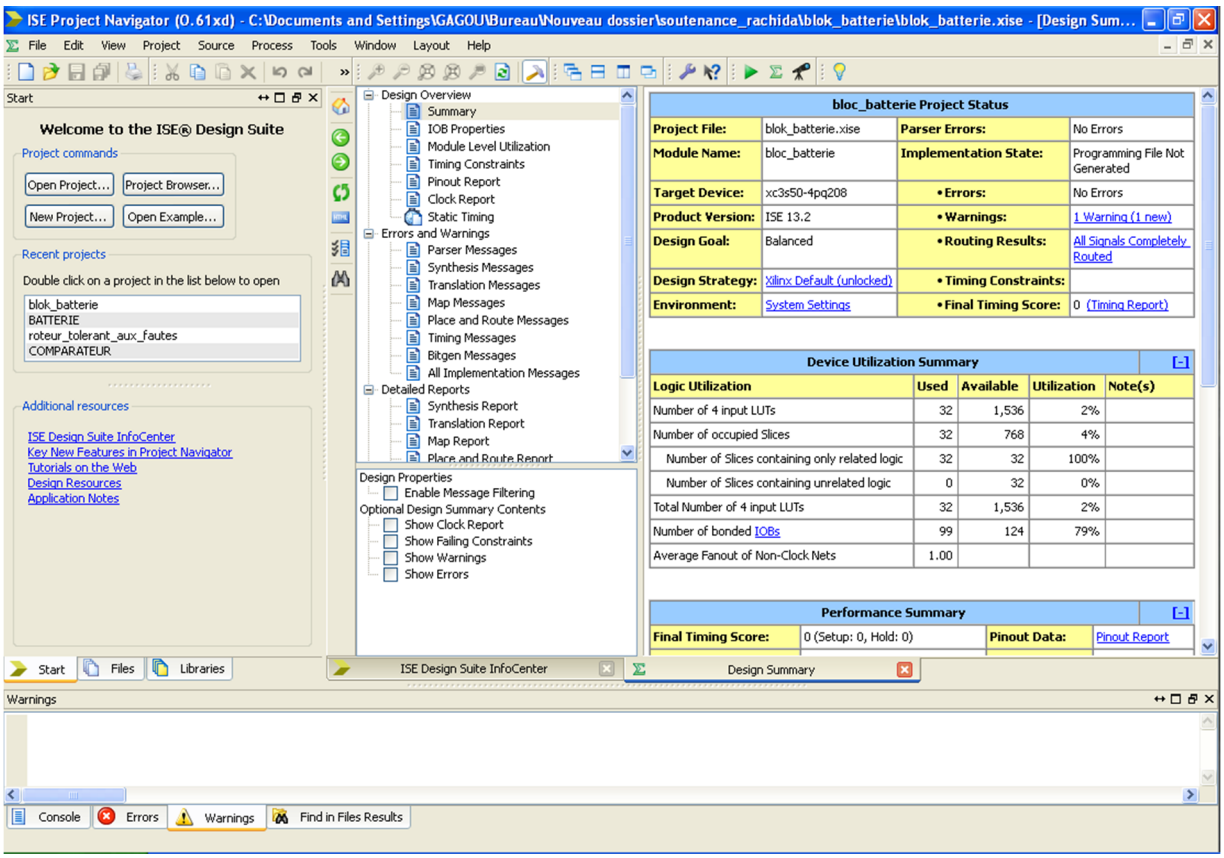


Figure IV.02 Interface d'acceuil de XILINX ISE 13.2_1.

Remarque : Les designs peuvent être décrits sous 3 formes principales, de façon schématique, par le langage de description matérielle (HDL) comme le VHDL et le Verilog ou alors par des diagrammes d'états.

IV.2 Implémentation et simulation des résultats des exemples d'application VHDL sous XILINX ISE :

IV.2.1 Premier exemple : modèle de gestion d'énergie électrique par batterie pour un système multi source :

IV.2.1.1 Spécification :

Dans cette étape on définit globalement le système à concevoir, à savoir les entrées, les sorties, et les entrées sorties, concernant notre système, les entrées sont : la tension de batterie *(U_Batterie)*, la tension de sortie du groupe électrogène *(GE)*, qui sont des signaux numérique sur 32 bits, et les sorties seront : des commandes (ON/OFF) de type bit (0/1) pour (*décharge 1)* , (*décharge 2)* et une commande manipulant l'allumage du GE (cmd_GE), ainsi que la demande de consommation *(demande)* : un signal numérique sur 32 bits. Pour réaliser cela il faut suivre les étapes suivantes :

1) Création d'un projet

Un projet permet de regrouper plusieurs fichiers-sources pour un laboratoire ou un module en particulier. Après avoir lancé le programme, on fait ***File*** puis ***New Project*** puis on choisit un nom représentatif pour notre projet : blok_batterie (figure IV.03). Il faut aussi spécifier un répertoire où le projet sera sauvegardé. Il est important que le chemin de ce répertoire ne contienne pas d'espaces, parce que certains outils invoqués peuvent ne pas les accepter. Conservez le « Top-level source type » en ***HDL*** et cliquez sur ***Next***.

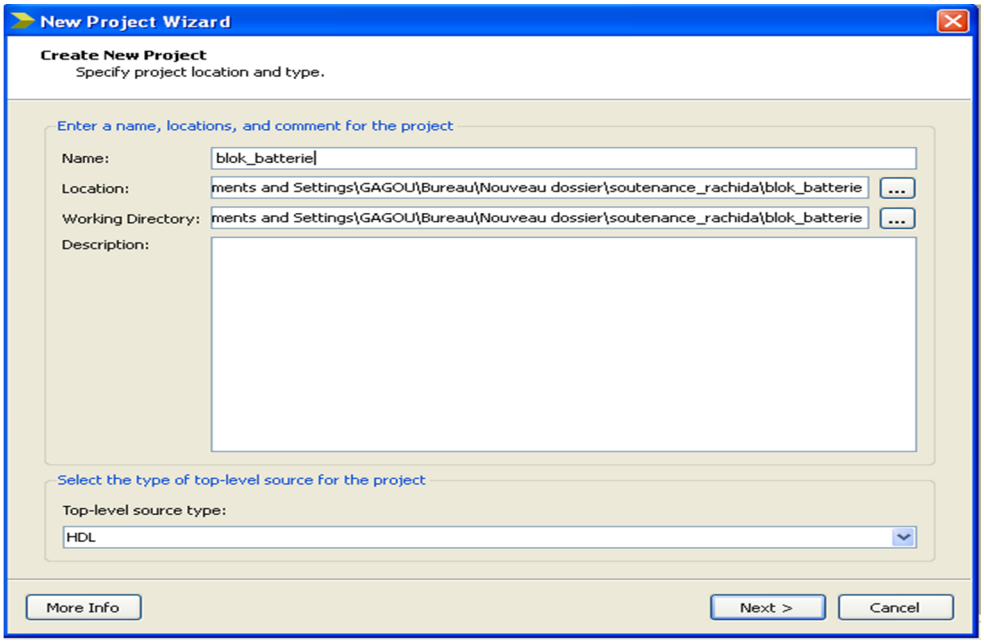


Figure IV.03 Création d'un projet VHDL.

On doit remplir certains champs correspondants aux différents outils et FPGA utilisés. Les choix reproduits sont à la figure IV.30, puis cliquez sur ***Next***, puis sur ***Finish***.

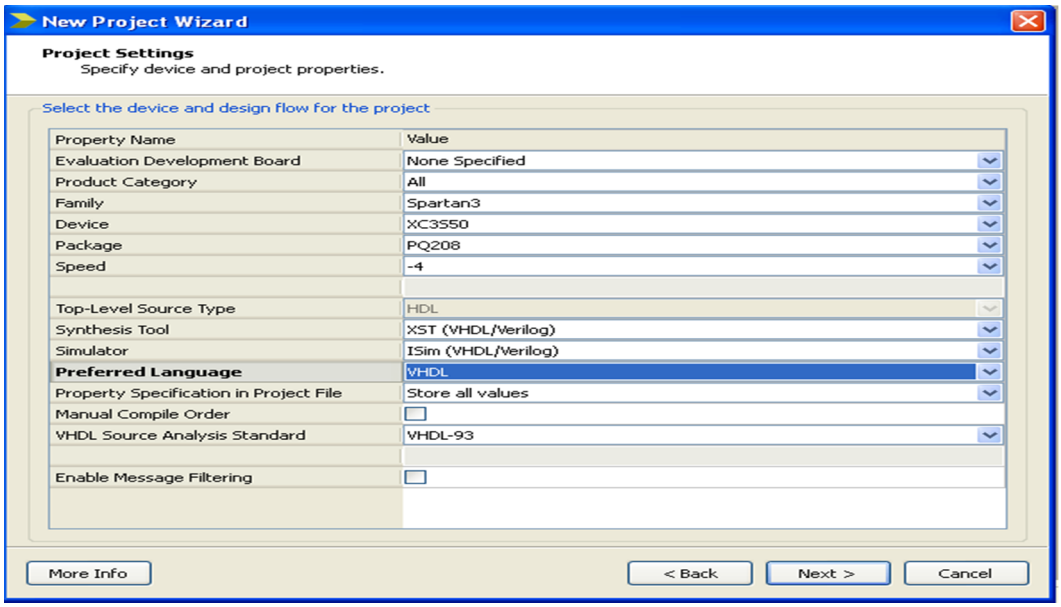


Figure IV.04 Remplissage des champs correspondants aux différents outils et FPGA utilisés.

2) Création d'un fichier source

Dans la fenêtre de Design, on peut voir notre projet. Afin d'y ajouter des fichiers, Faites un clique droit sur le projet et cliquez sur ***New Source…***

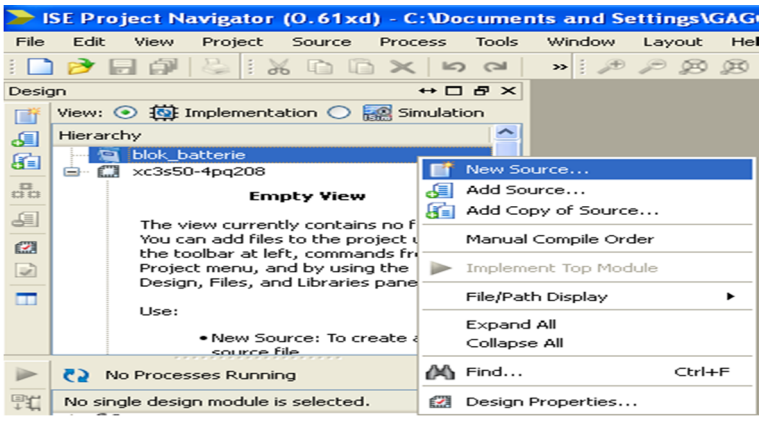


Figure IV.05 Création d'un fichier source.

Dans la fenêtre qui apparait, on choisit comme type ***VHDL Module*** et on donne un nom représentatif au fichier (ici ***bloc_batterie***). On clique sur ***Next***.

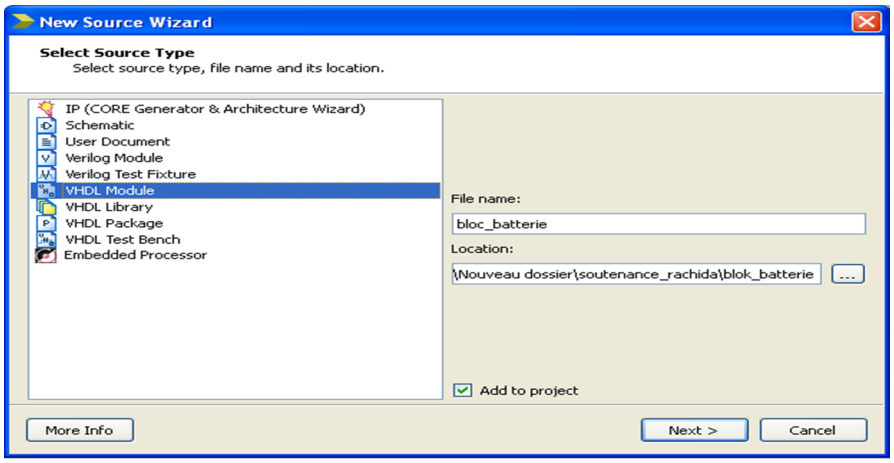


Figure IV.06 Spécification du langage VHDL pour le fichier source : blok_batterie.

Nous pouvons maintenant donner les entrées et sorties de notre module. On click sur ***Next***, puis ***Finish*.**

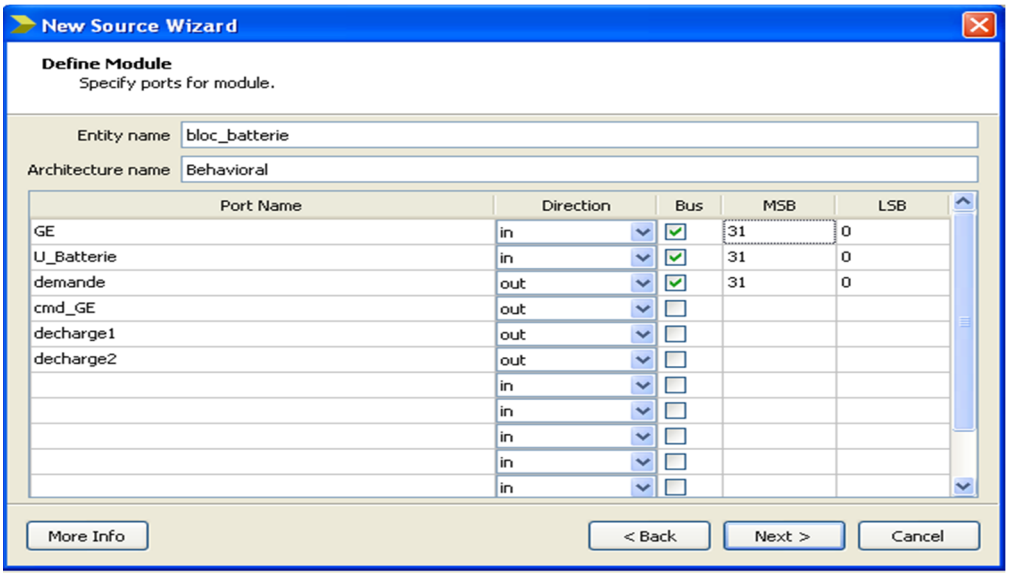


Figure IV.07 Remplissage des champs correspondants aux entrées, sorties et entrées-sorties du blok_batterie.

On verra alors un nouveau fichier VHDL dans notre projet qui comporte déjà un début de structure pour notre description matérielle. Là on est dans la phase ***implémentation :***

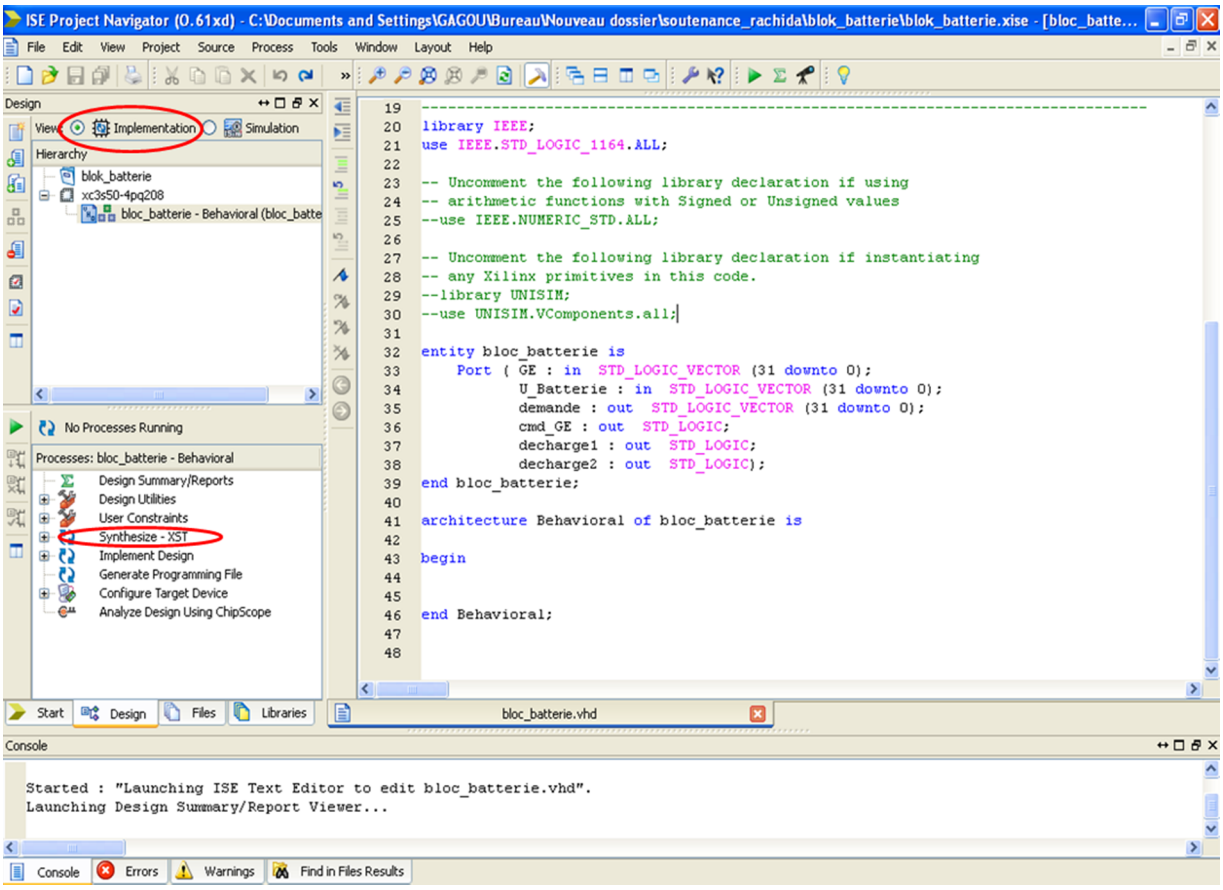


Figure IV.08 Visualisation du fichier blok_batterie résultant.

Dans la partie ***Process***, on click sur le signe (+) de ***Synthesize –XST***, on double click sur ***View Technology Schematic,*** on aura la figure suivante :

- *Synthesize –XST* : Permet de faire la synthèse du circuit en bloc configurable se retrouvant sur le FPGA.

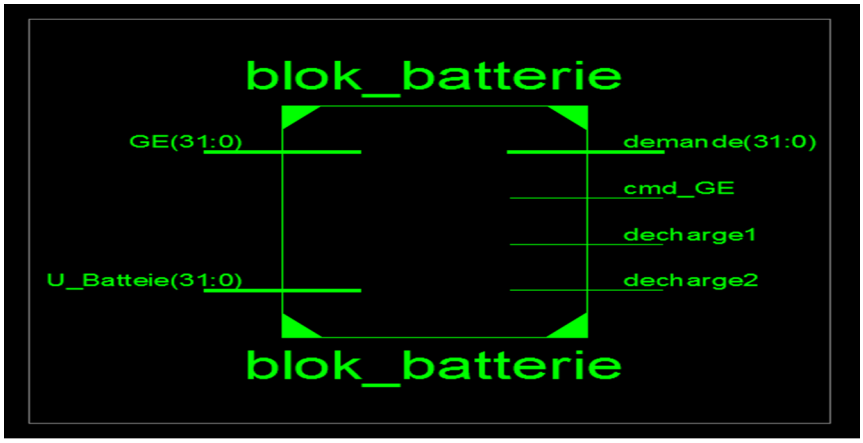


Figure IV.09 Vue d'ensemble du modèle de gestion blok_batterie.

IV.2.1.2 Analyse fonctionnelle (sur papier) : Le fonctionnement de blok_batterie est décrit dans le chapitre III (§ III.3.1.2).

IV.2.1.3 Description VHDL : C'est l'implémentation du l'algorithme descriptif des fonctionnalités du système à concevoir, (le code complet se trouve dans les annexes).

IV.2.1.4 Synthèse : Détaillons un peu dans le système pour visualiser le circuit obtenu, suivant la figure IV.10, pour le faire : on double click sur le schéma présenté en figure IV.09 avant, puis sur quelques blocs apparaissant pour visualiser les portes logiques employées.

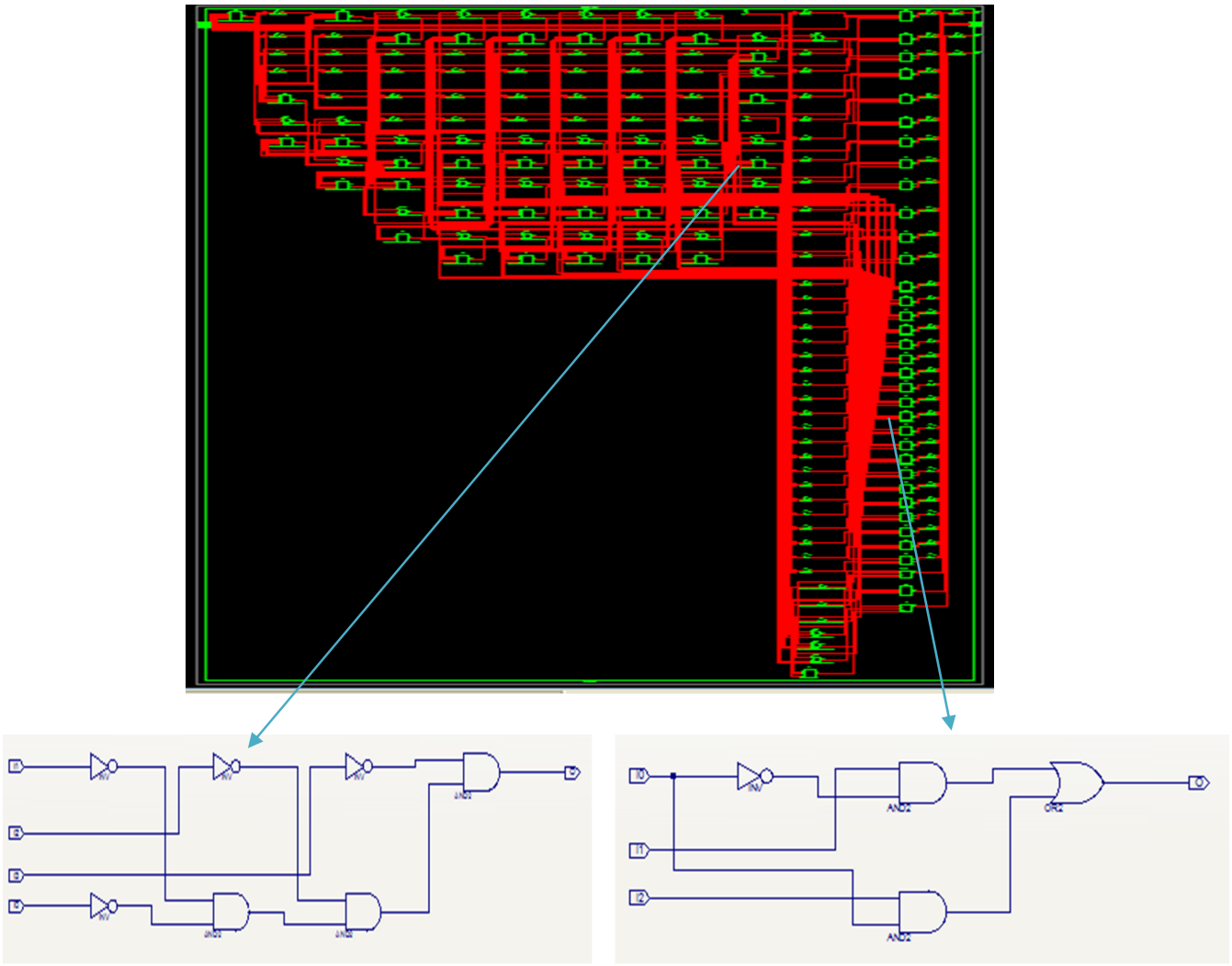


Figure IV.10 Synthèse du modèle de gestion blok_batterie.

ℜ **Simulation fonctionnelle :** Cette partie concerne la visualisation des résultats de la simulation.

D'abord if faut passer au mode ***simulation***, et créer un fichier de test (***tesbench***) pour le blok_batterie de même façon que la création d'un fichier source déjà vue. Ensuite en donne des valeurs de test (*stimuli*) pour vérifier le fonctionnement du l'algorithme implémenté. Puis on double click sur ***Simulate Behavioral Model***, comme c'est mentionné dans la figure suivante :

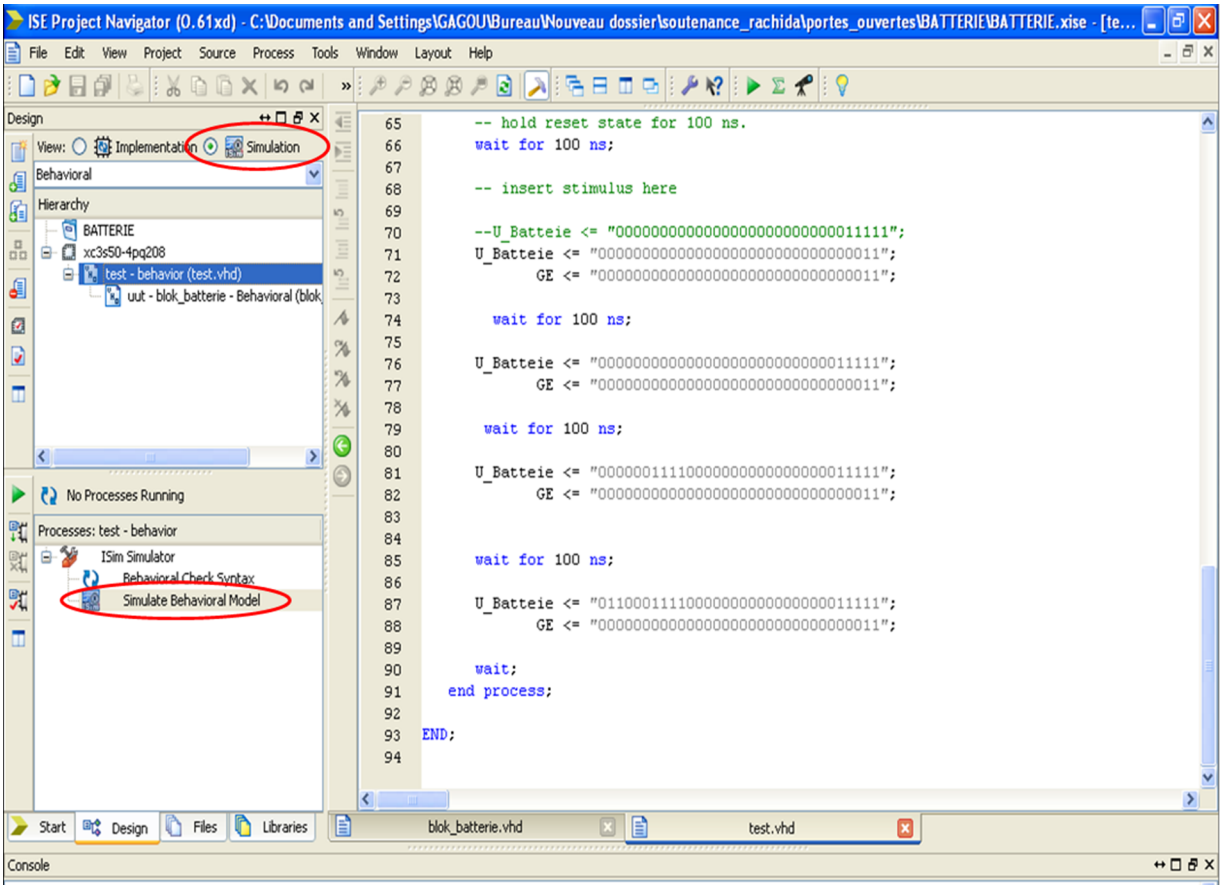


Figure IV.11 Les valeurs de test sous le fichier testbench.

On remarque que le chronogramme est subdivisé en deux parties : la première en haut représente les entrées/sorties sous forme de bus codé en binaire (par exemple la figure IV.12), la deuxième est la partie détaillée de bus de sortie encodé dans notre cas sur 32 bits (représentée par la figure IV.13), c'est nous qui avons ouvert le bus de sortie pour expliciter les résultats.

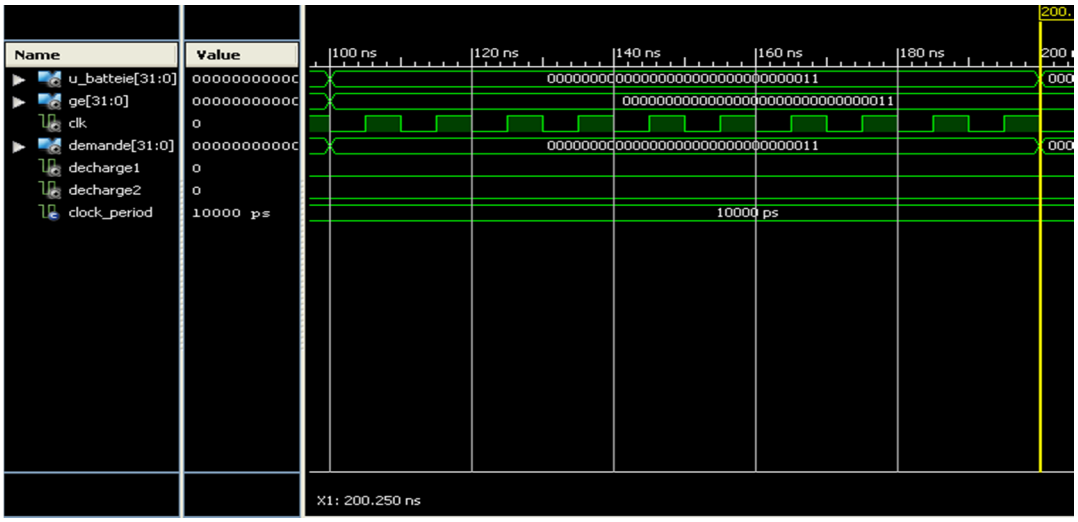


Figure IV.12 Premier cas : U_Batterie < Vmin.

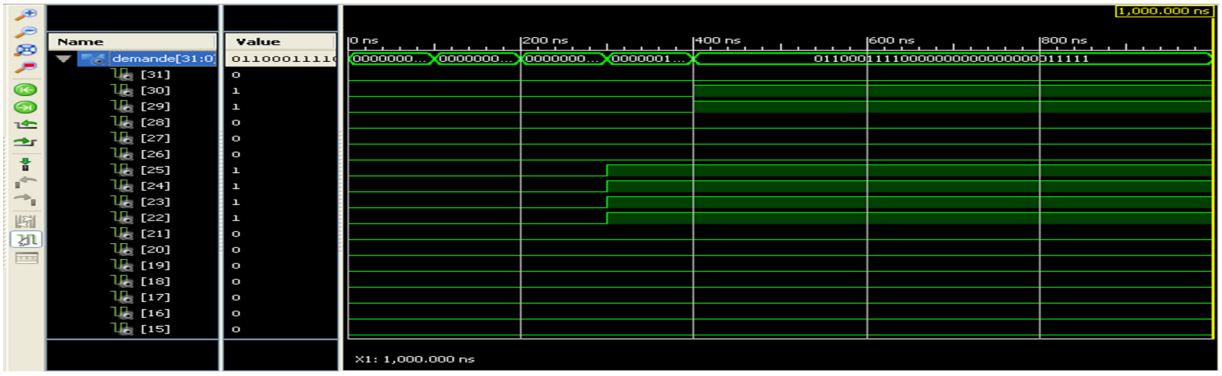


Figure IV.13 La partie détaillée du chronogramme.

Interprétation des résultats :

Nous avons testé le blok_batterie en donnant des valeurs qui peuvent représenter presque tous les cas de possible que peut rencontrer le système. Pour la figure IV.13: nous avons les valeurs suivantes (notons que les valeurs proposées sont en volt):

Les valeurs fixes :

Vmin = 00000000000000000000000000001111.

Vmax = 00000000000000000011111111111111.

deux_Vmax = 00110000000000000011111111111111.

GE = 00000000000000000000000000000011.

Les valeurs variables :

Après un temps de 100ns (par wait for 100 ns):

U_Batteie = 00000000000000000000000000000011.

On a U_Batteie < Vmin, d'après l'algorithme la demande d'alimentation sera fournie par le groupe électrogène (ge) et les deux décharges (decharge1, decharge2) sont éteintes et représentées par' 0'.

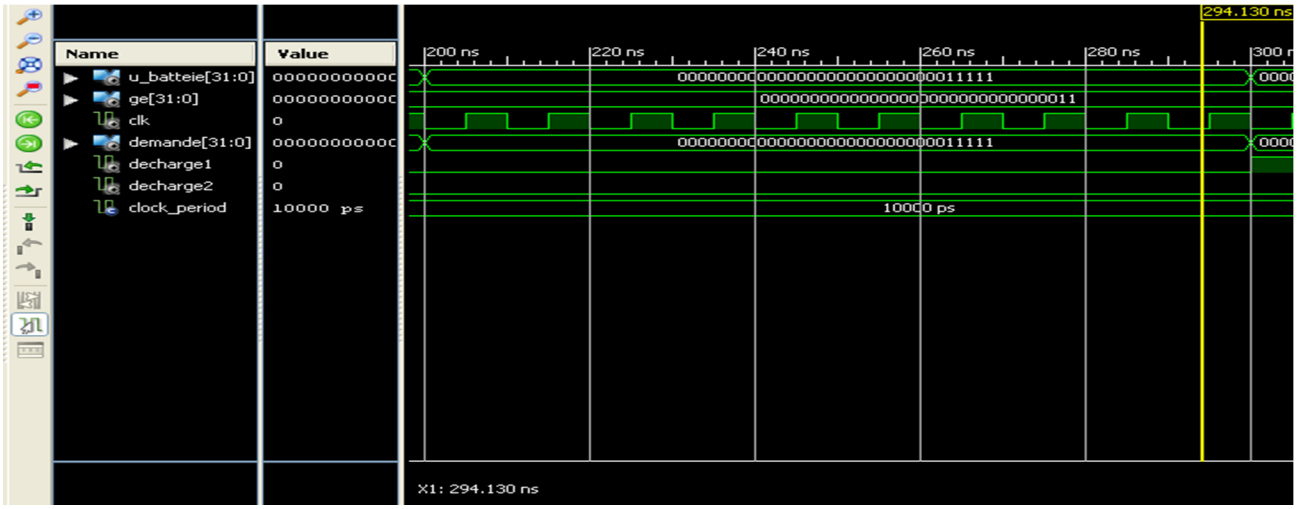


Figure IV.14 Deuxième cas : Vmin < U_Batterie < Vmax.

Interprétation des résultats :

Pour la figure IV.14: nous avons les valeurs suivantes :

Les valeurs variables :

wait for 100 ns: donc après 200ns

U_Batteie = 00000000000000000000000000011111.

On a Vmin < U_Batterie < Vmax, donc la demande d'alimentation sera fournie par U_Batteie et les deux décharges (decharge1, decharge2) sont toujours éteintes et représentées par ' 0'.

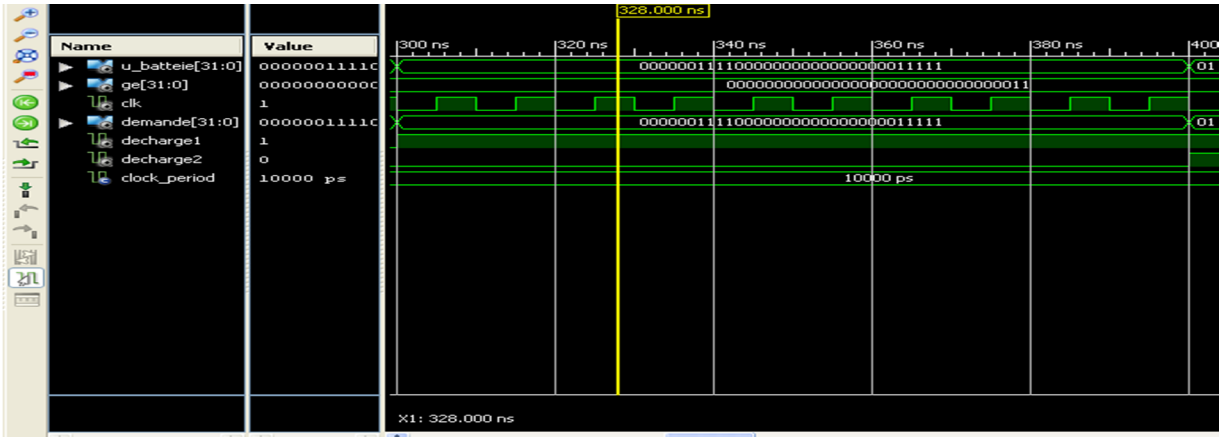


Figure IV.15 Troisième cas : U_Batterie > Vmax.

Interprétation des résultats :

Pour la figure IV.15: nous avons les valeurs suivantes :

Les valeurs variables :

wait for 100 ns: donc après 300ns

U_Batteie = 00000000000000000000000000011111.

On a U_Batterie > Vmax, donc la demande d'alimentation sera toujours fournie par U_Batteie avec déclanchement de la première décharge (decharge1) et représentée par '1'(Fran montant), pour la deuxième (decharge2) elle reste éteinte et représentées par' 0'.

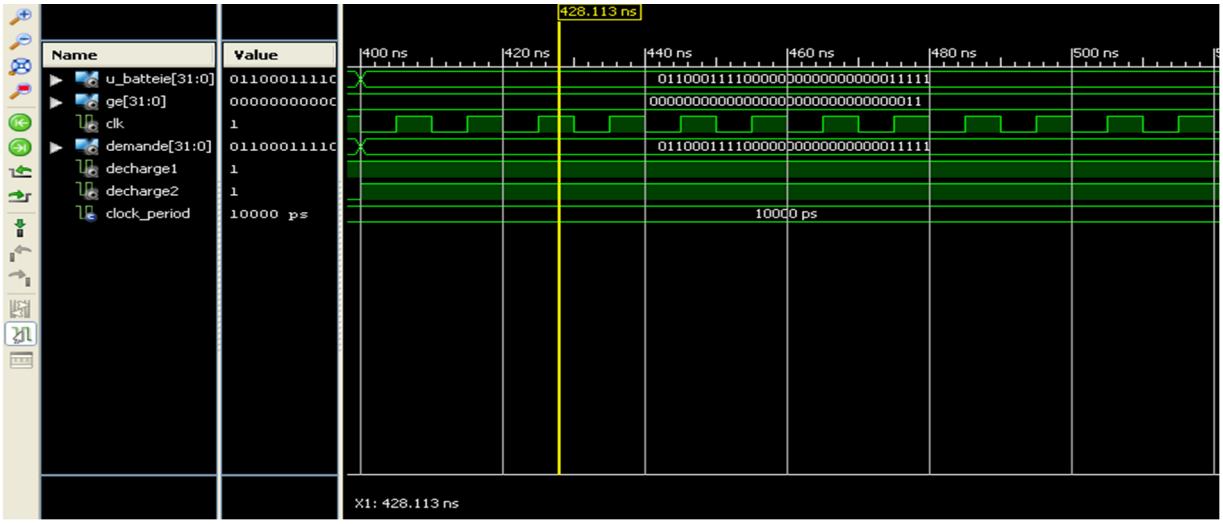


Figure IV.16 Dernier cas : U_Batterie > deux_Vmax.

Interprétation des résultats :

Pour la figure IV.16 : nous avons les valeurs suivantes :

Les valeurs variables :

wait for 100 ns: donc après 400ns

U_Batteie = 01100011110000000000000000011111.

On a U_Batterie > Vmax , donc la demande d'alimentation sera toujours fournie par U_Batteie avec déclanchement des deux décharges (decharge1, decharge2) et représentées par '1'.

IV.2.1.5 Implémentation :

Après la synthèse et la simulation de notre circuit en vue d'une implémentation sur le FPGA. Pour ce faire, il faudra utiliser la partie inférieure de l'onglet Design disponible dans ISE, que voici :

Pour réaliser les options, *Implement Design* et *Generate Programming File*, il faut faire un clique-droit sur chacun, puis *Run* (ou double click).

- *Implement Design* : Permet de faire les *mapping* et routage nécessaires pour placer le tout sur le FPGA.

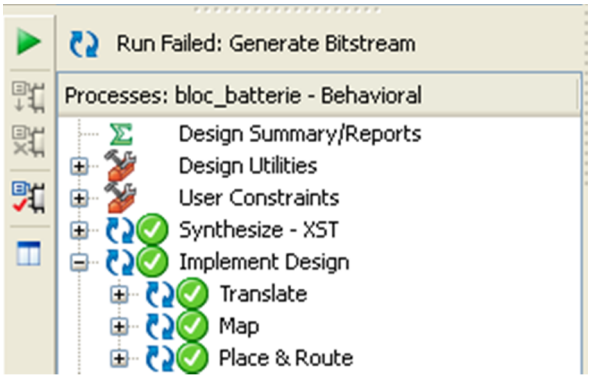


Figure IV.17 Validation de l'étape d'implémentation du modèle de gestion blok_batterie.

- *Generate Programming File* **:** Génère le fichier (.bit) utilisé pour programmer le FPGA.

Après chacune des étapes, un message de succès devrait s'afficher dans la console. Dans le cas contraire, il faut corriger les erreurs ou inspecter les avertissements (warnings) pour vous assurer qu'ils ne proviennent pas d'erreurs dans votre code.

Une fois que toutes les étapes de «compilation» sont réussies, un fichier *blok_batterie.bit* sera généré et placé dans le répertoire du projet et est prêt à être chargé sur le FPGA de la planchette de développement .

IV.2.1.6 Configuration : c'est la dernière étape du projet, la figure IV.18 montre les équipements nécessaires pour réaliser cette étape (au laboratoire). Pour le faire voici la procédure à suivre sous XINLINX ISE.

Dans Xilinx ISE, on clique deux fois sur ***Configure Target Device***. Lorsque l'outil ***iMPACT*** sera ouvert, il faut créer un projet en faisant ***File > New Project***. Si la fenêtre « A*utomatically create and save a project »* s'affiche, on clique sur ***No***. On choisit ***create a new project***, puis ***OK***, il faut s'assurer de mettre l'option de branchement automatique avant de cliquer de nouveau sur ***OK***. Si la fenêtre « *Auto Assign Configuration Files Query Dialog* » s'affiche, on clique sur **No**. Dans l'espace en blanc,on fait un click droit et on choisit ***Initialize Chain***, puis **Yes** si la fenêtre « *Auto Assign Configuration Files Query Dialog* » s'affiche. On choisit le fichier ***blok_batterie.bit*** qu'on trouvera dans le dossier projet et on clique sur ***Open***. Si le logiciel vous demande d'attacher un *SPI* ou un *BPI PROM*, répondez par ***No***. Cette option sert à charger la mémoire *PROM* de la planchette. Cliquez sur le symbole représentant le FPGA, puis un click droit : ***Program …*** puis sur ***OK*** dans la fenêtre *Device* ***Programming Properties***, sans rien changer.

Voilà! Vous pouvez maintenant vérifier votre design sur le FPGA, en essayant toutes les combinaisons possibles d'entrées et confirmer que les sorties sont bien conformes aux spécifications;

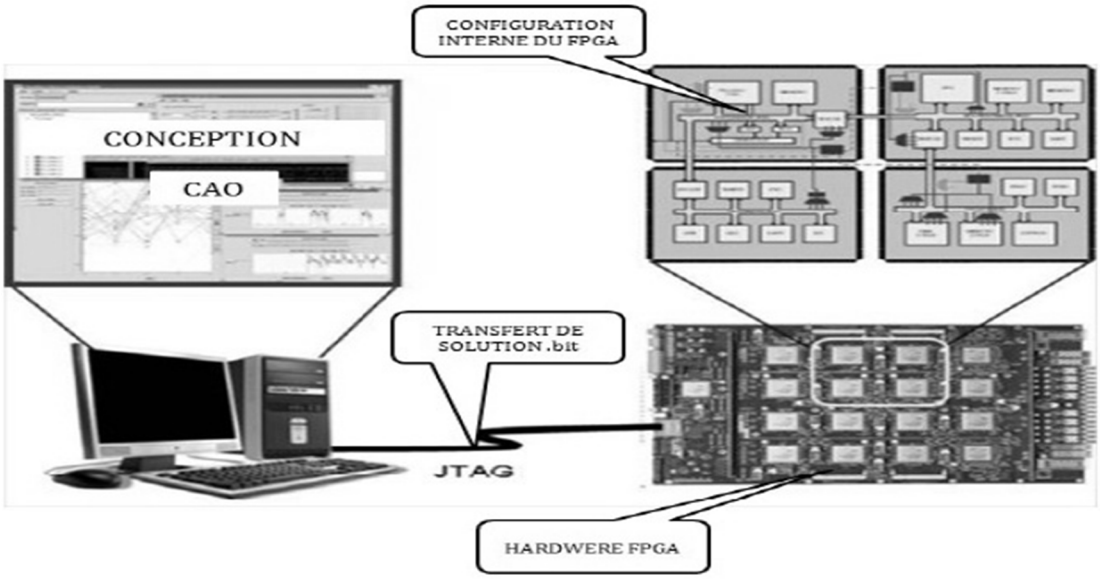


Figure IV.18 Configuration du FPGA par un câble JTAG.

IV.2.2 Deuxième exemple : Conception et implémentation d'un routeur tolérant aux fautes pour un réseau de communication sur puce NoC

IV.2.2.1 Spécification :

La figure IV.19 définit globalement le routeur à concevoir donc : d'abord le nom du projet est *routeur_NoC*, pour les entrées on a : le paquet de données (*paquet*), sur 8 bits, et la réception de l'acquittement (*reçu_ACK*) sur 1 bit qui sont des signaux numérique, et les sorties seront : le paquet émis (*paquet_emis*), sur 8 bits, les quatre directions : (*North*, *South*, *East* et *West)* plus l'envoi d'acquittement (*envoi_ACK*) ainsi que l'indication d'état fautif en cas d'une erreur pas corrigée (*etat_fautif*), qui sont sur 1 bit, sont des signaux numériques. Pour réaliser cela il faut suivre les étapes décrites en (§ IV.2.1.1). De même pour toutes les autres étapes.

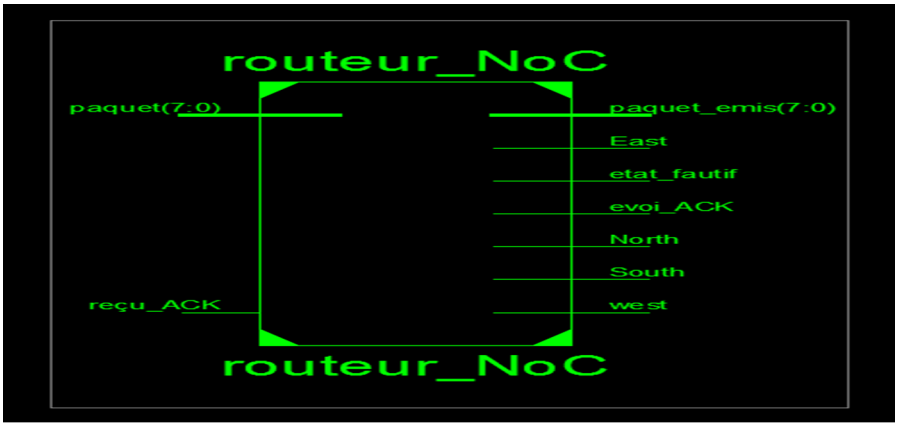


Figure IV.19 Vue d'ensemble du routeur_NoC.

IV.2.2.2 Analyse fonctionnelle (sur papier) : Le fonctionnement du *routeur_NoC* est décrit dans le schéma suivant :

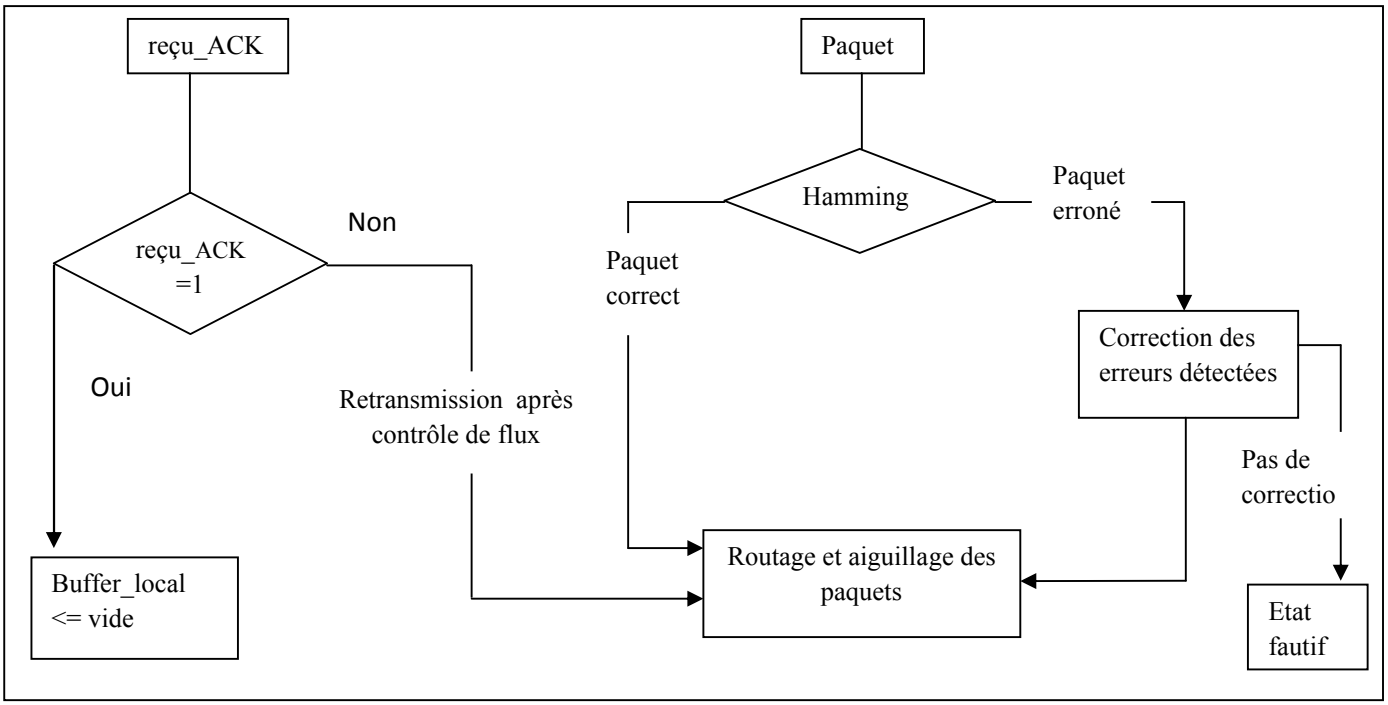


Figure IV.20 Principe de fonctionnement d'un routeur *routeur_NoC,* tolérant aux fautes sur un NoC.

Indication :

Quand un routeur (dans notre cas c'est le routeur d'adresse (x= 10, y=01)), reçoit un paquet de données, il lui applique le codage de Hamming pour détecter et corriger les erreurs, ou indication d'état fautif (le routeur est en état fautif) pour le reste du réseau, dans le cas contraire. Puis définit le chemin convenable pour le paquet traité selon l'algorithme de routage X-Y, déjà définit. Le routeur assure aussi un contrôle de flux de type ACK/NACK, concernant l'envoi et la réception de données via le réseau.

IV.2.2.3 Description VHDL : C'est l'implémentation du l'algorithme descriptif des fonctionnalités du routeur tolérant aux fautes. (Le code complet se trouve dans les annexes).

IV.2.2.4 Synthèse : Le circuit obtenu à cette étape est présenté comme suit:

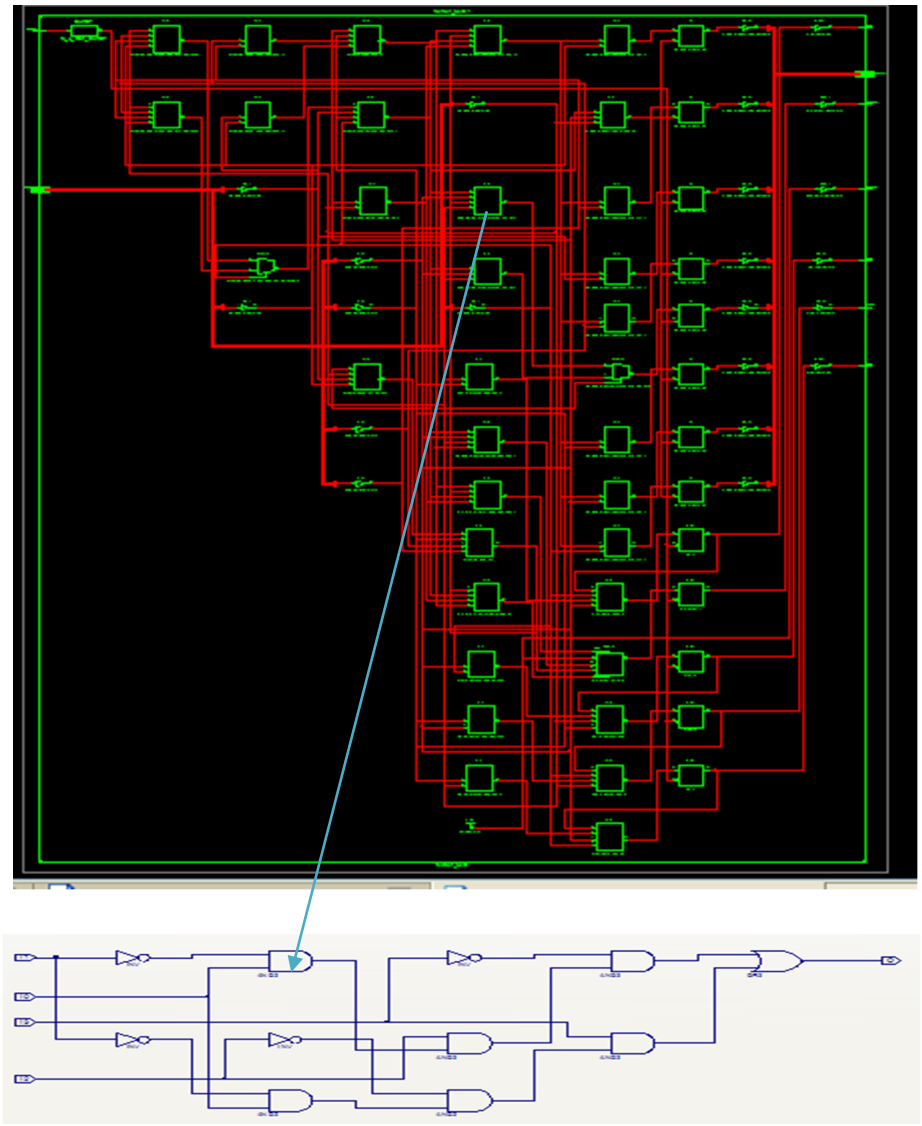


Figure IV.21 Synthèse du *routeur_NoC*.

ℜ Simulation fonctionnelle

D'abord il faut passer au mode ***simulation***, et créer un fichier de test (***tesbench***) pour le routeur_NoC, de même façon que la création d'un fichier test du blok_batterie déjà vue. Ensuite en donne des valeurs de test (*stimuli*) pour vérifier le fonctionnement du l'algorithme implémenté. Voici le résumé des tests effectués :

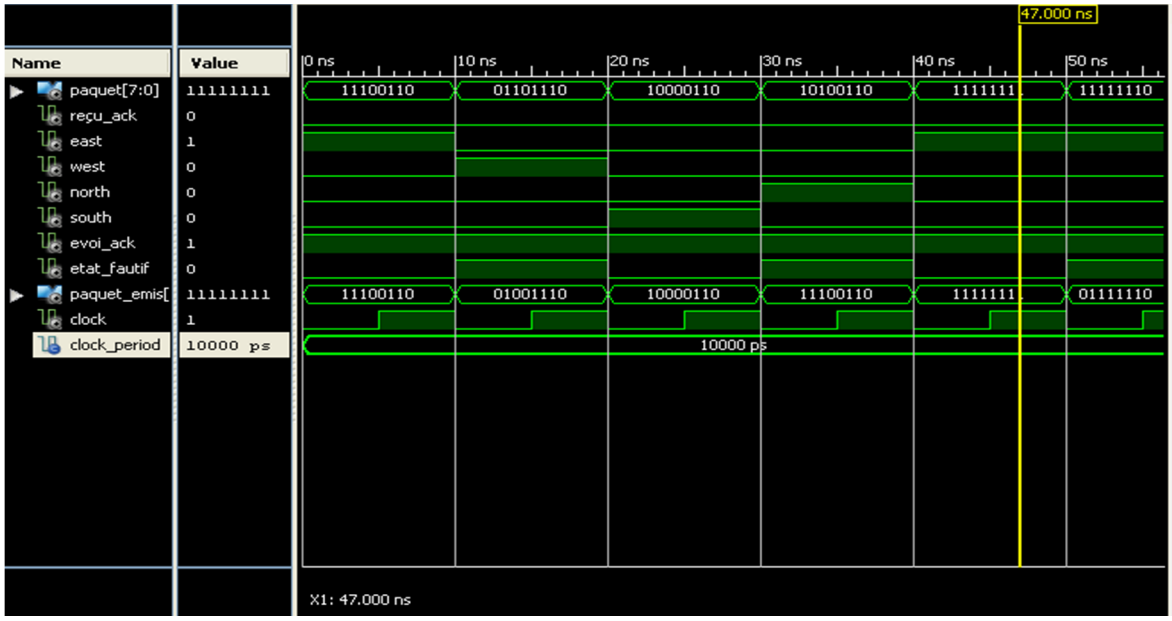


Figure IV.22 Résultats de la simulation du routeur_NoC.

Interprétation des résultats :

Supposons que notre routeur a reçu le paquet ' 11100110', le routeur envoi un ACK (*envoi_ACK*=1), nous remarquons qu'il y a pas d'erreurs (*etat_fautif* ='0'), la direction du paquet reçu d'après l'algorithme de routage sera *East (East*='1'). Le paquet émis sera le même reçu (figure IV.23).

Figure IV.23 Premier paquet.

Dans le deuxième cas (figure IV.24), il reçoit le paquet '01101110', ici une erreur est survenue, la correction des erreurs est faite d'après le codage de Hamming, donc le paquet émis sera '01001110' càd on a inversé le bit de 5eme position (v= 0101).

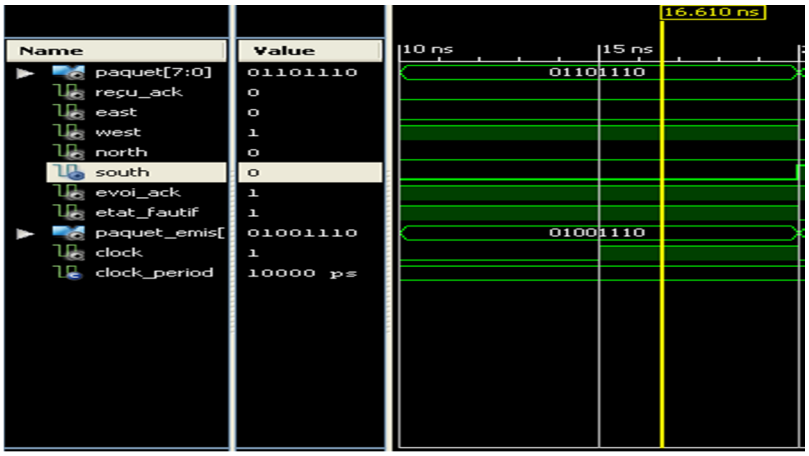


Figure IV.24 Deuxième paquet.

Le routeur reçoit le paquet de données ‘ 10000110’, la direction du paquet sera *North* (*north*=1), sans erreurs détectées.

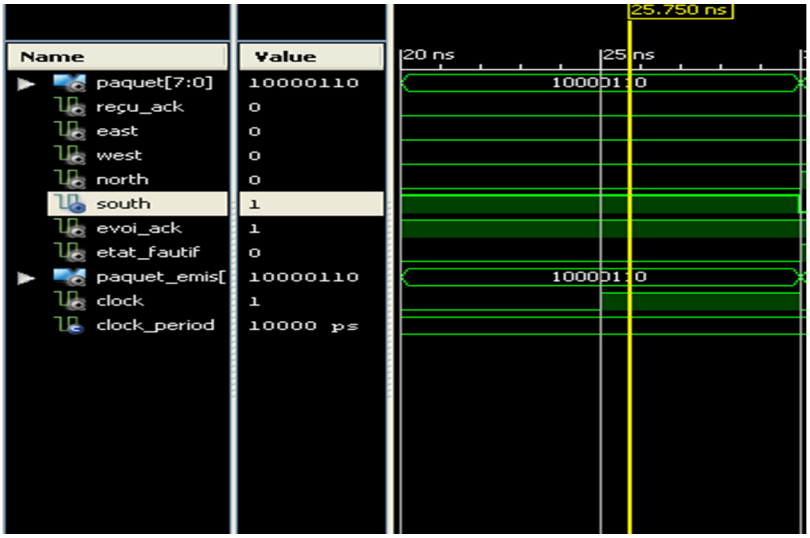


Figure IV.25 Troisième paquet

IV.2.1.5 Implémentation : on suit les mêmes instructions déjà présentés en (§IV.2.1.5).o aura la figure suivante :

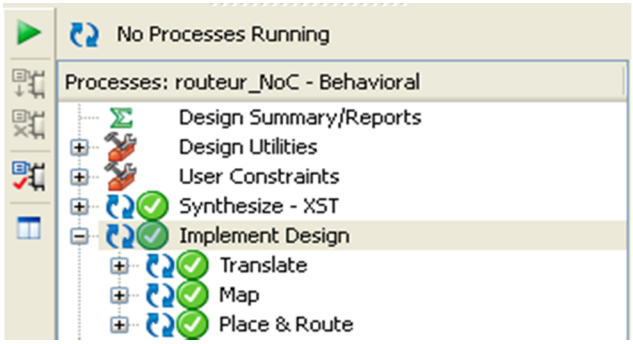


Figure IV.26 Validation de l'étape d'implémentation du routeur_NoC.

IV.2.1.6 Configuration : De même que pour le blok_batterie, c'est une étape qui se fait au laboratoire.

Conclusion

Dans ce chapitre, on a exposé les résultats obtenus des deux applications réalisées sous VHDL, a savoir : batterie dans le cadre d'une gestion optimisée d'énergie électrique pour un système multi sources et le routeur tolérant aux fautes pour un réseau sur puce.

Les résultats obtenus sont satisfaisant, les deux applications répondent aux objectifs fixés.

Conclusion générale

Nous constatons que la recherche dans le domaine de conception des systèmes intégrés implémentables sur les circuits numériques à architecture reconfigurable FPGA, est complexe car elle nécessite non seulement une maitrise des technologies relatives aux FPGA, mais aussi une très bonne connaissance des applications et de leurs environnements ainsi que les outils de CAO utilisés.

Durant le processus de conception nous avons constaté que malgré les progrès réalisés dans le domaine des architectures reconfigurables, les algorithmes de programmation matérielle s'écartent significativement des algorithmes de programmation logicielle. Nous avons démontrée dans ce mémoire la faisabilité d'implantation numérique câblé des systèmes de commande par un informaticien en combinaison avec un électronicien.

Dans le premier chapitre de ce mémoire, nous avons présenté une description générale des langages de programmation tout en particulier les langages de description matérielle HDL, ainsi que leur évolution au cours des années. Au deuxième chapitre, nous avons choisi un parmi ces langages qui est le VHDL, pour motiver notre choix, on a fait une comparaison entre le VHDL et Verilog, puis on s'est introduis au langage de programmation VHDL. Au troisième chapitre, nous avons présenté deux exemples d'applications de VHDL pour deux domaines différents : l'électronique : sur les systèmes de commande, dont on a choisi la gestion d'énergie électrique par batterie pour un système multi sources (uniquement une partie du système). Le deuxième domaine concerne les réseaux de communication sur puce NoC, dont on a proposé un modèle de routeur tolérant aux fautes vise à détecter et corriger les erreurs de transmission de données via un tel réseau. Comme on a développé une méthodologie de conception qui a la démarche théorique suivante : spécification du cahier de charge avec une étude théorique, conception architecturale détaillée, test, intégration et validation. Dans le quatrième chapitre, nous avons présenté les architectures détaillées des deux systèmes : la batterie et le routeur, puis on a visualisé les résultats de la simulation sous l'outil de développement XILINX ISE (version 13.2).

Finalement, l'approche traitée au cours de ce travail peut être avantageusement améliorée et facilement étendue sur d'autre application et les chercheurs du domaine vont donc devoir relever des défis encore plus importants.

Perspectives

Dans le cadre des systèmes à hautes performances, ce travail présente une contribution à la conception et à l'implantation de commande FPGA. Plusieurs travaux peuvent venir compléter ce travail qui a été fait jusqu'ici et particulièrement de prendre en compte les différents points qui n'ont pas été abordés dans ce manuscrit. Malgré l'importance quantitative des travaux scientifiques, et malgré les immenses progrès réalisés dans les architectures des réseaux sur puce NoC, de nombreuses problématiques restent à explorer. A cet effet, nous envisagerons comme perspectives, quelques axes de recherche pour améliorer les performances de conception des routeurs tolérants aux fautes pour les réseaux NoC. Ces axes sont récapitulés comme suit :

- Spécification des types des erreurs détectées à savoir les erreurs permanente, transitoire…etc. puis les corrigées selon leur types ;
- Une gestion optimisée en termes de performances des états fautifs du réseau ;
- L'implémentation des algorithmes sur FPGA, dans un laboratoire équipé, qui permet d'avoir le produit finale que présente le projet.

GLOSSAIRE

ASIC : littéralement « circuit intégré propre à une application » est un circuit intégré spécialisé. En général, il regroupe un grand nombre de fonctionnalités uniques et/ou sur mesure.

ADA : un langage de programmation orienté objet dont les premières versions remontent au début des années 1980, conçu par l'équipe de CII-Honeywell Bull en réponse à un cahier des charges établi par le département de la Défense des États-Unis (DoD).

Bibliothèque: un regroupement d'un ensemble d'éléments destinés à être réutilisés dans un développement ultérieur.

Câble JTAG : Groupe mixte d'action de test (JTAG) est le nom commun pour ce qui était plus tard normalisé en tant que IEEE 1149.1 d'accès de test standard et Boundary-Scan Architecture. Il a été initialement conçu pour tester les cartes de circuits imprimés en utilisant boundary scan et il est encore largement utilisé pour cette application .Aujourd'hui JTAG est également largement utilisée pour les ports de débogage de circuits intégrés.

CMOS : Complementary Metal-oxyde-semiconducteur : est une technologie pour construire des circuits intégrés. La technologie CMOS est utilisée dans les microprocesseurs, de microcontrôleurs, de mémoire RAM statique, et d'autres circuits logiques numériques. La technologie CMOS est également utilisée pour plusieurs circuits analogiques tels que les capteurs d'image (capteur CMOS), les convertisseurs de données, et émetteurs-récepteurs hautement intégrées pour de nombreux types de communication.

DSP : (Digital Signal Processor, en français « processeur de signal numérique ») est un microprocesseur optimisé pour les calculs mais aussi pour accéder très facilement à un grand nombre d'entrées-sorties (numériques ou analogiques). Son application principale est le traitement numérique du signal (filtrage, extraction de signaux, etc.).

ECAD : basée à Santa Clara, en Californie, était un fournisseur de logiciels de conception au début automatisation électronique. La société a été mieux connue pour son contrôle des règles de conception Dracula produit, mais aussi mise en page produit IC et logiciels de configuration PC. Finalement, dans une fusion avec SDA, il est devenu de Cadence Design Systems.

GLOSSAIRE

IP : La propriété intellectuelle (IP) est un terme controversé se référant à un certain nombre de types distincts de créations de l'esprit pour lequel un ensemble de droits sont reconnus en vertu des champs correspondants de la loi.

Microélectronique : est une spécialité du domaine de l'électronique qui s'intéresse à l'étude et à la fabrication de composants électroniques.

PLD : Un circuit logique programmable, ou réseau logique programmable, est un circuit intégré logique qui peut être reprogrammé après sa fabrication.

CPLD : un type de circuit complexe programmable ;

FPGA : (Field-Programmable Gate Array, en français un réseau de portes programmables in-situ) est un circuit intégré, constitué d'un réseau de cellules logiques élémentaires, librement assemblables, pouvant être reprogrammé après sa fabrication. Ce composant permet un prototypage aisé. De plus les FPGA permettent de plus en plus d'intégrer, sont de plus en plus rapide. Ils ont suffisamment de capacité et de rapidité pour permettre une approche multiprocesseur.

SoPC: « System on Programmable Chip» (système sur puce reprogrammable en français), désigne un système complet embarqué sur une puce reprogrammable de type FPGA, pouvant comprendre de la mémoire (data / code), un ou plusieurs processeurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue.

Testbench: Un banc d'essai est un environnement virtuel utilisé pour vérifier l'exactitude ou la solidité d'un dessin ou modèle (par exemple, un produit logiciel). Le terme a ses racines dans les essais des appareils électroniques, où un ingénieur serait assis à une table de laboratoire avec des outils de mesure et de la manipulation, tels que des oscilloscopes, des multimètres, des fers à souder, des coupe-fils, et ainsi de suite, et manuellement vérifier l'exactitude du dispositif sous test.

A

[01] A. BOUZEROURA & R. ALKAMA, « optimisation d'une alimentation d'énergie électrique multi-sources pour village isolé », communication, Université de Bejaia, février 2012.

[02] A. IGUERMIA & B. NBAHEDDA : « VHDL-AMS : un atout pour la conception des systèmes microélectroniques analogiques – numériques », cours pédagogique**,** École Polytechnique de Montréal, 1999-2000.

B

[03] B.S. BORROWY & Z.M. SALAMEH: « Methodology for Optimally Sizing the Combination Battery Bank and PV Array in a Wind/PV Hybrid System », IEEE Transaction on Energy Conversion, Vol. 12, N°1, pp. 73 - 78, March 1997.

C

[04] C. C. BRAHAM, J.WEIXING, Z.LINGYU: «The Design and Implementation of a Hierarchical Network on Chip », Procedia Engineering 29: 2966 – 2974, 2012.

[05] C. COMETE : « Co-design, conception conjointe logiciel-matériel », Editions Eyrolles, France, 1998.

[06] C. GUILLEMINOT : « Intégration numérique d'un système multicapteurs amrc de télécommunication basé sur le langage vhdl-ams », cours pédagogiques, École Nationale Supérieure d'Ingénieur Sud Alsace, (**ENSISA**), 2009.

[07] C. JOACHIM & L. PLEVERT: « La révolution invisible, Seuil », ISBN 978-2-02-086703-0, Paris, 2008.

[08] C. TANOUGAST, S. JOVANOVIC, F. MONTEIRO, C. DIOU ET A. DANDACHE : « Initiation à la modélisation et co-simulation comportementale C-VHDL d'un réseau de communication sur Puce (*Network on Chip*) », 10ème JPCNFM, Saint Malo, 26-28, pp.27-32, Novembre 2008.

[09] C. GUEX : « Introduction au VHDL », cours pédagogique**,** Ecole d'Ingénieurs du Canton de Vaud (EIVD), France, 1998.

[10] C. H. PROVOST : «Initiation au langage VHDL **»,** cours pédagogique, Université de Marne-La-Vallée, France, 2010.

[11] C.T IBRAHIM: « Contribution au développement de modèles pour l'électronique de puissance en VHDL-AMS », thèse Doctorat, Institut National DES Sciences Appliquées de Lyon , France,2009.

[12] C.O.KLEIN : « Spécification et conception des systèmes embarqués »**,** cours pédagogique, école polytechnique Montréal, 2009.

[13] C.E. SANCHEZ : « Le langage VHDL», Cours pédagogique, Institut Universitaire de Technologie de CRETEIL-VITRY, 2011-2012.

[14] **C. HYMANS** : « Vérification de composants VHDL par interprétation abstraite», thèse Doctorat, Université de Madrid, Espagne, 2004.

D

[15] D. L. PERRY, « VHDL: Programming by Example», The McGraw-Hill Companies », United States of America, 2002.

[16] D. GUIHAL, *«* modélisation en langage vhdl-ams des systèmes pluridisciplinaires », thèse du doctorat, Université Toulouse III, 2007.

[17] D. GIACONA : « VHDL - Réseaux programmables Cours -partie 8 », cours pédagogique », École Nationale Supérieure d'Ingénieurs Sud Alsace (ENSISA), France, 2003.

[18] D. DUFFAND, « V.H.D.L (VHSIC, High level Description Language)», Institut universitaire de formation des maîtres (IUFM) de Créteil, cours pédagogique, France, 2009**.**

G

[19] G. De MICHELI & L. BENINI: « Networks on Chips, Technology and tools », Morgan Kaufmann publishers, 2006.

E

[20] E. GULIYEV, M. KAVATSYUK, P. LEMMENS, G. TAMBAVE, H .LOHNER: «VHDL implementation of feature-extraction algorithm for the PANDA electromagnetic calorimeter», Nuclear Instruments and Methods in Physics Research A 664 22–28, March 2012.

[21] E. O. HWANG: « Digital Logic and Microprocessor Design With VHDL», cours pédagogique, Institut National polytechnique de Grenoble, 2005.

F

[22] F. MORAES, N. CALAZANS, A. MELLO, L. MOLLER, AND L. OST. HERMES: « An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip Integration», the VLSI Journal, 38(1): 69–93, October 2004.

H

[23] H. MOUSSA : « architectures des réseaux sur puce pour décodeurs canal multiprocesseurs», Thèse de doctorat, 'université de Bretagne Sud, 2009.

I

[24] J. CARON & J. HAUTIER : **«** Commande de processus », Editions Ellipse, France, 1997.

[25] J. ROUILLARD : « Ecrire & comprendre VHDL & AMS 2008 », ISBN 978-1-4092-3689-4, Espagne, 2008.

[26] J. ROUILLARD : « Lire et comprendre VHDL et AMS 2008 », ISBN 978-1-4092-2787-8, Espagne, 2008.

[27] J. HANA, S.F. ZHANG, L. YANYANB, B.Y. CAOA & L.J. HUANGA: «A Fault-tolerant Router's Simulation and Design based-FPGA in Network on chips», Procedia Engineering 23: 77 – 83, 2011.

[28] J. CALVEZ : « Modélisation et conception des systèmes électroniques », Manuscrit auteur, publié dans Colloque national GDR SOC-SIP, France, 2008.

K

[29] K. SKAHILL : «VHDL for programmable logic », cours pédagogique, University of Adelaide South Australia, 1990.

[30] K. GHERFI: « Modélisation vhdl-ams et application à l'intégration de puissance », MEMOIRE de Magister, université MENTOURI, Constantine. 2005.

L

[31] L. BENINI & G. De MICHEL: « Networks on Chips: A New SoC Paradigm», IEEE Computer Journal, 35(1): 70–78, January 2002.

[32] L. ZHOU : «Modélisation VHDL-AMS multi-domaines de structures intelligentes, autonomes et distribuées à base de MEMS », thèse du doctorat, université Louis Pasteur - Strasbourg I, 2007.

[34] L. PIERRE : « Architecture des systèmes embarqués », cours pédagogique, Université Josèphe FOURIER Grenoble, 2010-2011.

M

[35] M. JRIDI, « étude, Modélisation et Amélioration des Performances des convertisseurs Analogique Numérique Entrelacés dans le Temps**»,** thèse de doctorat, Université Bordeaux 1, 2007.

N

[36] N. VISWANATHANA, K. PARAMASIVAMB, K. SOMASUNDARAMC: « Exploring Hierarchical, Cluster based 3D Topologies for 3D NoC», Procedia Engineering 30: 606 – 615, 2012.

[37] N. NOLHIER : « VHDL », cours pédagogique, Université Paul Sabatier, Toulouse, 1997.

O

[38] O. BOURRION: « INITIATION VHDL », cours pédagogique, Université de Montréal, 2009.

P

[39] P. Nouel : « Langage VHDL et conception de circuits », cours pédagogique, Ecole nationale supérieure électronique, informatique et radiocommunications, Bordeaux, 1999.

[40] P. M. ZEITZOFF & J.E. CHUNG: « A perspective from the 2003 ITRS», IEEE circuits & devices magazine, January/February 2005.

[41] P. LECARDONNEL & P. LETENNEUR : « Introduction à la synthèse logique VHDL **»,** cours pédagogique**,** Lycée Julliot de la Morandière*,* GRANVILLE, France, 2001.

R

[41] R. BERNY : « TP CAO Electronique Numérique », cours pédagogique, Université de Genève, 2005.

[42] R. RUELLAND : « Apport de la co-simulation dans la conception de l'architecture des dispositifs de commande numérique pour les systèmes électriques » thèse du doctorat, institut national polytechnique de Toulouse, 2002.

S

[43] S. VIGHETTI : « Photovoltaïque raccordé au réseau : choix et optimisation des étages de conversion », thèse Doctorat UJF, G2Elab, 2008.

[44] S. KUMAR, A. JANTSCH, J .P. SOININEN, M. FORSELL, M. MILLBERG, J. O berg, K. TIENSYRJA, & A. HEMANI: «A Network on Chip Architecture and Design Methodology ». IProceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), PP 105–112, Pittsburgh, PA, USA, April 2002.

[45] S. JOVANOVIC´: « Architecture reconfigurable de système embarqué auto-organisé », thèse du Doctorat, Université Henri Poincaré, Nancy France, 2009.

T

[46] T. CUVELIER : « Introduction au langage VHDL », cours pédagogique, MCENSL1 IUT de Troyes, 2011.

[47] T. BLOTIN : « Le langage de description VHDL », cours pédagogique**,** lycée PAUL ELUARD, Paris, 2007.

U

[48] U. SARAVANAKUMAR, R. RANGARAJAN: «Energy and Throughput Analysis of Multicast Routing Algorithm for 2D Mesh Network on Chip », Procedia Engineering 30 144 – 151, January 2012.

V

[49] V. A. PEDRONI: « Circuit Design with VHDL », cours pédagogique, Massachusetts Institute of Technology, England, 2004.

X

[50] X.T. TRAN : « Méthode de Test et Conception en Vue du Test pour les Réseaux sur Puce Asynchrones : Application au Réseau ANOC » thèse de doctorat, institut polytechnique de Grenoble ,2008 .

Y

[51] Y. SALAH, M. ZEGHID et R. TOURKI: « Conception et Intégration d'un IP de Sécurité pour les Architectures de Communication dans un SoC », communication, Université de Monastir, Tunisie 2012.

[52]Y. HERVE, « Formation VHDL, LAAS-CNRS », cours pédagogique, Université de Toulouse, 2006.

Z

[53] Z. AIT OUALI : «application des FPGA à la commande d'un moteur asynchrone » thèse de magister, Université MOULOUD MAMMRI de TIZI OUAZOU, 2010.

[54] Z. NAVADI: cours pédagogique, « VHDL analysis and Modeling of Digital Systems », université de Grenoble, 1993.

WEBOGRAPHIE

[55] www. wikipédia.com

[56] www .comelec.enst.fr

[57] www.fr.edaboard.com

[58] www.angelfire.com

[59] www.infoscience.epfl.ch

[60] www.lelangagevhdl.net

[61] www.comelec.enst.fr

[62] www.systemc.org.

1) Structure d'une description VHDL :

1. 1 Les en-têtes de fichier

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. L'IEEE les a normalisées et plus particulièrement la bibliothèque IEEE1164 **[11]**. Cela se fait de la manière suivante :

```
Library IEEE ;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

Tout d'abord, la librairie principale (en générale, IEEE). Ensuite, le mot clé *«use»*, qui indique quelle package de la librairie nous allons utiliser. Après cela, le nom du package. Enfin, le .all signifie que l'on souhaite utiliser tout ce qui se trouve dans ce package. Lorsque le nom de la librairie est précédé de l'IEEE, cela signifie que c'est une librairie qui est définie dans la norme IEEE, et que l'on retrouvera donc normalement dans tout logiciel. Les librairies IEEE principales sont :

- ✓ *IEEE.standard;*
- ✓ *IEEE.std_logic_1164;*
- ✓ *IEEE.numeric_std;*
- ✓ *IEEE.std_logic_arith* .

Attention : Il ne faut pas utiliser les librairies *numeric_std* et *std_logic_arith* en même temps : la librairie *std_logic_arith* est en fait une version améliorée de la *numeric_std*, et qui a été ensuite incorporé dans la norme IEEE. Le fait d'utiliser les deux librairies en même temps causera un conflit lors de l'utilisation de certaines fonctions **[2.7.9.10]**.

1. 2 Entité et architecture

En VHDL, une structure logique est décrite à l'aide d'une entité et d'une architecture de la façon suivante :

ℜ La spécification d'entité (ENTITY) définie les signaux d'entrées-sorties, leur type ainsi que leur mode (lecture seule, écriture seule, lecture-écriture) et les procédures éventuellement associées **[1.2.7.9.10]**. Voici la structure générale d'une entité :

ENTITY *Nom de l'entité* **IS**
Description des entrées et des sorties de la structure en explicitant pour chacune d'entre elles le nom, la direction (IN, OUT et INOUT) et le type.
END *Nom de l'entité* **;**

ℜ L'architecture (ARCHITECTURE) est relative à une entité. Elle contient les fonctionnalités et éventuellement les relations temporelles du modèle **[1.2.7.9.10]**. L'architecture décrit le comportement de l'entité suivant la structure ci-dessous :

ARCHITECTURE *Nom de l'architecture* **OF** *Nom de l'entité* **IS**
Zone de déclaration.
BEGIN
Description de la structure logique.
END *nom de l'architecture* **;**

Remarque : Il est possible de créer plusieurs architectures pour une même entité. Chacune de ces architectures décrira l'entité de façon différente.

II.4 .1. 2 Description comportementale et structurelle

Il existe deux façons distinctes de décrire une structure :

- ***Description comportementale*:** le comportement de la structure est directement inscrit dans l'architecture à l'aide d'instructions séquentielles ou sous forme de flot de données.
- ***Description structurelle:*** dans ce type de description, on relie entre eux des composants préalablement décrits en VHDL.

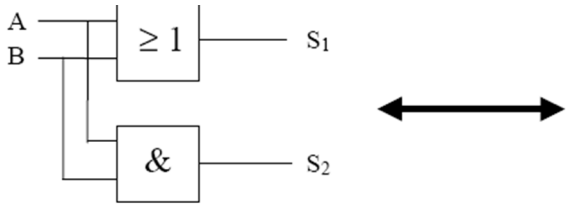
1. 3 Configuration, package et package body :

- La configuration permet, comme son nom l'indique, de configurer l'entité à laquelle elle est associée. On peut, par exemple, spécifier les architectures à utiliser pour tous les composants déclarés au sein d'une description ;
- Pour qu'une description soit portable, c'est-à-dire synthétisable ou simulable sur des outils différents et pour des composants cibles différents, il est préférable de n'avoir qu'une seule architecture par entité. Les configurations deviennent alors inutiles.
- La spécification de paquetage (PACKAGE et PACKAGE BODY) permet de regrouper des déclarations de types et/ou de sous-programmes (fonctions, procédures…) et en fait de construire des bibliothèques. Elle offre ainsi la possibilité d'exporter un de ces objets.

2) Les instructions concurrentes et séquentielles

2. 1 La nécessité d'utiliser des instructions concurrentes :

Lorsque l'on étudie une structure logique, il apparaît à l'évidence que les signaux qui la constituent évoluent de façon indépendante et parallèle.




```
architecture ET_OU of ET_OU is
begin
        S2 <= A and B;
        S1 <= A or B;
end ET_OU;
```

Figure 01 : Exemple d'une structure logique simple.

Dans cet exemple, les signaux S_1 et S_2 évoluent de façon indépendante et parallèle. Il est donc impossible de décrire leur évolution de façon séquentielle. Il faut avoir recours à des instructions de types concurrents qui s'exécutent en parallèle indépendamment de leur ordre d'écriture. Ainsi, à l'exécution de la description ci-dessus, c'est-à-dire lors de la simulation, les signaux S1 et S2 seront réactualisés en continu, en fonction des valeurs de A et de B et de façon parallèle. On parle alors d'instructions concurrentes.

On peut dissocier trois types d'instructions concurrentes :

- Les instructions d'affectation utilisée au sein de l'exemple précédent qui correspondent à l'écriture d'une équation booléenne,

Ainsi que les deux types d'instructions décrits dans le tableau suivant :

Instruction conditionnelle	Description
When else " *affectation When condition else ...*"	Affecter un signal à une expression (expression = signal, équation ou valeur). Exemple : S <= A when (Sel = "00") else B when (Sel = "01") else C when (Sel = "10") else '0';
With select when '*With signal select affectation When condition else ...*"	Semblable à la précédente avec en plus une précision préalable du signal sur lequel vont se porter les conditions. Exemple : With Sel select S <= A when "00", B when "01", C when "10", '0' when others;

Tableau 01 : Description des instructions concurrentes.

2. 2 Les instructions séquentielles :

1. ***Les process :***

Sont des ensembles regroupant des instructions séquentielles et se comportant, d'un point de vue externe, comme des instructions concurrentes, cela pour palier l'incompatibilité qui existe entre les instructions séquentielles et le fonctionnement des montages que l'on souhaite décrire **[1]**. Voici un exemple d'équivalence entre les deux descriptions :

A base d'instructions concurrentes :

```
architecture Arch of Ent is
begin
S1 <= A and B;
S2 <= A when B = '1' else
/A when C = '1' else
D;
end Arch;
```

A base d'instructions séquentielles regroupées au sein d'un process :

```
architecture Arch of Ent is
begin
Process (A, B)
begin
if (A = '1' and B = '1') then S1 <= '1';
else S1 <= '0';
```

```
end if;
end process;
Process (B, C)
begin
if (B = '0') then S2 <= A;
elsif (C = '1') then S2 <= not(A);
else S2 <= D;
end if;
end process;
end Arch;
```

- la déclaration d'un process débute par le mot clé "*process*", suivi d'une liste de noms de signaux exemple *process (A, B)*. Cette liste est appelée "liste de sensibilité", elle contient le nom des signaux dont le changement d'état va provoquer l'exécution du process.
- L'exécution d'un process est concurrente même si les instructions qu'il contient sont séquentielles ;
- L'exécution d'un process n'a lieu qu'en cas de changement d'état de l'un (ou de plusieurs) signal (aux) compris dans la liste de sensibilité.

2. *Les boucles et instructions:*

Les instructions séquentielles du type *if, then, else...* sont très puissantes, elles sont écrites à l'intérieur d'un process et permet une écriture des descriptions très lisible. Le tableau ci-dessous contient quelques exemples de traductions de boucles ou d'ensembles d'instructions conditionnelles.

Instructions et boucles	Description
if	**if** *condition1* **then** *instructions séquentielles*; **elsif** *condition2* **then** *instructions séquentielles*; **elsif** *condition3* **then** *instructions séquentielles*; **else** *instructions séquentielles*; **end if;**
for	**for** *paramètre* **in** *intervalle* **loop** *instructions séquentielles*; **end loop;** //**pour** *le paramètre* **compris** *dans l'intervalle* **exécuter** *instructions séquentielles*; **fin de la boucle;**
case "switch case",	**case** *signal* **is** **when** *valeur1* => *instructions séquentielles*; **when** *valeur2* => *instructions séquentielles*; **when** *valeur3* => *instructions séquentielles*; **when others** =>*instructions séquentielles*; **end case;**

Tableau02 : Description des instructions et boucles utilisée en VHDL.

3. *Les vecteurs de signaux :*

Le langage de description VHDL permet la déclaration et la manipulation des signaux de type vecteurs ou de bus de signaux : plusieurs signaux, pas uniquement un.

4. *La déclaration GENERIC*

Au sein de la description VHDL d'une structure logique, il est possible de rajouter une déclaration de GENERIC correspondant à des paramètres. Ces paramètres pourront, par

la suite, être modifiés lors de l'utilisation de la description en temps que composant. Typiquement, on utilise une déclaration de GENERIC pour spécifier des temps de propagation et autres délais de portes logiques. Ces temps seront alors modifiables lors de l'utilisation de ces portes logiques dans une description structurelle.
Exemple :

Entity OU is
GENERIC (TP: time := 20 ns);
port (E1, E2 : in bit; S1 : out bit);
end OU;

3) Fonctions et procédures

3.1 Rôle, principe et fonctionnement :

Le langage VHDL permet l'utilisation et la création de fonctions ou de procédures que l'on peut appeler à partir d'une architecture. Le rôle de ces fonctions ou procédures est de permettre, au sein d'une description, la création d'outils dédiés à certaines tâches pour un type déterminé de signaux.

3.2 Déclaration :

La syntaxe d'une fonction est la suivante :

function *nom de la fonction* (*liste des paramètres de la fonction avec leur type*)
return *type du paramètre de retour* is
zone de déclaration des variables;
begin
instructions séquentielles;
return *nom de la variable de retour ou valeur de retour*;
end;

Et pour l'appel d'une fonction, il peut être fait à partir d'instructions séquentielles ou concurrentes. Dans le cas des instructions concurrentes, la fonction sera toujours vérifiée.
En ce qui concerne les procédures, elles diffèrent des fonctions par le fait qu'elles acceptent des paramètres dont la direction peut être IN, INOUT et OUT. Une procédure ne possède donc pas un ensemble de paramètres d'entrées et un paramètre de sortie mais un ensemble de paramètres d'entrées-sorties et aucun paramètre spécifique de sortie **[1] [17]**. Prenons la syntaxe suivante :

procedure nom [(liste de paramètres formels)] is
[déclarations]
begin
instructions séquentielles
end [nom] ;

Pour l'appel, la syntaxe sera : *nom (liste de paramètres réels) ;*

❖ Toute fonction ou procédure déclarée au sein d'un package est décrite dans le package body associé à ce package.

4) Les types prédéfinis ou non

Définissons d'abord quelques notions à savoir :

- Objet (object): information manipulée par le langage et répartie en quatre classes:
- Constante (constant): objet de valeur fixe ;
- Variable (variable): objet de valeur modifiable par affectation (:=) ;

- Signal (signal): objet spécifique communiqué entre les parties (processus) d'un modèle et dont l'affectation (<=) ne modifie pas la valeur présente mais les valeurs futures prévues ;
- Fichier (file).
- Type (type): ensemble des valeurs prises par un objet et regroupées en quatre classes

Le tableau suivant montre les quatre classes, types associés et les expressions qui permettent de les définir **[1,3]** :

Classe	Type	Description	Expression
Scalaires (scalar)	entier (integer)	Nombre positif ou négatif sur 32 bits.	A: integer;
	réel (real)	Valeurs réelles.	A: real;
	énuméré (enumerated)	Caractérisation d'un objet par la liste complète de ses valeurs.	TYPE type_etat IS (init, debut, fin);
	Bit (bit)	Deux valeurs possibles '0' ou '1'	A: A: bit;
	Booléen (boolean)	Deux valeurs possibles True ou False	A: boolean;
	Caractère (character)	Caractères a, b, c ..., 1, 2 ...	A: character;

	physique (physical)
	
	Temps (time)
		Nombre réel de temps fs, ps ...,
		
		Caractérisation d'un objet par Son unité de base (base unit), l'intervalle de ses valeurs et ses éventuelles sous-unités.
			TYPE time IS RANGE -2147483647 TO 2147483647
			UNITS
			fs;
			ps = 1000 fs;
			ns = 1000 ps;
			us = 1000 ns;
			ms = 1000 us;
			sec = 1000 ms;
			min = 60 sec;
			hr = 60 min;
			END UNITS;

Composites (composite)	tableau (array)	Type composite consistant en un groupe d'objets dont le type est identique et la position indexée.	TYPE string IS ARRAY (positive RANGE <> OF character);
	Enregistrement (record)		TYPE date IS RECORD jour : natural RANGE 1 TO 31; mois : string; annee : integer RANGE 0 TO 4000; END RECORD;

ableau03 : Les types prédéfinis utilisée en VHDL.

Remarque :

Les types prédéfinis proposés par le langage VHDL initial sont vite devenus insuffisants dans la plupart des cas. L'instruction qui permet de définir un type est :

type nom is (valeurs possibles) ; **[3].**

5) Les attributs

5.1 Présentation des attributs, leurs rôles

Le langage VHDL propose un outil appelé attribut que l'on peut, en quelque sorte, assimiler à des fonctions. Placé auprès d'un signal, l'attribut " 'event " va, par exemple, retourner une information vraie ou fausse (type booléen) sur le fait que le signal en question ait subi ou non un événement (un changement de valeur). Le fonctionnement de cet attribut ressemble au comportement d'une fonction dont le paramètre de retour serait de type booléen, Il existe en réalité deux sortes d'attributs :

- Les attributs destinés à retourner des informations concernant le comportement d'un signal (événements, dernière valeur...).
- Les attributs destinés à retourner des informations concernant les caractéristiques d'un vecteur (longueur, dimension...).

Pour ces deux sortes d'attributs, on parle d'attributs prédéfinis, au même titre que l'on parle de types prédéfinis (bit, boolean, etc.). Voici une liste non exhaustive des différents attributs prédéfinis proposés par le langage VHDL **[1]**.

Attribut	Définitions - informations de retour
'high	Placé près d'un nom de tableau, il retourne la valeur du rang le plus haut (la valeur de retour est de type entier).
'low	Placé près d'un nom de tableau, il retourne la valeur du rang le plus bas (la valeur de retour est de type entier).
'left	Placé près d'un nom de vecteur ou tableau, il retourne la valeur du rang le plus à gauche (la valeur de retour est de type entier).
'right	Placé près d'un nom de vecteur ou tableau, il retourne la valeur du rang le plus à droite (la valeur de retour est de type entier).
'range	Placé près d'un signal, il retourne la valeur de l'intervalle spécifié par l'instruction range lors de la déclaration du signal ou du type utilisé.
'reverse_ range	Placé près d'un signal, il retourne la valeur de l'intervalle inverse spécifié par l'instruction range lors de la déclaration du signal ou du type utilisé
'length	Retourne X'high - X'low + 1 (sous la forme d'un entier).
'event	Vrai ou faux si le signal auquel il est associé vient de subir un changement de valeur.
'active	Vrai ou faux si le signal auquel il est associé vient d'être affecté.
'last_ value	Retourne la valeur précédente du signal auquel il est associé.
'last_ event ou 'last_ active	Retournent des informations de temps concernant la date d'affectation ou de transition des signaux auxquels ils sont affectés
'stable (T)	Vrai ou faux si le signal auquel il est associé n'est pas modifié pendant la durée T.
'quiet	Vrai ou faux si le signal auquel il est associé n'est pas affecté pendant la durée T.
'transaction	Retourne une information de type bit qui change d'état lorsque le signal auquel il est associé est affecté.

Tableau04 : Description des attributs utilisés dans le VHDL.

5.2 Définition des attributs

Comme il est possible de définir des types, il est également possible de définir des attributs. La différence réside dans le fait que le simulateur va ignorer ces nouveaux attributs et ne reconnaître que les attributs de type prédéfinis. La définition d'un nouvel attribut ne sert qu'à ajouter des informations à une description. Ces informations seront par la suite utilisées par d'autres outils. Les outils de synthèses logiques se servent souvent profitent de cette opportunité pour introduire des paramètres de synthèse et spécifier **[1]**, Voici l'exemple d'une création d'attribut :

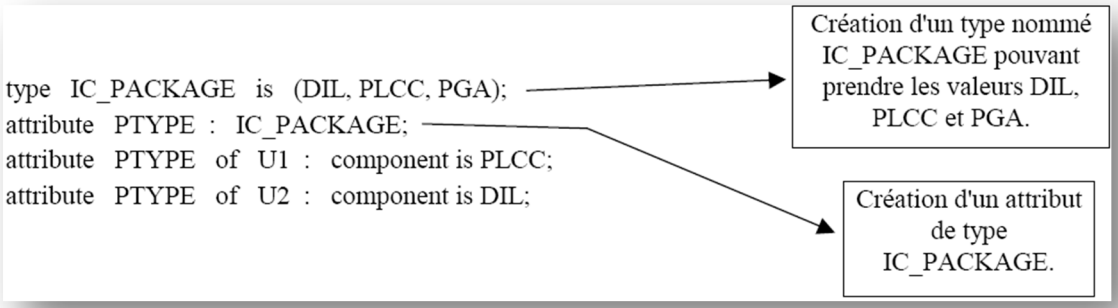


Figure 02 : création d'un attribut et d'un type.

5) Synthèse d'une description VHDL

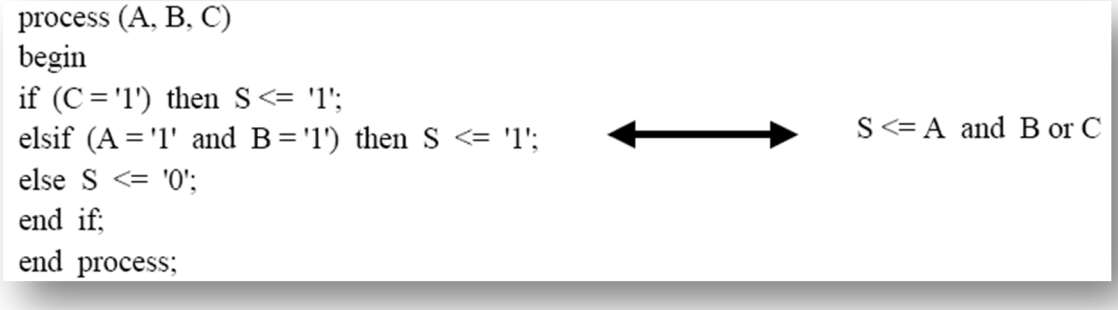
La synthèse d'une description VHDL représente une étape essentielle dans le processus de conception d'un composant programmable. Le résultat de cette synthèse va déterminer l'ampleur ou les caractéristiques du composant cible. Il est donc important de savoir orienter une synthèse de façon à optimiser le résultat. Il est possible de créer deux descriptions équivalentes à la simulation dont la synthèse ne donnera pas le même résultat. Il y a donc tout un savoir-faire à acquérir pour rédiger une description VHDL de façon à orienter le résultat de la phase de synthèse **[1]**.

5.1 Fonctions combinatoires

Une fonction combinatoire est une structure pour laquelle chaque combinaison d'entrée fait correspondre en sortie une combinaison déterminée et indépendante des combinaisons d'entrées et de sorties précédentes. On peut donc décrire ces structures à l'aide d'équations booléennes qui seront toujours vérifiées comme le sont les instructions concurrentes.

En effet, l'instruction "**S <= A and B or C**" est toujours vérifiée et correspond à l'équation booléenne S = A.B + C. Mais il est aussi possible de décrire une structure combinatoire en utilisant les instructions séquentielles d'un process.

L'équation précédente peut, par exemple, s'écrire :



L'issue de synthèse de ces deux types de description est identique, mais laquelle parmi les deux descriptions est à utiliser ?

En réalité, même si le résultat de la synthèse est identique, il y a une différence fondamentale entre ces deux descriptions. L'une n'est que le reflet des équations logiques de la structure combinatoire à décrire, alors que l'autre se base sur l'expression de son comportement.

Pour tenter de résumer les différences entre ces deux méthodes de description, le tableau peut lister les avantages et les inconvénients de chacune d'elles :

Méthode de description	Avantages	Inconvénients
A l'aide d'instructions concurrentes	La description obtenue est lisible et simple, la description obtenue reflète bien la réalité.	❖ Décrire une fonction combinatoire en utilisant des instructions concurrentes sous-entend que la taille de sa table de vérité le permet ou que les simplifications de cette dernière ont été faites.
A l'aide de process	❖ Pour des fonctions compliquées dans lesquelles les équations possèdent de nombreux termes, cette méthode est parfois plus simple. Lorsque la description comportementale est plus simple que la description algorithmique, cette méthode est avantageuse ; ❖ Il n'est pas nécessaire de faire les simplifications de la table de vérité, elles sont faites automatiquement par le synthétiseur lors de la synthèse.	❖ Dans certains cas, cette écriture est peu lisible et par conséquent très difficilement modifiable par une personne extérieure.

Tableau05 : Comparaison entre les deus méthodes de description des fonctions combinatoires.

Il n'y a donc pas de règle générale mais plutôt la nécessité de procéder à une démarche logique d'observation avant la description pour choisir la méthode de description adéquate.

5. 2 Fonctions séquentielles

La synthèse d'une fonction logique séquentielle est beaucoup plus complexe que celle d'une fonction combinatoire. Les possibilités de description offertes par le langage VHDL sont vastes et bien souvent irréalisables dans l'état actuel des technologies. Prenons l'exemple des attributs, ils sont puissants, efficaces, mais posent, pour certains, de gros problèmes aux outils de synthèse qui ne savent pas comment réaliser des structures capables de représenter le fonctionnement de ces attributs.

L'attribut *'quiet(T),* ne représente aucune structure électronique simple. Les synthétiseurs actuels refusent donc cet attribut faute de savoir quoi en faire. Par contre, il existe certaines descriptions typiques que le synthétiseur va reconnaître et traduire directement, comme l'attribut *'event*. Alors, on peut se poser la question suivante : comment doit-on rédiger la description VHDL d'une fonction séquentielle pour que la synthèse soit optimale et utilise le moins de composants possible ?

Or, là aussi, il n'existe pas de règle de rédaction miracle qui permette de garantir une synthèse optimale. Toutefois, on sait que toute fonction logique séquentielle

(synchrone) peut être réalisée grâce à une structure du type machines de MOORE ou de MEALY (appelées aussi séquenceur ou machine à états). La synthèse d'une description de fonctions logiques séquentielles donnera donc une de ces structures (ou une extension de ces structures).

- Comportementale, se limitant à décrire le comportement de la structure à l'aide d'instructions *if, then, else, case, when*...,
- Structurelle ou semi-structurelle, décrivant chacune des parties constituant les machines de MOORE ou de MEALY.

Notons qu'il est souvent intéressant d'utiliser ce genre de description lorsque l'on souhaite imposer des critères de synthèse du type : utiliser un minimum de bascules, privilégier la rapidité du montage, garantir certaines contraintes de temps, etc.

5.3 Synthèse d'un diagramme d'état

Il existe plusieurs formalismes capables de décrire l'évolution et le comportement d'un système séquentiel. Parmi eux se trouve le diagramme d'états (FSM), qui présente l'avantage d'être directement traduisible en langage VHDL.

6) Code source du modèle de la batterie :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity blok_batterie is
port (
U_Batteie :in std_logic_vector (31 downto 0);
GE : in  std_logic_vector (31 downto 0);
--reset :in std_logic;
demande :out  std_logic_vector (31 downto 0);
decharge1 :out std_logic;
decharge2 :out std_logic ;
cmd_GE :out std_logic
);
end  blok_batterie;

architecture Behavioral of blok_batterie is
----------------------------------------------------------------------
--*********************  declaration  ****************************
----------------------------------------------------------------------
signal  etat, C_dech2: std_logic;
constant Vmin: std_logic_vector (31 downto 0):="00000000000000000000000000001111";
constant Vmax: std_logic_vector (31 downto 0):="00000000000000000011111111111111";
constant deux_Vmax: std_logic_vector (31 downto 0):="00110000000000000011111111111111";
signal count : std_logic_vector( 3 downto 0);--:="0000";
signal E_dech1: std_logic :='0';
signal clk :  std_logic;
signal tmp: std_logic_vector(3 downto 0);

begin
```

```
-----------------------------------------------------------------
--*********************  programmation  **************************
-----------------------------------------------------------------
test1 : process(U_Batteie,clk, E_dech1)
begin
   if (U_Batteie< Vmin ) then
       cmd_GE <= '1'; decharge1 <= '0'; decharge2 <= '0';
                          demande <= GE;
   else
                    if (( U_Batteie >  Vmin)  and (U_Batteie <  Vmax )) then
                          cmd_GE <= '0'; decharge1 <= '0';  decharge2 <= '0';
                          demande <= U_Batteie;
                    else
                                demande <= U_Batteie;
                                      decharge1 <= '1'; E_dech1 <= '1'; C_dech2 <='0';
                                            if (U_Batteie >= deux_Vmax ) then
                                                  C_dech2 <='1';
                                                  decharge2 <= '1';
                                            end if;
                  end if;
end if;

end process;

-----------------------------------
C_etat : process(U_Batteie, C_dech2)
begin
   if (U_Batteie < Vmax ) then
       etat <='0';
          else
                    if C_dech2 = '0' then
                                etat <= '0';
                          else
                                etat <= '1';
                          end if;
         end if;
end process;
---------------------------------------

end Behavioral;
```

7) Code source du modèle de la batterie :

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

entity routeur_NoC is
port ( paquet: in std_logic_vector( 7 downto 0);
reçu_ACK: in std_logic;
```

```
East,west, North, South: out std_logic;
--------- transmission de hamming
evoi_ACK, etat_fautif: out std_logic;
paquet_emis : out std_logic_vector( 7 downto 0)
);
end routeur_NoC;

architecture Behavioral of routeur_NoC is
signal x_destination, y_destination: std_logic_vector(1 downto 0);

signal buffer_local: std_logic_vector( 7 downto 0);
signal p1,p2,p3,p4,p5,p, v1,v2,v3,v4,d1,d2,d3,d4,d5,d6,d7,d8: std_logic;
signal v: std_logic_vector(3 downto 0);

begin
-- algorithme de routage  X_Y (l'aiguillage des paquets)
 routage : process (paquet,v,p, x_destination ,y_destination)
 begin


--codage de hamming pour la detection et la correction des erreurs

 d1 <= paquet(7); d2 <= paquet(6); d3 <= paquet(5); d4 <= paquet(4); d5 <=
paquet(3);
 d6 <= paquet(2); d7 <= paquet(1); d8 <= paquet(0);

 x_destination  <=  d1 & d2;   y_destination <=  d3 & d4;
 -- -- l'algorithme de detection
 p1 <= d1 xor d2 xor d4 xor d5 xor d7;
 p2 <= d1 xor d3 xor d4 xor d6 xor d8;
 p3 <= d2 xor d3 xor d4 xor d8;
 p4 <= d5 xor d6 xor d7 xor d8;

 p5 <= p1 xor p2 xor d1 xor p3 xor d2 xor d3 xor  d4 xor p4 xor d5 xor d6 xor d7 xor
d8;

v1<= p1 xor d1 xor d2 xor d4 xor d5 xor d7;
v2<= p2 xor d1 xor d3 xor d4 xor d6 xor d8;
v3 <=p3 xor d2 xor d3 xor d4 xor d8;
v4<=p4 xor  d5 xor d6 xor d7 xor d8;
p<= p5 xor p1 xor p2 xor d1 xor p3 xor d2 xor d3 xor  d4 xor p4 xor d5 xor d6 xor d7
xor d8;

v<=v1 & v2 & v3 & v4;

--transmission_hamming <= p1 & p2 & d1 & p3 & d2 & d3 & d4 & p4 & d5 d6 &
d7 & d8 & p5;

  if  ((v="0000")and ( p='0')) then
```

```
 --nb_erreurs = 0
 etat_fautif <='0'; paquet_emis <= paquet;
-- (' aucune erreur est détectée');
 else
                if ((v/= "0000") and (p='1')) then
                --nb_erreurs = 1
                        --*********correction*********
                        if (v= "0001") then paquet_emis<= d1 & d2 & d3 & d4 & d5
& d6 & (not d7) & d8;

                        else
                                if (v= "0010") then paquet_emis<=  d1 & d2 & d3 & d4
& d5 & (not d6) & d7 & d8;

                                else
                                        if (v= "0011") then paquet_emis<= d1 & d2 &
d3 & d4 & (not d5) & d6 & d7 & d8;
                                        else
                                                if (v= "0100")then paquet_emis<= d1 &
d2 & d3 & (not d4) & d5 & d6 & d7 & d8;

                                                else
                                                        if          (v=          "0101")then
paquet_emis<= d1 & d2 & ( not d3) & d4 & d5 & d6 & d7 & d8;

                                                        else
                                                                if      (v=      "0110")then
paquet_emis<= d1 & (not d2) & d3 & d4 & d5 & d6 & d7 & d8;

                                                                else
                                                                        if (v= "0111")then
paquet_emis<= (not d1) & d2 & d3 & d4 & d5 & d6 & d7 & d8;

                                                                        else
                                                                        end if;

                                                                        end if;
                                                                end if;
                                                        end if;
                                                end if;
                                        end if;
                                end if;
                        etat_fautif <='0';
                else
                                if ((v/= "0000") and (p='0')) then
                                --nb_erreurs = 2
                                --*********correction d'une seule erreur*********
                        if (v= "0001") then paquet_emis<= d1 & d2 & d3 & d4 & d5
& d6 & (not d7) & d8;
                        else
```

```
                        if (v= "0010") then paquet_emis<= d1 & d2 & d3 & d4
& d5 & (not d6) & d7 & d8;
                        else
                                if (v= "0011") then paquet_emis<= d1 & d2 &
d3 & d4 & (not d5) & d6 & d7 & d8;
                                else
                                        if (v= "0100")then paquet_emis<= d1 &
d2 & d3 & (not d4) & d5 & d6 & d7 & d8;
                                        else
                                                if (v= "0101")then
paquet_emis<= d1 & d2 & ( not d3) & d4 & d5 & d6 & d7 & d8;
                                                else
                                                        if (v= "0110")then
paquet_emis<= d1 & (not d2) & d3 & d4 & d5 & d6 & d7 & d8;
                                                        else
                                                                if (v= "0111")then
paquet_emis<= (not d1) & d2 & d3 & d4 & d5 & d6 & d7 & d8;
                                                                end if;
                                                        end if;
                                                end if;
                                        end if;
                                end if;
                        end if;
                etat_fautif <='1';
                        else
                                if ((v= "0000") and (p='1')) then
                                --nb_erreurs = 1
                                etat_fautif <='1'; paquet_emis
<="00000000";
                                end if;
                        end if;
        end if;
end if;
----------------------------------------------------------------
 if ("10" < x_destination) then East <='1'; west <='0';
 South<='0'; North <='0';

else
        if ("10" > x_destination) then west <='1';
          East <='0'; South<='0'; North <='0';
        else
                if (("10" = x_destination) and ("01" > y_destination)) then
South<='1';
                                East <='0';west <='0';  North <='0';
                else
                        if (("10" = x_destination) and ("01" < y_destination))
then North <='1';
                                South<='0'; East <='0'; west <='0';
```

```
                                        end if;
                                end if;
                        end if;
end if;
-------------------------------------------
--*****************contrôl de flux************
evoi_ACK <= '1';
 if ( reçu_ACK ='1') then
buffer_local <= "00000000";
 else
 buffer_local <= buffer_local;
 end if;
-------------------------------------------
end process;
-----------------------------------------------------------------
end Behavioral;
```

Résumé

Ce travail de mémoire s'inscrit dans le cadre de développement des systèmes embarqués sur FPGA (Field Programmable Gate Array). C'est une nouvelle approche pour améliorer les performances du contrôle numérique des systèmes à processeur complet SoC à l'aide des outils de CAO. Nous avons exposés les principales caractéristiques des langages de description matérielle HDL, en suite nous avons marqués l'apport de l'informatique sur l'électronique. A cet effet, nous avons posés comme objectif la conception de la circuiterie de commande pour deux domaines différents : le premier concerne l'électronique, un modèle de batterie pour une gestion optimisée d'énergie électrique d'un système multi sources, le deuxième concerne les réseaux de communication sur puce, par une proposition d'un modèle de routeur tolérant aux fautes (détection et correction des erreurs). En s'intéressant aux différents aspects de la conception des systèmes intégrés : description, simulation et synthèse. Nous avons utilisé le langage de description matériel VHDL pour la modélisation comportementale des composants et un environnement logiciel XILINX ISE (version 13.2_1) pour la conception et la simulation en vue de vérification des systèmes conçus.

Abstract

This memory is part of embedded systems on FPGA (Field Programmable Gate Array). This is a new approach to improve the performance of digital control systems to complete SoC processor using CAD tools. We have exposed the main features of hardware description languages HDL, after we scored in the contribution of information technology on electronics. To this end, we are posed as objective to design the control circuitry for two different areas: one for the electronics, a battery model for optimal management of electrical energy of a multi sources (work developed for renewables), the second is a hot topic for communications networks on chip, for a proposal of a model of fault-tolerant router (detection and error correction). By looking at different aspects of the design of integrated systems: description, simulation and synthesis. We used the VHDL hardware description language for behavioral modeling of components and Xilinx ISE software environment (version 13.2_1) for design and simulation for verification of system designed.