MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF
UFAS (ALGERIE)

MEMOIRE

Présenté à la Faculté des Sciences de L'Ingénieur

Département d'Électronique

Pour l'obtention du diplôme de

MAGISTER

Option : Contrôle

Par

BENHAMADOUCHE ABDELOUAHAB

Thème:

CONCEPTION D'UN SYSTEME RECONFIGURABLE DE CONTROLE FLOU SUR PUCE

Soutenu le: / /2007 devant la commission d'examen:

JURY :

A. KHELLAF	**Prof à l'université de Sétif**	**Président**
T. MOHAMADI	**Prof à l'université de Sétif**	**Examinateur**
A. ZEGADI	**Maître de conférence à l'université de Sétif**	**Examinateur**
N. KHENFER	**Prof à l'université de Sétif**	**Rapporteur**
H. CHEMALI	**Chargé de cours à l'université de Sétif**	**Invité**

Remerciements

A l'issue de l'élaboration du présent mémoire, je me plie à cette aimable tradition, qui est de remercier toutes les personnes, qui de près ou de loin, par leur compréhension, leur coopération m'ont facilité la tâche et ont contribué à l'élaboration de ce travail.

Mais avant, je remercie Dieu le tout puissant qui m'a donné le courage, la volonté ainsi que la patience, pour terminer ce travail.

Je voudrais tout d'abord exprimer ma profonde reconnaissance à Mr N. KHENFER et Mr H. CHEMALI pour leur aide et leur disponibilité, et je tiens à les remercier d'avoir accepté de diriger ce travail et de m'avoir bien orienté.

Je tiens également à remercier Mrs A. KHELLAF, T. MOHAMADI, A. ZEGADI pour avoir accepté d'être membres du jury de ce mémoire.

Je ne peux manquer enfin d'exprimer ma gratitude et mes remerciements aux, enseignants, collègues et amis, au près desquels j'ai trouvé un excellent esprit de coopération.

Résumé

Une plateforme de Xilinx ISE 9.1i et un simulateur de ModelSim 6.0a sont utilisés pour la conception d'un système flou reconfigurable. Il s'agit d'une implémentation d'un système flou développé en VHDL sur une famille de FPGAs 'Virtex II Pro' et exploité pour la gestion de la température des IPs d'un SoC moyennant la reconfiguration des modes de fonctionnement des IPs en utilisant le bus local de communication des SoCs 'SMBUS'.

Une étude des techniques logicielles et matérielles de reconfiguration des systèmes a permis la conception de blocs reconfigurables au sein du système flou développé, apportant souplesse et flexibilité de prise en charge de contextes multiples et ciblant l'optimisation du fonctionnement global.

L'analyse des performances obtenues grâce à la reconfiguration a montrée non seulement un apport conséquent du degré de validation des méthodes de reconfiguration, mais aussi une nouvelle vision de conception. En effet le concepteur ne se trouvera plus fixé sur une solution unique, mais au contraire se confrontera à une sélection d'alternatives dont les retombées sont généralement liées à la structure active et à l'indice de performance ciblé.

TABLE DES MATIERES

Table des matières

INTRODUCTION 1

CHAPITRE I : SYSTÈMES SUR PUCE (SoCs)

Introduction 3
I.1 Méthodologie de conception des SoCs 3
I.1.1 Système sur puce 3
I.1.2 Flot de conception des SoCs 4
I.1.3 Composant virtuel 6
I.2 Les architecture de communication 7
I.2.1 Le bus AMBA 8
I.2.2 Le bus CoreConnect 9
I.2.3 Le bus Wishbone 10
Détails techniques 11
I.3 Test des SoCs 11
II.3.1 Architecture de test des SoC 11
a) Les "source" et "sink" 12
a) Le wrapper 12
b) Le TAM 12
I.3.2 Test thermique 12
I.3.2.1 DFTT (Design for Thermal Testability) 13
b) Built-In Temperature sensor 13
c) Boundary Scan (BS) 14
I.3.2.2 Test scheduling 14
I.4 Management thermique 15
Conclusion 16

CHAPITRE II : SYSTÈMES RECONFIGURABLES

Introduction 18
II.1 Types de reconfiguration 18
II.1.1 Reconfiguration logique 18
II.1.2 Reconfiguration par jeu d'instruction 19
II.1.3 Reconfiguration statique et dynamique 19
II.1.3.1 Reconfiguration statique 19
II.1.3.2 Reconfiguration dynamique 19
II.2 Classification architecturale des systèmes reconfigurables 20
II.2.1 Classification selon la granularité 20
II.2.2 Classification selon le schéma de reconfiguration 20
II.2.2.1 Architecture statiquement reconfigurable 20

II.2.2.2 Architecture dynamiquement reconfigurable 21
La reconfiguration dynamique à contexte unique 21
La reconfiguration dynamique multi-contexte 21
Les architectures reconfigurables partiellement 22
II.3 Classification selon le mode de traitement 22
II.3.1 Architecture orientée chemin de donnée 22
II.3.2 Architecture orientée traitement logique 22
II.3.3 Architecture orientée mémoire 23
II.4 Conception des SoCs reconfigurables 23
II.4.1 Les exigences au niveau système pour les SoCs reconfigurable 24
II.4.1.1 Spécification du système 24
II.4.1.2 Partitionnement et mapping hardware/software. 24
II.4.1.3 Conception de l'architecture. 25
II.4.2 Flot de conception des SoCs reconfigurables 25
II.4.2.1 Conception au niveau système (*System Level Design-SLD*) 27
II.4.2.2 Conception détaillée (*Detailed Design-DD*) 27
II.4.2.3 Implémentation (*Implementation Design-ID*) 28
Conclusion 28

CHAPITRE III : PROCESSEURS FLOUS

Introduction 29
III.1 La Logique floue 29
III. 1. 2 Mise en œuvre d'un système à base de la logique floue 29
a) Bloc de fuzzification 30
b) La base de connaissance floue 30
c) Engin d'inférence 30
d) Bloc de défuzzification 30
III. 2. Processeurs et contrôleurs flous 31
III.3 Architecture classique des processeurs flous 33
III.3.1 Architecture série 33
Architecture de l'unité d'inférence 33
Architecture série de la défuzzification 33
III.3.2 Architecture parallèle 34
Architecture du processeur 34
Performances 35
III.3.2 Architecture parallèle (*Pipelined Subdivision*) 35
Performances : 36
III.3.3 Coprocesseur flou WARP 36
Fonctionnement 36
Performance 37
III.4 Architecture reconfigurable 37
III.4.1 Processeur flou pour plateforme dynamiquement reconfigurable 38
III.4.2 Processeur flou reconfigurable statiquement 38
III.4.3 Architecture à défuzzification reconfigurable dynamiquement 38
Conclusion 40

CHAPITRE IV : CONCEPTION D'UN SYSTÈME FLOU RECONFIGURABLE

Introduction **41**

IV.1 Conception au niveau système **42**

IV.1.1 Langage et outils de conception 42

IV.1.2 Modèle général 43

IV.1.2.1 Blocs de fuzzification 43

IV.1.2.2 Bloc de la base des règles 44

IV.1.2.3 Bloc d'inférence 45

IV.1 2.4 Bloc de défuzzification 45

IV.1.3 Éléments de reconfiguration 45

IV.1.3.1 Technique de reconfiguration 45

IV.1.3.2 Chemin de données (Datapath) 46

IV.1.3.3 Réseau d'interconnexions reconfigurables 46

IV.1.4 Modèle de Partitionnement 46

IV.1.4.1 Allocation 47

d) Modules 'hardware' 47

e) Module 'hardware reconfigurable' 47

f) Module 'software' 47

IV.1.4.2 Partitionnement spatial 47

IV.1.4.3 Partitionnement temporel 48

IV.1.4.4 Ordonnancement 49

IV.2 Conception structurelle **49**

IV.2.1 Hardware non-reconfigurable 50

IV.2.1.1 Fuzzification 50

IV.2.1.2 Base des règles 52

IV.2.2 Hardware reconfigurable 54

IV.2.2.1 Éléments de calcul 54

a) Comparateur (*CMP*) 54

b) Additionneur (*ADD*) 55

c) Multiplicateur (*MUL*) 56

d) Diviseur (*DIV*) 57

IV.2.2.2 Éléments de sauvegarde et de traitement 59

e) Élément de traitement d'inférence 59

f) Élément de sauvegarde (*File System Register : FSR*) 60

IV.2.2.3 Matrice de connexion 61

IV.2.3 Contrôle de reconfiguration 63

IV.2.4 Assemblage du système flou 66

IV.2.4.1 Synthèse 66

IV.2.4.2 Exemple de contrôle flou 68

IV.3 Conception du système flou sous forme d'IP-Soft **69**

IV.3.1 CPU 70

IV.3.2 Interface wishbone 71

IV.3.3 Contrôleur SMBus 72

Conception du contrôleur SMBus 72

IV.4 Application au contrôle de la température d'un SOC **74**

IV.4.1 Architecture système 74

IV.4.2 Contrôleur flou 75

IV.4.2.1Paramètres de contrôle flou 75
g) Fonction d'appartenance des entrée X1 et X2 76
h) Règles floues 77
IV.4.2.1Paramètres de reconfiguration 78
IV.4.3 Simulation 79
IV.4.4 2ème approche de gestion de la température 80
a) Méthode par calcul de la valeur moyenne 80
b) Méthode par transcodage de la sortie 83
c) Méthode par transcodage des entrées et de la sortie 85
Conclusion 88

CONCLUSION GÉNÉRALE ET PERSPECTIVES 90

BIBLIOGRAPHIE 92

INTRODUCTION GENERALE

Introduction

L'augmentation de la complexité des algorithmes de traitement du signal, de l'image et du contrôle, notamment dans les applications embarquées, nécessite de grandes puissances de calcul pour satisfaire de nouvelles contraintes. C'est le cas par exemple des contraintes liées au contexte temps réel. Ceci conduit les concepteurs à s'orienter de plus en plus vers des architectures matérielles composées de différentes unités de calcul opérant en parallèle, présentant des capacités de calcul différentes et permettant de grandes optimisations. C'est le cas des systèmes sur puce ou SoC (System on Chip). Les composants présents dans ces systèmes peuvent être des unités de calcul programmables, des unités de calcul spécialisées ou reconfigurables, ou encore des chemins de données spécialisés.

En particulier, la présence d'unités reconfigurables, permet d'adapter l'architecture à différents traitements. Un accélérateur reconfigurable au sein de ce type de plate-forme permet d'apporter un degré important de flexibilité à l'architecture. Le concept de Platform based Design peut ici être pleinement exploité pour assurer la pérennité d'une architecture vis-à-vis de modifications de ses fonctionnalités ou de ses conditions de fonctionnement (changements de standards, de missions, etc.).

La flexibilité de cette architecture est assurée, en fonction de la technologie reconfigurable employée, par la possibilité d'allouer dynamiquement différents opérateurs de traitement dédiés au sein même de l'unité reconfigurable.

La gestion globale d'une telle architecture fortement hétérogène nécessite de plus en plus la présence d'un exécutif adapté capable, au minimum, de centraliser la gestion de la mémoire, d'ordonnancer sous différentes contraintes les tâches à exécuter, d'assurer le partage des moyens de communication qui soit indépendant de la technologie et de la structure matérielle.

Le travail, présenté dans ce mémoire, est issu des réflexions menées au sein de l'équipe du labo de test, Il ambitionne une modélisation système complète d'une plate-forme Reconfigurable System on Chip, incluant l'ensemble des éléments logiciels et matériels ainsi que leurs interactions. Nous souhaitons donc, au travers de ce mémoire, identifier en premier lieu les différents composants matériels et logiciels intervenant dans les système sur puce (SoC). En particulier, nous tentons de définir le rôle et l'organisation précise de d'une telle plate-forme. L'objectif est de se doter de méthode formelle pour la conception de ce type d'architecture.

L'élaboration d'un système de contrôle flou est notre premier objectif, il va nous permettre de vérifier les méthodes de conception des systèmes sur puce étudié dans ce mémoire. Notre souci est de rendre notre modèle de système flou le plus reconfigurable possible, supportant la plus grande variété de plates-formes et rassemblant la plupart des techniques développé dans le domaine du contrôle flou.

Pour vérifié l'efficacité d'une telle méthodologie, nous allons essayer de configurer notre système flou de tel manière qu'il puisse prendre en charge le management thermique des composants IPs d'un SoC, en incluant dans ces derniers des mécanismes de reconfiguration dépendent de la température on adaptant leurs paramètres fonctionnels au nouveau mode d'opération préconiser par le contrôleur flou.

Ce mémoire se décompose en quatre chapitres. Le premier chapitre décrit le contexte de la conception et du test des systèmes sur puce. Cette description montre l'évolution de la conception des circuits numériques. Cette étude permet d'exhiber la complexité des systèmes sur puce et de leur conception.

Le deuxième chapitre propose une introduction aux architectures reconfigurables et leurs classifications, il est aussi consacré la présentation des récentes avancées technologiques des systèmes reconfigurable sur puce.

Le troisième chapitre de ce mémoire est dédié à une présentation générale de la logique floue et il dresse un panorama des solutions architecturales pour l'implémentation des systèmes flous, Nous traitons, en particulier, de façon générale, l'architecture et le fonctionnement des processeurs numériques.

Le quatrième chapitre aborde plus en détail la mise en œuvre d'un système flou reconfigurable, ainsi que les différentes étapes de la démarche et de la méthodologie de conception utilisée. À la fin de ce chapitre un exemple d'application du domaine du management thermique des systèmes sur puce est présenté.

Le mémoire s'achève par la conclusion et par la description des perspectives qu'offrent les travaux menés dans ce mémoire.

Chapitre I

SYSTEMES SUR PUCE (SOCS)

Introduction

*Ce premier chapitre réalise la présentation des systèmes sur puce (*System on Chip-SoC*), et montre les outils de conception qui permettent l'assemblage de composants virtuels appelés IP-core (*Intellectual property core*) fortement hétérogène sur la même puce.*

Dans un deuxième temps, ce chapitre présente les mécanismes de tests des SoCs les plus utilisés et les problèmes liés à leur intégration. La description de ces mécanismes et suivie par une présentation détaillée de la technique de test thermique et de la gestion thermique des systèmes intégrés.

I.1 Méthodologie de conception des SoCs

I.1.1 Système sur puce

Un SoC est un système dans un circuit intégré où il rassemble des IPs hardwares et softwares qui visent à définir la fonctionnalité et le comportement d'un système donné.

Dans la plupart des cas le système désigne une application spécifique, qu'on peut trouver dans des systèmes complexe tel que les systèmes de télécommunication, de traitement de donnée et dans d'autre secteur de l'industrie, l'exemple présenté sur la Figure I.1 présente une des applications des SoCs, qui décrit les communications via protocole internet (Voice Over IP - VOIP). (1)

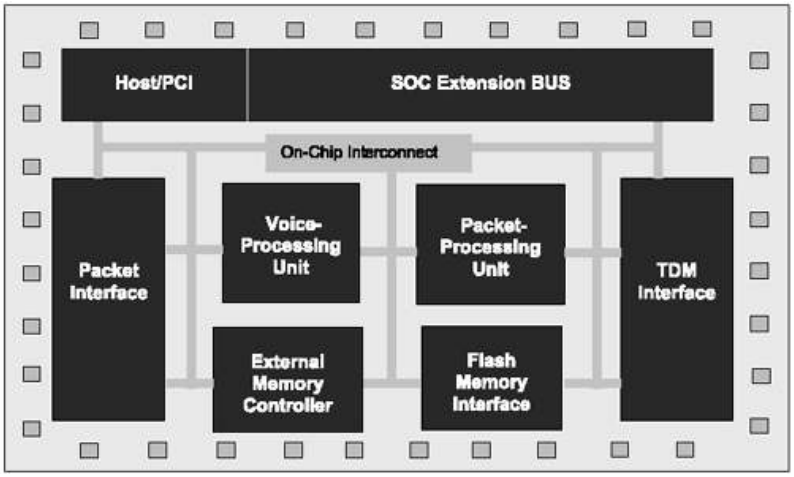


Figure I.1 : Architecture typique des SoCs

Les systèmes sur puce sont composés de parties logicielles et matérielles. Afin de décrire un tel système de manière plus explicite, la Figure I.2 donne une représentation de la structure en couche d'un tel système. Dans cette structure on note qu'il y a quatre éléments principaux (2):

- Le logiciel applicatif.
- Les processeurs.

- Les blocs matériels.
- Le réseau de communication.

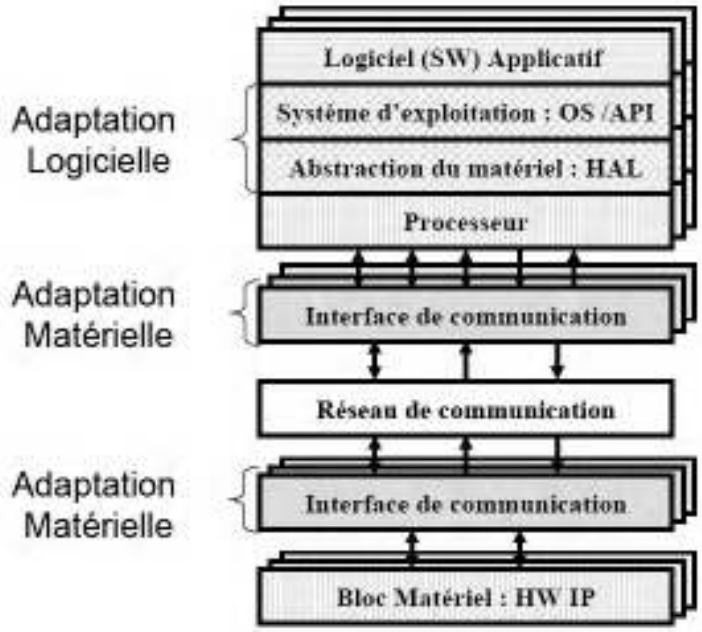


Figure I.2 : Représentation en couche de la structure d'un SoCs

D'autre part, lors de la conception d'un système sur puce, il est nécessaire pour effectuer l'intégration de tous ces éléments de réaliser à la fois des adaptations logicielles et des adaptations matérielles (Figure I.2). L'adaptation logicielle réalise l'interface entre le logiciel applicatif et le processeur. L'adaptation matérielle réalise quant à elle l'interface pour la communication entre les différents composants matériels et le réseau de communication.

I.1.2 flot de conception des SoCs

Ces méthodes reposent sur l'orthogonalisation de différents aspects qui permet une plus large exploration de l'espace des solutions d'implémentation (3). L'orthogonalisation concerne : (1) la fonctionnalité et l'architecture, (2) les calculs et la communication et (3) les besoins/contraintes et les choix d'implémentation. Cette dissociation requiert la définition de modèles, de niveaux d'abstraction, de méthodes permettant le passage d'un niveau de raffinement à un autre et de flots de conception adaptés. Un flot de conception repose donc sur un ensemble d'étapes permettant de fournir, à partir d'une spécification initiale, l'implémentation d'une application. La complexité de ce problème de transformation nécessite un processus de raffinement incrémental et la définition de modèles intermédiaires d'abstraction adéquats. Un flot de conception est constitué d'une suite d'alternances de phases d'exploration à un niveau d'abstraction et de raffinements entre niveaux d'abstraction. Cette gestion par palier permet de réduire la quantité d'information que le concepteur doit gérer à un niveau donné évitant ainsi les interactions et effets de bords avec les niveaux inférieurs.

Les nombreuses solutions apportées par la recherche académique ou industrielle peuvent être regroupées en trois familles (3) : La synthèse à partir d'une spécification système fonctionnelle (*Synthesis from specification* ou *system synthesis*), la conception s'appuyant sur l'utilisation de plate-forme (*Platform-Based Design*) et la conception basée sur la réutilisation de composant (*Component-Based Design* ou *IP assembly*).

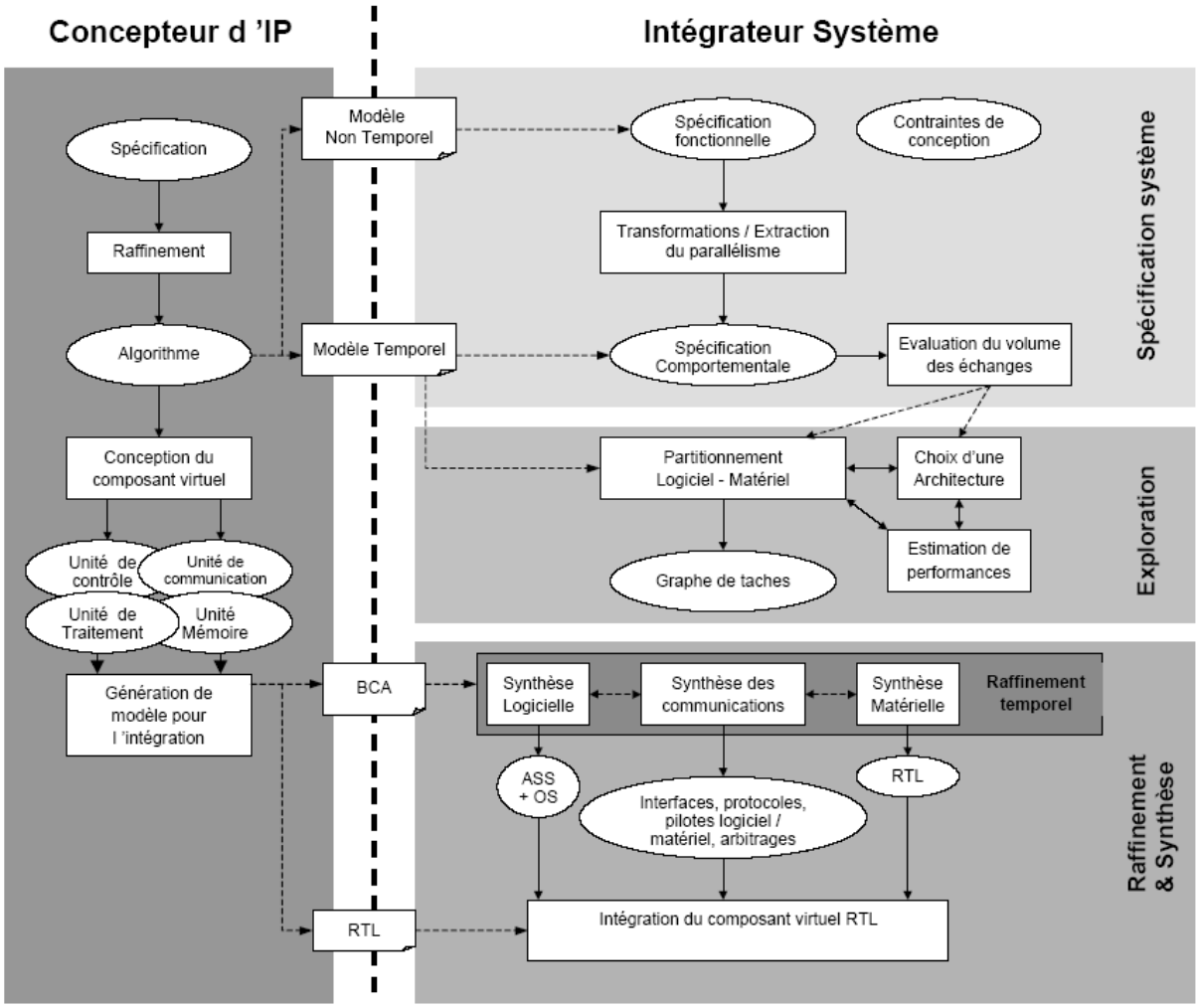


Figure I.3 : Flot de conception d'un SoC pour l'intégration des IPs.

La Figure I.3 présente les différentes étapes du flot de conception d'un SoC réutilisant des composants virtuels, allant de la spécification jusqu'à la synthèse ; pour lesquelles des phases de vérification et des rebouclages sont nécessaire pour la validation du SoC. Le flot de conception n'est pas unique mais chaque firme dispose de ses propres méthodes de développement et de réutilisation, il est cependant important de noter qu'en fonction de la méthodologie, les phases de conception peuvent ne pas être dissociées. Par exemple, dans la Figure I.3 l'étape de synthèse et de raffinement des communications succède la phase d'exploration architecturale. Or dans certaines méthodologies ces deux étapes sont réalisées conjointement.

I.1.3 Composant virtuel

Les composants virtuels aident à la conception par utilisation de blocs préconçus, qui sont souvent vendus en tant que cœurs IP (*Intellectual property*). C'est le principe de réutilisation (*design reuse*), qui contribue à augmenter la productivité et à diminuer les temps de conception (*time to market*) ; des facteurs cruciaux dans des contextes industriels à forte compétitivité. Le marché de la téléphonie mobile en est un exemple caractéristique, où un constructeur doit proposer une nouvelle génération de terminaux tous les 6 mois en moyenne (4).

Les SoCs fournissent d'ailleurs des solutions de choix pour une gamme extrêmement large de compromis coûts/performances. Les cœurs actuels peuvent être de trois types distincts :

- **Les Soft cores** : ce sont des composants décrits dans un langage de description matériel (*HDL : Hardware Description Language*) synthétisable. Ils sont indépendants de toute technologie et sont donc très souples d'utilisation. En contre partie, la phase de synthèse dépend de l'outil et elle est à la charge du concepteur du système.
- **Les Firm cores** : sont des composants ayant subis une optimisation en surface et vitesse par des techniques de placement relatives aux outils de conception utilisé dans l'intégration des SoCs. Ces composants sont décrits en HDL structurel, ils font appel à des blocs élémentaires d'une librairie générique. Cette implantation autorise des évaluations plus fines des ressources utilisées et des performances.
- **Les Hard cores** : sont des composants optimisés et placés-routés pour une technologie donnée. Le bloc est donc implicitement caractérisé en termes de performances et de surface.

Tableau I.1 : Caractéristiques des niveaux de description.

	Flot de conception	Représentation	Librairies	Technologie	Portabilité
Soft Non prédictible Très flexible	Conception système	Comportementale		Indépendante	Illimitée
	Conception RTL	RTL			
Firm Flexible Prédictible	Synthèse du floor plan	RTL + Placement de sous-blocs	Librairie de référence	Générique	Library mapping
	Placement	Netlist	Modèle de temps et d'interconnexions		
Hard Pas flexible Très Prédictible	Vérification du routage	Polygones	Spécifique au process Règles de dessin	Fixe	Process mapping

Le Tableau I.1 récapitule les caractéristiques de chacune de ces familles de blocs IP. On peut noter que le rapport performances/flexibilité est constant. Plus un bloc sera proche de la technologie cible, plus il sera performant et moins il sera flexible.

Ce concept de réutilisation matériel a tendance à se généraliser aujourd'hui, la recherche de performances électriques optimisées ou/et de temps de conception réduit conduit à la réutilisation de composants "virtuels". Dans ce contexte, de nombreuses contraintes additionnelles rendent le recours aux systèmes sur puce quasi-impératif :

- **La compacité :** La course à la miniaturisation des systèmes actuels exclut le recours systématique à l'utilisation d'un circuit par fonction et impose donc le regroupement de celles-ci sur un même circuit.
- **La consommation :** Les puissances à mettre en jeu pour transmettre des signaux entre différents circuits intégrés sur le circuit imprimé (*PCB : Printed Circuit Board*) sont largement plus importantes que celles requises pour des communications sur le même substrat de silicium. Conjointement, la diminution de la taille des éléments de puissance imposée par la miniaturisation nécessite une réduction de la consommation, pour assurer une autonomie au moins constante.
- **Les performances :** les fréquences envisageables sur des protocoles de communication inter-circuit (sur le PCB) sont bien inférieures à celles autorisée sur silicium, le regroupement des fonctionnalités sur un même substrat devient alors un impératif, surtout quand la partie numérique du traitement n'est plus prise en charge par une seule unité de traitement.

I.2 Les architecture de communication

Un point clé des architectures des SoCs est l'utilisation de modèles génériques d'interface pour adapter les composantes au réseau de communication sur puce. Ces interfaces agissent en tant que coprocesseurs de communication, permettant de dissocier les CPUs et les IPs du réseau de communication. Ces interfaces de communication peuvent être très simples ou très sophistiquées selon le composant à adapter et le réseau de communication. Il peut également être vu comme un pont entre le bus interne du CPU (ou IP) et le réseau de communication. Aussi, les protocoles de communication qui peuvent être employés n'étant pas limités à un modèle spécifique, l'interface de communication peut supporter des protocoles divers et complexes de communication.

Chaque interface de communication se compose de trois parties (Figure I.4) :

1. Adaptateur de composant.
2. Adaptateurs de canaux de communication.
3. Bus interne reliant ces deux derniers. Cette interface agit comme un pont (*bridge*) entre le composant et le réseau de communication.

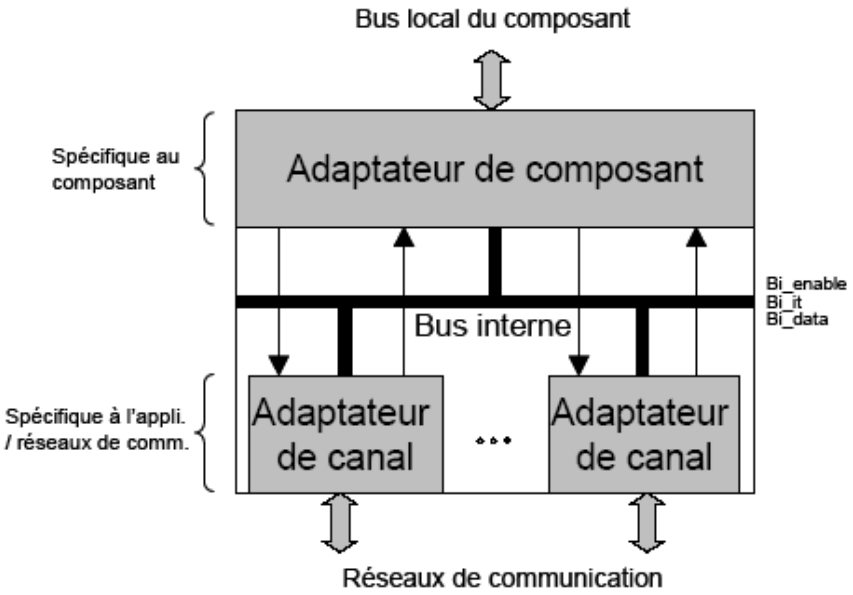


Figure I.4 : Interface de communication.

Le réseau de communication et la colonne vertébrale du SoC, il permet de réaliser les échanges de données entre les différents composants du système. Ce réseau de communication est généralement spécifique au système. Il existe un grand nombre de réseaux de communication pouvant être réalisés par :

- Un ensemble de connexion point à point ;
- Un ensemble de bus tel que le bus système AMBA-AHB™, STBus™, Sonics™…
- Un réseau sur puce (*NoC: Network on Chip*).

Dans notre étude, nous nous intéressons on particulier au bus intégré (*on-chip buses*), dans ce qui suit nous allons décrire les bus les plus utilisés dans l'intégration des systèmes sur puce.

I.2.1 Le bus AMBA

Les communications sont principalement réalisées par le bus AMBA (5). Ce bus a été conçu par ARM pour les systèmes sur puce, il existe en fait trois types de bus : AHB, ASB, APB. Le bus AHB accepte plusieurs maîtres, à des performances élevées et permet des transferts de données par paquets (*burst mode*). Le bus ASB possède les mêmes caractéristiques mais ne permet pas les transferts par paquets. Le bus APB est plus simple et moins performant. Il n'autorise qu'un seul maître, mais il consomme moins d'énergie. La Figure suivante illustre une configuration typique d'un bus AMBA.

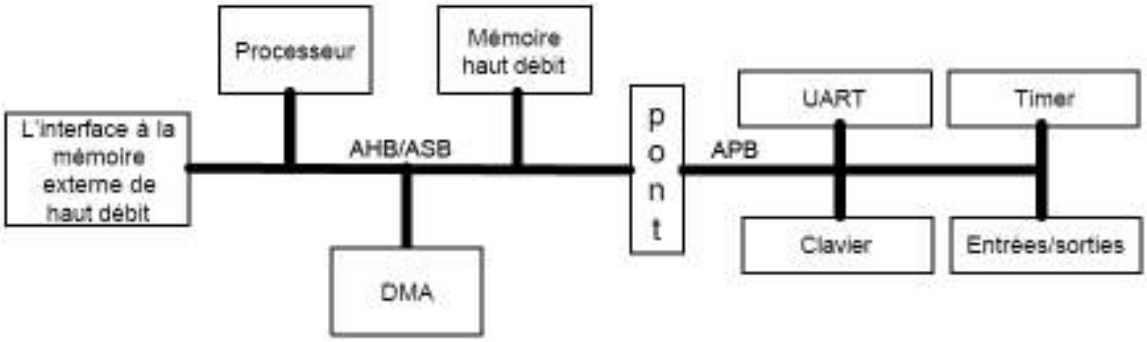


Figure I.5 : La configuration typique du bus AMBA.

I.2.2 Le bus CoreConnect

Le CoreConnect d'IBM est une architecture de bus embarqué disponible sous un accord de licence libre d'IBM (6). Les composants qui sont connectés au bus doivent être conformes à la norme *Blue Logic* d'IBM. Son architecture hiérarchique fournit trois bus synchrones (Figure I.6).

- Processor Local Bus (PLB).
- On-Chip Peripheral Bus (OPB).
- Device Control Register (DCR) Bus.

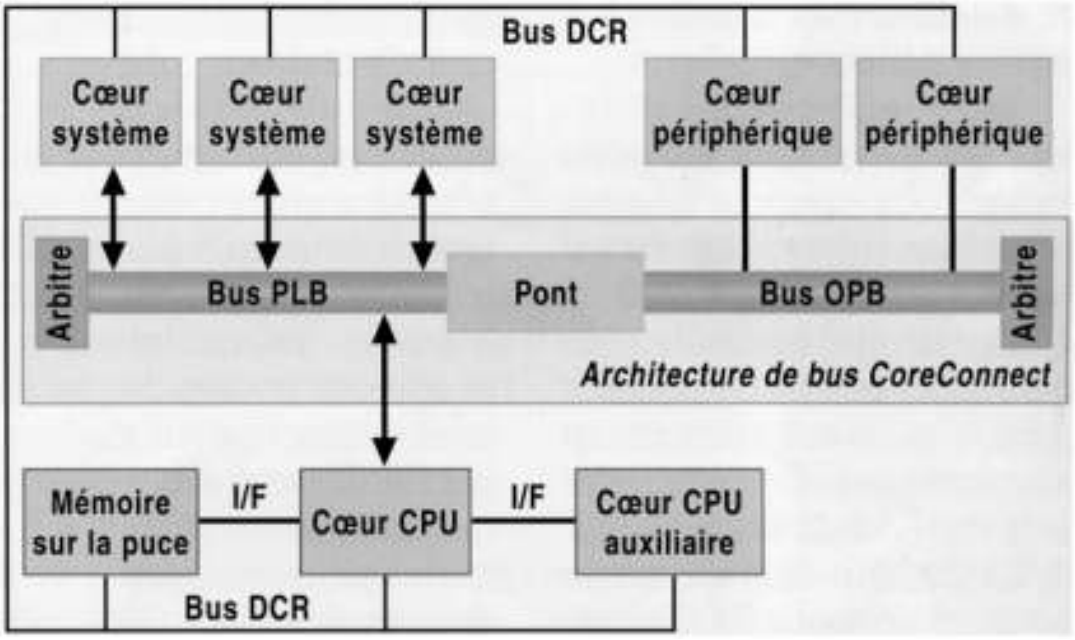


Figure I.6 : Modèle architectural du CoreConnect.

L'architecture du PLB est très semblable à celle de l'AHB d'AMBA avec des bus séparés de lecture/écriture de données, multi-maître, pipeline, " *split-transactions* " et les transferts en rafale. Son but est de fournir une performance élevée, une faible latence, et une flexibilité de conception en connectant des composants de haut débit, tels que des CPUs,

des interfaces externes de mémoire et des contrôleurs DMA. La largeur des données est de 32 ou 64 bits, extensible à 128 et à 256 bits.

I.2.3 Le bus Wishbone

Le bus d'interconnexion Wishbone est une méthode pour connecter des composants numériques sur un circuit intégré, les spécifications utilisées pour ce bus différent de celle des autres bus qu'on a déjà vu, elles sont régies par des règles, des recommandations, des suggestions et des observations. Sa permet au Wishbone d'être simple, libre et hautement configurable (7) (Figure I.7).

À cause de ses interfaces configurable, l'utilisateur doit crée ses propres standards Wishbone, spécifié l'ordre des données et autres paramètres, ce qui cause déjà la non-généricité avec d'autres règles de conception.

Pour qu'un module soit certifié compatible avec Wishbone, la spécification exige que de la documentation soit produite pour le module (un "*Wishbone Datasheet*"). La documentation nécessaire doit respecter un certain format et contenir certaines informations précises. Cette approche est intéressante, elle impose une discipline au concepteur, le forçant à documenter son travail. La réutilisation du module en est simplifiée parce que l'intégrateur n'a pas à deviner son fonctionnement.

Une autre particularité du protocole est qu'il énonce sous formes de règles précises les spécifications à respecter. En parcourant la liste des règles, on peut cerner rapidement si un module respecte la spécification ou non. Cette liste claire des règles permet au concepteur d'apprendre le protocole rapidement.

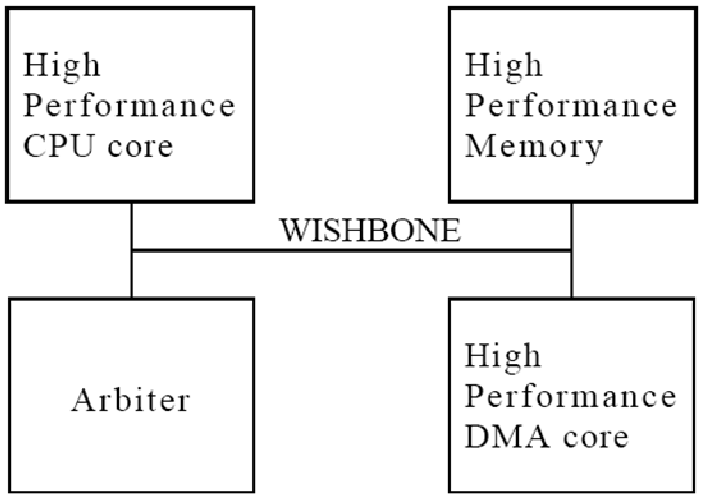


Figure I.7 : Structure du bus wishbone.

Opencore recommande l'utilisation du Wishbone pour l'interfaçage des IPs sur les FPGAs et les ASIC, c'est un standard libre et il appartient au domaine public (8).

Détails techniques

- Un seul bus pour toute l'architecture.
- Une architecture simple et compacte.
- Possibilité de plusieurs maîtres.
- Une espace d'adressage de 64 bits.
- Un bus de donnée de 8 et 64 bits (extensible).
- Un seul cycle de lecture et d'écriture.
- Supporte l'interfaçage aux mémoires.
- Arbitrage défini par l'utilisateur.

I.3 Test des SoCs

Plusieurs difficultés émergent lors du test des SoCs. Des problèmes apparaissent dès la conception de tels systèmes où les questions de testabilité doivent être prises en compte. Les problèmes rencontrés sont l'accessibilité aux cœurs, le temps de test du système, le test mixte, la puissance dissipée au cours du test, le partage des ressources de test. Sur un système sur puce, il n'y a pas de test individuel des cœurs avant l'assemblage. C'est à l'intégrateur de s'occuper du test des cœurs et non au fournisseur de cœur.

Généralement les cœurs sont fournis avec un test intégré (*Built In Selft Test-BIST*), et éventuellement une interface compatible boundary scan (9).

II.3.1 Architecture de test des SoC

D'une façon générale l'architecture de test d'un système intégré sur une puce peut se décomposer en trois principaux éléments comme le montre la Figure I.8 : les "source/sink", le "wrapper" et le TAM (10).

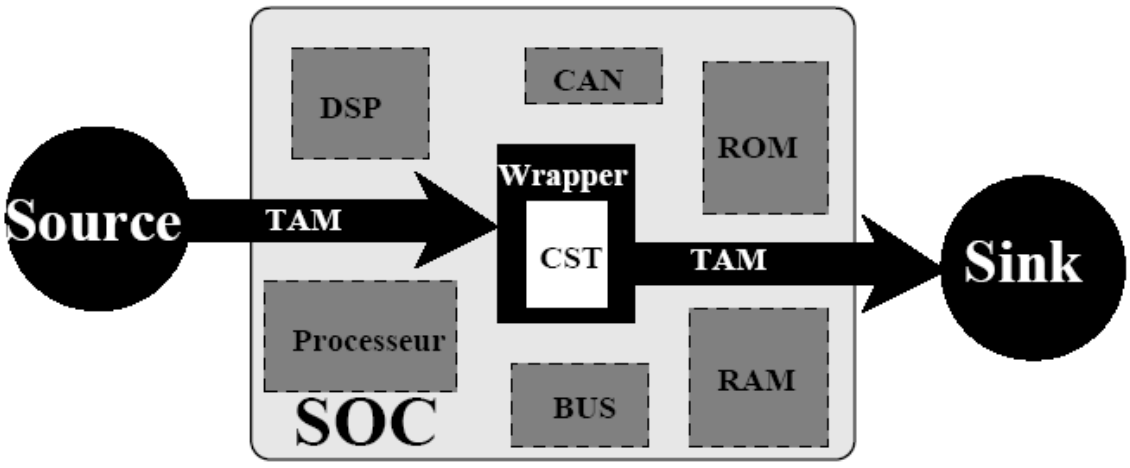


Figure I.8 : Architecture conceptuelle pour le test des SoCs.

a) Les "source" et "sink"

Le "source" est le terme désignant le matériel fournissant les vecteurs de test. Il peut s'agir d'un testeur industriel quand le "source" est externe au circuit ou d'un générateur de type BIST quand le "source" est interne. Parallèlement au "source" le "sink" correspond à l'analyseur de réponses aux stimuli appliqués. De même il peut être interne ou externe au SoC, sous la forme d'un analyseur de signature ou d'un testeur.

a) Le wrapper

Pour que le cœur soit intégrable facilement dans le système embarqué, le fournisseur doit délivrer son composant virtuel avec une interface de test, interface appelé communément "wrapper". Cette interface doit permettre l'accès du test aux entrées/sorties du cœur. Elle permet d'isoler électriquement l'IP quand le SoC est en mode test. Il est ainsi protégé des perturbations qui peuvent être provoquées par les IPs voisins. Le concepteur d'IP peut aussi grâce à ce wrapper implémenter les modes de contrôle du cœur. Les principaux modes de fonctionnement du bloc virtuel sont le mode normal, le mode test interne et le mode test externe. Ce dernier correspond au test des interconnexions entre IPs au niveau du SoC. Le wrapper permet donc les échanges de données de test entre le TAM et le circuit sous test.

b) Le TAM

Le TAM (Test Access Mechanism) est un terme qui englobe les moyens mis en œuvre pour transporter les stimuli et réponses de test des entrées/sorties du SoC aux entrées/sorties de chaque élément du circuit (cœur et UDL). Le choix du TAM dépend des "source/sink", des types de cœurs intégrés et des contraintes physiques imposés à l'intégrateur système.

I.3.2 Test thermique

La miniaturisation et la densification des circuits intégrés rendent les problèmes liés à la température de plus en plus importants. D'après les récentes statistiques (11), la plus grande partie des défauts sont issus d'un surchauffement, dut à une conception thermique inappropriée.

Les techniques de test et de monitoring thermique existent pour deux raison. La première est qu'elles sont efficaces pour l'analogique aussi bien que pour le numérique. Pour accédé au point chaud du circuit nous avons besoin d'intégré des capteurs ou de les fixés sur la surface de la puce, de ce fait ces capteurs sont toujours indépendants de la fonction du circuit. La seconde et que ces techniques n'ont aucune incidence sur les performances et la qualité du circuit (12).

Chaque circuit intégré à sa propre signature thermique, où son altération révèle dans la plupart des cas un disfonctionnement du circuit (Figure I.9).

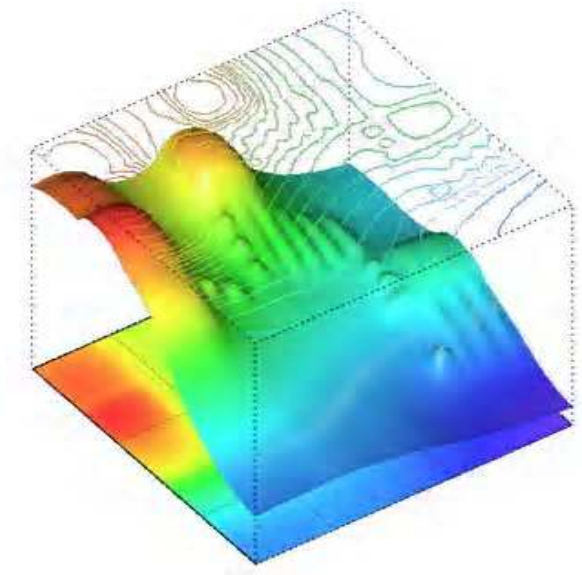


Figure I.9: signature thermique d'une puce

Plusieurs approches existent concernant la manière et le but de l'utilisation des techniques thermiques, dans les paragraphes suivants nous allons discuter quelques une de ces approches.

I.3.2.1 DFTT (Design for Thermal Testability)

Pour accroitre la fiabilité du système, la température du fonctionnement du circuit intégré devra être surveillée en permanence, ce qui nécessitera l'ajout de circuiterie intégrée de test thermique, c'est l'idée de la conception en vue du test thermique (*DFTT*). Dans ce cas s'il est signalé qu'il y a augmentation de la température au delà d'une certain limite dite de sécurité, une unité de contrôle peut immédiatement agir en diminuant la vitesse de fonctionnement ou en isolant des composants du système susceptibles d'être la source d'un mauvais fonctionnement ou des composant dont on connaît leur consommation excessifs de puissance (13) (14).

Comme pour le test électrique des circuits intégrés, les principales stratégies de test sont comme suit :

b) Built-In Temperature sensor

La difficulté dans la conception des capteurs intégrés est que les défauts ne peuvent être localisés dans le fonctionnement normal du système (pas de surchauffe). Dans ce cas les défauts ne peuvent être détectés dans les conditions normales de travail du circuit, alors le capteur doit intégrer une self-excitation pour générer périodiquement des conditions particulières (ex : un fort courant), puis les données relevées peuvent êtres collectées et traitées pour faire le diagnostique du système.

c) Boundary Scan (BS)

Cette technique et la même utilisé dans le test électrique (9), et elle convient très bien pour le test thermique, seulement après l'ajout d'un mécanisme d'adaptation pour la capture de température, un capteur de température fréquentiel par exemple. Un compteur de 12 à 14 bits est nécessaire, on plus de la circuiterie du BS (Figure I.10). Quelques instructions sont à définir pour activer et désactiver le comptage et pour scanner la température. L'avantage de cette méthode est qu'elle n'a pas besoin de pins supplémentaires, l'utilisation des ressources boundary scan sont largement suffisantes (15).

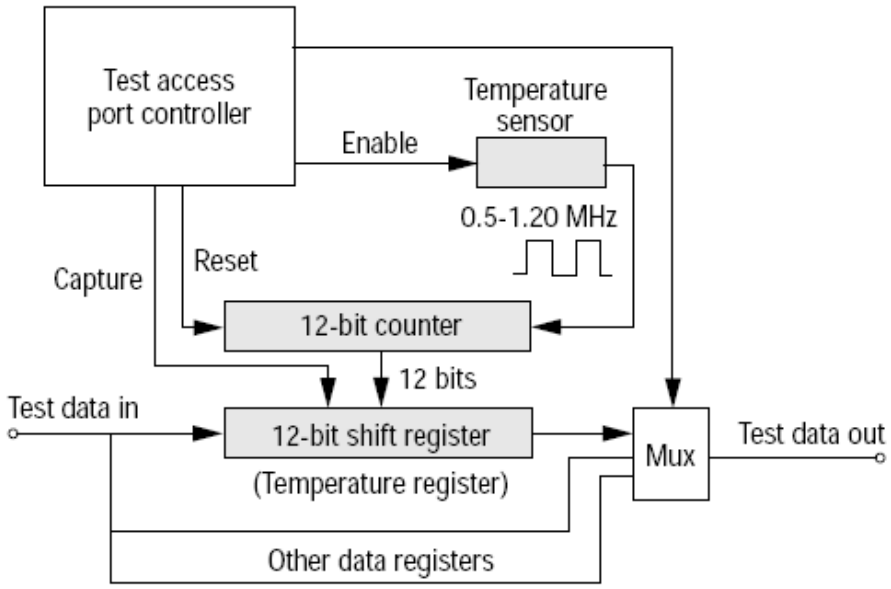


Figure I.10 : Block supplémentaire du BS pour le test thermique.

La mesure de la température peut être accomplie durant le test en production ou durant le test fonctionnel du circuit, dans les deux cas le traitement des données récupérées est effectué par un dispositif externe au circuit intégré.

I.3.2.2 Test scheduling

Le surchauffement et connu comme étant le problème majeur durant le test des SoCs, de récentes étude ont montrées que la puissance consommée durant le test peut augmentée jusqu'à 30 fois par rapport à la consommation dans le fonctionnement normal du système (16). Des contraintes de puissance sont alors fixées pour pallier à ce problème, mais elles ne sont toujours pas suffisantes pour éliminer les points chauds (*Hot Spot*), parce que elles ne prennent pas en considération la non uniformité de la consommation des IPs et celle de la dissipation de la température dans le SoC, aussi la simultanéité du test des IPs n'est pas considérée (Figure I.11).

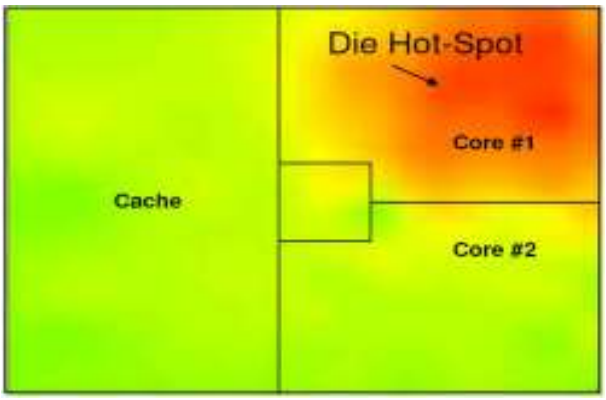


Figure I.11 : Comportement thermique d'un SoC pour un seul IP active.

Alors d'autres approches ont été proposées pour ordonnancer le test des IPs suivant leurs caractéristiques thermiques pour garantir une fiabilité de test et de fonctionnement optimale (16).

I.4 Management thermique

Les disfonctionnement d'origine thermique peuvent être évités par un management thermique, ou la gestion de la température commence lors de la conception par la simulation électrothermique du circuit qui fournira des informations sur le meilleur emplacement des composantes de ce dernier (17).

Au cours du fonctionnement normal des circuits intégrés on prévoit l'utilisation d'un ou plusieurs capteurs thermique qu'y sont implantés, ou mis à l'extérieur du circuit intégré. Habituellement ces techniques sont appliquées sur les micro-processeurs, où la température est maintenue dans un niveau admissible par : un refroidissement externe, le réglage de la fréquence d'exécution ou par l'asservissement de la tension d'alimentation.

Concernant la gestion thermique dans les systèmes sur puce plusieurs études ont été menées dans le but d'utiliser les techniques déjà existantes pour les micro-processeurs en les adaptant aux architectures des SoCs, ces techniques se basent sur l'intégration de capteurs et d'unités de contrôle sous la forme de soft-IP.

La Figure I.12 présente l'intégration de système de gestion (*Thermal Aware Power Management-TAPM*) dans un SoC, le TAPM proposé dans (18) est conçu comme un module synthétisable écrit dans le langage de description matériel VHDL.

À l'intérieur du TAPM, une interface est conçue pour garantir la communication entre le TAPM et les autres IPs en liaison SMBus qui fournie un accès rapide au différents composants sans avoir à interrompre le fonctionnement du système.

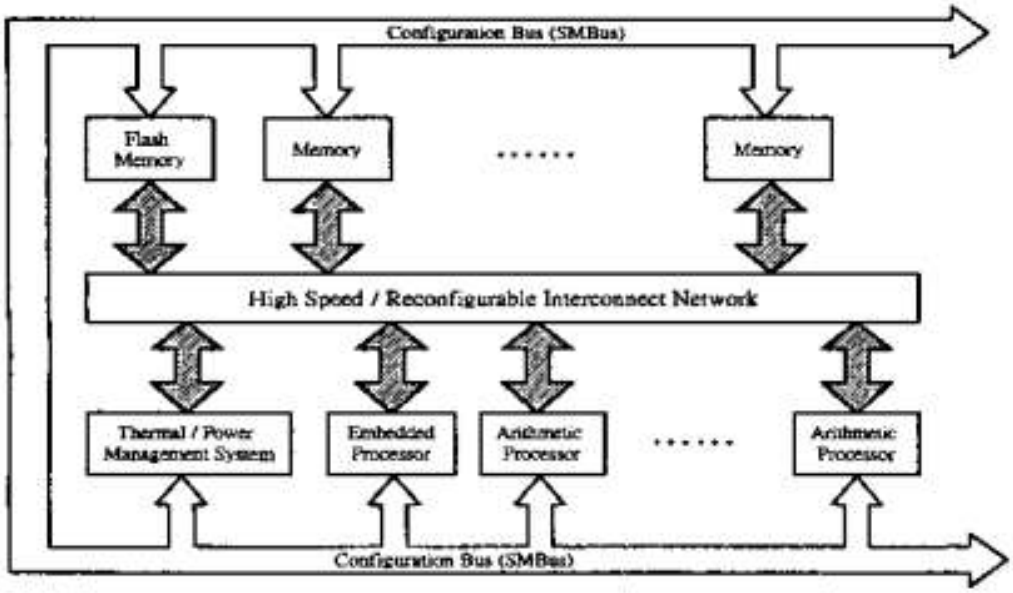


Figure I.12 : Architecture système du TAPM

La structure du TAPM est présentée sur la Figure I.13, le système est composé d'une unité de management thermique (*TMU*), d'un contrôleur de puissance et de refroidissement (*PC&AC*), d'un capteur de température et d'une interface. Le TMU peut être programmé par différents algorithme de gestion qui conviennent aux exigences thermiques du SoC.

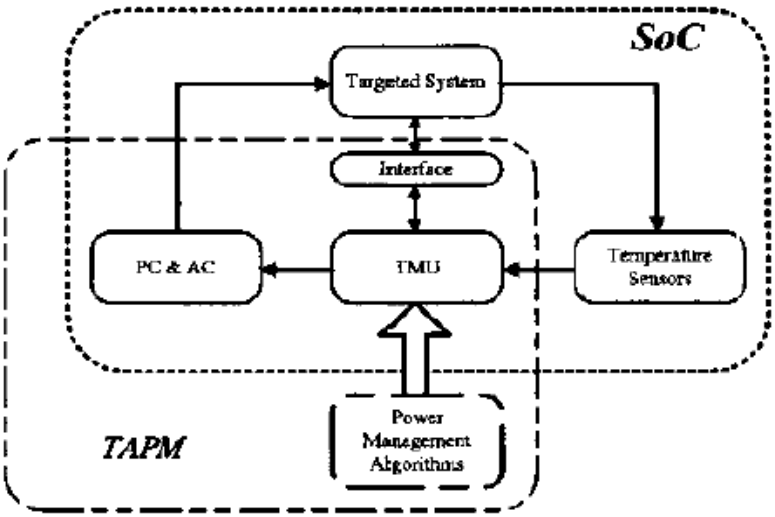


Figure I.13 : Éléments fonctionnels du TAPM.

Conclusion

Dans ce chapitre nous avons pu avoir un aperçu sur les architectures des SoCs où les différentes composantes de ces systèmes ont été présentées, aussi la méthodologie de conception a été introduite pour avoir une idée plus précise sur la complexité de leur implémentation. Finalement nous avons terminé ce chapitre par la présentation des

techniques de test et de gestion des SoCs, en particulier le management thermique qui sera le but de notre application.

Le développement des systèmes sur puce, a conduit à l'extension de leur domaine d'utilisation et leur adaptation à des applications particulières où leur reconfiguration est devenue indispensable, dans le chapitre suivant nous allons discuter de ce cas particulier des architectures des SoCs.

Chapitre II

SYSTEMES RECONFIGURABLES

Introduction

Une architecture est qualifiée de reconfigurable des lors qu'elle dispose d'un support lui permettant de s'adapter aux traitements qui lui sont assignées. Ces architectures se révèlent être des plateformes particulièrement adaptées pour les applications embarquées qui combinent le contrôle et le calcul intensif. Les performances de ces architectures restent souvent limitées par les délais de communication et de reconfiguration.

Dans ce chapitre, nous introduisons les principaux concepts des architectures reconfigurables, et les différents types de reconfiguration. Nous présenterons aussi une classification détaillée de ces systèmes en ce qui concerne leur granularité, leur structure et les plans de reconfiguration possible au niveau système, d'autres critères de classification sont aussi présentés.

II.1 Types de reconfiguration

II.1.1 Reconfiguration logique

Les architectures reconfigurables au niveau porte les plus répandues sont des architectures à base de FPGA (Figure II.14). Il existe deux types de technologies pour ces composants reconfigurables selon que la configuration est effectuée en utilisant des éléments *Anti-Fuse* ou des mémoires *SRAM*.

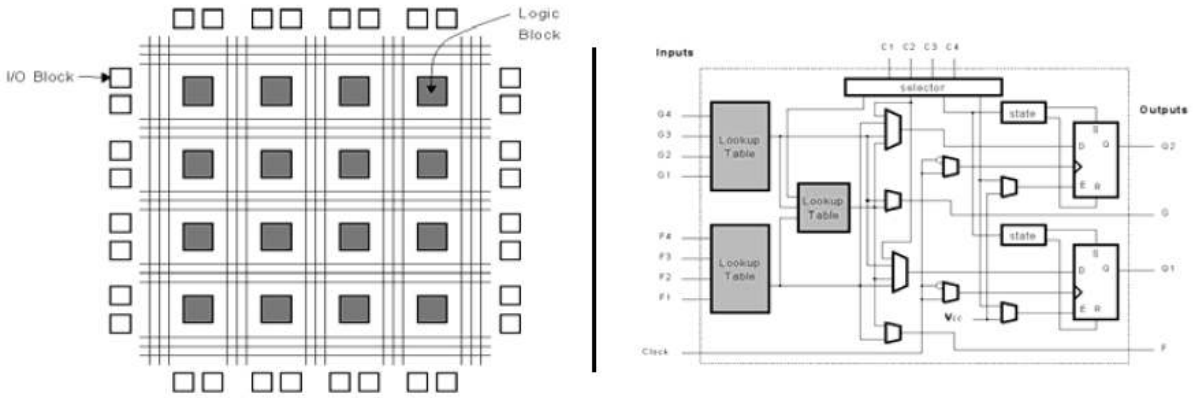


Figure II.14. Architecture générique d'un FPGA et de son bloc logique

La technologie *Anti-fuse* utilise un fort courant électrique pour créer la connexion entre deux fils et elle est généralement non-reprogrammable. La technologie basée sur les SRAMs, permet de reconfigurer le composant FPGA en modifiant la fonctionnalité des blocs logiques (CLB) et le réseau d'interconnexion par le chargement d'un flot binaire dans les cellules mémoire SRAM (19) (20).

II.1.2 Reconfiguration par jeu d'instruction

Le concept de ce type de reconfiguration se réfère aux architectures hybrides qui consistent en un microprocesseur lié à une logique reconfigurable. Son avantage est une combinaison de la souplesse du logiciel avec le haut rendement du matériel. Une des approches prometteuses de ces systèmes est les processeurs RISP (*reconfigurable instruction set processors*) qui ont la capacité à adapter leurs jeux d'instructions à l'application exécutée à travers une reconfiguration hardware. Le résultat est un reconfigurable et une architecture extensible du processeur qui pourraient s'adapté aux besoins spécifiques des concepteurs.

II.1.3 Reconfiguration statique et dynamique

Il existe deux importantes approches de reconfiguration : dynamique et statique (21).

II.1.3.1 Reconfiguration statique

La reconfiguration statique est l'approche la plus simple et la plus commune pour implémenter des applications avec une logique reconfigurable. La reconfiguration statique nécessite des changements hardwares à une vitesse relativement lente. C'est une stratégie de la mise en application statique où chaque application consiste en une configuration. Le principal objectif est d'améliorer les performances du système visé.

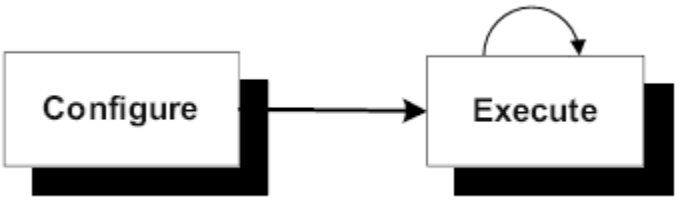


Figure II.15: principe de la reconfiguration statique

II.1.3.2 Reconfiguration dynamique

La reconfiguration dynamique utilise un plan de l'affectation dynamique qui réattribue le hardware au cours d'exécution. C'est une technique avancée et elle est considérée comme une réalisation flexible du compromis espace/temps. Elle peut augmenter les performances du système en utilisant des circuits hautement optimisés qui sont chargés et déchargés dynamiquement pendant le fonctionnement du système (Figure II.16). Dans cette reconfiguration la flexibilité du système est maintenue et la densité fonctionnelle du hardware est augmentée.

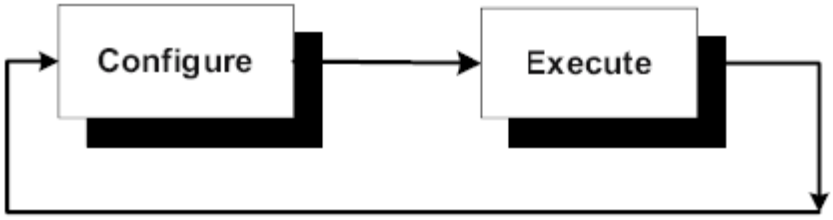


Figure II.16: principe de la reconfiguration statique

II.2 Classification architecturale des systèmes reconfigurables

Les architectures reconfigurables peuvent être classifiées selon différents paramètres. Dans cette section nous distinguons les classifications architecturales suivantes (22) :

II.2.1 Classification selon la granularité

La granularité d'un composant reconfigurable représente la taille du plus petit élément fonctionnel. Les FPGAs ont typiquement une faible granularité avec des unités fonctionnelles à deux ou à quatre bits. D'autres architectures reconfigurables implémentent des unités arithmétiques à plus gros grain.

Un composant reconfigurable est configuré en chargeant dans celui-ci une séquence de bits (ou *bit-stream*). Le temps de chargement est directement proportionnel à la taille de la séquence. Les architectures à gros grain nécessitent des séquences plus courtes que les architectures à grain fin donc un temps de configuration nettement plus faible.

Une faible granularité procure une plus grande flexibilité d'adaptation du matériel à la structure de calcul souhaitée mais elle entraîne un temps de configuration important. De plus, le grand nombre d'interconnexions entre les unités rajoute un délai conséquent à l'exécution du module implanté (malgré l'existence de structures permettant de réduire ce délai, comme les chaînes de propagation de retenue que l'on peut trouver maintenant sur quelques FPGA).

II.2.2 Classification selon le schéma de reconfiguration

II.2.2.1 Architecture statiquement reconfigurable

Les architectures reconfigurables traditionnelles sont statiquement reconfigurables (*compile-time reconfiguration : CTR*), où les ressources de reconfiguration sont lancés au démarrage de l'exécution et restent inchangé durant l'application. Afin de reconfigurer ce type d'architecture, le système doit être arrêté pendant la reconfiguration est relancée avec la nouvelle configuration.

Les architectures FPGA traditionnelles ont été programmées à l'origine de façon sérielle avec un seul contexte, où seulement une configuration peut être chargé à la fois. Ce type de

FPGAs est programmé en utilisant un flux sériel de données, il nécessite la reconfiguration totale pour tout changement.

II.2.2.2 Architecture dynamiquement reconfigurable

Les architectures dynamiquement reconfigurables (*run-time reconfiguration : RTR*) permettent de procéder à la reconfiguration et à l'exécution en même temps. Les différentes reconfigurations dynamiques sont représentées sur la Figure II.17, et elles sont discutées dans les paragraphes suivants.

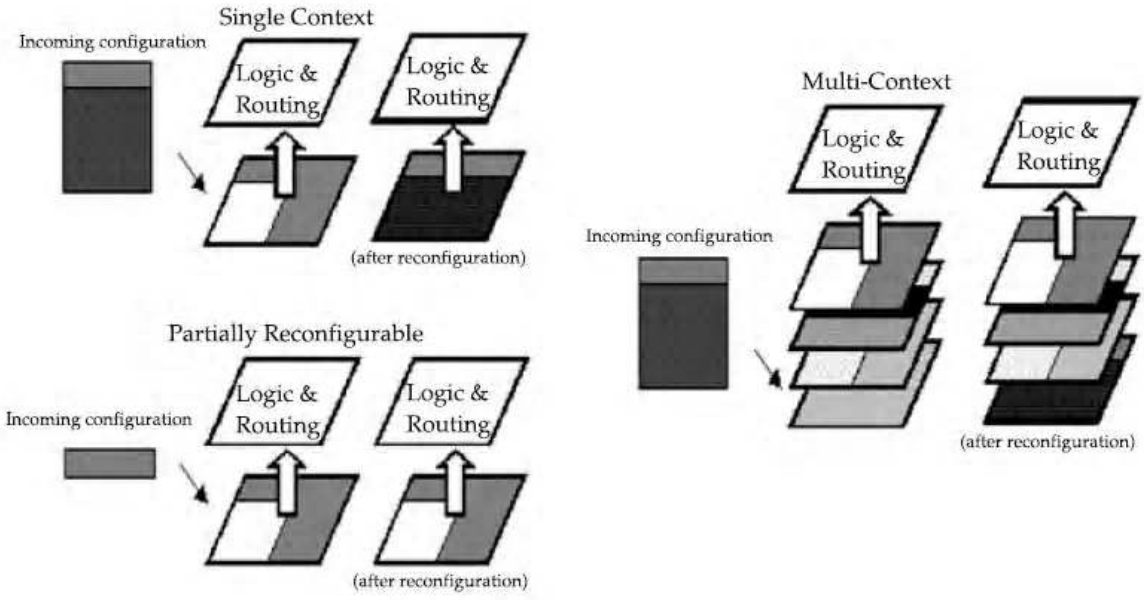


Figure II.17: Les différents modèles de reconfiguration dynamique

La reconfiguration dynamique à contexte unique

Bien que les architectures à contexte unique puissent être reconfigurées statiquement, une reconfiguration dynamique sur FPGA peut aussi être implémentée. Où les configurations sont groupées dans des contextes et chaque contexte est échangé si besoin. Il faut prêter attention à ce que les configurations entre les contextes doivent être bien partitionnées pour minimiser les délais de reconfiguration.

La reconfiguration dynamique multi-contexte

L'architecture reconfigurable à contexte multiple possède plusieurs plans de configuration. Un plan de configuration unique peut être activé à un certain moment mais le passage à un autre plan peut être très rapide (de l'ordre de quelques nanosecondes). Elle consiste à stocker un nombre de configurations dans des buffers (généralement en mémoire locale du composant) et de changer de contexte pendant l'exécution en un temps assez faible par rapport à un chargement de contexte à partir d'une mémoire externe. L'architecture Chameleon par exemple, possède un cache permettant de stocker une configuration et de reconfigurer le composant à partir de ce cache en un seul cycle (23).

Les architectures reconfigurables partiellement

Dans les architectures à reconfiguration partielle, la couche de programmation sous-jacente opère comme une RAM. En utilisant des adresses pour spécifier l'emplacement visé des données de configuration. Deux cas sont possible selon que le reste des segments non-reconfiguré continu ou non leur exécution :

- Configuration partielle fixe.
- Configuration partielle active

Les architectures à reconfiguration partielle permettent de reconfigurer sélectivement une zone de l'architecture pendant que le reste du composant poursuit son fonctionnement. Elle est basée sur le concept de "matériel virtuel" qui est similaire à l'idée de mémoire virtuelle. Plusieurs contextes peuvent être actifs à la fois sur le composant selon le nombre de zones actives. Le temps de reconfiguration d'un contexte sur une zone de ce composant est masqué par l'exécution d'un autre contexte sur une autre zone.

II.3 classification selon le mode de traitement

Le type de traitement peut être considéré comme une caractéristique fondamentale du système, puisque il influence les autres caractéristiques des noyaux de ce dernier. D'après ce raisonnement, une classification générale des architectures reconfigurables basées sur le type de traitement comme principal critère est donnée (24).

II.3.1 Architecture orientée chemin de donnée

Les architectures à chemin de donnée visent des noyaux dominés par la logique combinatoire, et l'arithmétique en particulier. Dans de tels noyaux, les opérations sur les blocs à grain grossier et les opérations avec la propagation de retenu sont typiques. Ce type de traitement est particulièrement adapté au traitement du signal numérique (DSP).

La reconfiguration ne s'effectue plus seulement sur les interconnexions inter-modules, mais également à l'intérieur des modules, on ne considère que la reconfiguration du chemin de données. Les opérateurs que contient le modules ne sont par configurables mais sont optimisés, ce qui permet une augmentation des performances et une diminution de la flexibilité de l'architecture par rapport à une architecture classique de type FPGA.

II.3.2 Architecture orientée traitement logique

Ces architectures sont destinées à l'origine pour le mappage de gros bloque de logique combinatoire, comme le traitement logique fusionné avec les chemins de donnée. Et comme les éléments du traitement logique sont utilisés pour implémenter des éléments arithmétiques, les architectures orienté traitement de donnée devraient valider le mappage de fonctions arithmétiques avec la propagation de retenu.

Les architectures reconfigurables du type traitement logique conviennent à l'implémentation d'accélérateur matériel dans les applications de cryptographie, par exemple.

II.3.3 Architecture orientée mémoire

Les architectures orientées mémoire visent les noyaux de traitement qui nécessitent des ressources de stockage, ces ressources sont nécessaire parce que :

- Les besoins des algorithmes : les mémoires appliquent des délais au moyen de stockage local (registre, LUT..) ou par des éléments à délai variable (FIFO, buffer..).
- Les besoins de l'implémentation : les éléments mémoires dépendent de la méthode d'implémentation.

D'habitude, la mise en place des noyaux orientés mémoire exigera aussi le flot de donnée et de la logiques câblées. Par conséquent, pour garantir l'efficacité d'ordonnancement de tels noyaux, ces architectures devront être à fonctions multiples ou à usage général.

II.4 Conception des SoCs reconfigurables

L'augmentation de la complexité des algorithmes de traitement du signal et de contrôle, notamment dans les applications embarquées, nécessite de grandes puissances de calcul pour satisfaire de nouvelles contraintes. Ceci conduit les concepteurs à s'orienter de plus en plus vers des architectures matérielles composées de différentes unités de calcul opérant en parallèle, présentant des capacités de calcul différentes et permettant de grandes optimisations, c'est le cas des systèmes sur puce ou (SoC) (25) (21). Les composants présents dans ces systèmes peuvent être des unités de calcul programmables, des unités de calcul spécialisées ou reconfigurables ou encore des chemins de données spécialisés.

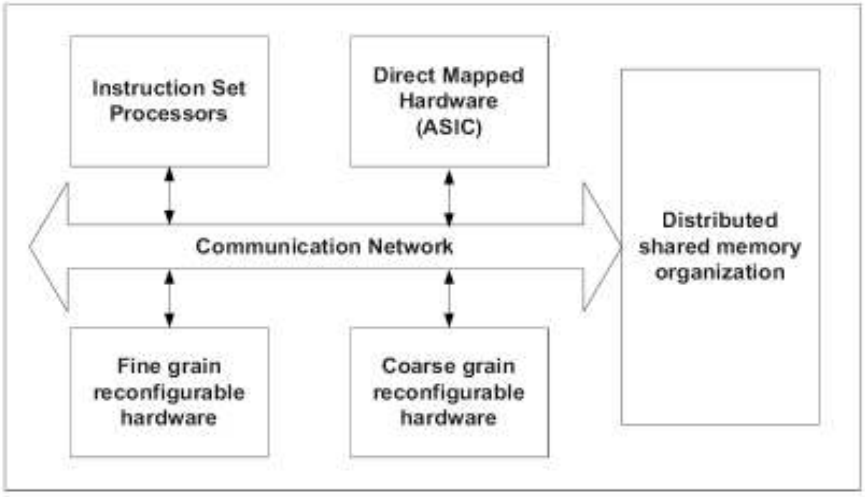


Figure II.18 : une vue globale des SOCs reconfigurable

En particulier, la présence d'unités reconfigurables, ou ARD (*Accélérateur Reconfigurable Dynamiquement*), permet d'adapter l'architecture à différents traitements. Un accélérateur reconfigurable au sein de ce type de plateforme permet d'apporter un degré important de flexibilité à l'architecture.

II.4.1 Les exigences au niveau système pour les SoCs reconfigurable

L'introduction des ressources reconfigurables dans les système-sur-puce a crée le besoin de modification et de prolongement des flots de conception classiques, principalement dans les niveaux d'abstraction plus élevé où les décisions les plus importante sont prises.

Comme il a été noté dans le 1er chapitre, habituellement, il est convenu que le flot de conception générique des systèmes intégré nécessite les phases clés suivantes:

- Spécification du système.
- Partitionnement et mapping hardware/software.
- Conception de l'architecture.
- Simulation au niveau système.
- Fabrication du hardware et du software.

La manière dont la présence des ressources reconfigurables affecte les principales étapes du flot de conception et les besoins supplémentaires qu'y sont liés sont discutés dans les paragraphes suivants.

II.4.1.1 Spécification du système

Dans cette phase, les besoins, les restrictions et les spécifications sont rassemblés sans les ressources reconfigurables, mais un effort supplémentaire doit être consacré pour l'identification des parties qui sont candidates à être implantées avec du matériel reconfigurable.

Les effets des blocs reconfigurables sur la surface, la vitesse et la puissance absorbée devraient être complètement appréhendés avant qu'ils puissent être utilisés efficacement.

II.4.1.2 Partitionnement et mapping hardware/software.

Durant cette phase, une nouvelle dimension va être ajouté aux systèmes, la partie du système visé qui sera réalisée sur un matériel reconfigurable devra être identifiée, quelque règles seront nécessaire pour y parvenir.

- Si l'application à besoin d'accélérateurs matériels, qui ne sont pas utilisés en même temps ; alors un bloc reconfigurable dynamiquement sera une solution optimal que la logique câblée.
- Si éventuellement, les spécifications d'une ou plusieurs parties de l'application changent, ces parties seront implantées sur un hardware reconfigurable.

- Si de nouvelle génération de l'application sont envisageable pour mettre à jour le système, alors la partie concernée sera reconfigurable.

De plus, pour les systèmes reconfigurables, les problèmes d'allocation temporelle et d'ordonnancement seront pris en considération, toutes en respectant les autres contraintes liées aux systèmes traditionnels, où tout cela augmente d'avantage la complexité du système ; d'autant plus que les problèmes liés aux techniques reconfigurables sont hérités de la conception du hardware en plus de la conception du software.

II.4.1.3 Conception de l'architecture.

Beaucoup de questions sont liées au choix de l'architecture, qui concerne la technologie de reconfiguration utilisée et la partie du système qui doit être reconfigurable. La méthode la plus utilisée est la simulation ; Où le partitionnement, le mapping et l'ordonnancement sont fait manuellement par le concepteur, et les résultats et le rendement sont fait par simulation. Aussi cette approche et assez efficace car il n'existe toujours pas d'outil capable d'intégré un partie reconfigurable sur une plateforme hardware.

Avant de choisir la technologie reconfigurable adéquate pour la modélisation du système, il est nécessaire d'identifier certain paramètre : (a) le volume des blocs reconfigurable, (b) le volume des contextes de reconfiguration, (c) le temps de reconfiguration et la possibilité de reconfiguration partiel, (d) la puissance consommée. Ces paramètre donneront au concepteur une idée sur comment les différentes technologies affecteront les performances globales du système.

Les résultats obtenus qui vérifies si le modèle satisfera ou non les performances du système sont :

- L'utilisation spatiale, qui valide ou non la taille des blocs et la granularité du système.
- L'utilisation temporelle, qui donne les temps nécessaires à la reconfiguration et à l'activation des calculs.
- La mémoire et le bus de chargement des contextes.
- La surface et la puissance consommée, qui seront comparées à l'implantation hardware et software.

II.4.2 Flot de conception des SoCs reconfigurables

Dans cette section nous fournirons la structure générale d'un des flots de conception utilisé dans modélisation des SoCs reconfigurables. Ce flot à pour objectif d'amélioré le processus de conception d'un SoC, afin d'utiliser les outils disponibles d'une manière optimale (25).

L'idée principale de ce flot de conception est d'identifier les parties du système où l'ajout de technologies reconfigurables a le plus grand effet. C'est très important, car il n'y

toujours pas d'outils commerciaux ou de méthodologies pour supporter la reconfiguration. Le flot de conception est divisé en trois parties comme montrer sur la Figure II.19.

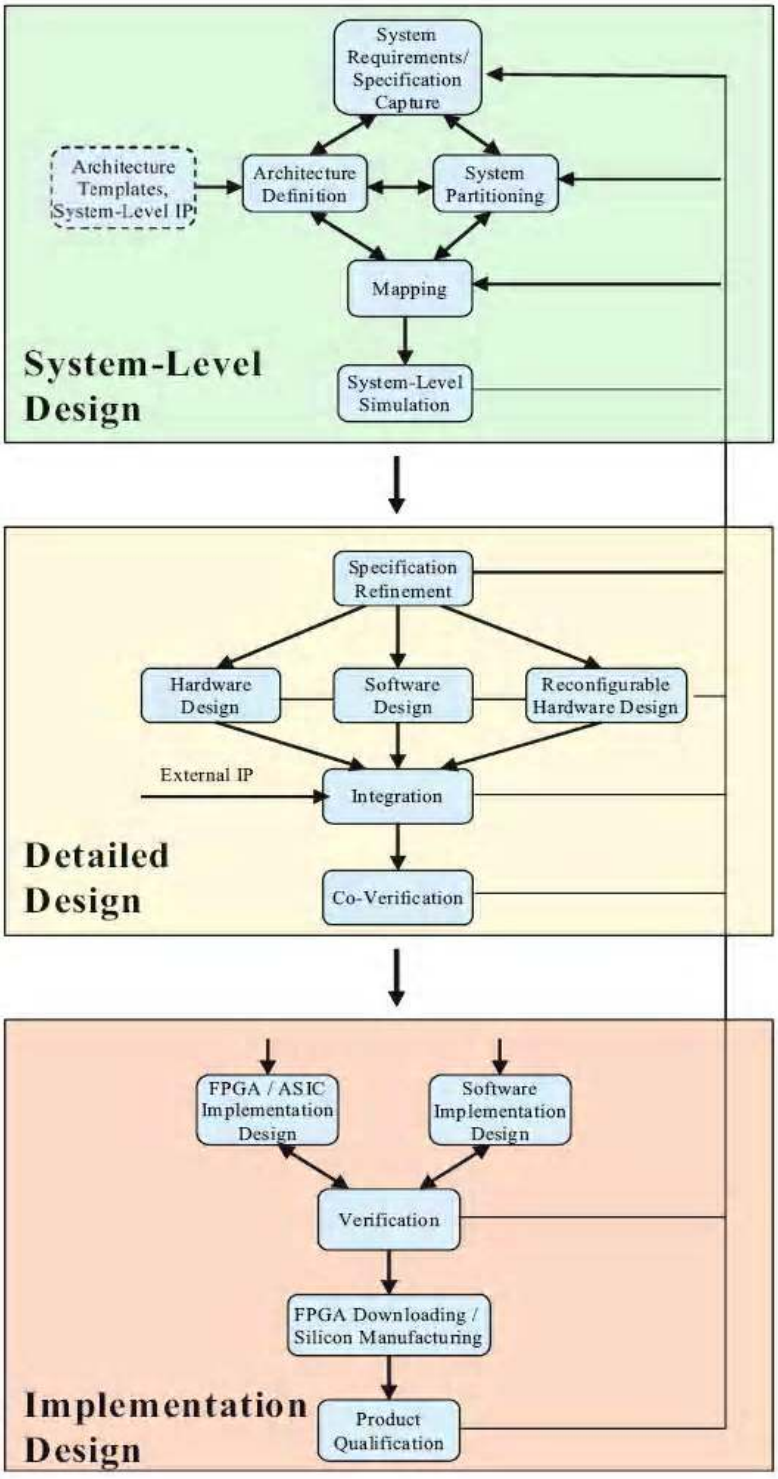


Figure II.19: Flot de conception

La conception au niveau système concerne le niveau le plus élevé du flot, cependant la conception détaillée et l'implémentation correspondent aux parties vitales de la méthodologie.

II.4.2.1 Conception au niveau système (*System Level Design-SLD*)

Les objectifs principaux de cette phase sont :

- Le développement des spécifications de l'application.
- La conception de l'architecture du SoC.
- La sélection des techniques d'implémentation.
- Partitionnement de l'application pour l'implémentation en hardware, software ou hardware reconfigurable.
- L'évaluation des performances de partitionnement.

Concernant la phase de partitionnement, le modèle fonctionnel de l'application est partagé en software, hardware et hardware reconfigurable, ces parties sont ensuite mappées dans l'architecture, tout en leurs associant des spécifications temporelles et matérielles locales nécessaire dans la phase de mapping.

Dans cette phase, les questions liées à la reconfiguration s'exprime sous les formes suivantes :

- Le but de la reconfiguration avec toutes les contraintes qu'y sont associées est identifié dans l'étape de la spécification.
- Durant le partitionnement du système, La partie reconfigurable se manifeste comme étant une ressource de calcule semblable au processeur, sa apporte une nouvelle dimension au partitionnement spatiale du SoC.

II.4.2.2 Conception détaillée (*Detailed Design-DD*)

Dans cette phase, les spécifications sont affinées et la vérification est planifiée en fonction de la technologie ciblée. Après ceci, le partitionnement individuel des blocs hardware, software et reconfigurable est conçu et vérifié.

Dans cette phase, les questions liées à la reconfiguration s'exprime selon les points suivants :

- Le bloc hardware nécessite l'ajout de moyen de communication et dans le cas de module reconfigurable dynamiquement, un mécanisme de gestion des contextes et nécessaire.
- L'intégration et la co-vérification associent le module reconfigurable aux autres composantes du SoC sur une même plateforme et rend possible la vérification de tous le système. Le module hardware et simulé avec un simulateur HDL ou émuler sur FPGA.
- Des règles de modélisation spécifiques à l'outil HDL doivent être suivies pour la reconfiguration dynamique multi-contexte.
- Les modules reconfigurables doivent être implémentés avec leur bloc de contrôle en utilisant la technologie sélectionnée.

II.4.2.3 Implémentation (*Implementation Design-ID*)

Dans cette phase, les questions liées à la reconfiguration s'exprime sous les formes suivantes :

- La reconfiguration dynamique nécessite la gestion des contextes (*bit-stream*).
- Les règles et les contraintes spécifiques au système doivent être respectées pour la reconfiguration dynamique multi-contexte.

Conclusion

Nous avons présenté dans ce chapitre une synthèse générale sur les architectures reconfigurables, en particulier les architectures reconfigurables dynamiquement, et leurs types communément utilisés dans la littérature, ainsi que leurs caractéristiques qui on permit de les classer. Nous avons aussi pu voir les modifications apportées au flot de conception des SoCs pour leur intégrer des modules reconfigurable, une réadaptation de l'architecture globale du SoC est nécessaire pour permettre une adéquation des modules ajoutés avec les IPs déjà présents sur le système.

L'utilisation des architectures reconfigurables se développe rapidement dans les domaines des télécommunications, et de contrôle notamment dans les systèmes évolutifs tel que le contrôle flou, une application des techniques reconfigurable aux systèmes flous sera présentée dans les chapitres suivants.

Chapitre III

PROCESSEURS FLOUS

Introduction

Les composants flous peuvent être classés en deux catégories selon leur technique de conception : numérique ou analogique. L'approche numérique a commencée par les travaux de Togai et Watanabe (26). *Depuis, plusieurs approches ont été proposées.*

Dans ce chapitre, nous donnerons un bref aperçu sur la logique flou, après une présentation générale des solutions matérielles en particulier numériques pour implanter les contrôleurs flous. Aussi, nous détaillerons quelques exemples de processeurs flous numériques commerciaux.

III.1 La Logique floue

La logique floue intervient dans la manipulation des connaissances imparfaites. Elle aide à formaliser la représentation et le traitement des connaissances imprécises ou approximatives, elle est approche de la démarche humaine dans le sens que les variables traitées ne sont pas des variables logiques (au sens de la logique binaire) mais des variables linguistiques, proches du langage humain.

III. 1. 2 Mise en œuvre d'un système à base de la logique floue

Un système à base de la logique floue peut être décrit par un ensemble ou une série de règles linguistiques de la forme : SI condition ALORS faire.

Ces règles permettent de convertir le raisonnement ou la stratégie de contrôle d'un expert humain du système en une stratégie de contrôle automatique bien adaptée au monde réel (27). Ce système est composé de quatre blocs (Figure III.20) qui sont :

- Une interface de fuzzification à l'entrée.
- Une base de connaissance floue.
- Un engin d'inférence ou bloc de prise de décision.
- Une interface de défuzzification.

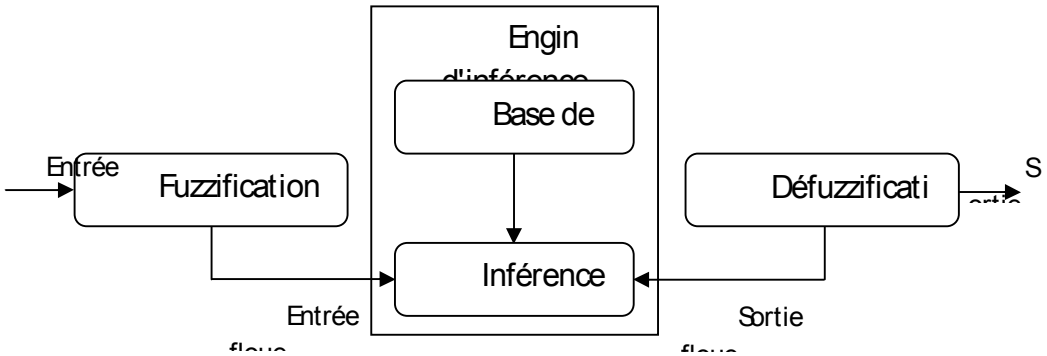


Figure III.20 : Structure de base d'un système flou.

a) Bloc de fuzzification

Ce bloc permet la conversion de la grandeur déterministe de l'entrée en une entité floue, qui est définie par des variables linguistiques, et des fonctions d'appartenances; qui peuvent ensuite être traitées par l'engin d'inférence.

Les fonctions d'appartenance peuvent avoir différentes formes, mais les formes trapézoïdales, triangulaires ou issues de modèles mathématiques sont les plus couramment utilisées.

b) La base de connaissance floue

Les règles floues représentent le cœur d'un système flou, et permettent d'exprimer sous forme linguistique les variables d'entrée du système par les variables de commande du processus. Un exemple typique de règles est donné par :

Si x1 est "positif" et x2 est "zéro" alors u est "négatif".

Où 'x1' et 'x2' représentent deux variables d'entrée du système : l'écart de réglage, et sa variation et 'u' la commande. L'expérience humaine joue un rôle important dans l'élaboration de ces règles.

c) Engin d'inférence

Il permet de définir la stratégie de contrôle ou de réglage pour les appliquer aux processus à contrôler en utilisant toutes les implications floues qui lient les différentes variables de chaque règle. Parmi les principales méthodes d'inférence, on cite :

- La méthode d'inférence Max-Min.
- La méthode d'inférence Max-Prod.
- La méthode d'inférence Som-Prod.

d) Bloc de défuzzification

Le résultat de la combinaison des règles appliquées aux intervalles flous de la variable de sortie définit une fonction d'appartenance. Il s'agit de convertir cette information en une grandeur qui peut être appliquée à un processus comme étant un signal de commande.

Il existe plusieurs méthodes de défuzzification qui nécessitent des ressources de calcul plus ou moins importantes, mais en pratique on utilise surtout les méthodes suivantes :

Défuzzification par calcul de la moyenne de maxima

Cette approche est intuitive, elle choisit le point avec la plus forte incidence, c.-à-d. la fonction d'appartenance maximale U_{mm} (Figure III.21) Il se peut qu'il existe plusieurs points, alors, on calcule la moyenne des maximas. Cette méthode ne tient pas compte de la forme des sous-ensembles flous, mais elle est très simple du point de vue calcul.

Défuzzification par calcul du centre de gravité

Il s'agit de calculer le centre de gravité de la fonction d'appartenance de la variable de sortie, où il suffit de calculer l'abscisse U_{cg} de la Figure III.21.

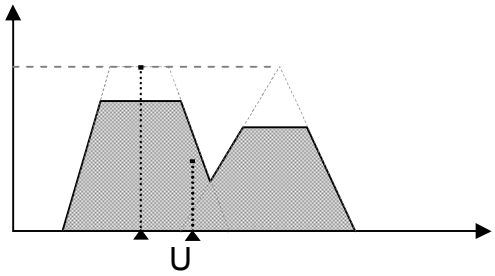


Figure III.21 : Défuzzification de la sortie floue.

L'abscisse U du centre de gravité peut être calculée par la fonction suivante :

$$\frac{\sum_{i=1}^{m} u_i \mu_i(u_i)}{\sum_{i=1}^{m} \mu_i(u_i)}$$

Dans le cas discret. Le calcul du centre de gravité permet bien d'obtenir une seule valeur pour la grandeur de sortie mais son calcul est relativement complexe.

III. 2. Processeurs et contrôleurs flous

Ces dernières années, les applications de la logique floue en milieu industriel ont augmentées de façon très importante. Généralement, l'implémentation d'un projet flou est décrite par les points suivants (28) :

- Le nombre des entrées et le nombre des sorties.
- Le nombre et la forme des fonctions d'appartenance des entrées et des sorties.
- Le nombre et le format (nombre des prémisses et conclusions) des règles floues.
- Résolution des entrées et des sorties.
- Résolution des valeurs alpha (degrés d'appartenance des fonctions d'appartenance associées aux entrées).
- L'algorithme flou utilisé.

Les performances et la rapidité d'un système flou peuvent être mesurées par les paramètres suivants :

- Le nombre de règles par seconde (*Fuzzy Logic Inference per Second-FLIPS*).
- Le nombre de sorties par seconde.
- Le temps de réponse du système représentant la période de la commande.

L'implémentation matérielle "hardware" d'un système à base de logique floue peut s'effectuer de plusieurs façons :

- Les composants de type FPLA (Fuzzy Programmable Logic Array).

- Les tables de transcodage (look-up tables).
- Les microcontrôleurs classiques.
- Les microcontrôleurs contenant en plus des instructions classiques des instructions spécialisées relatives à la logique floue.
- Les composants flous capables de fonctionner sans un processeur hôte.
- Les processeurs ou coprocesseurs flous numériques.
- Les processeurs ou coprocesseurs flous analogiques.

Le choix d'une solution particulière dépend de plusieurs paramètres :

- Le coût de l'implémentation.
- La simplicité de la réalisation.
- La structure et la complexité du projet flou utilisé dans l'application.
- La contrainte de rapidité (dépendant du temps de réponse de l'application).

Le choix d'une solution pour la réalisation du projet flou repose généralement sur un compromis entre les paramètres précédents.

Beaucoup d'architecture sont basé sur des processeurs de type RISC, Ces processeurs sont généralement rapides, flexibles parce qu'ils peuvent être programmés pour un projet quelconque, à condition qu'il ne dépasse pas certaines limites sur les caractéristiques. Généralement ils sont construits en utilisant directement la technologie CMOS dans un seul composant, comme par exemple, les processeurs flous WARP et SAE-81C99A (28).

On trouve parfois des processeurs flous qui utilisent des circuits discrets. Ce type de processeur est souvent un premier prototype de laboratoire, ou une architecture spécialement conçu pour des applications spécifiques.

Néanmoins, la structure d'un tel processeur reste généralement composée des blocs représentés sur la Figure III.22.

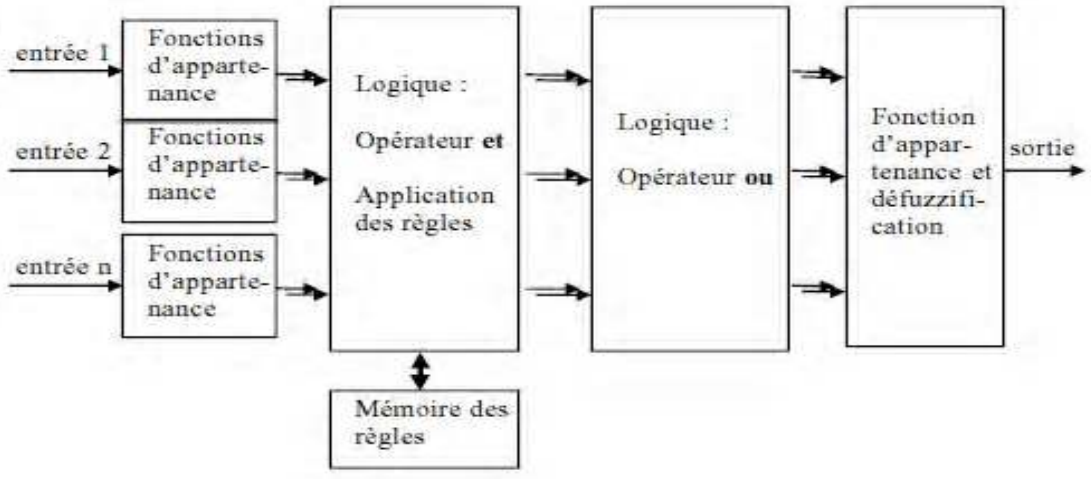


Figure III.22 : Structure générale du processeur flou.

Une autre caractéristique que nous ajoutant dans cette thèse pour la classification des processeurs flous et leur capacité d'adaptation au changement de la application visé, dans les paragraphes suivant nous allons détailler quelques architectures qui seront le début de nos réflexions sur la conception de notre système flou.

III.3 Architecture classique des contrôleurs flous

III.3.1 Architecture série

Le plus grand problème des processeurs flous est la grande quantité de données à traitée. Une architecture de systèmes flous numériques peut utiliser des bus larges pour supporter les échanges entre modules. Mais cette approche rend l'implantation de circuits efficaces dotés d'un nombre élevée d'E/Sdifficile (29).

L'architecture série est utilisée pour pallier à ce problème. Elle permet de capter un vecteur de bits et de le traiter ensuite de façon séquentielle.

Architecture de l'unité d'inférence

L'unité d'inférence est l'unité qui prend les valeurs antécédentes floues et génère les résultats dans le domaine flou. Cette unité comprend plusieurs modules qui dépendent de l'application souhaitée, mais d'une manière générale, elle peut avoir la configuration suivante (Figure III.23).

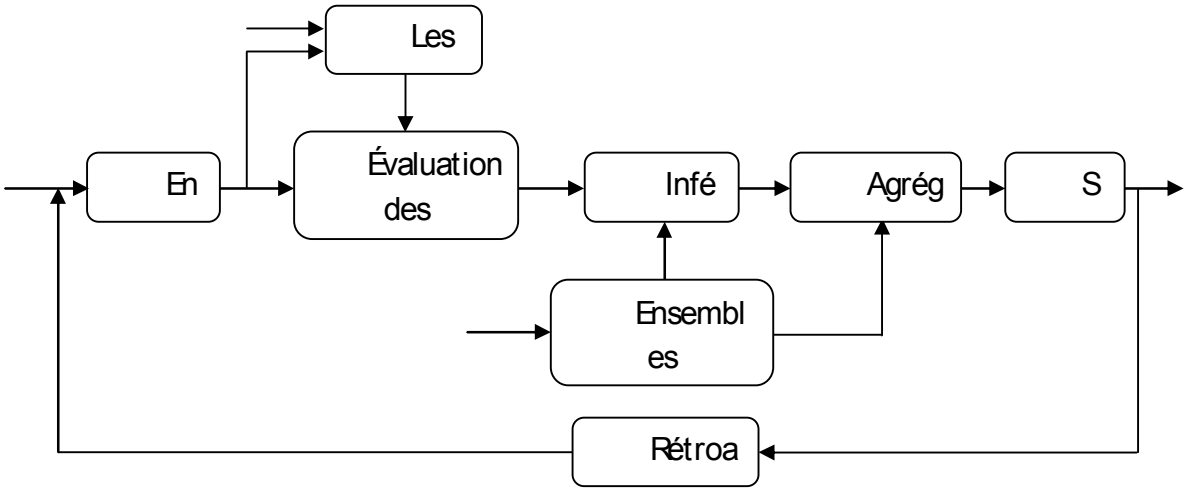


Figure III.23 : Configuration générale de l'unité d'inférence.

Du point de vue implémentation matérielle, la méthode d'inférence Min-Max est la plus adéquate et la plus simple à implanter.

Architecture série de la défuzzification

Le processus de défuzzification est le point le plus critique pour la performance des processeurs flous. Il existe plusieurs algorithmes pour chaque type d'architecture ; mais ces

algorithmes doivent traiter le problème de la division. A noter que, c'est la division qui contribue à donner au model flou sa nature non linéaire.

Il y a deux méthodes pour réaliser la division, la première est d'élaborer un algorithme dans le même sens, et la seconde est d'implanter une table qui sera adressée par le numérateur et le dénominateur du centre de gravité.

III.3.2 Architecture parallèle

Cette architecture exploite, le parallélisme inhérent dans les inférences. Le processeur exploite aussi le fait qu'une partie des règles est activée pour réduire le nombre de règles traitées, ceci est une conséquence directe de la manière avec laquelle les règles sont mémorisées. Cette méthode réduit considérablement la taille de la mémoire requise et simplifie le processus d'activation des règles (30).

Architecture du processeur

Ce genre de processeurs se compose de l'assemblage des unités suivantes Figure III.24:

- L'interface d'entrée reçoit les variables linguistiques.
- Quatre unités de détection qui ont pour tâche de récupérer seulement les intervalles flous avec des degrés d'appartenance positifs ; le résultat est stocké dans le registre "Ant-register".
- Quatre unités d'antécédent qui calculent le degré d'appartenance en se basant sur les données stockées dans le registre précédent.
- Quatre unités de règles qui traitent les règles, en calculant le degré d'activation de chaque règle active seule.
- Le défuzzificateur qui utilise la méthode de "Yager".
- L'unité de contrôle a pour rôle de coordonner les activités des autres unités.

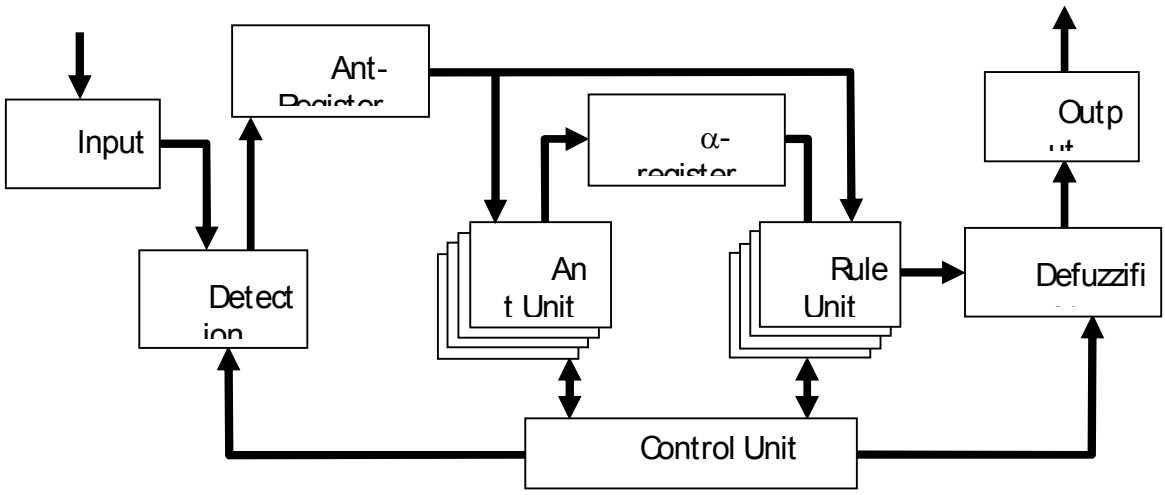


Figure III.24 : Architecture du processeur parallèle.

Performances

L'horloge de travail de ce genre de processeur est de 60 MHz. Cette horloge est déterminée par la synthèse du processeur, qui est modélisée en VHDL, par l'outil de développement et de synthèse Simplify de Synopsys en utilisant la technologie HCMOS5.

Le FIPS est déterminé par $N_{INF} = 1/ (N_{cycles} * 16.67 * 10^{-9})$, où N_{cycles} est le nombre de cycles d'horloge de l'étage le plus lent.

Si le nombre de règle active est de 16, le temps de réponse de l'étage le plus lent est de 24 cycles d'horloge, donc les performances obtenues sont de 2.5 MFIPS.

La surface de Silicium est de 30 mm2, dont 18 mm2 sont occupés par les mémoires.

III.3.2 Architecture parallèle (*Pipelined Subdivision*)

Les différentes approches de conception des processeurs flous à base de VLSI doivent trouver un compromis entre la vitesse, la flexibilité et la surface utilisée. Cette architecture a été divisée en plusieurs blocs illustrés dans la Figure III.25, cette disposition des modules permet une progression du traitement en pipeline et en parallèle (31).

Le premier des modules permet la fuzzification des entrées A0 et A1 qui adressent directement une ROM où se trouve une décomposition des fonctions d'appartenance.

Ce processus retourne aussi les degrés d'appartenance "αi" qui correspondent à cette décomposition, de plus le processeur sélectionne les règles floues actives qui sont stockées dans une autre mémoire adressable par les fonctions d'appartenance en quatre impulsions d'horloge.

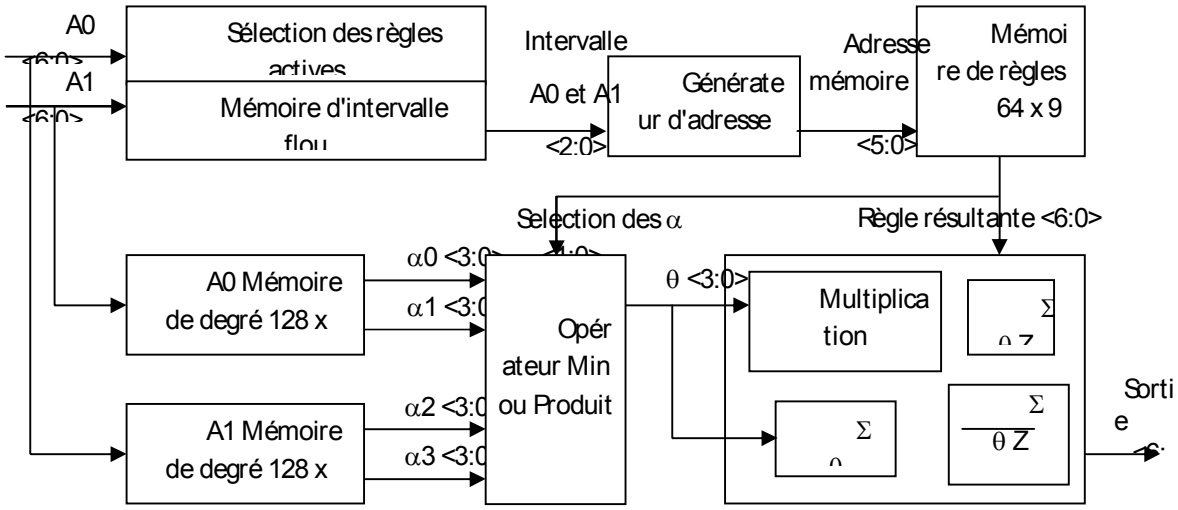


Figure III.25 : Schéma bloc du processeur.

À l'intérieur de cette mémoire se trouve les sorties conséquentes et un code qui permet la sélection du degré de vérité ; ce dernier est calculé par la méthode du minimum ou celle du produit. Enfin le bloc de défuzzification utilise la méthode du centre de gravité en utilisant la sortie de la mémoire des règles et les degrés de vérité pour le calcul de la sortie finale.

Performances :

La fréquence d'horloge atteint 133 MHz qui est limitée par la fréquence de la RAM ; chaque règle est traitée en un cycle d'horloge. Le temps global du traitement est la somme du temps requis à la synchronisation (1 période = 7.5ns) et le temps nécessaire pour les blocs en pipeline (12 périodes = 90ns) plus le temps de la division (4 périodes = 30ns), donc le temps de réponse est de 127.5 ns soit 7.8 MFIPS, et l'erreur de calcul est de 1%, la surface utilisée est de 3 mm2 avec la Technologie 0.35 µ CMOS, pour une consommation de 200 mW.

III.3.3 Coprocesseur flou WARP

Le coprocesseur flou WARP est développée et fabriquée par SGS-THOMSON en technologie HCMOS4T 0.7µm. Il peut être utilisé comme coprocesseur avec un processeur ordinaire ou comme un contrôleur flou indépendant. Ce coprocesseur effectue des inférences en 33.1µs pour un projet flou à 4 entrées, 2 sorties et 32 règles (32).

Fonctionnement

Le WARP a deux modes de fonctionnement, le mode de chargement et le mode de traitement. Dans le mode de chargement, les données concernant le projet flou (le nombre des entrées, le nombre des sorties, les fonctions d'appartenance des entrées et des sorties et la base de règles reliant les mots des entrées à ceux des sorties) sont chargées dans des mémoires internes. Ce mode prépare le WARP pour le mode de traitement où il lit les valeurs des entrées depuis un bus d'entrée en utilisant des signaux de synchronisation, calcule les valeurs des sorties et les envoie vers un bus de sortie.

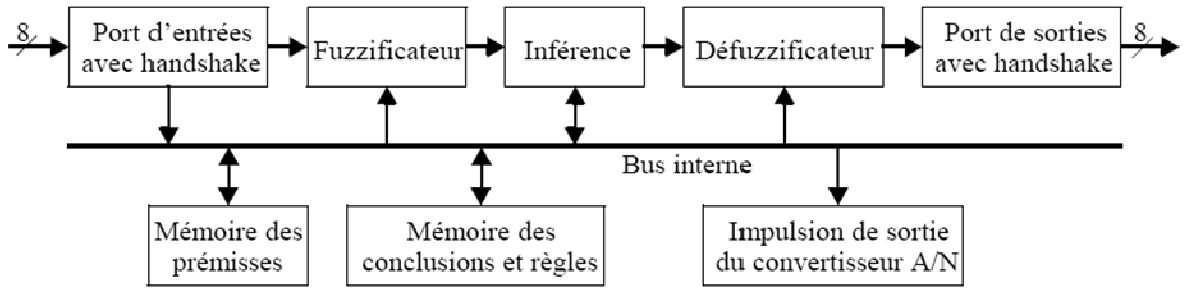


Figure III.26 : Schéma bloc du processeur WARP.

Le schéma bloc du WARP est donné à la (Figure III.26). Le WARP a trois mémoires internes. La première mémoire est celle des prémisses AM (*Antecedent Memory*) qui contient les fonctions d'appartenance de toutes les entrées, la deuxième mémoire est celle des conclusions et des règles PCM (*Programme/Consequent Memory*) qui contient les fonctions d'appartenance de toutes les sorties et la troisième mémoire est le registre de configuration RB (*Register Bench*).

Le cycle de traitement commence par la saisie des valeurs des entrées une par une via le bus des entrées. Ces entrées doivent être fournies avec leur numéro d'identification ce qui signifie que cette saisie peut s'effectuer dans n'importe quel ordre. Le WARP peut effectuer un cycle de traitement sans saisir toutes les valeurs des entrées. Pour une entrée non saisie, sa dernière valeur est prise en compte. Cette propriété peut être très intéressante quand les entrées ont des dynamiques différentes. La saisie d'une valeur d'entrée s'effectue en 8 périodes d'horloge.

Les valeurs des entrées sont récupérées, stockées et routées par le port des entrées vers le "fuzzificateur". Le fuzzificateur calcule les degrés d'appartenance de la valeur reçue aux fonctions d'appartenance associées (valeurs alpha).

L'unité d'inférence utilise ensuite la méthode max-min sur les valeurs alpha pour obtenir les valeurs oméga qui représentent les pondérations des fonctions d'appartenance des sorties lors du calcul des valeurs numériques de ces sorties par le défuzzificateur. La méthode utilisée dans le défuzzificateur est la méthode du centre de gravité.

Finalement, les valeurs numériques calculées par la défuzzificateur et codées sur 8 bits sont envoyées au bus de données des sorties via le port des sorties. Les valeurs de sortie sont envoyées avec leur numéro d'identification. Un signal servant à synchroniser le WARP avec des circuits extérieurs lents peut être fourni à ce niveau.

Performance

La capacité du WARP est de 8 entrées, 4 sorties et 256 règles d'un format de 4 prémisses et une conclusion. Les entrées et les sorties sont codées sur 8 bits. Le nombre des fonctions d'appartenance peut atteindre 16 pour une entrée et 256 pour toutes les sorties. Les fonctions d'appartenance des entrées sont soit de forme triangulaire, soit de forme trapézoïdale tandis que celles des sorties sont de forme singleton. La fréquence maximale supportée par le WARP est de 40 Mhz.

III.4 Architecture reconfigurable

Dans les systèmes flous conventionnels, la stratégie de contrôle est figée mais si les conditions de fonctionnement change, ceci peut crée des problèmes dès que le procédé devient instable. Afin de conserver des performances admissibles pour le système, le processeur flou évolutif devient une nécessité (33). Ces systèmes évolutif ou reconfigurable,

sont une alternative attrayante au processeur traditionnel puisque les ressources du processeur sont adaptées aux divers applications possibles, en modifiant par exemple l'univers du discours, les règles ou toutes autres éléments de l'architecture (34).

Nous avons choisi quelques exemples de systèmes flous reconfigurables trouvés en littérature, qui seront présentés dans ce qui suit :

III.4.1 Processeur flou pour plateforme dynamiquement reconfigurable

Beaucoup d'étude se sont intéressé à cette possibilité (35) (36) (37), ces processeur utilisent la capacité de reconfiguration des circuits reprogrammables tel que les FPGA, où les possibilités de changement de contextes de façon dynamique sont utilisés pour adapter le système à son environnement (Figure III.27), un processeur hôte est utilisé pour charger les bit-stream du plan de configuration de l'extérieur.

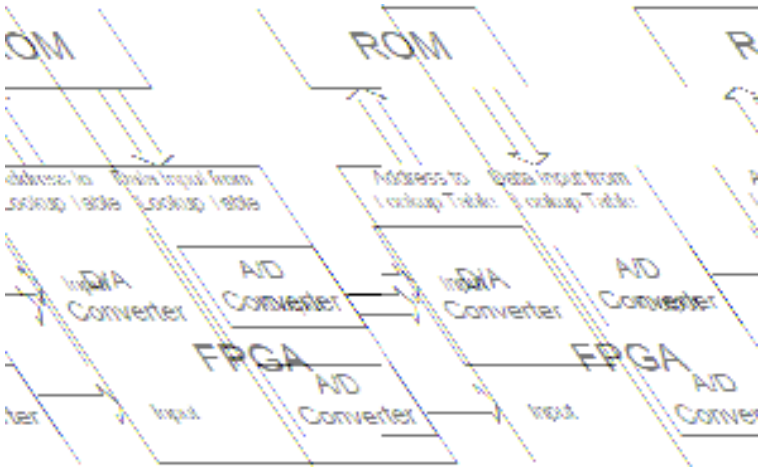


Figure III.27 : Carte de contrôle flou à base de FPGA.

III.4.2 Processeur flou reconfigurable statiquement

C'est la méthode la plus simple, elle consiste en la description comportementale du processeur avec un langage de description matériel tel que VHDL, où l'utilisation de paramètre générique est nécessaire pour permettre à l'utilisateur de redéfinir les valeurs des composantes de son système tel que le nombre d'entrées/sorties, la largeur du flot de donnée, le contenu des mémoires ROMs afin de les adaptés à son application (38). L'arrêt du système puis la reprogrammation du processeur est nécessaire.

III.4.3 Architecture à défuzzification reconfigurable dynamiquement

Cette architecture décrite dans (39), propose d'intégré plusieurs méthodes d'inférence et de défuzzification sur un hardware flexible qui permet de reconfigurer dynamiquement le système flou en basculant d'une méthode à l'autre sans interrompre le fonctionnement du procédé.

Le module de défuzzification combine 4 méthodes décrites dans les équations suivantes :

$$\hat{y} = \sum_{i=1}^{r} \alpha_i \cdot c_i / \sum_{i=1}^{r} \alpha_i \quad (1) \qquad \hat{y} = \sum_{i=1}^{r} \alpha_i \cdot w_i \cdot c_i / \sum_{i=1}^{r} \alpha_i \cdot w_i \quad (2)$$

$$\hat{y} = \sum_{i=1}^{r} \alpha_i \cdot (p_i + q_i \cdot \alpha_i) / \sum_{i=1}^{r} \alpha_i \quad (3) \qquad \hat{y} = \sum_{i=1}^{r} c_i \cdot (1 - \beta_i^2) / \sum_{i=1}^{r} (1 - \beta_i^2) \quad (4)$$

Ces différentes méthodes de défuzzification utilisent les mêmes blocs arithmétiques (deux multiplicateurs, un diviseur et plusieurs additionneurs), la Figure III.28 montre les circuits équivalents aux 4 équations, les blocs encadrées par des lignes discontinus sont communes dans les 4 méthodes.

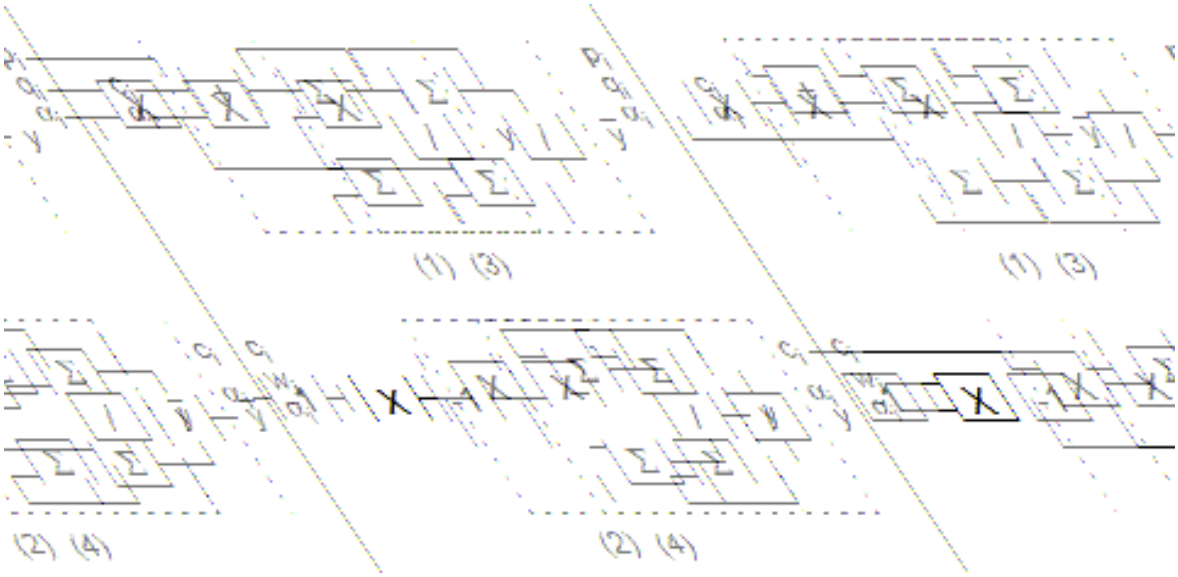
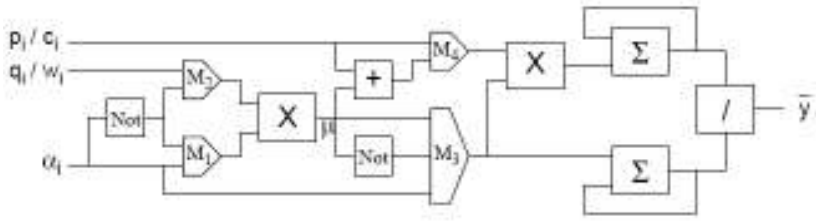


Figure III.28 : blocs fonctionnels des méthodes de défuzzification

Le multiplexage des multiplicateurs (Figure III.29.a) permet le partage des ressources de calcul, de cette façon le même hardware peut implémenter plusieurs méthode de défuzzification. La table de la (Figure III.29.b) présente les signaux de contrôles pour basculer d'une configuration à une autre.



Eq.	M_1	M_2	M_3	M_4
(1)	–	-	α_i	c_i
(2)	α_i	w_i	μ	c_i
(3)	α_i	q_i	α_i	$\mu + p_i$
(4)	$1-\alpha_i$	$1-\alpha_i$	$1-\mu$	c_i

(a) (b)

Figure III.29 : Défuzzificateur multifonction a) bloc-diagramme b) selection des multiplicateurs.

Conclusion

Dans ce chapitre, nous avons exploré les solutions matérielles possibles pour réaliser un composant flou (un processeur flou en général). Selon les techniques de conception, nous avons pu voir les possibilités d'intégration des systèmes numériques sur des plateformes classiques et reconfigurables.

L'approche numérique est relativement facile à développer comparé aux approches analogiques. Les processeurs flous numériques offrent une plus grande flexibilité de configuration et de programmation. Leurs performances par contre, restent dépendantes des méthodes d'inférence utilisées et surtout très limitées par les méthodes de défuzzification.

Chapitre IV

CONCEPTION D'UN SYSTEME FLOU RECONFIGURABLE

Introduction

Nous avons pu voir dans les chapitres précédents une étude exhaustive de tous les éléments qui vont nous aidés pour la mis en œuvre de notre système flou reconfigurable. Ce travail vient de l'idée de rassembler sur le même système toutes les méthodes possibles utilisées dans le traitement flou, pour les évaluer et même pour créer de nouvelles méthodes et les tester dans leurs contextes.

La démarche suivie pour la réalisation de nos objectifs est définie ci-dessous :

- Conception du système flou et de ses différents blocs constitutifs.
- Ajout des interfaces de communication qui permettent l'intégration du système flou sous forme d'IP.
- Application du système flou pour le contrôle et la gestion de la température sur un SoC.

Concernant la première partie, nous avons essayé de hiérarchiser notre travail de conception conformément aux points suivants :

Au niveau système

- Définition des spécifications du système suivant le cahier des charges.
- Définition d'une première architecture.
- Identification des éléments susceptibles d'inclure du hardware reconfigurable.
- Sélection des techniques de reconfiguration.
- Partitionnement pour l'implémentation en hardware, software et hardware reconfigurable.
- Sélection des techniques d'implémentation.

Au niveau structurel

- Affinement de l'architecture du système.
- Modélisation des modules hardware et hardware reconfigurable, ces modules sont conçus et simulés suivant les règles de modélisation spécifique à l'outil HDL, dans notre cas il s'agit du langage VHDL sous la plateforme de développement Xilinx ISE 9.1i.
- Vérification.
- Modélisation de la partie contrôle des modules reconfigurables.
- Conception d'une unité de contrôle, qui va gérer le flot de donnée dans le système flou.
- Co-vérification de la parie software, hardware et hardware reconfigurable.

En second lieu, nous allons essayer de structurer notre système pour qu'il soit intégré dans un SoC sous forme d'un IP-Soft, capable de communiquer avec les autre IPs du

système ; une interface composée du bus système 'Wishbone' et du bus de contrôle 'SMBus' est ajoutée.

Et enfin, comme application nous allons exploiter notre système flou reconfigurable pour la gestion de la température d'un SoC.

IV.1 Conception au niveau système

IV.1.1 Langage et outils de conception

Le choix d'un langage spécifique est généralement lié à plusieurs conditions, au niveau d'abstraction et au champ d'application. Il permet d'expérimenter et de matérialiser les différentes idées pour une exploitation future. Le langage que nous avons choisit doit permettre de faire des descriptions de haut niveau en plus des descriptions structurelles de bas niveau.

Notre choix s'est porté sur le VHDL qui permet de décrire le comportement d'un circuit et de simuler les différentes étapes de conception. La plateforme de développement "Xilinx ISE 9.1i" de la société "Xilinx" qui intègre tous les outils nécessaires à la conception et à la simulation des applications à base de FPGA ainsi que l'outil de simulation "ModelSim 6.0a" de "Mentor Graphics" sont exploités pour la mise en œuvre de notre système flou reconfigurable. Le flot de conception basé sur le langage VHDL est représenté sur la Figure IV.30.

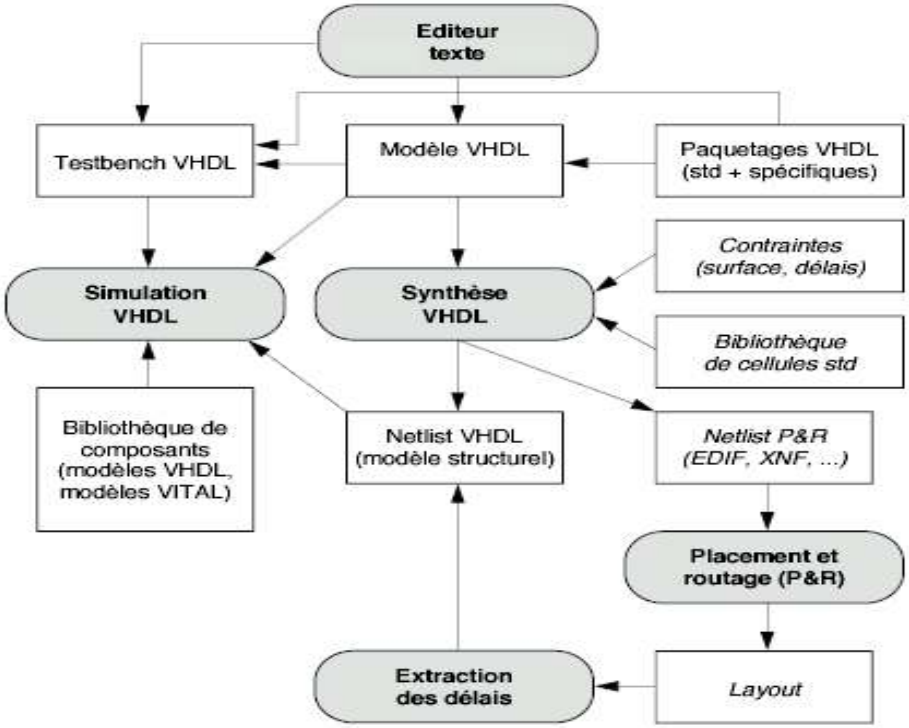


Figure IV.30 : Flot de conception basé sur VHDL

IV.1.2 Modèle général

La structure globale de notre système respecte le modèle d'un système flou généralisé. Elle est divisée en cinq blocs : deux blocs de fuzzification pour chacune des entrées, un bloc pour la base des règles, un bloc d'inférence, et un bloc de défuzzification. Nos besoins en ressources systèmes dépondent essentiellement du nombre des entrées et des sorties, de la forme de l'univers du discours et des méthodes d'inférence et défuzzification.

Notre cahier des charges impose et spécifie les points suivants :

a) 2 entrées normalisées et discrétisées.
b) 1 sortie normalisée et discrétisée.
c) La largeur du bus de donnée est de 8 bits.
d) L'univers du discours est divisé en un maximum de 7 fonctions d'appartenance.
e) Un chevauchement double des fonctions d'appartenance.
f) 4 règles actives parmi 49 règles au maximum.

Dans cette phase de conception l'architecture est basée sur les 5 blocs cités plus haut (Figure IV.31) et nous n'avons pas encore à ce stade déterminé quelle partie du hardware doit être reconfigurée.

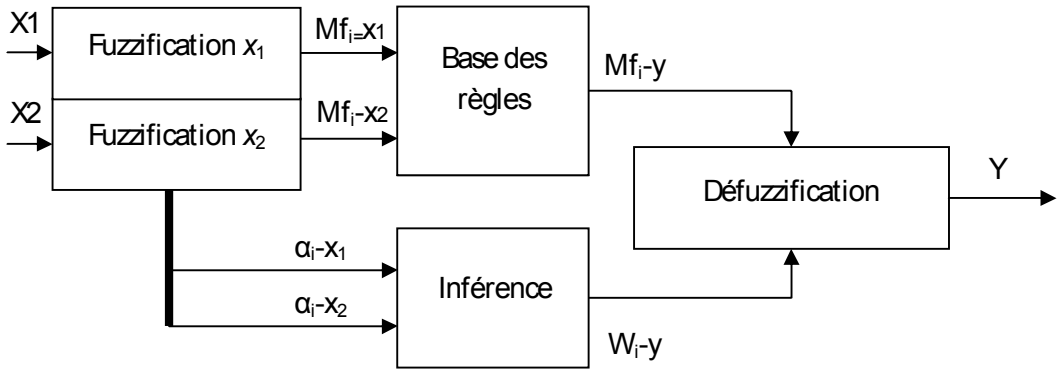


Figure IV.31 : Architecture du contrôleur flou.

Dans les paragraphes suivants nous discuterons de l'implantation de ces modules et les possibilités de reconfiguration qui y sont associées.

IV.1.2.1 Blocs de fuzzification

Les entrées du système flou doivent être normalisées et discrétisées ensuite la fuzzification transforme les valeurs de l'entrée numérique 'x' en une partie floue numérique sur un espace de représentation de l'univers de discours, des méthodes nécessitant des calculs sont rarement utilisées pour l'implantation de cette partie du système flou, car les fonctions d'appartenance résultantes peuvent avoir des formes passants du simples au complexes (triangulaires, trapézoïdales, gaussiennes…ou toute autre forme) (Figure IV.32).

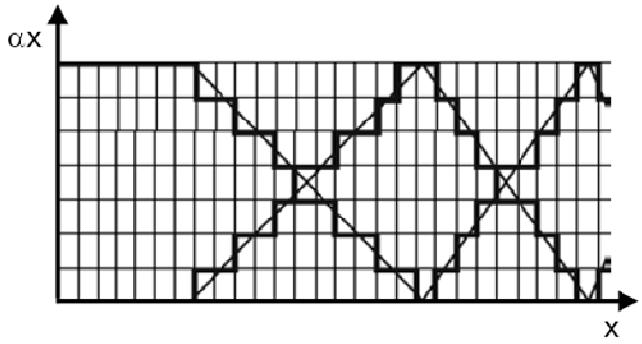


Figure IV.32 : Données normalisées

D'où l'utilisation de mémoires pour stocker l'univers du discours correspondant à chaque entrée, un autre élément décisive de l'implantation de ce bloc est l'existence d'un double chevauchement entre deux fonctions d'appartenance. Alors, chaque variable d'entrée est exprimée par deux fonctions d'appartenance et deux degrés d'appartenance.

Les valeurs obtenues sont enregistrées selon la configuration suivante :

Mf1-xi	α1-xi	Mf2-xi	α2-xi

Où: Mf1, Mf2 : fonctions d'appartenance codées sur 3 bits chacune.

α1, α2 : degrés d'appartenance codés sur 5 bits chacun.

IV.1.2.2 Bloc de la base des règles

Ce bloc reçoit les fonctions d'appartenance résultantes de la fuzzification. Il permet la sélection des règles floues qui sont généralement sous la forme suivant :

If X is Mf1 and Y is Mf2 then U is Mf1-y.

Avec 7 fonctions d'appartenance au maximum pour chaque variable d'entrée, le nombre de règles est de 7 x 7 soit 49 règles possibles (Tableau IV.2).

Tableau IV.2 : Matrice des règles.

U		**Y**						
		NG	**NM**	**NS**	**Z**	**PS**	**PM**	**PG**
X	**NG**	NG	NG	NG	NG	NG	NG	NG
	NM	NM	NM	NM	NG	NG	NG	NG
	NS	NM	NM	NS	NS	NS	NS	PS
	Z	NS	NS	NS	*Z*	*Z*	Z	PS
	PS	NS	NS	Z	*PS*	*Z*	PM	PM
	PM	PS	PS	PS	PM	PM	PM	PM
	PG	PG	PG	PG	PG	PG	PG	PG

IV.1.2.3 Bloc d'inférence

Généralement, pour calculer la valeur correspondante du degré de vérité de chaque règle active, les opérateurs de la logique floue sont implantés selon la méthode d'inférence de Mamdani, mais d'autres méthodes d'inférence conviennent aussi pour certaines applications.

IV.1 2.4 Bloc de défuzzification

Ce module convertit les résultats de l'inférence en valeurs exploitables dans la commande, où les performances du système en termes de stabilité, de précision et de vitesse dépendent fortement de la conception de ce bloc. Il est impératif donc de choisir des algorithmes qui produisent de bon compromis temps/performances.

Les algorithmes utilisés en défuzzification dépendent largement des applications visées, ces algorithmes consomment beaucoup de ressources système ; car ils sont décrits par des fonctions mathématiques plus ou moins complexes, généralement elles sont toutes implantées par les mêmes opérateurs arithmétiques (multiplicateur, additionneur, diviseur…) **[39]**.

IV.1.3 Éléments de reconfiguration

Dans les paragraphes précédents, nous avons décrit la manière dont les différents blocs du contrôleur flou seront implantés, il est clair que les blocs nécessitant des calculs seront candidats à la reconfiguration **[40]**.

Le calcul des inférences et la défuzzification emploient tous les deux les mêmes opérateurs mathématiques, alors nous allons les combiner dans un même module qui va prendre en charge les différents algorithmes employés dans ces blocs. Le module résultant de cette combinaison sera reconfigurable pour permettre au système flou de gérer toutes les méthodes possibles d'inférence et de défuzzification.

IV.1.3.1 Technique de reconfiguration

Pour satisfaire les contraintes de notre cahier des charges et les contraintes liées à la plateforme de développement, nous avons choisi la technique de reconfiguration en prenant comme critères les points suivants :

- Réduire le hardware au minimum (contrainte de surface liée au FPGA).
- Portabilité de toute la structure sur d'autre plateforme.
- Temps de reconfiguration minimum.
- Reconfiguration au niveau des ressources de calcul.
- Reconfiguration au niveau des échanges de donnée.
- Satisfaire les besoins en éléments mémoires.

La technique appropriée pour l'implémentation du module reconfigurable est la reconfiguration dynamique en chemin de données (Datapath) par un maillage à 2 dimensions (R-Mesh) **[40] [24]**.

IV.1.3.2 Chemin de données (Datapath)

Le chemin de donnée constitue le cœur du hardware reconfigurable, il est composé d'éléments de gros grain (bloc arithmétique, registre..) qui permettent d'effectuer une grande variété d'opérations sur les données.

IV.1.3.3 Réseau d'interconnexions reconfigurables

Au sein d'un composant reconfigurable, le réseau d'interconnexion assure la distribution des données entre les différentes unités de calculs. Sa topologie a un impact de tout premier ordre sur les performances, la consommation et la flexibilité du système. Trois grandes catégories de réseaux peuvent être distinguées **[41]**.

- Les réseaux d'interconnexion globaux
- Les réseaux d'interconnexion point-à-point
- Les réseaux d'interconnexion hiérarchiques

Nous avons choisi pour notre architecture un réseau d'interconnexion global, qui va nous garantir une connectivité totale des ressources, au sens où chaque élément (calcul, mémorisation, contrôle, etc.) peut éventuellement communiquer avec tous les autres.

Ce mode d'interconnexion permet des reconfigurations dynamiques multi-contexte, cependant, il reste assez complexe à mettre en œuvre et à gérer de part le nombre important de points d'interconnexion.

IV.1.4 Modèle de Partitionnement

Pour arriver à mettre au point un système reconfigurable, une étape très importante du processus de conception est alors exécutée, c'est l'étape de partitionnement. Dans cette phase on cherche à répartir les différentes fonctionnalités et spécifications du système sur l'espace architectural du système flou, cette répartition étant déterminée de manière à garantir certaines performances et/ou optimiser un ou plusieurs critères.

Le problème du partitionnement impose de modéliser l'application cible de l'architecture considérée, et également de préciser le schéma d'exécution retenu de l'application sur la structure du système. Définir des modèles précis permet d'obtenir des comportements réalistes mais un excès de précision peut accroître fortement les temps de partitionnement. Donc nous allons établir des modèles globaux pour le software, le hardware et le hardware reconfigurable pour permettre une bonne gestion des

ressources système. Dans ce qui suit, nous détaillerons notre méthode de partitionnement.

IV.1.4.1 Allocation

Dans l'étape d'allocation nous déterminons les parties du système qui seront implantées en software ou hardware (reconfigurable ou non-reconfigurable), d'où la définition des différents modules:

d) Modules 'hardware'

Comme il a été indiqué précédemment, nous réaliserons les modules de fuzzification et celui de la base des règles en hardware non reconfigurable, plus exactement en mémoire, mais aussi une partie du contrôleur de reconfiguration (Contrôleur d'unité reconfigurable dynamiquement) du système flou sera réalisée en hardware.

e) Module 'hardware reconfigurable'

Nous implanterons la défuzzification et le calcul des inférences en hardware reconfigurable, qui va être basé sur une matrice de connexions (R-Mesh) qui reliera quatre blocs de calcul : un comparateur, un additionneur, un multiplicateur et un diviseur. De plus, des registres (rassemblés sou forme d'une mémoire) sont associés à ces blocs, pour permettre un bon contrôle du flux de données.

f) Module 'software'

La gestion de la reconfiguration et le contrôle des différents échanges de données entre les blocs cités plus haut est exécuté sous forme logiciel, où des procédures de contrôle sont exécutées en utilisant des programmes (ensembles d'instructions) implantés dans un microcontrôleur ou processeur de type RISC, ou bien sur des machines à états finis (FSM) qui génèrent les signaux de contrôle de façon séquentielle.

IV.1.4.2 Partitionnement spatiale

Le partitionnement spatial vise à affecter sur le hardware et le software les tâches exécutées par le système flou, par leur allocation architecturale.

Les blocs alloués précédemment (Figure IV.33) définissent une structure plus fine, où l'assignation des tâches du traitement flou est définie comme suit :

Fuzzification : cette tâche est exécutée dans la partie hardware.

Sélection des règles : cette tâche est réalisée par la partie hardware via l'adressage de la mémoire des règles.

Inférence et défuzzification : ces deux parties sont intégrées ensembles dans le hardware reconfigurable, où les tâches d'inférence et de défuzzification sont exécutées en combinant les 4 éléments de calcul, par la réorganisation des connexions qui relient tous ces

éléments, d'autres modules qui sont nécessaires pour la gestion de la reconfiguration et pour le contrôle des entrées/sorties des différents éléments seront ajoutés si besoins dans les phases d'assemblage et de vérification.

Le contrôleur de reconfiguration sera implanté en hardware et en software pour pouvoir contrôler la partie reconfigurable et gérer les différentes composantes du système flou.

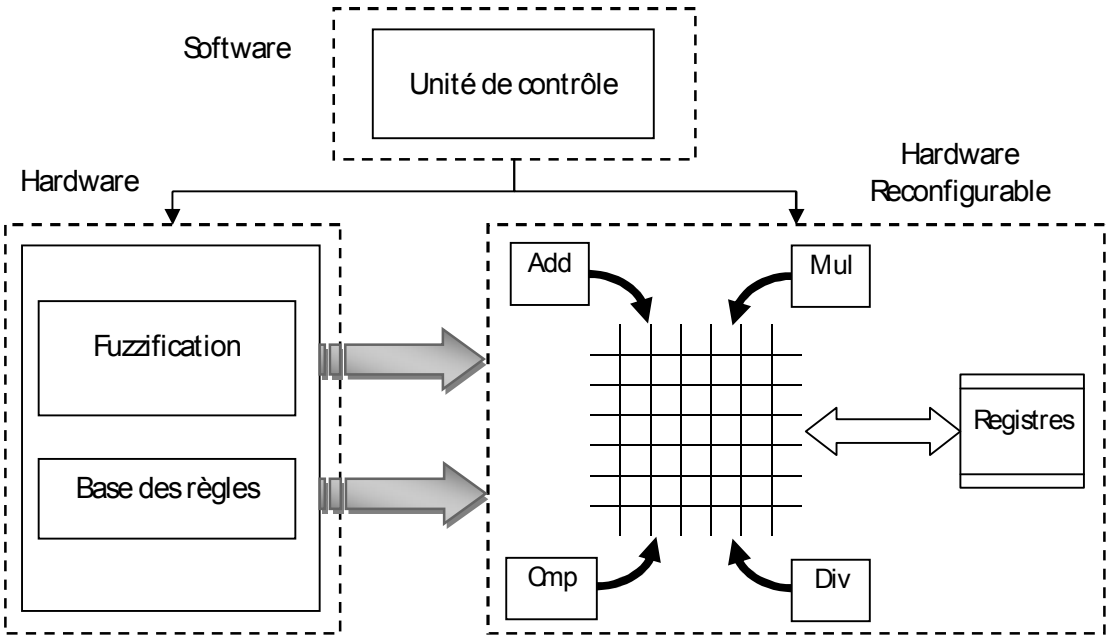


Figure IV.33 : Partitionnement spatial.

IV.1.4.3 Partitionnement temporel

C'est l'étape clé du partitionnement des systèmes reconfigurables, elle intervient lorsque les composants matériels sont beaucoup plus petits que la somme des ressources nécessaires pour toutes les configurations nécessaires au système. Au lieu d'essayer de redimensionner le matériel ou de diminuer les tailles des configurations, on charge (et décharge) les configurations dans les composants lorsqu'on en a besoin. Dans notre système, cette étape consiste à regrouper les tâches nécessaires à l'exécution des méthodes d'inférence et de défuzzification au sein de plusieurs contextes qui seront chargés et déchargés si besoin de façon séquentielle.

Donc si une méthode de défuzzification complexe nécessitera des ressources matérielles dépassants les composants existants dans le hardware reconfigurable, deux solutions sont envisageables ; dans un premier temps nous utiliserons des registres pour temporiser les échanges de données entres les différents blocs reconfigurables. Une autre alternative serait possible où plusieurs contextes seront crées et chargées dans l'élément reconfigurable pour pouvoir effectuer cette tâche, et entre chaque chargement de contexte une sauvegarde systématique des données

intermédiaires de calcul sera indispensable pour la continuité de l'exécution sans la moindre perte d'information (Figure IV.34).

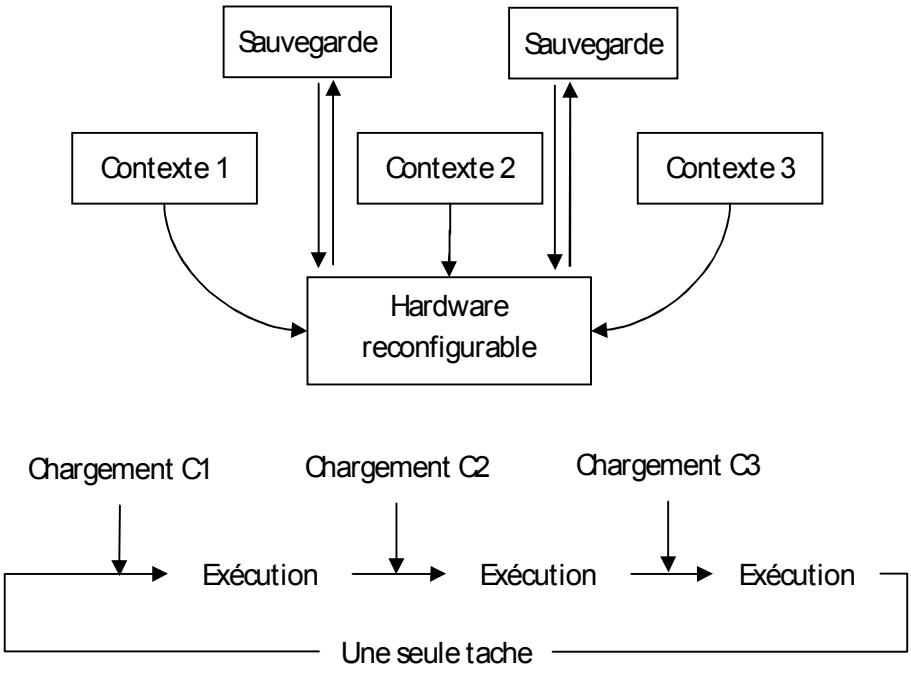


Figure IV.34 : Partitionnement multi-contexte.

IV.1.4.4 Ordonnancement

Cette étape a pour but d'ordonnancer les exécutions des tâches, les communications et les reconfigurations du hardware, elle permet aussi d'adapter l'allocation des ressources en fonction des évolutions des temps d'exécution des taches. Pour arriver à un tel niveau de maitrise des composants reconfigurables dans le temps et dans l'espace, des moyens de contrôle très complexes sont nécessaires.

IV.2 Conception structurelle

Dans cette section, nous allons affiner la structure des différents blocs du système flou et nous dériverons les moyens et algorithmes utilisés pour leurs synthèses et leurs implémentations.

La plateforme de développement Xilinx ISE 9.1i est utilisée pour la compilation des modules VHDL, leurs synthèses et leurs implémentations. Les modules VHDL seront décrits par des composants structurels ou comportementaux, suivant le niveau d'abstraction approprié pour la conception de chaque bloc du système et suivant les ressources existantes sur le FPGA.

Le circuit FPGA choisi pour l'implémentation de notre système flou est le Virtex™II Pro, que nous avons estimé qu'il serait le mieux adapté pour notre application, parce qu'il

comprend des éléments mémoire de grande capacité avec des temps d'accès inférieurs à ceux des autres FPGAs, aussi il comporte des modules de calcul intégrés très rapide comme nous allons le voir par la suite.

Tableau IV.3 : Caractéristiques du circuit FPGA Virtex II pro (xc2vp2-7fg256)

Élément	Nombre
Nombre de Slices	1,408
Nombre de LUTs	2,816
Nombre d'IOBs	140
Nombre de RAM16	14
Nombre de MULT18X18s	12
Nombre de GCLKs	16

IV.2.1 Hardware non-reconfigurable

IV.2.1.1 Fuzzification

Comme nous l'avons déjà fait remarquer, ce bloc est constitué essentiellement de mémoire, Pour y parvenir, nous avons généré les valeurs floues correspondantes à chaque entrée en tenant compte de la forme de l'univers du discours; ces valeurs sont enregistrées dans une RAM qui peut contenir jusqu'à 256 mots (chaque entrée est codée sur 8 bits), chaque mot contient deux fonctions d'appartenance et leurs degrés d'appartenance correspondants (Figure IV.35).

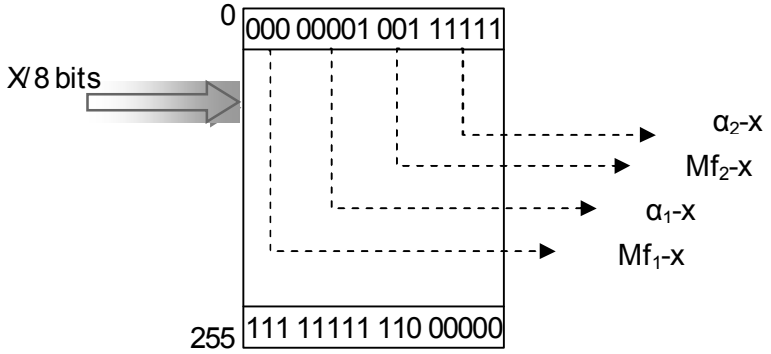


Figure IV.35 : RAM de fuzzification

Les fonctions d'appartenance sont codées sur 3 bits (leur nombre est de 7). Cependant le codage correspondant des degrés d'appartenance dépend de la précision voulue, car l'augmentation du nombre de bits des degrés d'appartenance (petit pas d'échantillonnage) améliore la précision du contrôleur mais augmente fortement les latences du système.

Cette méthode nous permet d'éviter l'utilisation de deux mémoires l'une pour stocker les fonctions d'appartenance, et l'autre pour stocker les degrés d'appartenance, ainsi elle

permet de sélectionner les fonctions d'appartenance et leurs degrés correspondants en même temps par un adressage direct.

L'implémentation du module de fuzzification repose sur l'utilisation d'une des RAMs se trouvant sur notre FPGA, nous avons besoin de quelques signaux de commande et d'entrées nécessaires au chargement de l'univers du discours (Figure IV.36).

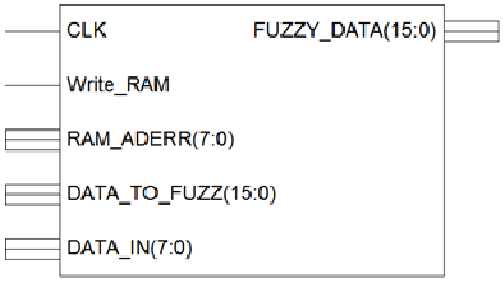


Figure IV.36 : Module de fuzzification

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.4, où on voit clairement que l'outil d'implantation du programme VHDL à bien sélectionné un bloc de RAM, au lieu de l'utilisation de la logique câblée (utilisée dans l'implémentation de certains FPGA).

Tableau IV.4 : Résultats de synthèse de la fuzzification sur xc2vp2-7fg256

Logic Utilization	Used	Available	Utilization
Number of Slices containing only related logic	0	0	0%
Number of Block RAMs	1	12	8%
Number of GCLKs	1	16	6%
Total equivalent gate count for design	65,539		

Concernant la fréquence maximale d'exécution de ce module, elle est généralement tirée des résultats donnés par Xilinx ISE dans le rapport de synthèse, où les temps de propagation pour la fuzzification sont :

Temps de validation avant horloge (tv) = 2.134 ns.

Temps de propagation après horloge (tp) = 2.924 ns.

Ce qui donne une période minimale de 5.058 ns et une fréquence maximale d'exécution de 197.7 Mhz.

Simulation : la simulation de la fuzzification est présentée sur la Figure IV.37, l'écriture d'une donnée dans la RAM de fuzzification se fait après validation du signal de contrôle "write_ram", et l'application de l'adresse cible sur le bus

d'adresse "ram_addr". Le fonctionnement en mode de fuzzification se fait en mettant "write_ram" à '0', quelques exemples sont ainsi donnés.

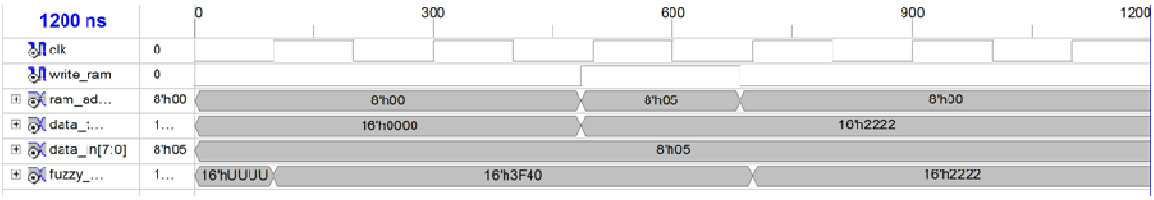


Figure IV.37 : Simulation de la fuzzification.

IV.2.1.2 Base des règles

Les implications des règles sont enregistrées dans une mémoire RAM de 147bits (soit : 49 règles codées sur 3 bits (Tableau IV.5)) pour former la matrice de commande du système flou, où les implications sont codées sur 3 bits pour réduire la taille de la RAM. Pour sélectionner les 4 règles actives ; on adresse de la mémoire des règles par le résultat de la combinaison des fonctions d'appartenance issue du bloc de fuzzification comme montré sur la Figure IV.38.

Tableau IV.5 : Codage des règles

Conclusion	Abréviation	Code binaire
Positive large	PL	111
Positive moyenne	PM	110
Positive petit	PS	101
Zéro	Z	100
négatif petit	NS	011
négatif moyen	Nm	010
Négatif large	NL	001

Ram (address) = Ram (Mf1-x1 & Mf2-x2).

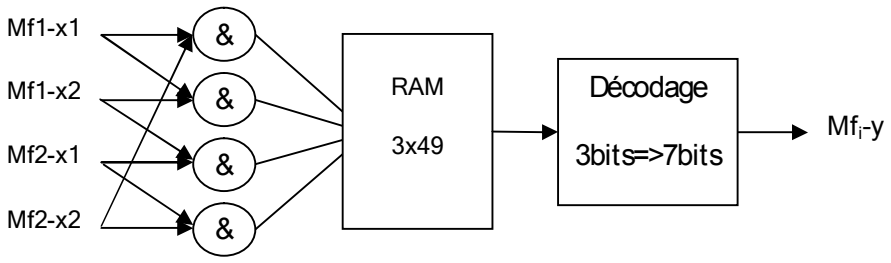


Figure IV.38 : Sélection des règles Actives.

Une règle est sélectionnée à chaque cycle d'horloge, ainsi les règles sont envoyées en sortie en mode pipeline. Dans la dernière phase de la sélection des règles, on convertit les 4 règles actives par un décodage de 3 à 8 bits pour obtenir les valeurs singleton des fonctions d'appartenance relatives à la sortie.

L'implantation du module est représentée sur la Figure IV.39. Et comme pour la fuzzification, l'écriture des différents paramètres nécessaires pour sont fonctionnement dont la table des règles se fait de l'extérieur via le bus d'adresse 'rules_Aderr' et le bus de donnée 'Rule_in'.

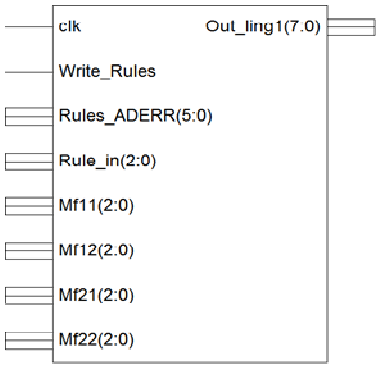


Figure IV.39 : Module de la sélection des règles.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.6, où l'implantation du module se fait sur des RAMs d'un autre type à cause du multiplexage qui se fait avant l'adressage de la table des règles.

Tableau IV.6 : Résultats de synthèse de la sélection des règles sur xc2vp2-7fg256

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	5	2,816	1%
Total Number of 4 input LUTs	34	2,816	1%
Number used for Dual Port RAMs	24		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	3,210		

La fréquence maximale d'exécution est de 150.8 Mhz (tv= 3.168 ns, tp= 3.461ns, T_{min} = 6.629ns).

Simulation : l'écriture d'une donnée dans la RAM des règles se fait après validation du signal de contrôle "write_rules". Quelques exemples montrent le fonctionnement en mode sélection des règles (Figure IV.40), où pour chaque 4 fonctions d'appartenance

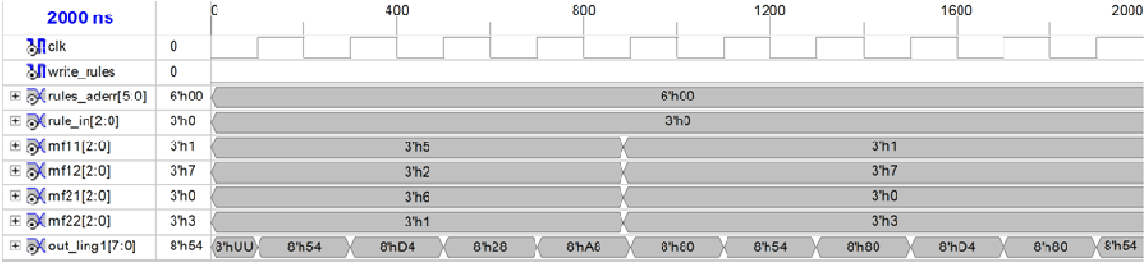


Figure IV.40 : Résultats de simulation de la sélection des règles.

'mfi' 4 règles sont sélectionnées et envoyées en séries.

IV.2.2 Hardware reconfigurable

C'est la partie la plus difficile à mettre en œuvre, car il s'agit d'une composition hétérogène d'éléments de calcul, qui nécessitent une bonne maîtrise de l'outil de conception et de la description de chaque élément. De plus, une mauvaise configuration de toutes ses composantes compliquera d'avantage le contrôle de ce bloc.

Le hardware reconfigurable doit permettre au système flou d'exécuter toutes les méthodes de calcul des inférences et de défuzzification, c'est pourquoi nous allons concevoir les éléments de calcul de façon à ce qu'ils soient capables de s'adapter aux différents algorithmes. Dans les paragraphes suivants nous détaillerons tous les modules nécessaires pour l'implémentation de ce bloc.

IV.2.2.1 Éléments de calcul

a) Comparateur (*CMP*)

Cet élément est nécessaire pour effectuer les comparaisons des degrés d'appartenance, il est utilisé de plusieurs façons pour exécuter certaines méthodes de calcul des inférences et de défuzzification.

Cet élément doit être capable de :

- Trouver le maximum ou le minimum d'une série de variables.
- Donner le rang de l'implication qui correspond au maximum ou au minimum d'une série de variables données.

Pour que le comparateur puisse effectuer ces différentes opérations, il doit être lui-même reconfigurable, de ce fait nous avons choisi l'implémentation suivante :

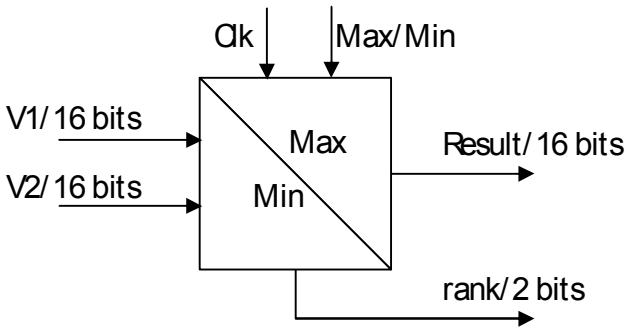


Figure IV.41 : Comparateur reconfigurable.

Parce que les variables sont acheminées vers le bloc de comparaison par série de 4 variables, alors les comparaisons vont se succéder sur 3 ou 4 tops d'horloge pour donner le résultat et le rang correspondant.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.7, où nous remarquons que ce module est implanté en totalité en utilisant la logique câblée (LUT).

Tableau IV.7 : Résultats de synthèse du comparateur sur xc2vp2-7fg256

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	4	2,816	1%
Number of 4 input LUTs	29	2,816	1%
Number of occupied Slices	17	1,408	1%
IOB Flip Flops	10		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	337		

La fréquence maximale d'exécution est de 127.6 Mhz (tv = 4.540 ns, tp = 3.293ns, T_{min} = 7.833ns).

Simulation : pour quelques valeurs d'entrées, l'exécution du comparateur est donnée sur la figure suivante. Chaque 4 tops d'horloge, la valeur la plus grande est donnée en sortie avec sans classement.

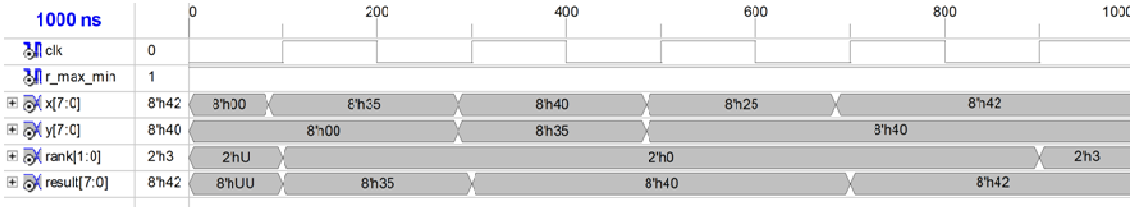


Figure IV.42 : Résulats de simulation de CMP.

b) Additionneur (*ADD*)

Cet élément est aussi considéré reconfigurable, car il permet de faire l'addition et la soustraction sans changé le hardware, suivant que le signal "Add_Sub" est mis ou non à '1'.

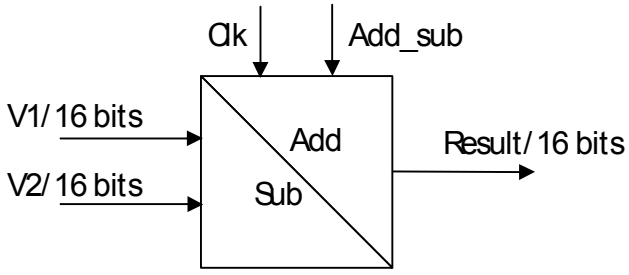


Figure IV.43 : Additionneur/ Soustracteur reconfigurable.

Résultats de synthèse : l'existence de blocs de 'additionneurs partiels' configurables en soustracteurs sur les FPGAs a réduit considérablement la surface utilisée pour

l'implantation de ce module, les résultats de synthèse sont résumés dans le Tableau IV.8.

Tableau IV.8 : Résultats de synthèse de Add/ Sub sur xc2vp2-7fg256

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	16	2,816	1%
Number of 4 input LUTs	32	2,816	1%
Number of occupied Slices	16	1,408	1%
Number of GCLKs	1	16	6%
Total equivalent gate count for design	416		

La fréquence maximale d'exécution est de 146.9 Mhz (tv = 3.512 ns, tp = 3.293ns, T_{min} = 6.805ns).

Simulation : La simulation de 'Add/ Sub' pour les deux modes de fonctionnement est donnée sur la figure suivante.

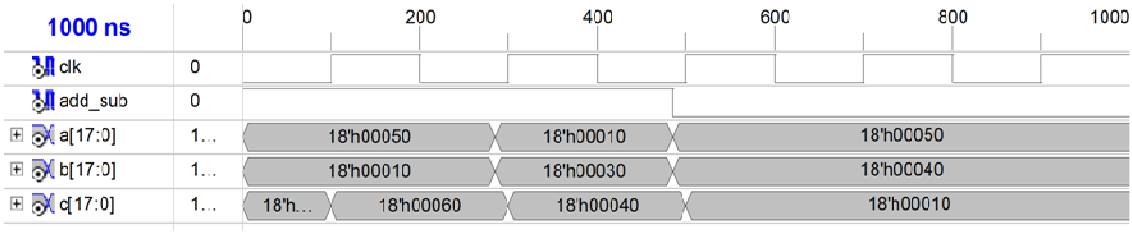


Figure IV.44 : Résulats de simulation de Add/ Sub

c) Multiplicateur (*MUL*)

Plusieurs algorithmes peuvent être utilisés pour implémenter la multiplication, mais tous les algorithmes de multiplication binaires qui utilisent les méthodes d'additions ou celles des décalages successifs produits des retards très importants dans les réponses du système. Alors, pour que la multiplication soit la plus performante possible nous utiliserons un des 12 multiplicateurs hardwares qui sont intégrés sur le Virtex II pro. Ces multiplicateurs peuvent faire des multiplications de 18 bits, mais nous les configurerons pour des multiplications de 8 bits. Si nous aurons besoin de plus, un simple changement de paramètre générique suffira pour les adapter.

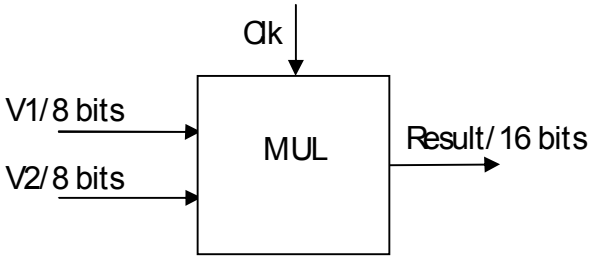


Figure IV.45 : Multiplicateur.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.9.

Tableau IV.9 : Résultats de synthèse de Mul sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	8	2,816	1%
Number of occupied Slices	4	1,408	1%
IOB Flip Flops	16		
Number of MULT18X18s	1	12	8%
Number of GCLKs	1	16	6%
Total equivalent gate count for design	4,179		

La fréquence maximale d'exécution est de 147.4 Mhz (tv = 3.069 ns, tp = 3.714 ns, T_{min} = 6.783ns).

Simulation : Quelques exemples de simulation du module de multiplication sont donnés sur la Figure IV.46, où on remarque qu'un seul cycle d'horloge est suffisant pour que la réponse soit disponible sur le port de sortie.

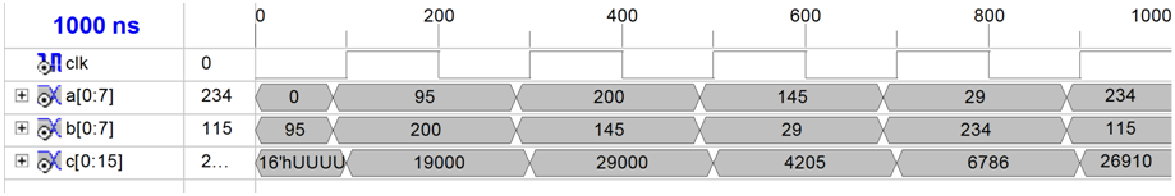


Figure IV.46 : Résulats de simulation de Mul.

d) Diviseur (*DIV*)

Il existe une variété d'algorithmes de division intégrables sur hardware **[42]**, qui utilisent des méthodes assez complexes et qui nécessitent beaucoup de temps de calcul ou beaucoup plus de surface allouable (dans le cas des FPGAs on parle plus du nombre de blocs CLBs utilisés).

La vitesse et la surface du système flou dépendent en général de façon très critique de l'algorithme de division appliqué. En effet, la division consomme un pourcentage élevé de la surface du FPGA. Pour cela et pour empêcher la dégradation des performances globales du système, nous avons utilisé la division binaire à décalage (Binary Shift Division in Data Path) **[43]**.

La Figure IV.47 illustre un exemple de cette division (8 bits E/S), cette méthode peut être implémentée pour être executée en un top d'horloge, ou bien sur plusieurs tops d'horloge (8 tops d'horloge pour un mot de 8 bits), afin de réduire la surface consommée sur le FPGA, mais elle entraine des réactions très lentes du système aux changements qui peuvent survenir sur les entrées.

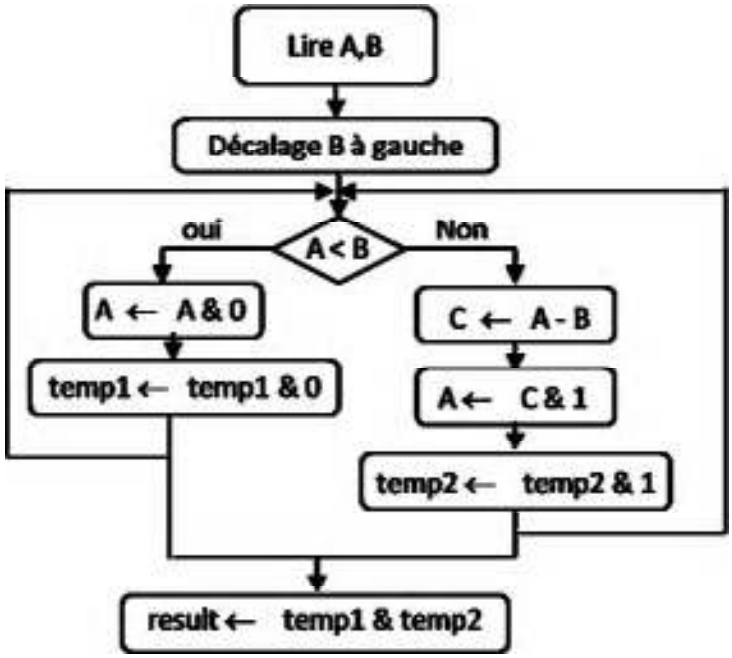


Figure IV.47 : Algorithme de division.

Même si cet algorithme est exécuté en mode pipeline pour accélérer la division **[44] [45]**, nous aurons de grandes difficultés pour le contrôler et par conséquent de contrôler tout le hardware reconfigurable, pour toutes ces raisons nous avons estimé que la meilleure solution est que le module de division soit exécuté en un seul top d'horloge.

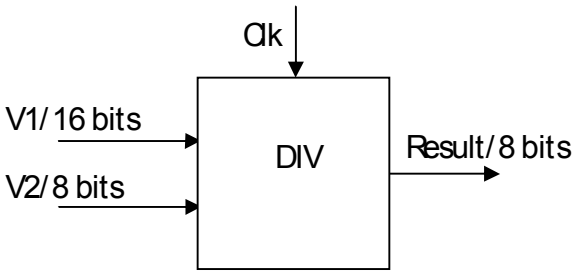


Figure IV.48 : Module de division.

La Figure IV.48 présente le bloc le module de division, où 'V1' est le dividende, 'V2' est le diviseur et 'result' est le quotient. Cette division est spécialement adaptée pour les calculs de la logique floue.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.10.

Tableau IV.10 : Résultats de synthèse de Div sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	267	2,816	9%
Number of occupied Slices	137	1,408	9%
IOB Flip Flops	8		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	2,458		

La fréquence maximale d'exécution est de 37.8 Mhz (tv = 23.177 ns, tp = 3.293ns, T_{min} = 26.41ns), cette faible fréquence va limiter sérieusement les performances du système flou.

Simulation : Des exemples de simulation sont données sur la Figure IV.49, où le résultat qui représente le quotient est donné en un top d'horloge :

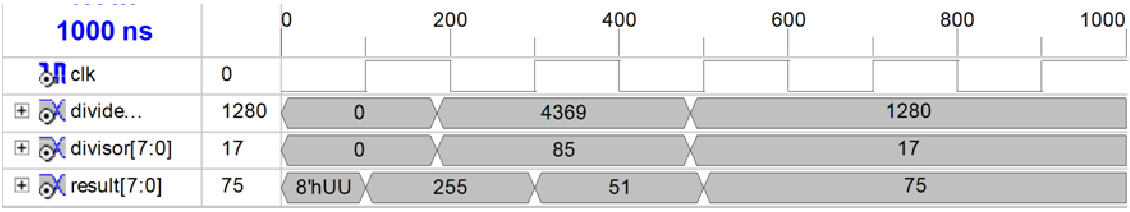


Figure IV.49 : Résultats de simulation de Div.

IV.2.2.2 Éléments de sauvegarde et de traitement

Des éléments sont ajoutés à la structure du système pour la préparation et la sauvegarde des données, ceci étant nécessaire pour la mise en marche du système flou.

e) Élément de traitement d'inférence

Il permet de conditionner les degrés d'appartenance émis par le bloc de fuzzification, pour faciliter leur traitement dans les modules du hardware reconfigurable.

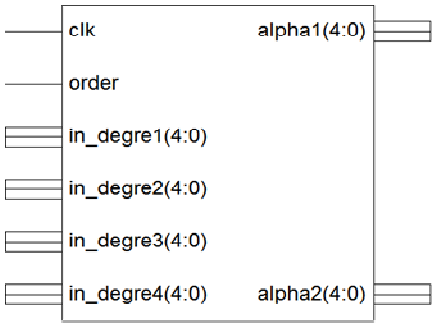


Figure IV.50 : Module de traitement d'inférence

Ce module reconfigurable dispose de deux modes de fonctionnement ; le premier assure l'envoi des α_i (degrés d'appartenance) selon un ordre précis sur un seul port pour permettre leur identification suivant leur règle respective, le deuxième réalise la transmission deux à deux pour permettre leur comparaison.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Figure IV.39:

Tableau IV.11 : Résultats de synthèse de Infer sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	2	2,816	1%
Number of 4 input LUTs	26	2,816	1%
Number of occupied Slices	14	1,408	1%
IOB Flip Flops	15		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	325		

La fréquence maximale d'exécution est de 155.8 Mhz (tv = 3.122 ns, tp = 3.293 ns, T_{min} = 6.415ns).

Simulation : cette simulation se fait en deux temps, le premier en mettant le signal "order" à '0' pour avoir les α_i sur les 2 sorties du module et le second en mettant le signal "order" à '1' pour envoyer les α_i sur une seule sortie et en forçant l'autre en haute impédance.

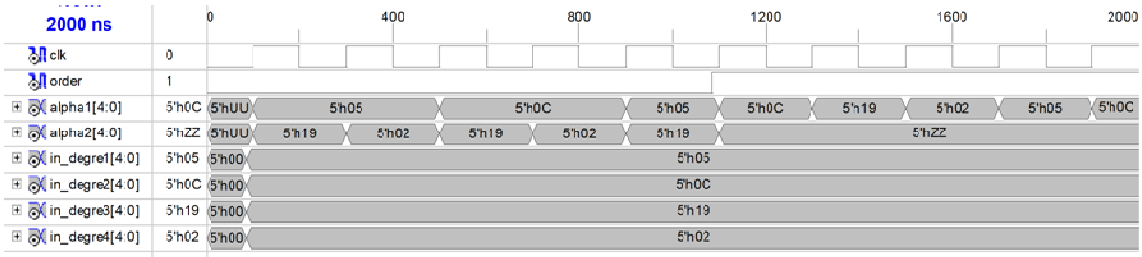


Figure IV.51 : Résulats de simulation du traitement d'inférence.

f) Élément de sauvegarde (*File System Register : FSR*)

Cet élément est très important pour la gestion du flux de données, il permet entre autres de sauvegarder les données intermédiaires de calcul, il est composé de 16 registre de 16 bits.

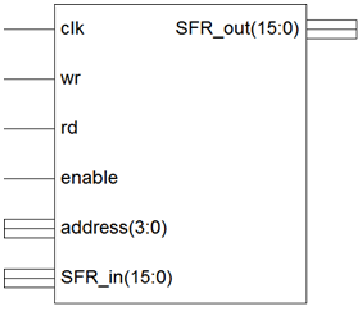


Figure IV.52 : Module FSR

Deux manières peuvent être utilisées pour la description de ces registres, l'une est d'utiliser des bascules, alors que l'autre utilise une mémoire RAM, cette dernière est choisie dans notre application car la première consomme beaucoup de surface.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.12.

Tableau IV.12 : Résultats de synthèse de FSR sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	1	2,816	1%
Number of occupied Slices	9	1,408	1%
Number used as 16x1 RAMs	16		
IOB Flip Flops	32		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	2,361		

La fréquence maximale d'exécution est de 164.5 Mhz (tv = 2.783 ns, tp = 3.293 ns, T_{min} = 6.076ns).

Simulation : la simulation du module se fait en mettant le signal "ram_enable" à '1', ensuite on valide le signal "wr" pour permettre l'écriture des données dans leurs adresses spécifiques, et en mettant le signal 'rd' à '1' on peut lire le contenu des registres.

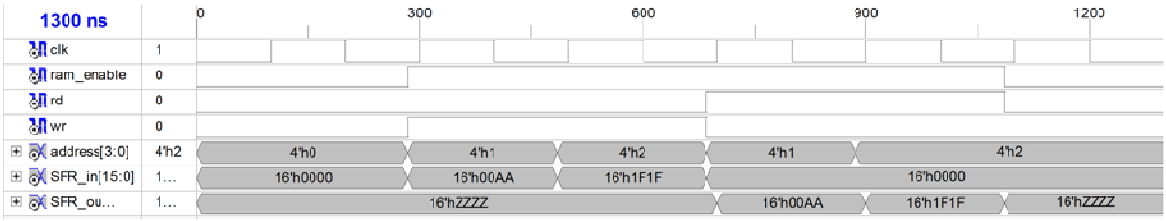


Figure IV.53 : Résulats de simulation d l'élément de sauvegarde(FSR).

IV.2.2.3 Matrice de connexion

Après avoir conçu les 4 éléments de calcul, et les éléments de sauvegarde et d'adaptation, la mission qui reste est leur assemblage pour former le hardware reconfigurable. La manière la plus convenable pour le faire, tout en prenant en compte l'hétérogénéité de ces éléments est d'utiliser une matrice de connexion dont la description VHDL sera écrite sous forme comportementale, ce qui permettra de gérer le flux de données entre les différents blocs du hardware reconfigurable de façon très intelligente évitant ainsi de rassembler une structure complexe au niveau logique (mux, nand..) qui sera inappropriée pour notre système. Ce choix architectural est expliqué plus tard.

Pour connecter tous les éléments hardwares efficacement nous les représenterons par des blocs symboliques avec leurs entrées/sorties utiles au traitement flou et qui doivent être reliées à la matrice d'interconnexion.

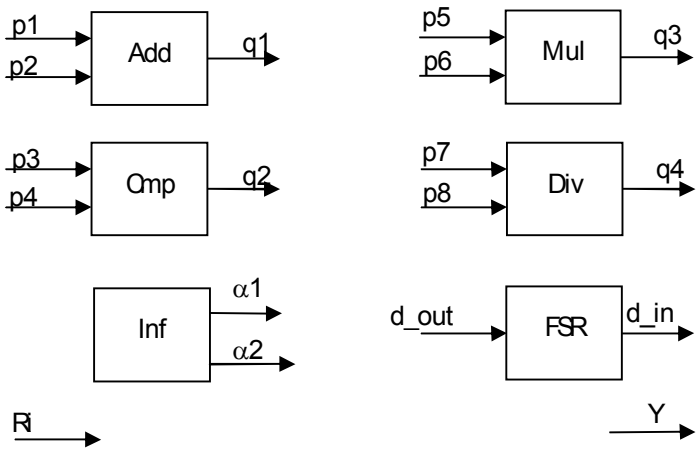


Figure IV.54 : Représentation symbolique des éléments reconfigurables.

Où : p1 – p10 et q1 – q4 correspondent respectivement aux entrées et aux sorties des éléments de calcul (les qi sont vue par la matrice de connexion comme des entrée, et les pi sont ses sorties).

Ri : les règles issues du bloc de la base des règles (entées de la matrice de connexion).

αi : les degrés d'appartenance (entrées de la matrice de connexion).

Y : sortie de la matrice de connexion, aussi c'est la sortie du système flou.

Notre solution pour l'implémentation de cette matrice est inspirée de la structure des FPLA (*Fied Programmable Logic array).* La description comportementale de ce module en VHDL permet l'adaptation de la largeur des données et l'optimisation des points de connexions, ce qui est très difficile à réaliser dans une description structurelle comportant des multiplexeurs. Le module d'interconnexion est représenté sur la Figure IV.55, les points représentent les connexions possibles pas contre les croix sont des jonctions interdites car les

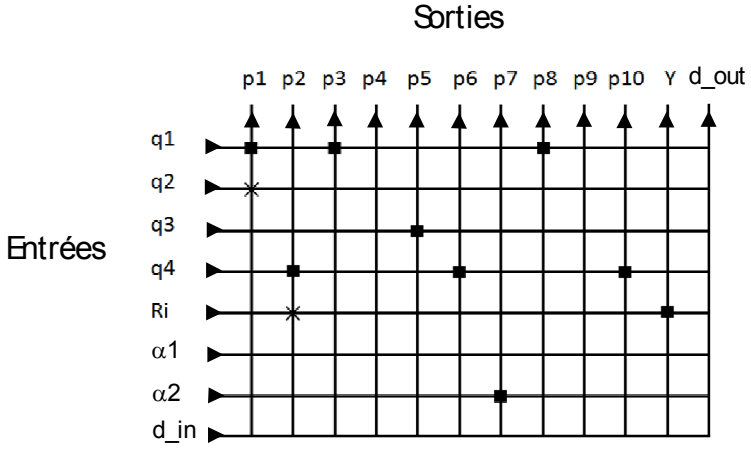


Figure IV.55 : Module d'interconnexion.

sorties ne peuvent avoir qu'une seule source. Un code pour les colonnes et un autre pour les lignes permet l'adressage de la matrice pour définir les points de connexions souhaitées.

Résultats de synthèse : les résultats de synthèse du module d'interconnexion sont résumés dans le Tableau IV.13.

Tableau IV.13 : Résultats de synthèse de 'interconnect' sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	1	2,816	1%
Number of occupied Slices	9	1,408	1%
Number used as 16x1 RAMs	16		
IOB Flip Flops	32		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	2,361		

La fréquence maximale d'exécution est de 130.1 Mhz (tv = 4.322 ns, tp = 3.359 ns, T_{min} = 7.681 ns), cette fréquence va être décisive pour déterminer le temps de reconfiguration, car elle est largement supérieure à la fréquence de l'ensemble des éléments reconfigurable.

IV.2.3 Contrôle de reconfiguration

Une fois tous les éléments hardware (reconfigurable et non-reconfigurable) sont étudiés, nous passons à l'assemblage de tout le système flou, mais avant de procéder à cet assemblage nous avons besoin d'un contrôleur de reconfiguration qui permettra de générer

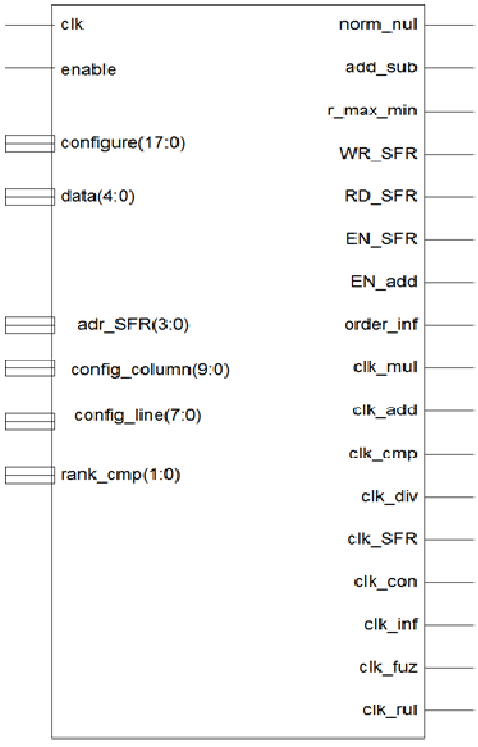


Figure IV.56 : Contrôleur reconfigurable.

tous les signaux de contrôle pour la synchronisation des échanges de données entre les différents modules, et pour permettre le changement des plans de reconfiguration adéquats. À cet effet, beaucoup de méthodes sont utilisées **[46] [47]**, et leurs implémentations sont toujours dépendantes de l'application et de la plateforme visée.

Pour la gestion de la reconfiguration du système flou et du flot de données, nous avons choisi la méthode de gestion par machine d'états (FSM) **[48]**, où les états de la FSM sont chargés avec le contexte de reconfiguration à partir de l'extérieur. Les signaux de commande sont générés pour les modules reconfigurables, mais aussi pour la fuzzification et la base des règles floues, et par soucis de synchronisation nous avons préféré de les gérer avec le même contrôleur (Figure IV.56).

Ce contrôleur est divisé en trois parties gérées par la même machine FSM (Figure IV.57).

Figure IV.57 : Gestion de reconfiguration par FSM

- Gestion des contextes : cette partie renferme les plans de reconfiguration nécessaire pour le changement de contexte, son exécution nécessite un seul état de la FSM, où elle génère deux vecteurs de ligne et de colonne pour valider les points de connexion dans la matrice d'interconnexions, la sauvegarde des données intermédiaires est exécutée dans le même cycle horloge.
- Génération des horloges (multiplexage des horloges) : l'horloge de chacun des blocs connectés à la matrice est une image de l'horloge principale (synchronisation oblige), un multiplexage des horloges est alors exécuté pour valider ou non les horloges secondaires.

- Gestion des signaux de commande : chaque état de la FSM va générer les signaux de commande (‘wr’, ‘rd’, ‘enable’…) relatifs aux éléments hardwares qui sont connectés à la matrice.

Résultats de synthèse : les résultats de synthèse sont résumés ci-dessous :

Tableau IV.14 : Résultats de synthèse du controleur sur xc2vp2-7fg256

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	22	2,816	1%
Number of 4 input LUTs	36	2,816	1%
Total Number of 4 input LUTs	52	2,816	1%
IOB Latches	14		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	525		

La fréquence maximale d’exécution est de 177.4 Mhz (tv = 1.995 ns, tp = 3.460 ns, T_{min} = 5.635 ns), ce qui va garantir une bonne gestion du système sans dégrader ses performances.

Simulation : la simulation du module se fait après la validation du signal ‘enable’ est l’injection des états de la FSM, la Figure IV. 29 montre un exemple de simulation.

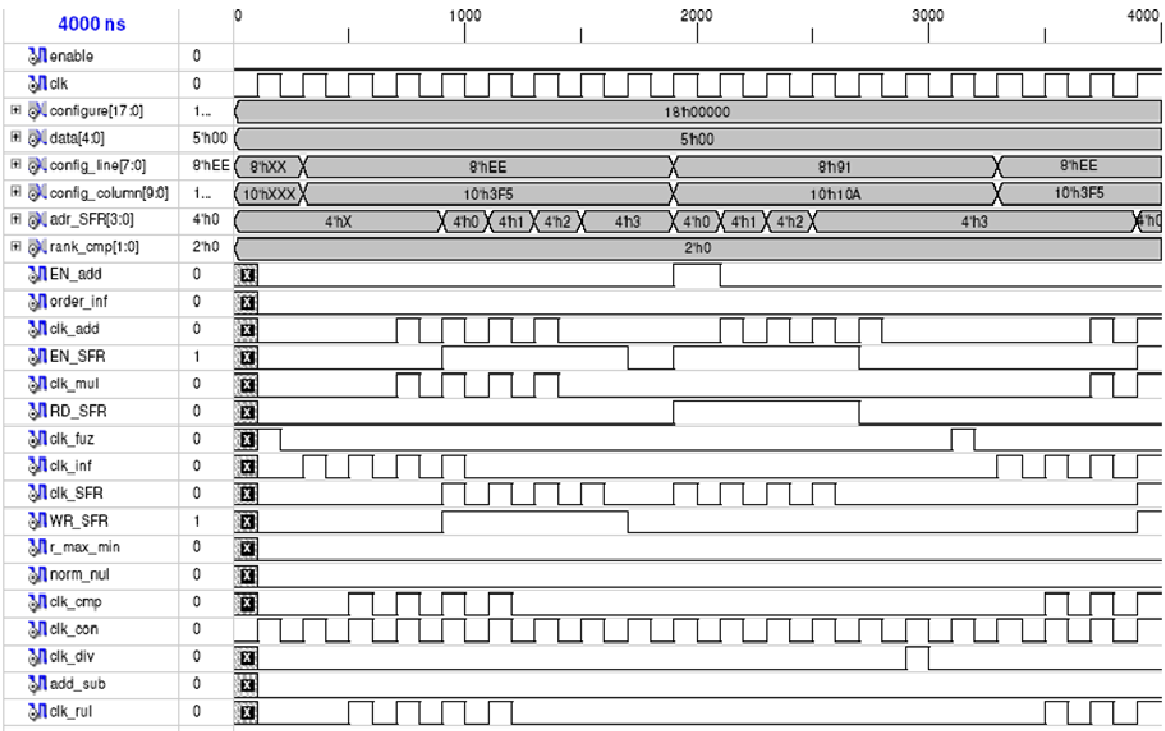


Figure IV.58 : Résultats de simulation du contrôleur de reconfiguration.

IV.2.4 Assemblage du système flou

Pour pouvoir assembler tous les blocs précédemment décrits, une description VHDL de type structurelle est suffisamment adaptée à cette tâche. À la fin de la synthèse et de l'implémentation du système nous arrivons à un composant de type software représenté par la Figure IV.59.

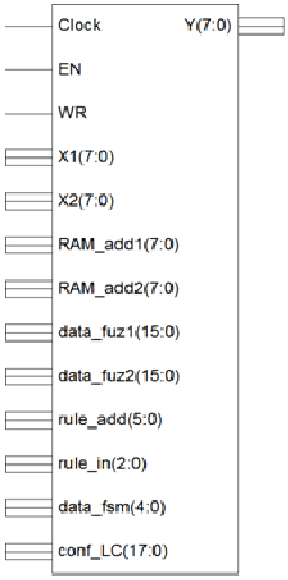


Figure IV.59 : Système flou.

Les entrées X1 et X2 et la sortie Y sont celles du système flou, tandis que les autres ports d'entrées sont celles du chargement des paramètres du contrôleur de reconfiguration et celles relatifs au chargement des données dans les mémoires de fuzzification et de la base des règles floues.

Ces entrées/sorties peuvent êtres gérées soit par un ordinateur hôte soit par un processeur externe à ce composant et qui peut être intégré sur la même puce, nous montrerons dans la suite une solution à ce problème au cours de l'intégration de notre système dans un environnement plus complet.

IV.2.4.1 Synthèse

La synthèse de ce composant a montré une densité acceptable de tout le hardware par rapport au circuit FPGA choisi, l'équivalence totale en porte logique est de 146,954 portes.

La fréquence maximale de fonctionnement dépend de l'élément le moins rapide de la structure soit la division, qui nécessite une fréquence maximale de 37.8 Mhz, et par conséquent la fréquence globale sera un peu moins de 37.0 Mhz.

Tableau IV.15 : Synthèse du système flou sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Total Number Slice Registers	255	2,816	9%
Number used as Flip Flops	241		
Number of 4 input LUTs	673	2,816	23%
Number of occupied Slices	370	1,408	26%
Number used as logic	633		
Number used for Dual Port RAMs	24		
Number used as 16x1 RAMs	16		
Number of RAMB16s	2	12	16%
Number of MULT18X18s	1	12	8%
Total equivalent gate count for design	146,954		
Number of bonded IOBs	107	140	76%

L'implémentation du système sur le Virtex™II Pro de type "xc2vp2-7fg256" est présentée sur la Figure IV.60, où on voit la distribution de tous les éléments hardwares reconfigurables et non reconfigurables sur la surface du FPGA, le nombre de ports nécessaire pour une telle structure est de 107 ports.

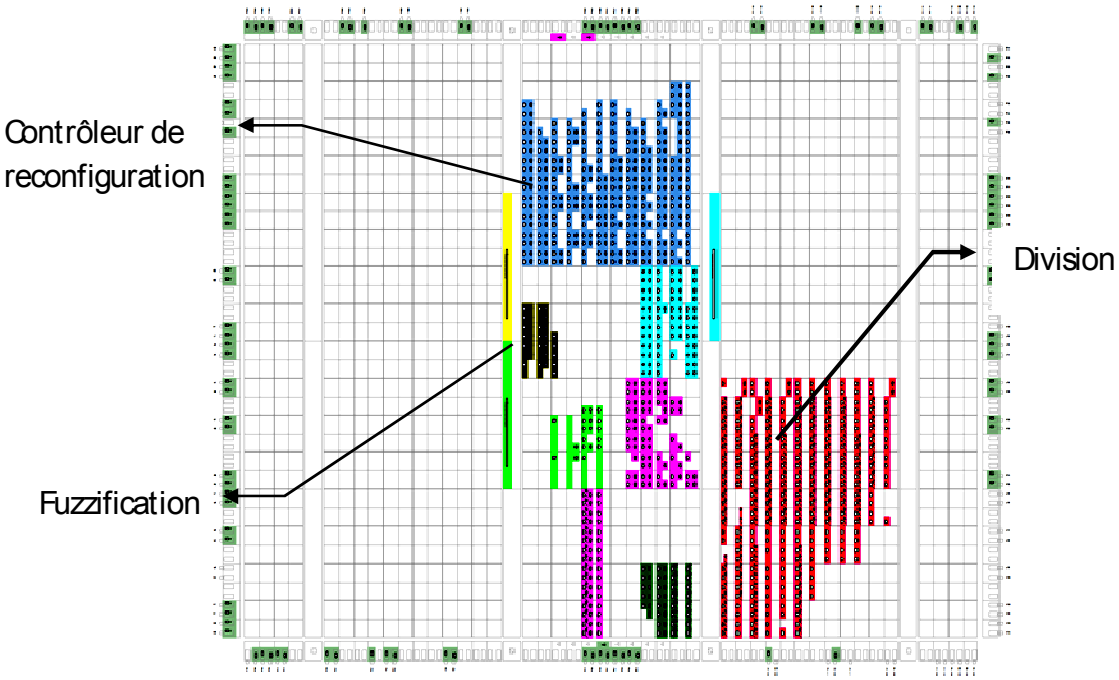


Figure IV.60 : Implémentation du système flou.

IV.2.4.2 Exemple de contrôle flou

Dans cette section un exemple de simulation est présenté, où les paramètres de réglage du contrôleur flou sont :

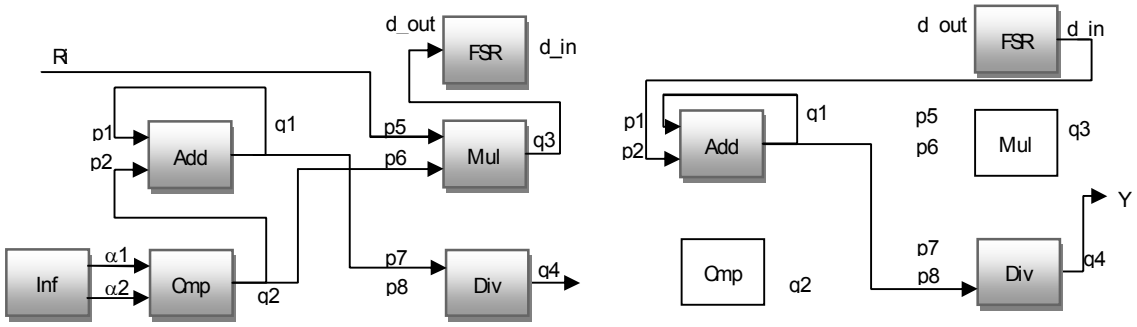
- Fuzzification avec fonction d'appartenances de forme triangulaire de X1 et X2.
- Règles floues (voir Tableau IV.16).
- Inférences par méthode de Mamdani.
- Défuzzification par centre de gravité avec des valeurs de type singletons.

Tableau IV.16 : Règles floues.

Y		X1						
		NG	NM	NS	Z	PS	PM	PG
X2	NG	NG	NM	NS	Z	PS	PM	PG
	NM	NM	NG	NM	NS	Z	PS	PM
	NS	NS	NM	NG	NM	NS	Z	PS
	Z	Z	NS	NM	NG	NM	NS	Z
	PS	PS	Z	NS	NM	NG	NM	NS
	PM	PM	PS	Z	NS	NM	NG	NM
	PG	PG	PM	PS	Z	NS	NM	NG

En ce qui concerne les mémoires de fuzzification et de règles floues, nous avons besoin de les charger de l'extérieur par les données nécessaire pour leur fonctionnement par une série d'adressage des RAMs internes. Par contre pour le calcul des inférences et de la défuzzification nous avons besoin d'une reconfiguration multi-contexte.

Deux contextes sont nécessaires pour effectuer les tâches de calcul, et un contrôle de reconfiguration sur 15 états pour la FSM, dont un seul état est consacré au chargement des plans de reconfiguration par le biais des codes de ligne et de colonne spécifiques à la matrice d'interconnexion.



Contexte n°1.
Ligne : `b'11101110`
`Colonne : b'1111110101`

Contexte n°2.
Ligne : `b'10010001`
`Colonne : b'0100001010`

Figure IV.61 : Reconfiguration à deux contextes.

La simulation du système flou pour cet exemple est effectuée dans un premier temps pour une seule valeur de X1 et X2 pour vérifier la synchronisation des signaux de commande, où la réponse est donnée après 15 cycles d'horloge conformément à la machine d'états finis correspondante (voir Annexe-3).

La Figure IV.63 représente la surface de commande de notre exemple de vérification, qui a été récupéré par simulation du traitement flou sur l'ensemble de l'univers du discours des entrées X1 et X2.

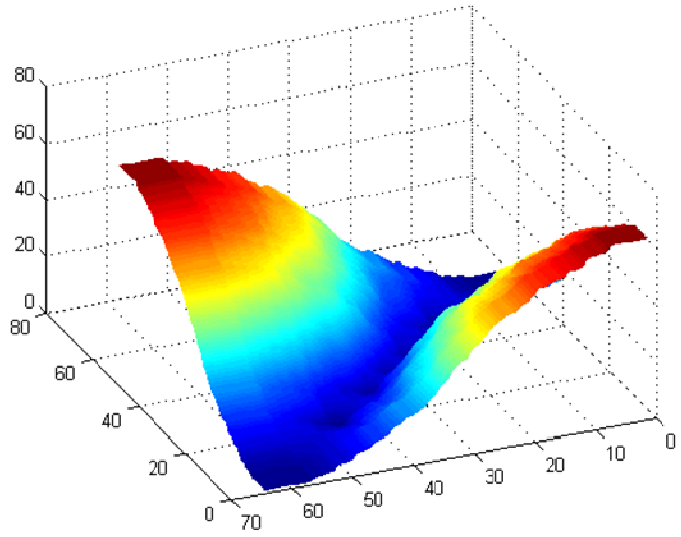


Figure IV.62 : Surface de commande spécifique à l'exemple choisi.

IV.3 Conception du système flou sous forme d'IP-Soft

Cette étape de conception est très importante pour l'intégration du système flou dans son environnement cible. Les besoins en moyens de communication avec d'autres composants virtuels (IP) est indispensable pour une bonne flexibilité et une meilleur adaptabilité du système sur un SoC.

Pour y arriver, nous avons pu respecter des critères de réutilisation **[49]** (voir Annexe-4). qui nous ont assurées une facilité de maintenance du code source durant toutes les phases de conception, toutefois le point le plus important pour l'intégration du système sous forme d'un IP-Soft, est une gestion centralisée de toutes les ressources systèmes et surtout le contrôle des ports de communication, d'où le besoin de l'utilisation d'un CPU de type RISC par exemple.

L'intégration de notre système flou dans un SoC, requiert son association à des interfaces de communication, la plus importante est le bus système commun avec les autres IPs du SoC (Figure IV.64) qui est le bus système Wishbone. Dans notre cas, il sera en mode esclave car notre IP n'aura pas besoin de prendre le contrôle du bus, ainsi il peut récupérer les données nécessaires pour sont fonctionnement ; telles que les configurations, mais aussi le contenu des mémoires de fuzzification et celle des règles.

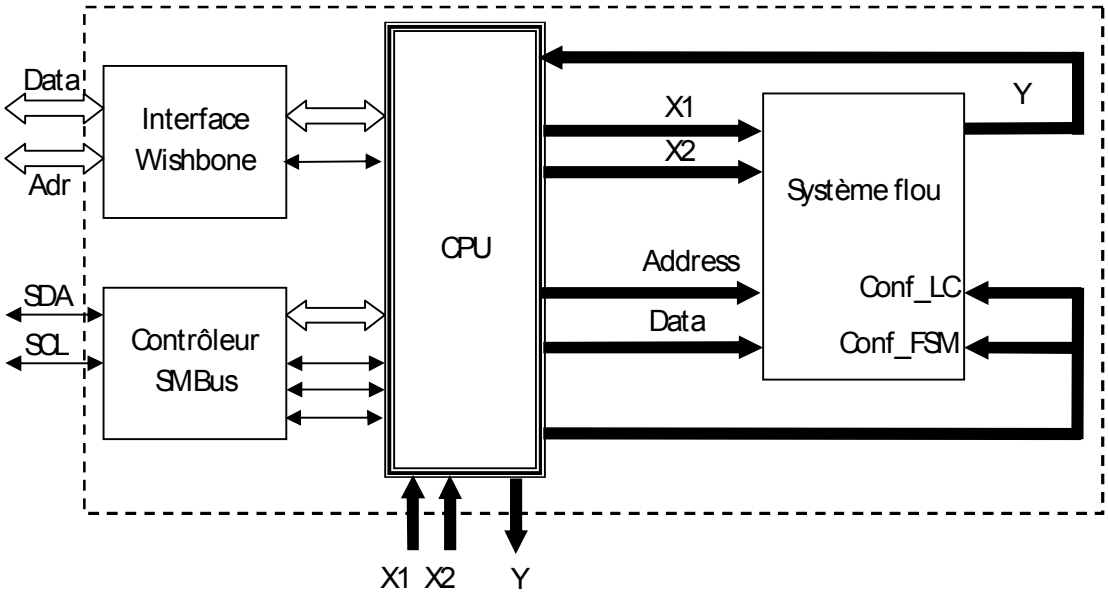


Figure IV.64 : Blocs fonctionnels du système flou

La deuxième interface est consacrée exclusivement au contrôle des modes fonctionnels du SoC, c'est l'interface SMBus ; où le système flou doit être configuré comme un maître pour pouvoir intervenir sur les IPs plus rapidement sans interrompre le fonctionnement du système.

IV.3.1 CPU

Les besoins de notre système en terme de ressources de communication sont relativement faibles comparées à d'autres IPs existants, donc nous avons seulement besoin d'un microcontrôleur de type RISC pour gérer les ports d'entrées/sorties du système flou, ainsi de prendre en charge les protocoles de communication.

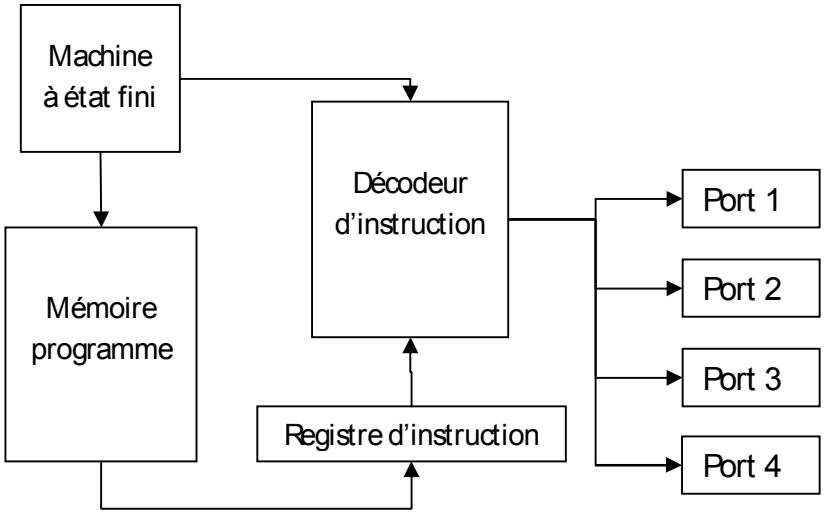


Figure IV.65 : Blocs fonctionnel du CPU

Ceci nécessite en général un temps de développement assez élevé et des ressources de calcul importantes qui ne sont pas disponibles sur le FPGA choisi. C'est pourquoi, dans cette phase de notre travail nous utiliserons une unité de contrôle réduite qui permettra une gestion simplifiée du système flou (Figure IV.65).

IV.3.2 Interface wishbone

Comme il a été signalé dans le premier chapitre concernant le bus Wishbone, il n'y pas de spécifications restrictives, mais plutôt des recommandations, c'est pourquoi il existe plusieurs façons de raccorder un CPU au bus, mais la plus simple est celle de la figure suivante **[6]** :

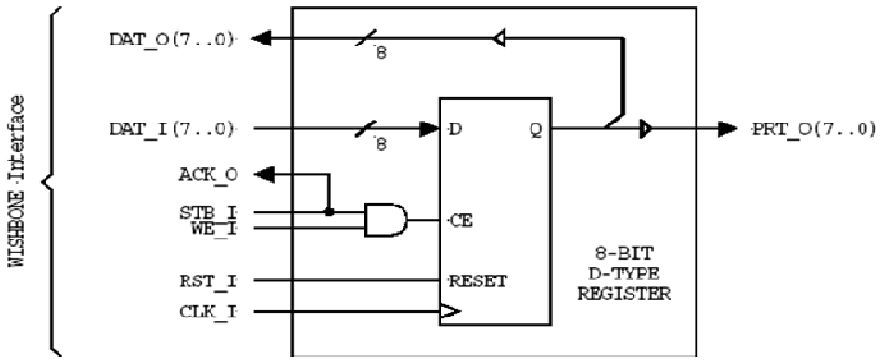


Figure IV.66 : interface Wishbone

Cette interface est celle du Wishbone esclave, toute la structure est implantée en utilisant un registre à bascule 'D' avec un reset synchrone et une seule porte 'AND' ; Durant les cycles d'écriture, la donnée est présentée sur le bus d'entrée DAT_I(7..0), et elle est introduite au front montant de CLK_I quand STB_I et WE_I sont validées.

L'état de la sortie DAT_O (7..0) est contrôlée par le maître, durant le cycle de lecture la porte 'AND' empêche le passage des données dans le registre.

Résultats de synthèse : les résultats de synthèse sont résumés dans le Tableau IV.17 :

Tableau IV.17 : Résultats de synthèse de Slave_Wishbone sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	1	2,816	1%
IOB Flip Flops	16		
Number of GCLKs	1	16	6%
Total equivalent gate count for design	137		

La fréquence maximale d'exécution est de 170.8 Mhz (tv = 2.560 ns, tp = 3.293 ns, T_{min} = 5.853 ns).

IV.3.3 Contrôleur SMBus

Le bus de gestion système SMBus (*System Management Bus*) est une interface à 2 ports, à travers laquelle de nombreux composants peuvent communiquer les uns avec les autres. Il est basé sur le principe de fonctionnement du bus I2C **[50]**.

Généralement, le SMBus fournit au système et aux outils de gestion de puissance un moyen de contrôle très robuste. Un système peut utiliser le SMBus pour passer des messages entre les composants afin de surveiller leurs performances, vérifier leurs identités, sauvegarder leurs états de fonctionnement et reporter différents types d'erreurs.

Le SMBus est un moyen pour connecter plusieurs composants, dont un seul composant maître est admis ; ce composant initialise le transfert des données avec des composants esclaves en leur fournissant le signal d'horloge.

Conception du contrôleur SMBus

Le contrôleur SMBus contient une interface microcontrôleur synchrone qui permet à un microprocesseur ou un microcontrôleur de se connecter au bus SMBus, et une autre interface maître pour gérer le protocole de communication (Figure IV.67).

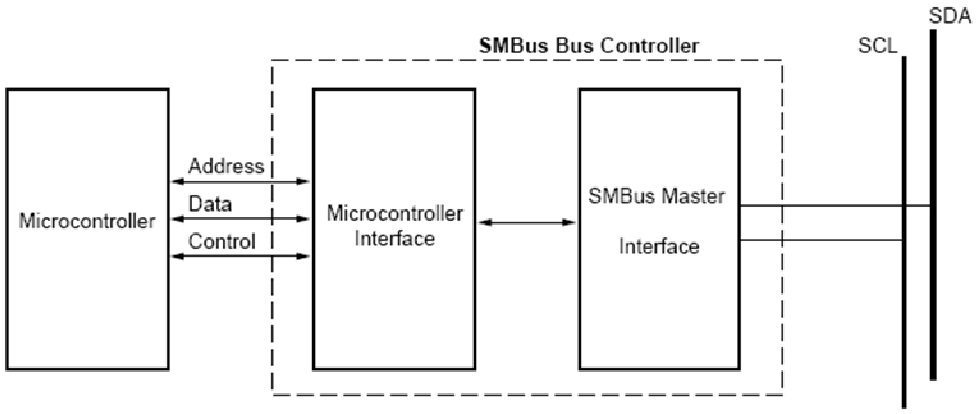


Figure IV.67 : Contrôleur SMBus.

Deux ports sont indispensables SDA (*serial data*) et SCL (*serial clock*), qui transportent l'information entre les composants connectés au bus. Il y a deux autres ports optionnels SMBUS et SMBALERT, qui ne sont pas inclus dans cette implémentation. Il n'y a pas de limitation sur la capacité du bus comme pour l'I2C, Les deux lignes SDA et SCL sont bidirectionnelles et elles sont connectées à une tension positive via des résistances pull-up.

Le bloc diagramme du contrôleur SMBus est présenté sur la Figure IV.68, où il est divisé en deux majeures parties ; une interface pour microcontrôleur et une interface SMBus. L'interface microcontrôleur supporte un protocole synchrone pour le contrôle du bus de

donnée, ce protocole définit la méthode selon laquelle le microcontrôleur lit et écrit les données dans les registres du contrôleur.

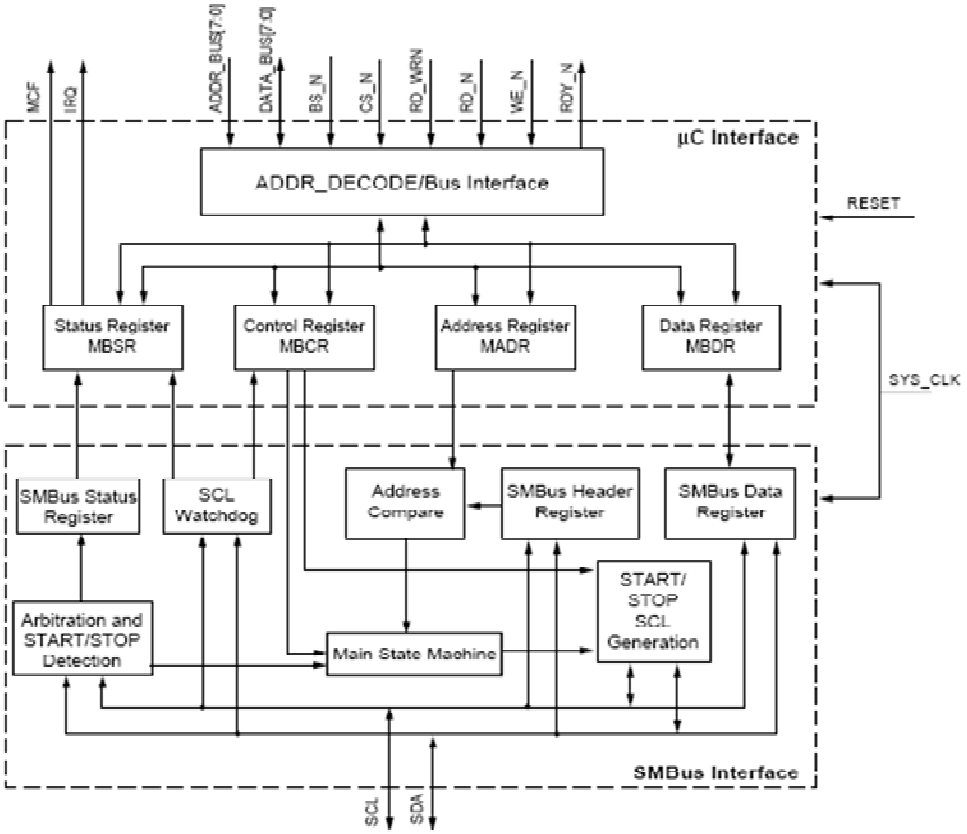


Figure IV.68 : Bloc diagramme du contrôleur SMBus.

Résultats de synthèse : les résultats de synthèse du contrôleur SMBus sont résumés dans le Tableau IV.18 :

Tableau IV.18 : Résultats de synthèse du SMbus sur xc2vp2-7fg256.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	119	2,816	4%
Number of occupied Slices	150	1,408	10%
Total Number of 4 input LUTs	230	2,816	8%
IOB Flip Flops	18		
Number of GCLKs	2	16	12%
Total equivalent gate count for design	2,665		

La synthèse de l'interface du microcontrôleur est de 663 portes logique et 2002 portes logiques pour l'interface du SMBus, aussi La fréquence maximale d'exécution est de 144.0 Mhz (tv = 3.328ns, tp = 3.614ns, Tmin = 6.942ns).

IV.4 Application au contrôle de la température d'un SOC

Comme exemple d'application de notre système flou, nous avons choisi une application très spécifique au SoC, où la reconfiguration a un apport très important dans le fonctionnement et la fiabilité du système.

Le développement des systèmes sur puce nécessite l'intégration de composants conçus par des technologies différentes, et supportant des caractéristiques spécifiques, qui sont généralement contraignantes par rapport aux autres composants du SoC, de ce fait il existe une distribution non uniforme de la puissance et par conséquent de la température sur la puce, surtout quand des vitesses élevées de fonctionnement et de commutations de l'activité dans certains composants sont exigées **[51]**. Des solutions existent pour mettre en œuvre de nouvelles techniques de partitionnement et de placement qui prennent en charge la consommation spécifique de chaque IP utilisé dans la conception d'un SoCs **[52] [53]**.

Dans cette dernière partie de ce mémoire, nous proposons une gestion dynamique de la température via une reconfiguration locale des IPs se trouvant sur la même puce, par une réorientation de l'activité de chaque IP de manière à gérer la consommation du système et donc trouver des solutions à la dissipation permise.

IV.4.1 Architecture système

L'architecture global du système (Figure IV.69) est composée de gestionnaire de température à base du système flou déjà décrit dans les sections précédentes, est le bus SMBus qui est typique dans les applications de management de la température et de la puissance.

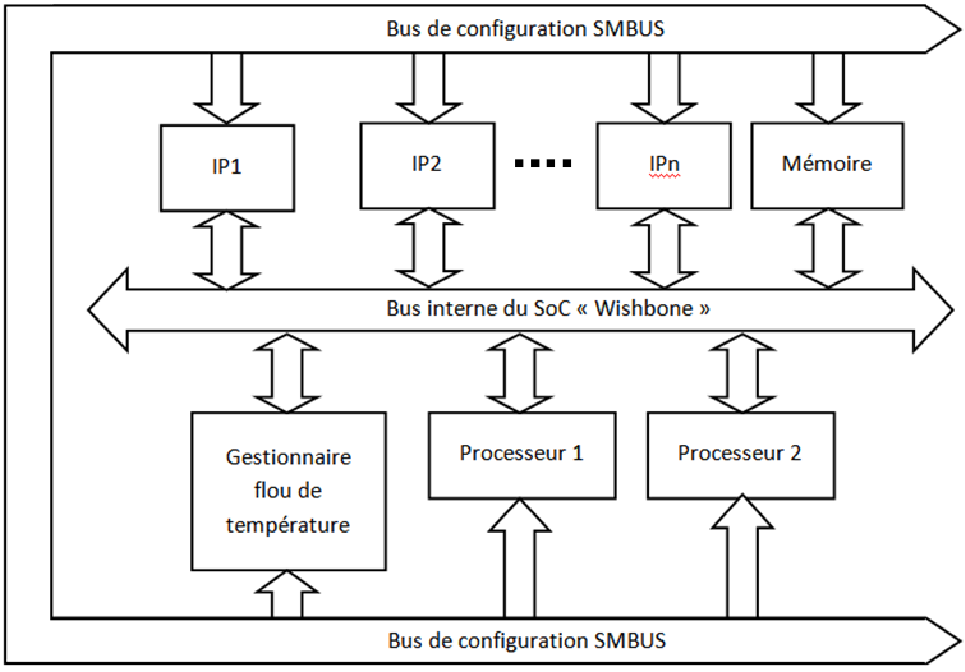


Figure IV.69 : Architecture proposée pour la gestion de la température d'un SOC.

Dans chaque IP une interface SMBus y est intégrée pour la communication avec le système flou, ceci pour ne pas interférer avec le fonctionnement normal du système, surtout que la gestion de la température se fait en mode interruption.

Chaque IP est doté d'un mécanisme spécifique de reconfiguration (Figure IV.70), où il réagit en adoptant le mode de fonctionnement proposé par le gestionnaire de température conformément à son activité qui est caractérisée par sa température locale.

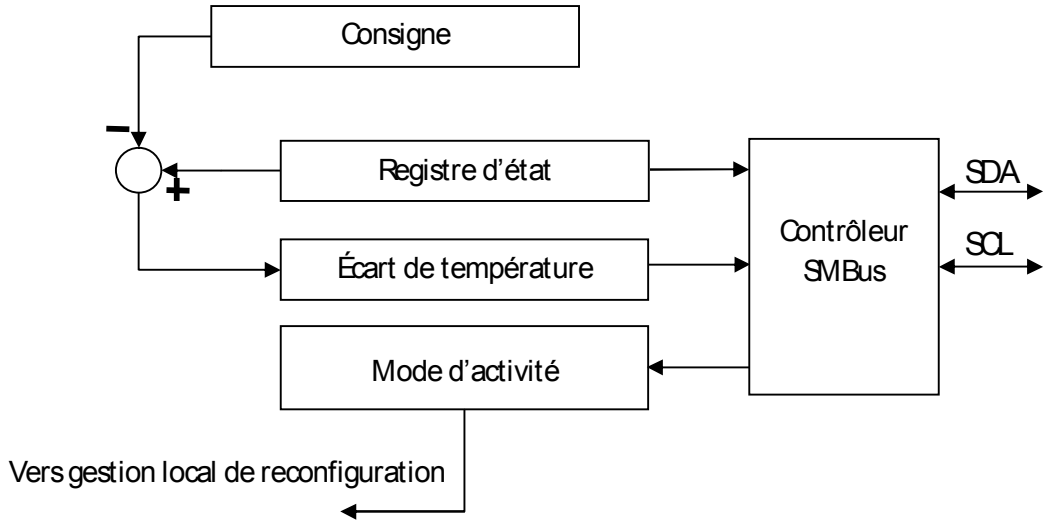


Figure IV.70 : Interface pour la gestion de la température.

Dans la suite, nous supposerons que les données sont générées par un certain nombre d'IPs reliés par le SMBus et nous essayerons de les manager sans se préoccuper de la manière dont l'IP va utiliser le résultat du traitement flou.

IV.4.2 Contrôleur flou

Le contrôleur flou est l'organe principal qui va gérer la température du SoC, il permettra de déterminer le mode d'activité le plus approprié pour une certaine plage de température. Pour cette raison nous utiliserons notre système flou, qui intègre déjà les interfaces nécessaires pour le contrôle de la température, notamment le contrôleur SMBus. Un réglage des paramètres du système est essentiel pour le rendre opérationnel.

IV.4.2.1 Paramètres de contrôle flou

Nous allons définir les besoins du contrôleur en matière de fonctions d'appartenance des entrées X1 et X2, de la base des règles floues et des fonctions d'appartenance de la sortie Y. Ces paramètres sont généralement définis après expertise du système et après plusieurs simulations.

g) Fonction d'appartenance des entrée X1 et X2

Notre système de contrôle est représenté sur la Figure IV.71:

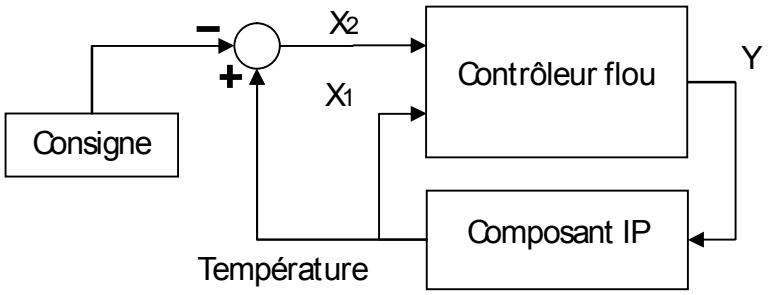


Figure IV.71 : Configuration du système flou.

De ce fait, nous avons besoin de deux entrées, l'entrée X1 représente la température instantanée du composant IP et l'entrée X2 définit l'écart de température par rapport à une consigne spécifique au composant IP.

Pour un premier modèle nous avons choisi d'utiliser le même univers du discours pour les deux entrées, où le système possède 7 fonctions d'appartenance comme illustré dans Tableau IV.19.

Tableau IV.19 : Fonctions d'appartenance.

Différence de température	
Conclusion	**Abréviation**
Positive large	PL
Positive medium	PM
Positive small	PS
Zero	Z
Negative small	NS
Negative medium	Nm
negative large	NL

Température	
Conclusion	**Abréviation**
Zero	Z
Zero positive	ZP
Medium small	MS
Medium large	ML
Great small	GS
Great large	GL
Huge	H

L'univers du discours des entrées est partitionné comme montré sur la Figure IV.72 :

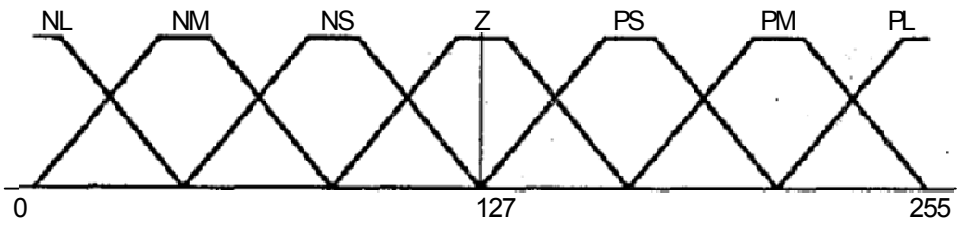


Figure IV.72 : Fonction d'appartenance et univers du discours

h) Règles floues

Le choix des règles est une phase délicate de la caractérisation du contrôleur flou, elle doit permettre de maintenir la température de chaque IP de manière individuelle dans un certain niveau pour que les performances du système ne soient pas affectées.

7 règles sont utilisées dans notre exemple (le nombre de règles peut être élargi vers 49 règles), la matrice des règles donnée dans le Tableau IV.20 est une première évaluation du système de gestion de la température du SoC, elle n'est en aucun cas définitive.

Tableau IV.20 : Règles floues.

Température

Différence de température	Y	X1						
		Z	ZP	MS	ML	GS	GL	H
X2	NL	R1	R1	R2	R3	R4	R5	R6
	NM	R1	R1	R2	R3	R4	R5	R7
	NS	R1	R1	R3	R3	R4	R5	R7
	Z	R1	R2	R3	R3	R5	R6	R7
	PS	R1	R2	R3	R3	R5	R6	R7
	PM	R1	R2	R3	R4	R5	R7	R7
	PL	R1	R2	R4	R4	R5	R7	R7

La base des règles contient une représentation symbolique de la conclusion du traitement flou, elle serait convertie ensuite en valeurs codées sur 8 bits qui représentent les fonctions d'appartenance 'singleton' de la sortie.

Ces règles converties représentent le mode de reconfiguration recommandé pour les composants IP, où le système doit être maintenu stable sans être en proie à des surchauffes fatales. Ces règles sont décrites comme étant des règles de reconfiguration qui seront interprétées par chaque IP suivant ses propres modes de reconfiguration interne.

Le tableau suivant montre quelques modes de reconfiguration possibles.

Tableau IV.21 : Modes de reconfiguration.

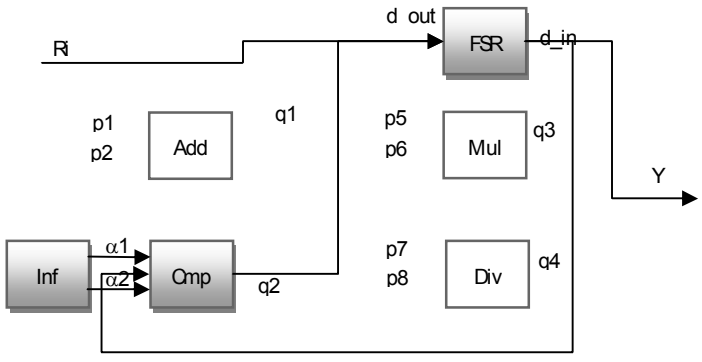
Conclusion	Abréviation
Vitesse très rapide	R1
Vitesse normale	R2
Vitesse réduite	R3
Vitesse réduite & puissance réduite	R4
VR& PR& déclanchement du refroidisseur	R5
Isolation de l'IP	R6
Arrêt du système	R7

IV.4.2.1 Paramètres de reconfiguration

Pour que le contrôleur soit capable de fournir des données exploitables dans les composants IP, ces données doivent être limitées en nombre, donc nous avons choisi une méthode de défuzzification et d'inférence qui assure la préservation des modes de reconfiguration (contrairement à la méthode centre de gravité qui balaye tout l'univers du discours de la sortie), et qui assure un traitement de données relativement rapide, pour que le SoC soit surveillé de manière efficace.

La méthode choisie est celle du max-min qui ne nécessite pas beaucoup de ressources ni de multiples reconfigurations, où le calcul du minimum est effectué pour trouver le degré de vérité des 4 règles activées. En ce qui concernant la défuzzification, un simple calcul du maximum des degrés de vérité correspondant à chaque règle est réalisé pour trouver la règle la plus adéquate, et par conséquent de trouver le mode de reconfiguration qui maintiendra le système stable.

La figure suivante représente le contexte de reconfiguration nécessaire pour l'exécution de la méthode max-min.



Contexte.
Ligne : b'10101110
Colonne : b'0010110101

Figure IV.73 : Reconfiguration pour la méthode Max-min.

Cette méthode utilise les ressources de contrôle plus que les ressources de calcul, elle dépend fortement de la gestion du flux de données entre les registres systèmes (FSR) et le comparateur.

IV.4.3 Simulation

Après paramétrage du contrôleur flou, nous pouvons maintenant vérifier son fonctionnement ; pour y parvenir nous devons créer un programme VHDL qui va simuler les réactions d'un IP aux modes de reconfigurations.

Ce modèle va réagir aux différents modes de reconfiguration, en changeant la température instantanée proportionnellement aux règles de configurations injectées.

La Figure IV.74 présente la surface de contrôle de l'IP, où chaque mode de reconfiguration est activé par un intervalle flou de X1 et X2

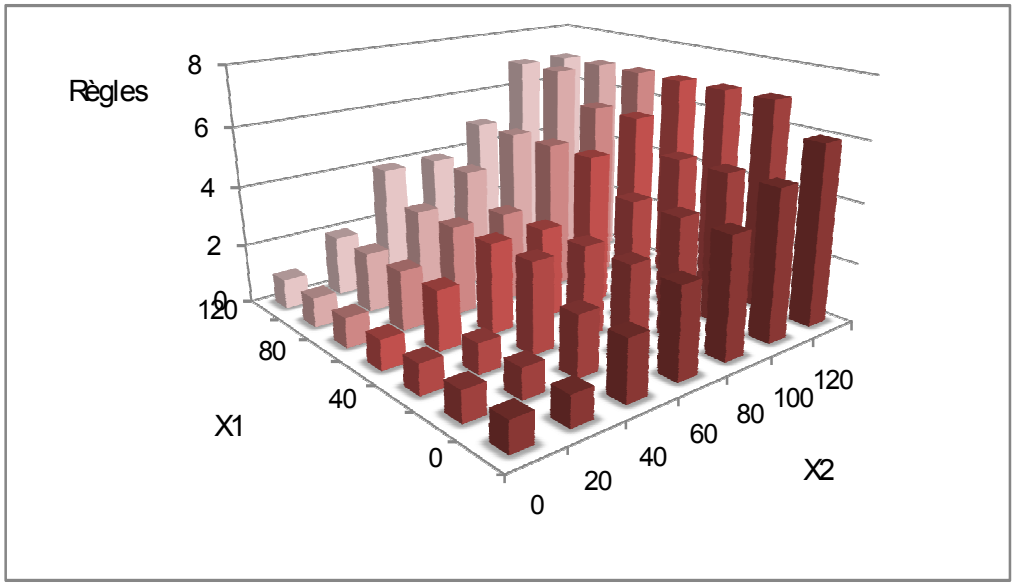


Figure IV.74 : Surface des règles de reconfiguration.

Des simulations comportementales aux niveaux du système sont réalisées dans le but de vérifier que le contrôleur développé procède vraiment à l'asservissement des modes de

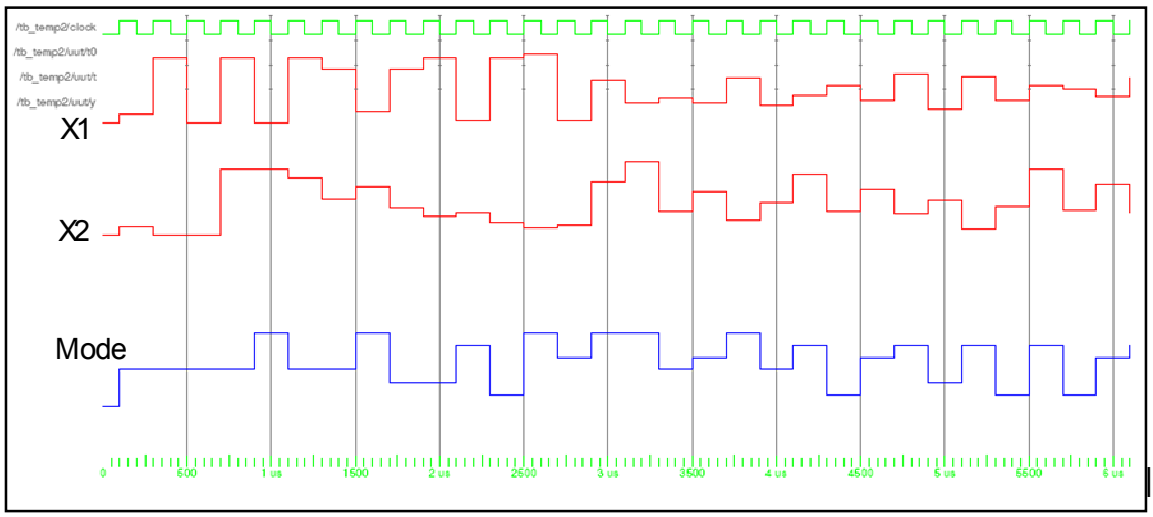


Figure IV.75 : Simulation de gestion de température par reconfiguration.

reconfiguration de l'IP et que les mécanismes de reconfiguration sont aussi activés (Figure IV.75).

Pour ce faire, nous avons relié l'IP au contrôleur flou et on observe la commande correspondante produite à chaque changement de température. Les courbes de la Figure IV.75 représentent le comportement de l'IP par rapport à la gestion du contrôleur à travers ses modes de reconfiguration.

IV.4.4 2ème approche de gestion de la température

Cette 2ème approche est destinée à la gestion de plusieurs IPs intégrés dans la même puce, elle utilise le même principe que celui de l'approche précédente, sauf qu'il faut ajouter au contrôleur flou une logique de calcul pour son adaptation aux nouvelles exigences de cette méthode.

Cette approche consiste à trouver un niveau d'alerte global pour la reconfiguration du SoC, alors nous allons utiliser les mêmes paramètres du contrôleur flou ; les fonctions d'appartenance, les règles floues, le calcul des inférences et la défuzzification.

Après nous adapterons l'environnement du contrôleur flou pour la gestion des 4 IPs selon la configuration suivante.

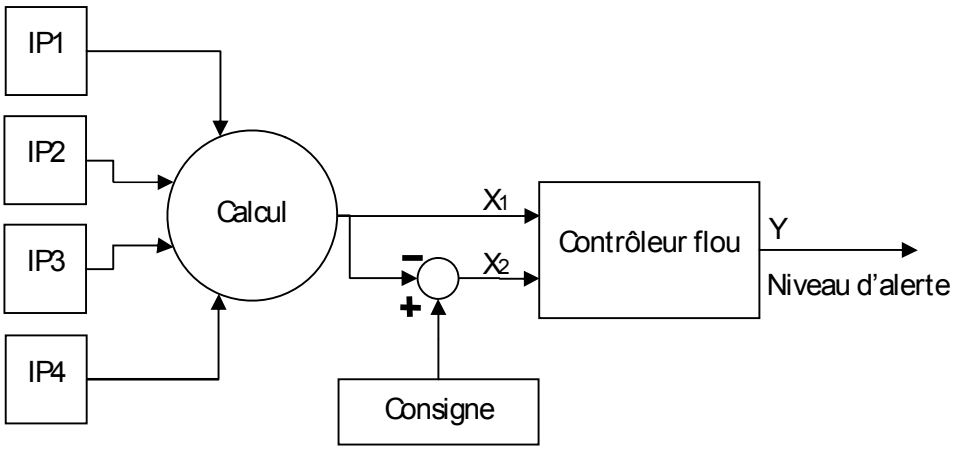


Figure IV.76 : 2ème approche de gestion de la température.

Figure IV.76 représente la structure initiale pour la réalisation de plusieurs méthodes possibles, ces méthodes sont décrites dans les prochains paragraphes.

a) Méthode par calcul de la valeur moyenne

Le calcul da la variable X1 se fait par la valeur moyenne de la température des IPs, et la

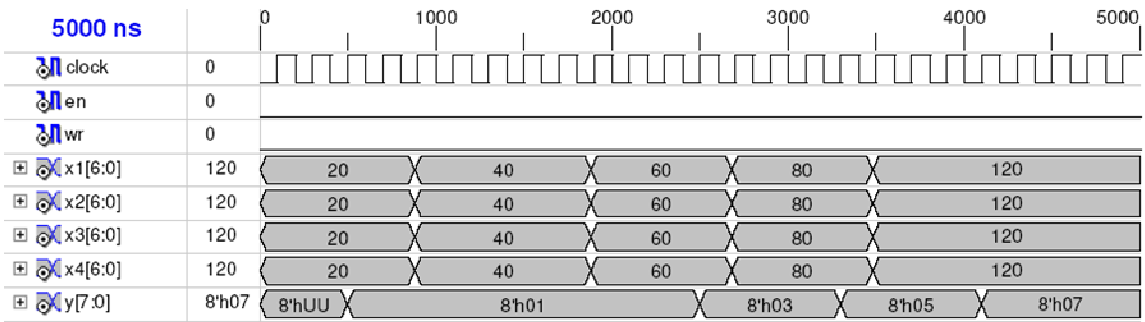


Figure IV.77 : Simulation pour consigne = 40°C

variable X2 représente la différence de la température moyenne par rapport à une consigne, cette consigne décrit le fonctionnement optimal du système. Les variations de la température sur la puce sont définies de 0°C à 127°C, et la valeur de consigne est définie entre 0 et 127°C.

La Figure IV.77 et la Figure IV.78 présentes deux simulations du système de gestion, pour des températures variant de 0°C à 120°C l'une pour une consigne de 40°C et l'autre pour 60°C

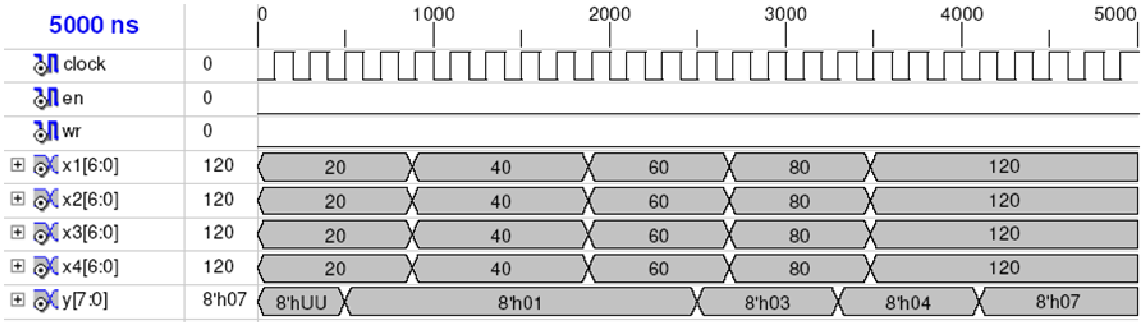


Figure IV.78 : Simulation pour consigne = 60°C

Au même moment où la température s'est écartée de sa consigne, la règle de reconfiguration n° 5 s'est activée pour la 1ère consigne et la règle n° 4 s'est activée pour la deuxième consigne, ce qui montre le bon fonctionnement de système ; dans la suite nous fixerons la consigne à 60°C.

La Figure IV.79 montre un deuxième exemple où les températures des différents IPs sont très éloignées, ainsi nous remarquons que la valeur moyenne a masqué l'augmentation de la température de l'IP n° 1, de ce fait aucun changement de règle n'a été enregistrée, et le système continu à travailler dans des conditions critiques.

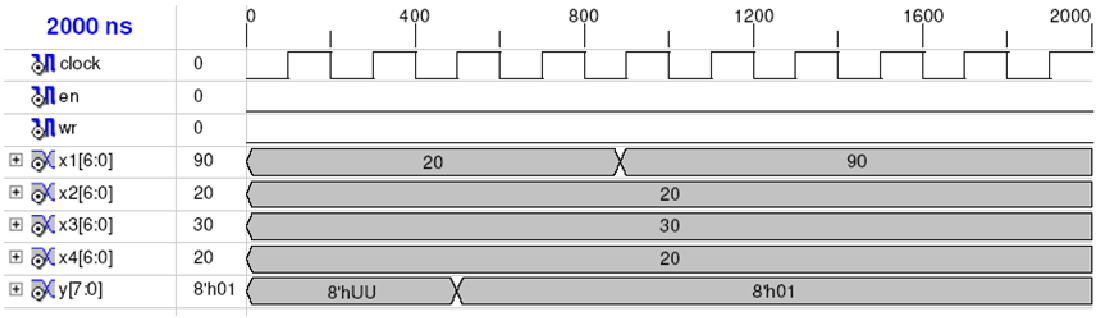


Figure IV.79 : Résultat de simulation pour une température critique.

Pour régler ce problème, une autre méthode est proposée mais reste impossible à décrire en VHDL et à simuler sur un simulateur numérique, cette solution utilise des fonctions mathématiques pour borner la température des IPs et renforcer le facteur risque de chaque IP pour les températures élevées.

Un autre point doit être étudié lors de la gestion de la température, qui est l'emplacement de l'IP et la surface qui doit occuper sur le SoC, quand un IP est tout le temps

activé, alors il va générer une température qui influera à son tour sur les autres IPs voisins sur la puce. Pour permettre à cette spécificité d'être prise en charge par le contrôleur flou, nous allons pondérons les IPs à plus grand risque par des coefficients qui vont augmenter ou diminuer l'effet d'un certain nombre de composants pour améliorer la qualité de la gestion de la température.

Un autre programme VHDL est écrit pour prendre en considération les coefficients de pondération qui varient entre 0.0 et 1.0, la valeur '1.0' définit les IPs à grand risque.

Dans la simulation de la Figure IV.80, nous avons pris le même exemple que celui de la Figure IV.79, mais en imposant les coefficients spécifiques suivants :

IP1 := 0.8 IP2 := 0.3 IP3 := 2.0 IP4 := 0.4

Après l'ajout des coefficients il y a une nette amélioration de la détection de la température, où la règle de reconfiguration n° 4 s'est activée.

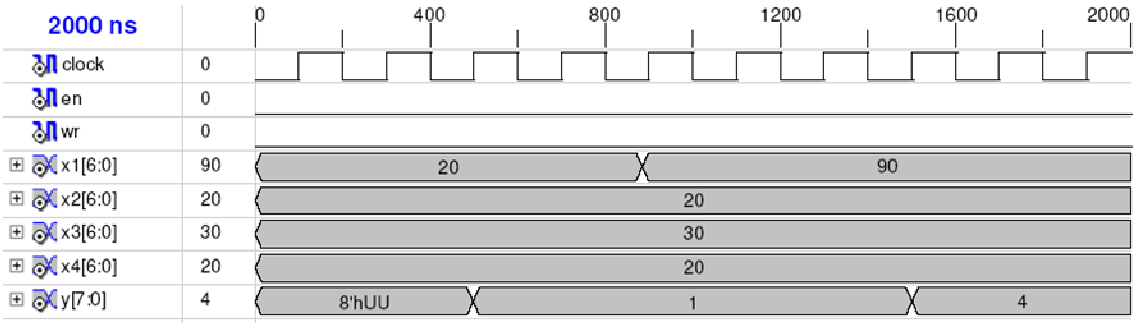


Figure IV.80 : Gestion pondérée de la température.

Pour conclure cette série de simulations, nous allons augmenter le nombre d'IPs dans le système jusqu'à 16 IPs (Figure IV.81), nous remarquons que plus le nombre de sources augmente, plus le contrôleur flou ne peut pas détecter les points chauds sur le SoC, ces valeurs sont dévaluées par le calcul de la valeur moyenne, pour cette raison l'amélioration

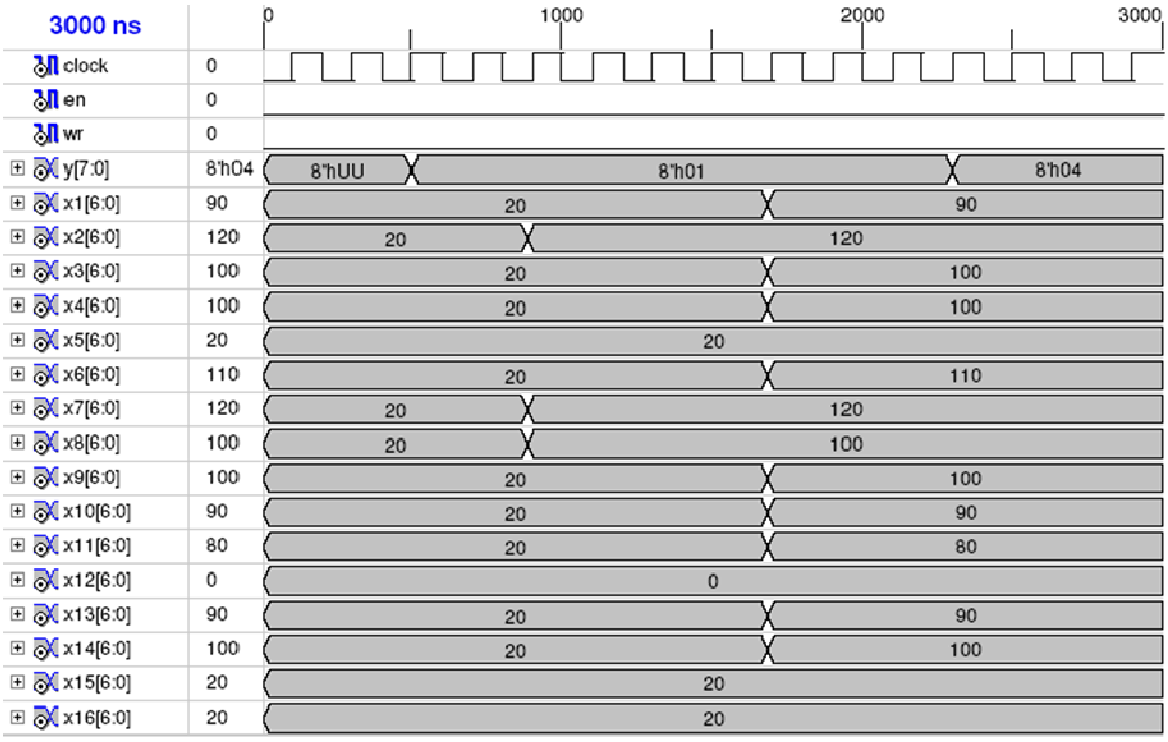


Figure IV.81 : Gestion de température pour 16 IPs.

ou la modification de la méthode de calcul pour la fusion des sources de température est nécessaire.

b) Méthode par transcodage de la sortie

Une extension de la méthode décrite dans la section précédente rend le système de gestion de la température plus efficace, en ce qui concerne la sensibilité de ce dernier à l'emplacement des IPs et à l'effet de leur proximité les uns des autres.

Nous utiliserons la même structure que celle de la Figure IV.76, mais en ajoutant un bloc de transcodage à la sortie de du contrôleur flou qui sera orienté par le bloc de calcul placé avant le contrôleur (Figure IV.82).

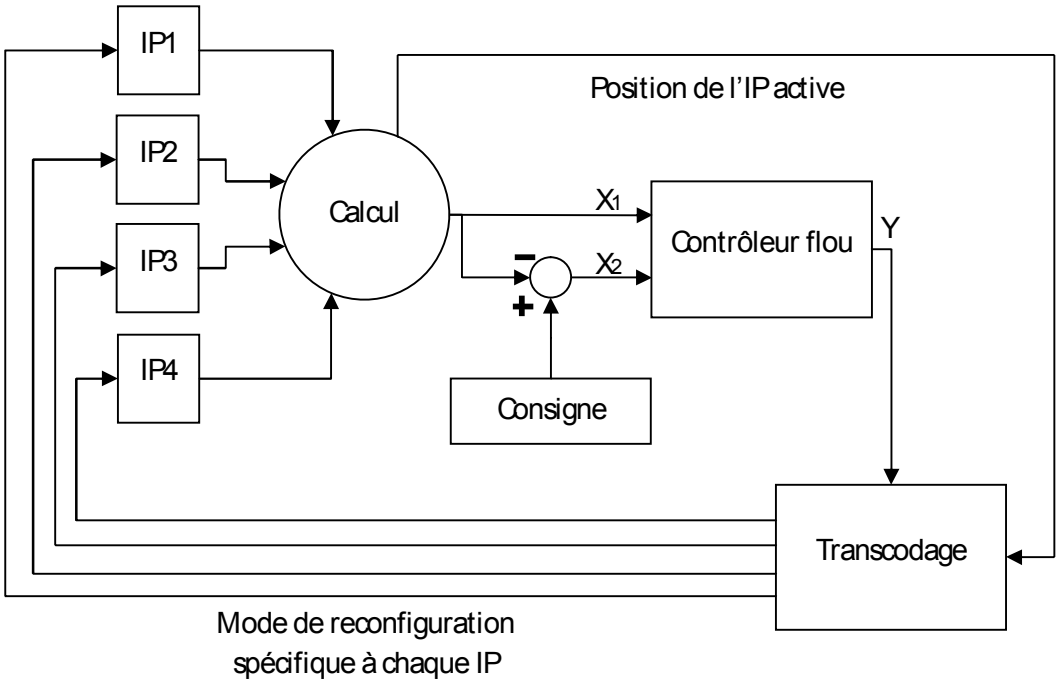


Figure IV.82 : Gestion de la température par transcodage.

Le transcodage se fait par une table qui contient toutes les valeurs possibles des modes de reconfiguration de chaque IP, cette table est adressée par la combinaison de la sortie 'Y' et de la position du IP qui est caractérisée par la température la plus élevée.

Un exemple de transcodage est présenté dans le Tableau VI.22:

Tableau VI.22 : Quelques valeurs de la table de transcodage.

Mode d'alerte 'Y'	IP active	Code de sortie			
		IP1	IP2	IP3	IP4
R5	IP1	R5	R4	R3	R1
R2	IP2	R1	R2	R1	R1
R4	IP3	R3	R3	R4	R2
R3	IP1	R3	R2	R2	R1

R5	IP4	R2	R3	R4	**R5**
R6	IP1	**R6**	R5	R4	R2

Ce tableau décrit quelques exemples de transcodage du mode de reconfiguration du SoC vers les modes de reconfiguration propres aux IPs, ce transcodage dépend de plusieurs paramètres qui nécessitent une connaissance parfaite du SoC et des emplacements spécifiques des IPs.

Une première simulation est donnée sur la Figure IV.83 :

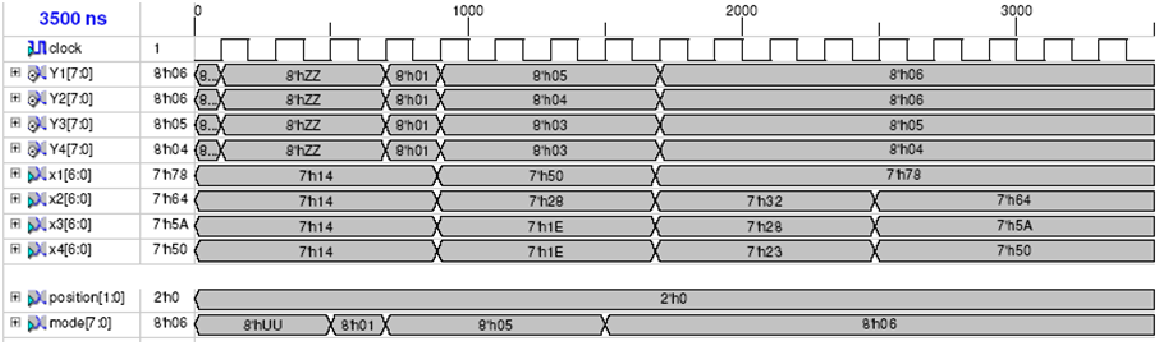


Figure IV.83 : Résultat de simulation de la gestion de température par transcodage.

Nous constatons dans cette dernière simulation, que non seulement le mode de reconfiguration de l'IP le plus active est changé, mais aussi les autres IPs qui lui sont adjacents sur le SoC, aussi un mode de reconfiguration globale est fixé.

Dans la simulation (Figure IV.84) nous avons essayé d'asservir l'IP par des modes de reconfigurations de sorte à montrer la convergence de la température de l'IP vers des états stables. Pour arriver à ce résultat, il faut avoir un modèle d'IP qui va réagir aux différents stimuli, alors nous avons utilisé le même modèle que celui de la section "IV.4.3 Simulation".

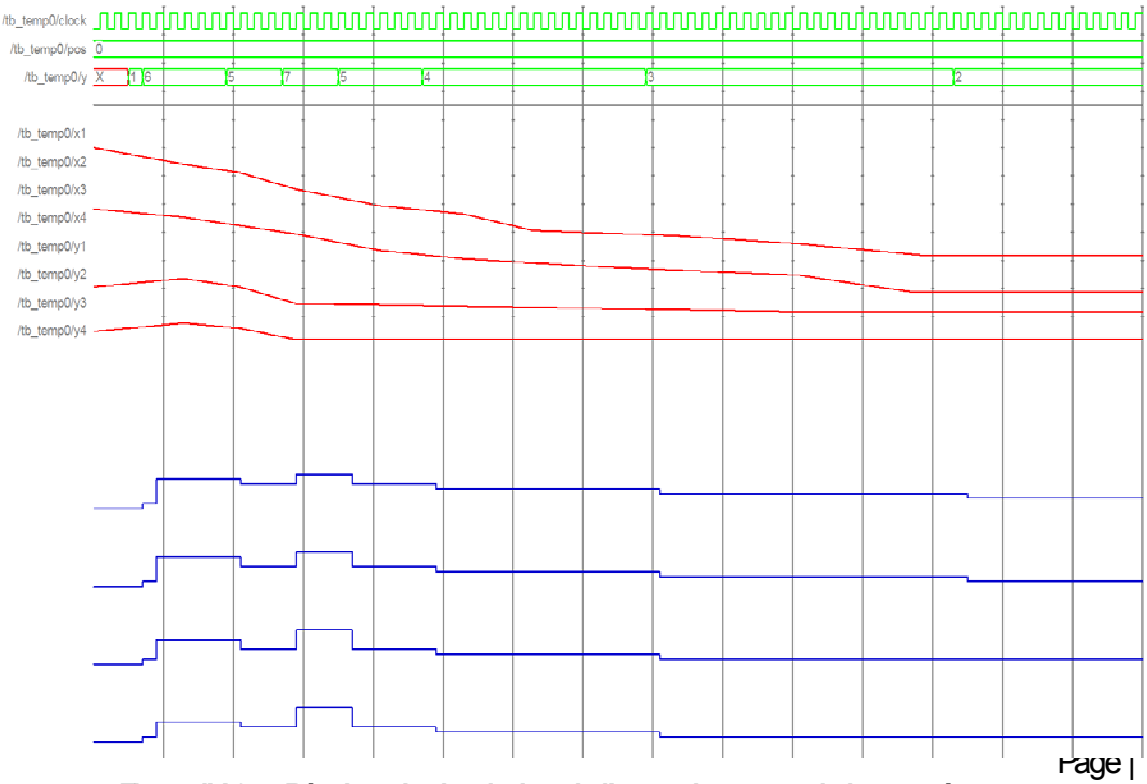


Figure IV.84 : Résultat de simulation de l'asservissement de la température.

La Figure IV.84 montre une simulation de l'asservissement de 4 IPs voisins sur un SoC, où leurs paramètres de transcodage respectifs sont identifiés pour prendre en charge les dépendances existantes entre leurs modes de reconfiguration correspondants.

Nous remarquons que la température de chaque IP (représentée en rouge) diminue suivant le mode de reconfiguration imposé à chaque IP (représenté en bleu), contrairement aux méthodes précédentes qui impose un mode de reconfiguration global sans prendre en charge les changements locaux de la température, ce qui montre l'efficacité de cette nouvelle méthode. Néanmoins, la prise en charge d'un nombre important d'IPs reste à développer.

c) Méthode par transcodage des entrées et de la sortie

En utilisant le même principe que celui de la méthode précédente, nous allons reconsidérer quelques éléments de la structure présentée sur la Figure IV.82. Dans certains cas l'intégrateur du SoC ne peut placer des capteurs de température dans chaque IP, alors cette tâche sera traitée par le fournisseur de composants virtuels, en implantant dans chaque IP un mécanisme de surveillance de l'activité.

Ce mécanisme va fournir au contrôleur flou un mot de 2 bits qui définit le mode d'activité de chaque IP, mais pour que le contrôleur flou soit capable de gérer ce format de donnée un changement de l'élément de calcul est nécessaire, d'où la configuration suivante :

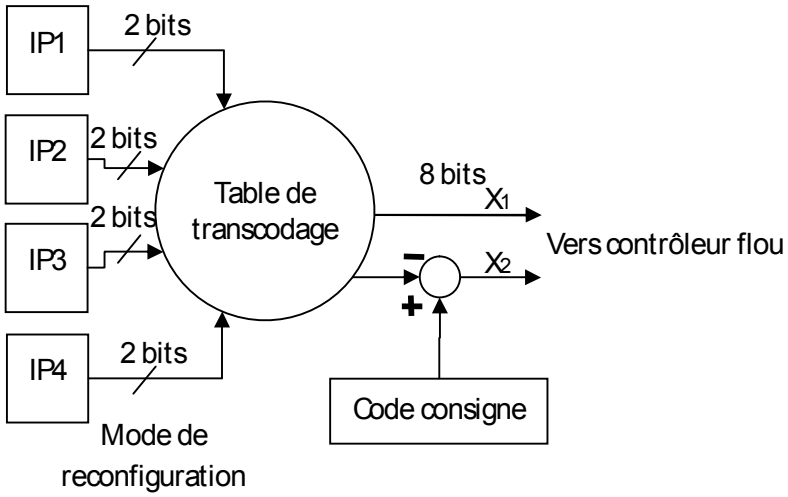


Figure IV.85 : Gestion de l'activité de L'IP par transcodage.

Les codes qui proviennent des IPs représentent leur activité, ces codes sont combinés ensembles pour adresser une table de transcodage (de 255 * 8bits) qui permet de générer des mots codées sur 8 bits qui peuvent être traitées par le contrôleur flou, ensuite le contrôleur flou va fournir un autre code qui va permettre à son tour de donner les modes de reconfiguration de chaque IP comme il a été réalisé dans la méthode précédente.

Les résultats de simulation de cette méthode sont représentés sur la Figure IV.86 :

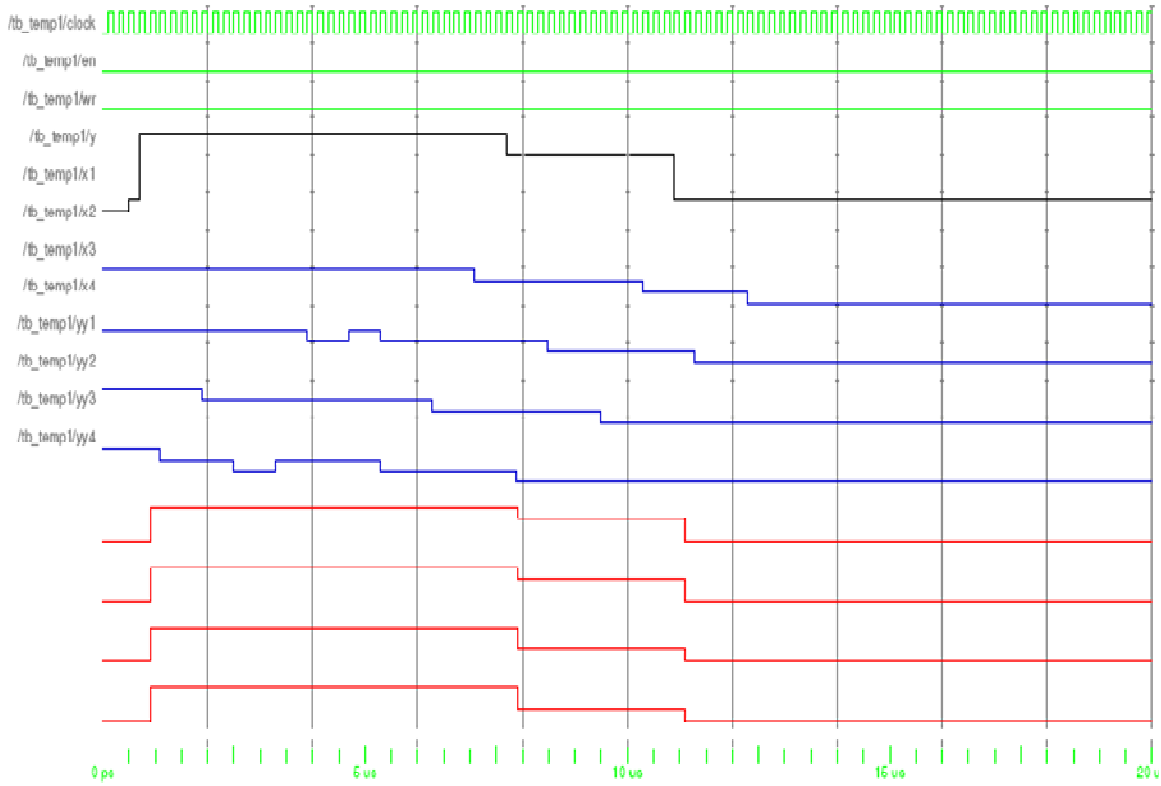
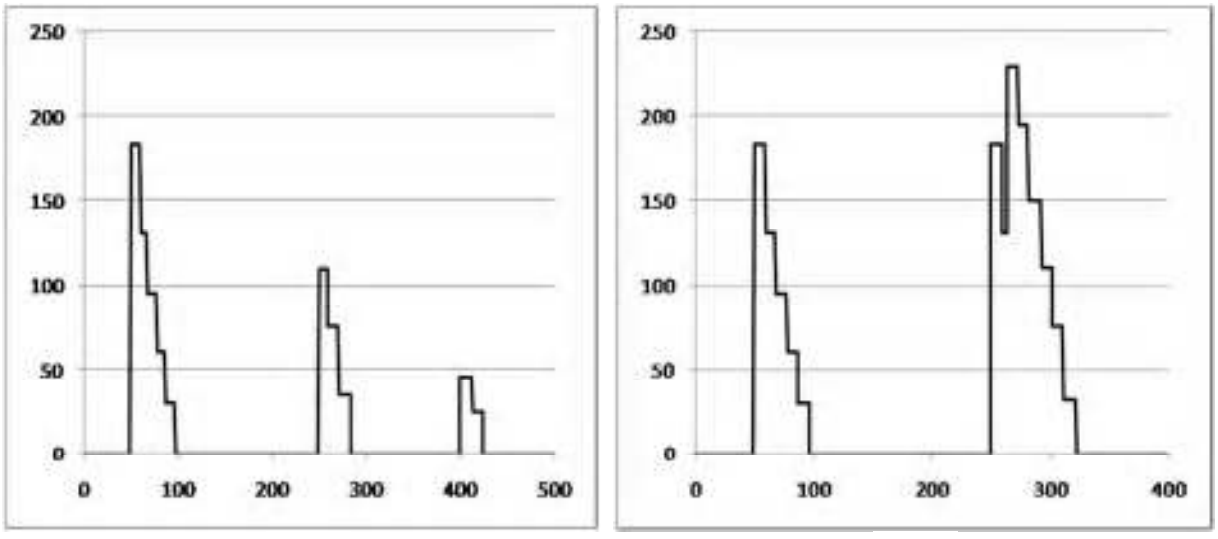


Figure IV.86 : Simulation de la méthode de gestion d'activité par transcodage

Pour faire fonctionner le système, nous devrions forcer des modes de reconfiguration initiaux pour chaque IP (représentés en bleu), après quelques cycles d'horloge nous remarquons que les modes de reconfiguration se stabilisent à '0', ainsi le système est remis dans un état plus sûr. La figure **IV.57** présente le fonctionnement du contrôleur pour quelques exemples spécifiques.



(a) (b)

Figure IV.87 : Exemple de commande des modes de reconfiguration des IPs-core.

La figure **IV.57.a** représente le fonctionnement du contrôleur pendant 50 µs, où nous avons forcé les modes de reconfiguration des 4 IPs (x1, x2, x3, x4) par les valeurs 10, 11, 10, 00 (soit la valeur décimale 184 après transcodage) à l'instant 5 µs et par les valeurs 01, 10, 11, 01 (soit la valeur décimale 109 après transcodage) à l'instant 25 µs et par les valeurs 00, 10, 11, 10 (soit la valeur décimale 46 après transcodage) à l'instant 40 µs, ainsi, nous avons constaté que le contrôleur a pu gérer les modes de reconfiguration des IPs de telle façon à les rendre moins dangereux pour le fonctionnement global du système. Le changement d'état est effectué en 10 cycles, ce qui représente un temps de réponse de 2 µs pour une fréquence de contrôle de 5 Mhz.

Le deuxième exemple représente l'apparition d'un nouveau mode qui se manifeste par une modification des modes de reconfiguration en cours de gestion, la première partie du graphe de la figure **IV.57.b** montre un exemple de fonctionnement normal du contrôleur, par contre dans la deuxième partie et pendant que le contrôleur régule les modes de reconfiguration, un changement brutal de ces modes est forcé pour vérifier si le contrôleur peut gérer ou non de nouveaux états imprévus du SoC, nous remarquons que le contrôleur à poursuivi la gestion des modes de reconfiguration, en prenant en charge les nouvelles valeurs imposées.

Comme pour la dernière méthode cette méthode est très prometteuse, mais les problèmes de paramétrage du contrôleur flou et du transcodage rendent la mise en œuvre de cette technique très compliquée.

Conclusion

Dans ce dernier chapitre de ce mémoire, nous avons présenté la conception d'un système flou reconfigurable sous forme d'un composant software (IP-Soft) intégrable sur SoC. En suivant scrupuleusement la méthodologie de conception des systèmes reconfigurables pour mettre au point notre système.

Dans un premier temps, nous avons fixé un certain nombre de contraintes définies dans le cahier des charges pour montrer la méthodologie de conception et les caractérisations des différents blocs constituants le système. Un bon partitionnement hardware et software des différentes tâches réalisables par le système est nécessaire. Par la suite, nous avons regroupé tous les blocs hardware et hardware reconfigurables au sein de la même description VHDL, mais ceci n'a pas pu se faire qu'après le développement d'un contrôleur qui se charge d'effectuer les différentes actions nécessaires à la gestion des flux de données entre tous les éléments du système et de prendre en charge les différents plans de reconfiguration indispensables au fonctionnement du système flou.

Le système que nous avons réalisé ne peut être intégré dans un SoC dans son état actuel, c'est pourquoi nous avons ajouté des ressources de communication et une unité de contrôle qui sont nécessaires à l'évolution du système vers un IP-Soft.

Enfin plusieurs approches de gestion intégrée de la température sur SoC sont réalisées, permettant ainsi d'assurer un fonctionnement optimal de l'ensemble des IPs cores réunis sur la même puce. Elles permettent de réadapter l'activité des IPs par des règles de reconfiguration recommandées par le contrôleur flou au niveau du système global et au niveau de chaque IP.

Les résultats et les simulations des différentes étapes de développement sont présentés, ainsi qu'une évaluation de notre approche de conception et nos approches de gestion de température.

CONCLUSION GENERALE

ET PRESPECTIVES

Conclusion générale et perspectives

Comme les SoCs regroupent un ensemble de composants non-homogènes et parfois résistants à la complète coopération sans interfaces, les avancées dans le domaine du 'IP & Reuse' et la nouvelle vision de reconfiguration ont réussi à développer et mettre au point une multitude de produits qui répondent aux spécificités du '*time to market*', mais qui se sont accompagnés d'un éventail de problèmes spécifiques et étalés sur toute la chaine de conception et réalisation.

Cette étude se basant sur les apports des techniques de reconfiguration et exploitant VHDL comme outil de programmation à l'échelle comportementale et structurelle pour modéliser un système flou, a montré qu'une nouvelle vision et stratégie est introduite pour la gestion des ressources par l'inclusion de la reconfiguration au niveau software et hardware et prenant en charge très tôt dans le concept de développement les partitionnements et les modules à restructurer. Une solution au problème délicat posé par la gestion de la dissipation de la température par l'ensemble des IPs dans un SoC est présentée. Elle peut être qualifiée de solution simple mais porteuse, car elle agit uniquement sur la fonctionnalité et évite tout recours à la restructuration des IPs, et son impact est immédiat indépendamment du nombre d'IPs et de leurs activités spécifiques.

Les étapes réussies dans ce travail peuvent être consignées et comptabilisées comme suit :

- Conception d'un système flou reconfigurable au niveau des unités de calcul et au niveau du chemin des données. On note aussi le caractère générique utilisé dans les différents sous modules du système pour assurer souplesse de programmation, portabilité du produit et permettre des extensions à volonté.
- La mise au point de la reconfiguration est réussie grâce au bon partitionnement en blocs fonctionnels. On tire une conclusion importante en qualifiant que le partitionnement est la clé de la reconfiguration mais l'optimisation globale des ressources se complique.
- En exploitant extensivement les moyens de communication dans les SoCs, on préserve le caractère IP des composants des SoCs et on assure la reconfiguration au niveau système.
- Comme l'optimisation des ressources est définie à l'échelle comportementale alors il est très difficile de prouver l'impact individuel dû à la reconfiguration.

Cet ensemble de règles et recommandations a permis l'implantation d'un système flou comprenant une structure floue reconfigurable (éléments d'interconnexion, éléments de

calcul…), un bus système 'Wishbone', une interface de communication série 'SMBus' et un gestionnaire des ressources et d'activation des tâches exécutées par le système.

Le système développé est implanté sur une famille de FPGA de type 'Virtex II P'. Pour la «virtex » la surface utilisée est de 25% et la fréquence maximale de travail est de 37.0 Mhz. Les autres caractéristiques sont :

- 2 entrées et une sortie normalisées et discrétisées.
- 7 fonctions d'appartenance avec un double chevauchement, 4 règles actives parmi 49 règles au maximum.
- La largeur du bus de données est de 8 bits.
- Présentation de notre système comme un IP-Soft pour intégration facile dans une SoC.

Pour mieux apprécier l'apport de cette architecture, nous avons reconfiguré le système flou sous forme d'un contrôleur consacré à la gestion de la température d'un SoC, où nous avons pus constater les difficultés de contrôle et de management d'une tel quantité physique dans un environnement fortement hétérogène.

Il est indispensable de rappeler les difficultés d'intégration des architectures reconfigurables dans les systèmes sur puce, en particulier celles dédiées à la gestion des performances et à l'exploration des défauts fonctionnels dans les SoCs.

L'optimisation des opérations de reconfiguration est complexe car la reconfiguration est fortement liée à l'application alors il serait intéressant d'étudier la décentralisation des mécanismes de reconfiguration d'une part et de faire évoluer le principe du multitâche doté de reconfiguration d'autre part. Et comme la reconfiguration est généralement accompagnée de circuiteries additionnelles, il est fortement recommandé de déterminer les dégradations locales en termes de vitesse et coût et les comparer à la souplesse et flexibilité tirées au niveau fonctionnement global.

La reconfiguration étudiée dans le domaine du contrôle flou au niveau traitement moyennant des restructurations logiques devrait être menée au mode flou pour éviter le fardeau de l'optimisation des ressources. Ainsi, une 'reconfiguration floue' dans l'environnement flou est recherchée.

La recherche de moyens au niveau comportemental d'auto-reconfiguration des IPs pourrait alléger le problème de dissipation de la chaleur et augmenter ainsi la fiabilité et la sécurité des systèmes sur puce.

Comme la reconfiguration fonctionnelle est accompagnée de circuits dont la structure est modifiée en permanence, un projet de recherche est en cours pour modéliser les nouveaux concepts de test et redéfinir les actions. Il serait intéressant d'y introduire la duplication algorithmique aux principales tâches de reconfiguration.

BIBLIOGRAPHIE

Bibliographie

1. **Nekoogar, Farzad et Nekoogar, Farak.** *From ASICs to SOCs: A Practical Approach.* New Jersey : Prentice Hall PTR, 2003. 0-13-033857-5.

2. **Hunsinger, Frédéric.** *Méthode de validation globale pour les systèmes monopuces.* INPG. 2006. Thèse de doctorat.

3. **COUSSY, PHILIPPE.** *Synthèse d'Interface de Communication pour les Composants Virtuels.* LESTER. 2003. Thèse de doctorat.

4. **Pouget, Julien.** *Thèse des systèmes sur puce: ordonnancement et exploration de l'éspace des solution architecturales.* UNIVERSITE MONTPELLIER II. 2002. Thèse de doctorat.

5. **ARM Limited.** *AMBA Specification (Rev 2.0).* s.l. : ARM Limited, 1999.

6. **IBM.** *The CoreConnect Bus Architecture.* 1999.

7. **OpenCores Organization.** *WISHBONE SoC Architecture Specification, Revision B.3.* 2002.

8. www.opencore.org. [En ligne] OpenCore Organisation, 2006.

9. **IEEE.** *Standard Test Access Port and Boundary-Scan Architecture.* 1990. Std 1149.1-1990.

10. **BENABDENBI, Mounir.** *CONCEPTION EN VUE DU TEST DE SYSTÈMES INTÉGRÉS SUR SILICIUM (SOC).* L'UNIVERSITÉ PARIS VI. 2002. Thèse de doctorat.

11. **Courtois, Bernard et Rencz, Marta.** http://tab.computer.org/tttc/index.html. *http://tab.computer.org/.* [En ligne] Test Technology Technical Council, 30 May 2004.

12. *IC Reliability and Test : What Will Deep Submicron Bring.* **IEEE.** 1999. IEEE DESIGN & TEST OF COMPUTERS.

13. **Székely, V, Rencz, M et Courtois, B.** Thermal testing methods to increase system reliability. *13th IEEE SEMI-THERM Symposium.* 1997, pp. 210-217.

14. *Design for thermal testability (DfTT) and a CMOS realization.* **v, Székely, et al.** 1996, Sensors and Actuators, Vol. A55, pp. 29-33.

15. *Thermal Monitoring and Testing of Electronic Systems.* **Székely, Vladimir, et al.** 2, s.l. : IEEE, 1999, IEEE TRANSACTIONS ON COMPONENTS AND PACKAGING TECHNOLOGY, Vol. 22, pp. 231-237.

16. *Thermal-Safe Test Scheduling for Core-Based System-on-Chip Integrated Circuits.* **Rosinger, Paul, Al-Hashimi, Bashir M. et Chakrabarty, Krishnendu.** 11, s.l. : IEEE, Novembre 2006, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, Vol. 25, pp. 2502-2512.

17. **CHIUEH, HERMING, DRAPER, JEFFREY et CHOMA, JOHN.** A Dynamic Thermal Management Circuit for System-On-Chip Designs. *Analog Integrated Circuits and Signal Processing.* Kluwer Academic Publishers, 2003, pp. 175-181.

18. **Chan, Chinhung, et al.** A Thermal-Aware Power Management Soft-IP for Platform-based SoC Designs. 2004, pp. 181-184.

19. **Xilinx.** *Xilinx 6200 Preliminary Data Sheet.* San Jose, CA : s.n., 1996.

20. **Altera.** *Stratix FPGA Family, Datasheet ver. 3.0.* décembre 2002.

21. **VOROS, NIKOLAOS S.** *System level design of reconfigurable systems-on-chip.* Dordrecht : Springer, 2005. 10 0-387-26103-6.

22. **BEN CHEHIDA, Karim.** *Méthodologie de Partitionnement Logiciel/Matériel pour Plateformes Reconfigurables Dynamiquement.* L'UNIVERSITE DE NICE-SOPHIA ANTIPOLIS. 2004. Thèse de doctorat.

23. Chameleon Systems. *http://www.chameleonsystems.com/.* [En ligne]

24. **Katarzyna, Leijten-Nowak.** *Template-Based Embedded Reconfigurable Computing.* Technische Universiteit Eindhoven. s.l. : Philips Electronics, 2004. PHD Thesis. 90-386-1583-3.

25. *Realization of wireless multimedia communication systems on reconfigurable platforms.* **Masselos, K, et al.** 4-6, 2003, Journal of systems architecture, Vol. 49, pp. 155-175.

26. *A VLSI Implementation of a Fuzzy-Inference Engine: Toward an Expert System on a Chip.* **TOGAI, MASAKI et WATANABE, HIROYUKI.** New York : Elsevier Science Publishing Co., Inc., 1986, INFORMATION SCIENCES, Vol. 38, pp. 147-163.

27. **Bouchon, B.** *La logique floue et ces applications.* Paris : Addison Wesley, 1995.

28. **ALSALEH, Hassan.** *Utilisation Classique et Extensions des Fonctionnalités des Processeurs Flous.* LAMII/CESALP. 1998. Thèse de doctorat.

29. *Serial Architectures for Efficient Digital Fuzzy Hardware Processing.* **Carrsco, L. et Getierrz, J.** Boston : Kluwer Academic Publishers, 1998, Fuzzy hardware: Architectures and Applications.

30. *A Parallel Processor Architecture for Real-Time Fuzzy Application.* **Ascia, G. et Catania, V.** Boston : Kluwer Academic Publishers, 1998, Fuzzy hardware: Architectures and Applications.

31. *A fast digital fuzzy processor.* **Gabrielli, A. et Gandolfi, E.** 1, Jan 1999, IEEE Micro., Vol. 19, pp. 68–79.

32. **SGS_THOMSON Microelectronics,.** *FUZZYSTUDIOTM 2.0 WARP2.0 Development.* 1996.

33. *A Context Switchable Fuzzy Inference Chip.* **Cao, Qi, et al.** 4, s.l. : IEEE, Août 2006, IEEE TRANSACTIONS ON FUZZY SYSTEMS,, Vol. 14, pp. 552-567.

34. *Processor, A Reconfigurable Parallel Architecture for a Fuzzy.* **ASCIA, G., et al.** New York : Elsevier Science Inc., 1996, INFORMATION SCIENCES, pp. 299-315.

35. *An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA System.* **Kim, Daijin.** 03, s.l. : IEEE, Juin 2000, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS,, Vol. 47, pp. 703-715.

36. *Implementing a Fuzzy System on a Field Programmable Gate Array.* **McKenna, Michael et Wilamowski, Bogdan M.** s.l. : IEEE, 2001, pp. 189-194.

37. **Muresan, Valentin, Orisu, Dan et Wang, Xiaojun.** FROM VHDL TO FPGA. A CASE STUDY OF A FUZZY LOGIC CONTROLLER. *Proceedings of the International Conference of Young Lecturers and PhD Students.* 1997, pp. 83 - 90.

38. **Ballouti, Adel.** *Techniques de test dans les systèmes reconfigurables : application au contrôleurs flous testables.* Département d'électronique, UFAS. 2007. Mémoire de magister.

39. **Jiménez, C. J., Solano, S. Sánchez et Barriga, A.** HARDWARE IMPLEMENTATION OF A GENERAL PURPOSE FUZZY CONTROLLER. *Sixth International Fuzzy Systems Association World Congress (IFSA'95).* Juillet 1995, Vol. 2, pp. 185-188.

40. **Vaidyanathan, Ramachandran et Trahan, Jerry L.** *DYNAMIC RECONFIGURATION, Architectures and Algorithms.* s.l. : Kluwer Academic Publishers, 2004. 0-306-48428-5.

41. *Du micro-processeur au circuit FPGA, une analyse sous l'angle de la reconfiguration.* **David, Raphaël, Lavenier, Dominique et Pillement, Sébastien.** 2003. Technique et science informatiques.

42. **Koren, Israel.** *Computer arithmetic algorithms.* Natick, Massachusetts : A K Peters, Ltd, 2002. 1-56881-160-8.

43. **Department of Computer Science, University of Manchester.** CS122 Electronics Laboratory Manual. 2000.

44. **Chemali, H., Ballouti, A. et Benhamadouche, A.** A Fast Testable Digital fuzzy logic Controller. *International computer system and information technology conference.* 2005, pp. 653-658.

45. **Chemali, H., Benhamadouche, A. et Ballouti, A.** Implementations of a Digital Fuzzy Logic Controller in FPGA. *1st International Symposium on Electromagnetism, Satellites and Cryptography.* 2005, pp. 210-213.

46. **Schaumont, Patrick, et al.** A Quick Safari Through the Reconfiguration Jungle. *DAC'01.* Juin 2001.

47. *Dynamic Hardware Multiplexing: Improving Adaptability with a Run Time Reconfiguration Manager.* **Benoit, P., et al.** s.l. : IEEE, 2006, Proceedings of the 2006 Emerging VLSI Technologies and Architectures.

48. **Sad, E.M., M.K., Ah& et Abutaleb, M.M.** Optimization of Reconfiguration Transitions for (Self-)reconfigurable FSM Using Decomposition. *Twenty Second Nntiond Radio Science Conference (NRSC 2005).* Mars 2005, pp. 1-10.

49. **Xilinx.** *Xilinx Design Reuse Methodology for ASIC and FPGA Designers.*

50. **SBS Implementers Forum.** *System Management Bus (SMBus) Specification.* s.l. : SBS Implementers Forum, 2000.

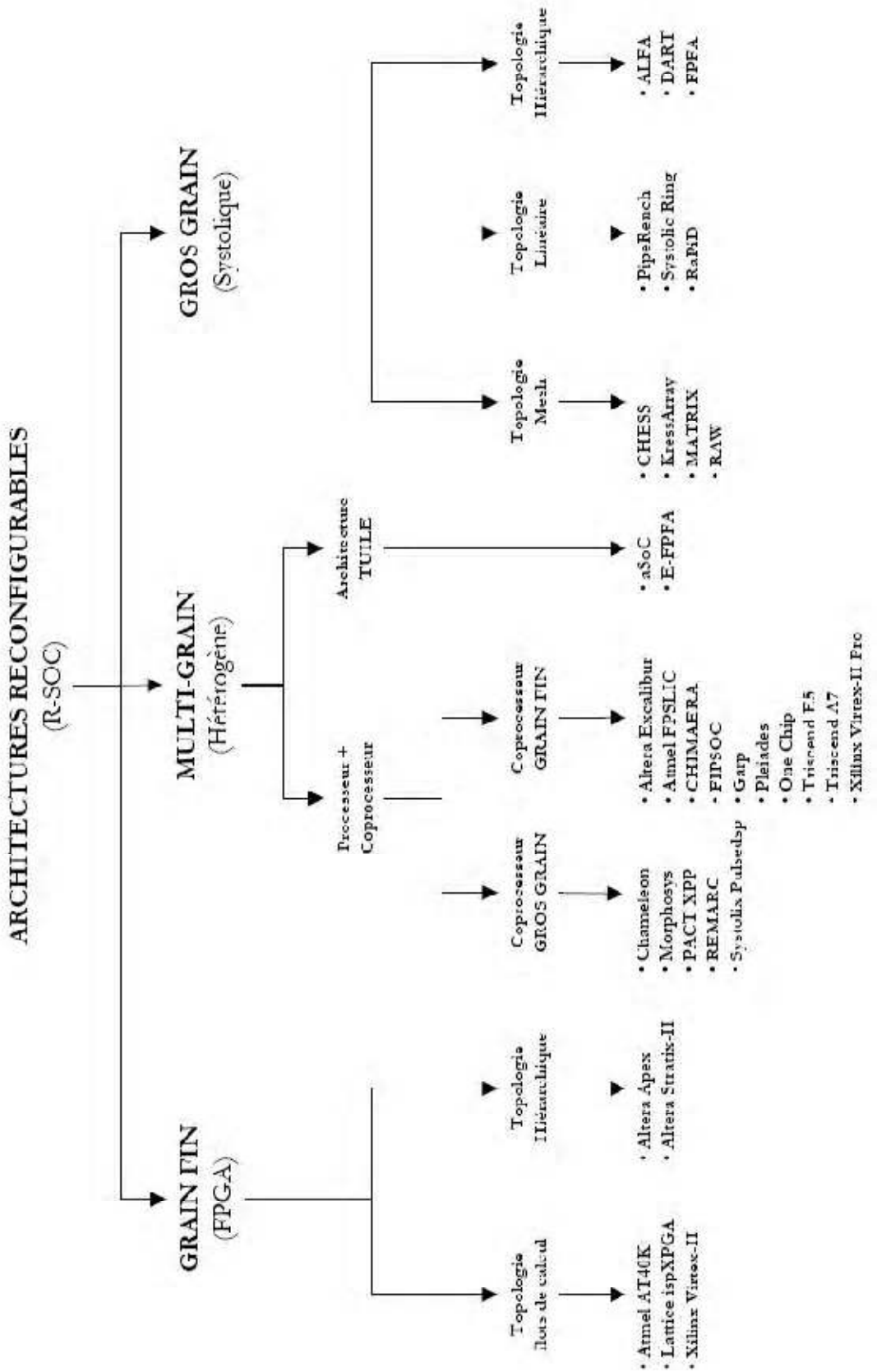
51. *Thermal Management for High Performance Integrated Circuits With Non-uniform Chip Power Considerations.* **Yuan, Tsorng-Dih et Hong, Bor Zen.** s.l. : IEEE, 2001, Seventeenth IEEE SEMI-THERM Symposium, pp. 95-101.

52. **Huang, Yu-Jung, Fu, Shen-Li et Chang, Yun-Wen.** FUZZY LOGIC BASED THERMAL DESIGN FOR MCM PLACEMENT. *IEMT/IMC Proceedings.* 1998, pp. 179-184.

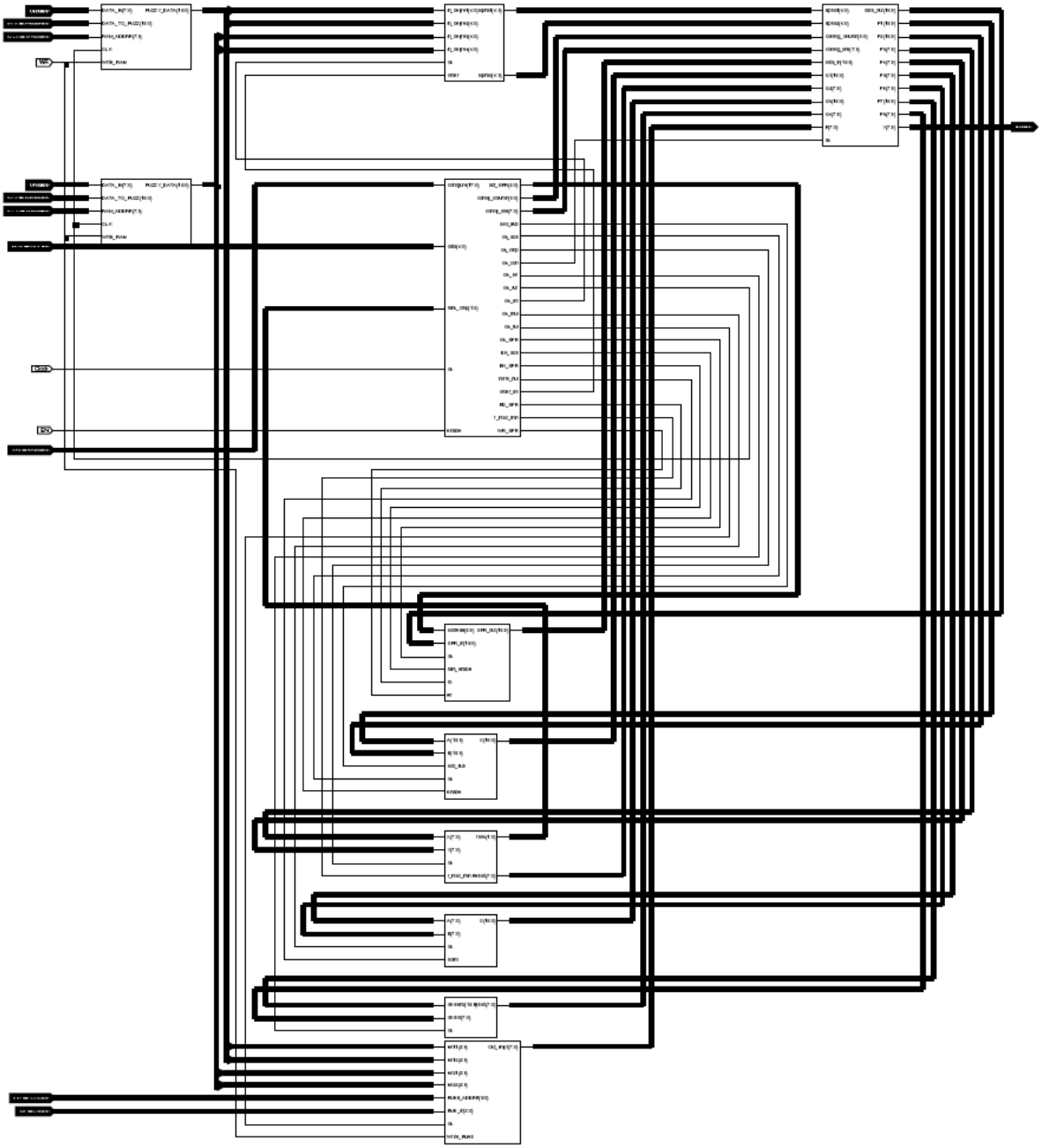
53. **Hung, W.-L., et al.** *Temperature-Aware Voltage Islands Architecting in System-on-Chip Design.* [prod.] IBM EDA Laboratory. New York : s.n., 2005.

ANNEXE

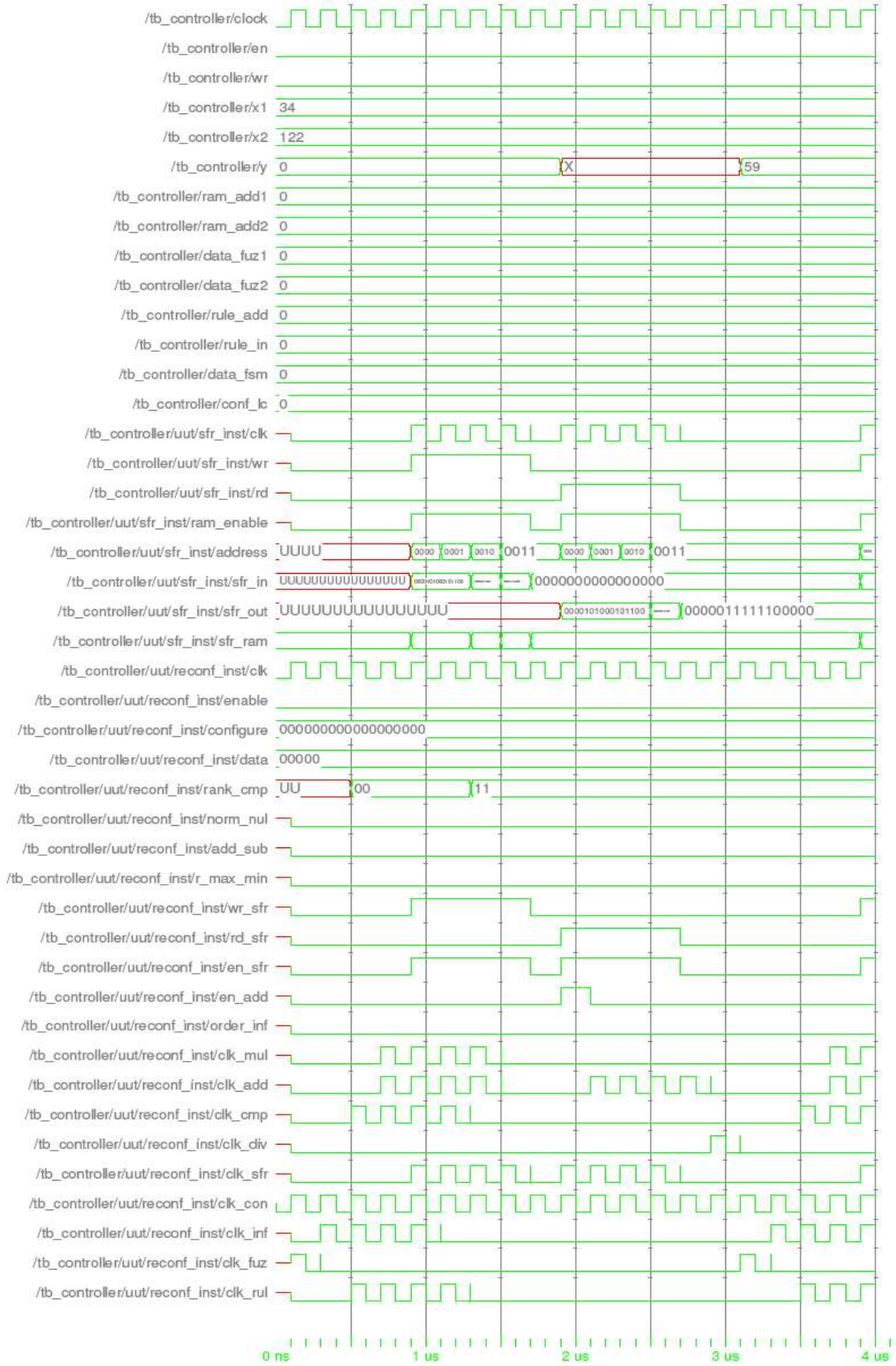
1. Classification des architectures reconfigurables.



2. Synthèse du système flou



3. Simulation du système flou pour une seule valeur



4. Règles de conception et réutilisation.

Voici quelques règles de conceptions :

1. Déclarer les ports dans un ordre logique, par fonctionnalité plutôt que par type.
2. Ne déclarer qu'un seul port par ligne, si possible avec le commentaire associé sur la même ligne.
3. Pour chaque groupe logique de ports, respecter l'ordre suivant :
    - Clocks,
    - Resets,
    - Enables,
    - Autre signaux de contrôle,
    - Données.
4. Utiliser des fonctions plutôt que répéter une même section de code. Ne pas oublier de commenter la fonction.
5. Utiliser des boucles et des tableaux pour améliorer la clarté du code.
6. Le nombre de domaines d'horloge ainsi que leurs fréquences sont bien documentés.
7. Les horloges au niveau système ne doivent utiliser que des arbres de distribution dédiés.
8. Si deux domaines d'horloge asynchrones doivent être utilisés, s'assurer qu'ils soient synchronisés dans un module simple, qui devrait juste être composé de registres permettant de rendre les deux domaines d'horloge synchrones.
9. La stratégie de reset doit être documentée au niveau système, en particulier pour les points suivants :
    - Synchrone ou asynchrone ?
    - Reset à la mise sous-tension interne ou externe ?
    - Chaque bloc doit-il être pouvoir être remis à zéro individuellement (pour déboguage)?
    - La stratégie de vérification au niveau système doit être développée et documentée avant de commencer la sélection (si le système est un assemblage de modules réutilisables) ou le codage des sous-blocs.
10. La stratégie de vérification au niveau des sous-blocs doit être développée et documentée avant d'en commencer la conception.
11. Soigner la contrôlabilité et l'observabilité pour faciliter le déboguage au niveau système. La contrôlabilité est facilitée en donnant la possibilité d'éteindre individuellement chaque sous-module, ou de le mettre en mode déboguage

(fonctionnement simplifié). L'observabilité est améliorée en ajoutant des bus permettant de contrôler l'état du système.

12. Les registres ne doivent être sensibles qu'à un seul front d'horloge (généralement montant).

13. Pas de logique sur l'horloge (gated clock).

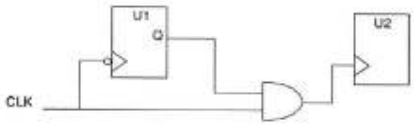


Figure 2. 1 : gated clock.

14. Pas d'horloge générées en interne (ou donner un moyen d'utiliser une horloge externe en phase de déboguage).

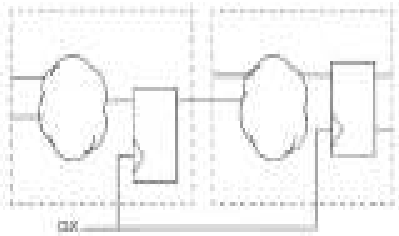


Figure 2. 2 : Clock idéal.

15. Pas de reset générés en interne ou de reset conditionnels (ou donner un moyen d'utiliser un reset externe en phase de déboguage).

16. Utiliser un pipelinage massif, permettant d'atteindre des bonnes performances.

Exemple :

Avant : Après :

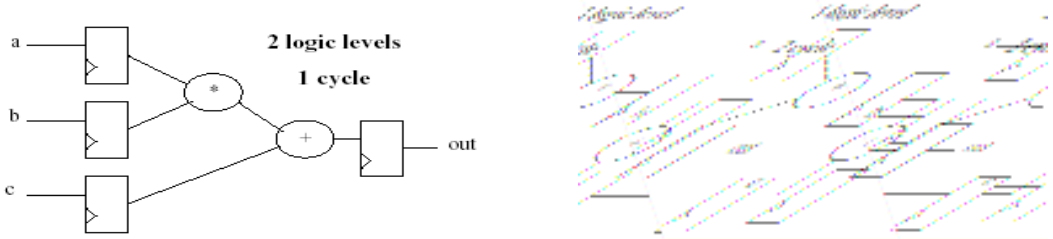


Figure 2. 3 : Architecture en pipeline

17. Utiliser des registres indépendants de la technologie lors du codage style RTL pour la logique séquentielle.

18. Le module doit être synchrone et basé sur des bascules D. Ne pas instancier de latches lors du codage RTL, particulièrement des bascules S-R.

19. Lors d'une assignation conditionnelle, utiliser l'une des règles suivantes pour ne pas inférer de latches :

    - Assigner une valeur par défaut en début d'un process,
    - Assigner une valeur de sortie pour chaque combinaison des entrées,
    - Terminer l'assignation conditionnelle par la condition else ou when others.
    - Pas de retour combinatoire, i.e. de boucle sur un process combinatoire.
    - Compléter toutes les listes de sensibilité des process :
    - L'horloge et les signaux asynchrones pour un process séquentiel,
    - Tous les signaux d'entrée d'un process combinatoire.
    - N'indiquer que les signaux nécessaires à la liste de sensibilité.

20. Utiliser des signaux plutôt que des variables. Exception possible pour les indices de tableau ainsi que les valeurs intermédiaires accumulées lors de boucles.

21. Utiliser des conditions case plutôt que des conditions if-then-else, quand approprié.

Exemple :

Description d'un multiplexeur avec le "If-then-Else"

```
process (sel,a,b,c,d)
begin
  if (sel = "00") then
          outi <= a;
  elsif (sel = "01") then
          outi <= b;
  elsif (sel = "10") then
          outi <= c;
  else
          outi <= d;
  end if;
end process ;
```

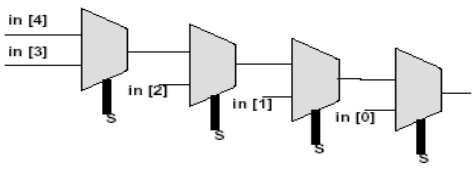


Figure 2. 4 : synthèse d'un multiplexeur avec le "If-then-Else"

Description d'un multiplexeur avec le " Case-when"

```
process (sel,a,b,c,d)
```

```
begin
  case sel is
        when "00" => outc <= a;
        when "01" => outc <= b;
        when "10" => outc <= c;
        when others => outc <= d;
  end case;
end process;
```

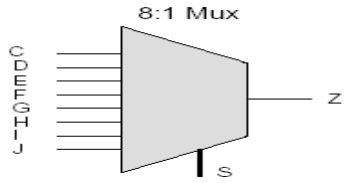


Figure 2. 5 : synthèse d'un multiplexeur avec le " Case-when"

22. La description d'une machine à états doit se faire en utilisant 2 process, un pour la logique séquentielle et un pour la logique combinatoire.

23. Créer un type énuméré pour décrire le vecteur d'état, permettant à l'outil de synthèse d'en optimiser le codage.

24. Pour une machine à état, assigner un état par défaut.

25. Mettre un registre en sortie de chaque sous-bloc (si le design est complètement synchrone, ce registre existe déjà)

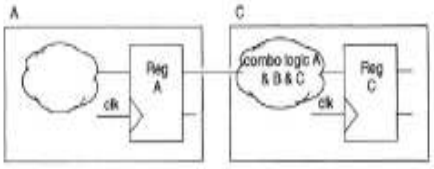


Figure 2. 6 : Registre en sortie de chaque sous bloc

26. Mettre un registre en entrée de chaque bloc (à partir d'une certaine complexité seulement, pas nécessaire pour les sous-blocs). Ceci permet de mieux prévoir les timings quelle que soit la complexité et la taille du système dans lequel est intégré le bloc.

5. Programme principale du système flou

```
1----------------------------------------------------------------------------
2     -- Company:
3     -- Engineer:
4     --
5     -- Create Date:             20:55:36 06/02/2007
6     -- Design Name:            controlleur du système flou
7     -- Module Name:             controller - Behavioral
8     -- Project Name:
9     -- Target Devices:
10    -- Tool versions:
11    -- Description:
12    --
13    -- Dependencies:
14    --
15    -- Revision:
16    -- Revision 0.01 - File Created
17    -- Additional Comments:
18    --
19    ----------------------------------------------------------------------------
20
21    library IEEE;
22    use IEEE.STD_LOGIC_1164.ALL;
23    use IEEE.STD_LOGIC_ARITH.ALL;
24    use IEEE.STD_LOGIC_UNSIGNED.ALL;
25    library work;
26    use work.ALL;
27
28    entity controller is
29    port ( Clock : in                std_logic;
30    EN                 : in  std_logic;
31    WR                 : in  std_logic;
32    X1                 : in  std_logic_vector (7 downto 0);
33    X2                 : in  std_logic_vector (7 downto 0);
34    Y                  : out std_logic_vector (7 downto 0);
35
36    RAM_add1           : in      std_logic_vector (7 downto 0);
37    RAM_add2           : in      std_logic_vector (7 downto 0);
38    data_fuz1 : in    std_logic_vector (15 downto 0);
39    data_fuz2 : in    std_logic_vector (15 downto 0);
40    rule_add           : in      std_logic_vector (5 downto 0);
41    rule_in            : in      std_logic_vector (2 downto 0);
42    data_fsm           : in      std_logic_vector (4 downto 0);
43    conf_LC            : in      std_logic_vector (17 downto 0));
44    end controller;
45
46    architecture BEHAVIORAL of controller is
47    signal bus_address : std_logic_vector (3 downto 0);
48    signal bus_column : std_logic_vector (9 downto 0);
49    signal bus_line : std_logic_vector (7 downto 0);
50
51    signal cnt_norm_nul               : std_logic;
52    signal cnt_add_sub                : std_logic;
53    signal cnt_r_max_min              : std_logic;
54    signal cnt_WR_SFR                 : std_logic;
55    signal cnt_RD_SFR                 : std_logic;
```

```
56    signal cnt_EN_SFR                : std_logic;
57    signal cnt_en_add                : std_logic;
58    signal cnt_order_inf             : std_logic;
59    signal cnt_clk_mul               : std_logic;
60    signal cnt_clk_add               : std_logic;
61    signal cnt_clk_cmp               : std_logic;
62    signal cnt_clk_div               : std_logic;
63    signal cnt_clk_SFR               : std_logic;
64    signal cnt_clk_con               : std_logic;
65    signal cnt_clk_inf               : std_logic;
66    signal cnt_clk_fuz               : std_logic;
67    signal cnt_clk_rul               : std_logic;
68
69    signal Bus_fuzzy1 : std_logic_vector(15 downto 0);
70    signal Bus_fuzzy2 : std_logic_vector(15 downto 0);
71
72    signal Bus_rank : std_logic_vector(1 downto 0);
73    signal Bus_P1 : std_logic_vector (15 downto 0);
74    signal Bus_P2 : std_logic_vector (15 downto 0);
75    signal Bus_P3 : std_logic_vector (7 downto 0);
76    signal Bus_P4 : std_logic_vector (7 downto 0);
77    signal Bus_P5 : std_logic_vector (7 downto 0);
78    signal Bus_P6 : std_logic_vector (7 downto 0);
79    signal Bus_P7 : std_logic_vector (15 downto 0);
80    signal Bus_P8 : std_logic_vector (7 downto 0);
81    signal Bus_Q1 : std_logic_vector (15 downto 0);
82    signal Bus_Q2 : std_logic_vector (7 downto 0);
83    signal Bus_Q3 : std_logic_vector (15 downto 0);
84    signal Bus_Q4 : std_logic_vector (7 downto 0);
85    signal Bus_alpha1 : std_logic_vector (4 downto 0);
86    signal Bus_alpha2 : std_logic_vector (4 downto 0);
87    signal Bus_R : std_logic_vector (7 downto 0);
88    signal Bus_data_in : std_logic_vector (15 downto 0);
89    signal Bus_data_out : std_logic_vector (15 downto 0);
90
91    COMPONENT reconfigure
92    PORT(
93    clk            : IN std_logic;
94    enable         : IN std_logic;
95    configure      : IN std_logic_vector(17 downto 0);
96    data           : IN std_logic_vector(4 downto 0);
97    rank_cmp       : in std_logic_vector (1 downto 0);
98    norm_nul       : OUT std_logic;
99    add_sub        : OUT std_logic;
100   r_max_min      : OUT std_logic;
101   WR_SFR         : OUT std_logic;
102   RD_SFR         : OUT std_logic;
103   EN_SFR         : OUT std_logic;
104   EN_add         : OUT std_logic;
105   order_inf      : OUT std_logic;
106   clk_mul        : OUT std_logic;
107   clk_add        : OUT std_logic;
108   clk_cmp        : OUT std_logic;
109   clk_div        : OUT std_logic;
110   clk_SFR        : OUT std_logic;
111   clk_con        : OUT std_logic;
112   clk_inf        : OUT std_logic;
113   clk_fuz        : OUT std_logic;
```

```
114   clk_rul        : OUT std_logic;
115   adr_SFR        : OUT std_logic_vector(3 downto 0);
116   config_column  : OUT std_logic_vector(9 downto 0);
117   config_line    : OUT std_logic_vector(7 downto 0)
118   );
119   END COMPONENT;
120
121   component max_min
122   port ( clk     : in             std_logic;
123   r_max_min : in        std_logic;
124   X                    : in      std_logic_vector (7 downto 0);
125   Y                    : in      std_logic_vector (7 downto 0);
126   rank                 : out     std_logic_vector (1 downto 0);
127   Result               : out     std_logic_vector (7 downto 0));
128   end component;
129
130   component add
131   port ( add_sub : in                  std_logic;
132   clk                  : in    std_logic;
133   enable               : in    std_logic;
134   A                    : in    std_logic_vector (15 downto 0);
135   B                    : in    std_logic_vector (15 downto 0);
136   C                    : out   std_logic_vector (15 downto 0));
137   end component;
138
139   component division
140   port ( clk     : in            std_logic;
141   dividend : in        std_logic_vector (15 downto 0);
142   divisor              : in     std_logic_vector (7 downto 0);
143   result               : out    std_logic_vector (7 downto 0));
144   end component;
145
146   component fuzzy_Rules
147   port ( clk     : in               std_logic;
148   Write_Rules : in     std_logic;
149   Rules_ADERR : in     std_logic_vector (5 downto 0);
150   Rule_in              : in        std_logic_vector (2 downto 0);
151   Mf11                 : in        std_logic_vector (2 downto 0);
152   Mf12                 : in        std_logic_vector (2 downto 0);
153   Mf21                 : in        std_logic_vector (2 downto 0);
154   Mf22                 : in        std_logic_vector (2 downto 0);
155   Out_ling1            : out       std_logic_vector (7 downto 0));
156   end component;
157
158   component mult1
159   port ( clk     : in        std_logic;
160   norm : in            std_logic;
161   A                    : in std_logic_vector (7 downto 0);
162   B                    : in std_logic_vector (7 downto 0);
163   C                    : outstd_logic_vector (15 downto 0));
164   end component;
165
166   component inference
167   port ( clk     : in            std_logic;
168   in_degre1 : in       std_logic_vector (4 downto 0);
169   in_degre2 : in       std_logic_vector (4 downto 0);
170   in_degre3 : in       std_logic_vector (4 downto 0);
171   in_degre4 : in       std_logic_vector (4 downto 0);
```

```
172   order                : in      std_logic;
173   alpha1               : out     std_logic_vector (4 downto 0);
174   alpha2               : out     std_logic_vector (4 downto 0));
175   end component;
176
177   --      component interconnexion_02
178   component interconnect
179   port ( cconfig_column : in                  std_logic_vector (9 downto
0);
180   cconfig_line         : in           std_logic_vector (7 downto 0);
181   Q1                   : in          std_logic_vector (15 downto 0);
182   Q2                   : in          std_logic_vector (7 downto 0);
183   Q3                   : in          std_logic_vector (15 downto 0);
184   Q4                   : in          std_logic_vector (7 downto 0);
185   R                    : in          std_logic_vector (7 downto 0);
186   alpha1               : in          std_logic_vector (4 downto 0);
187   alpha2               : in          std_logic_vector (4 downto 0);
188   data_in              : in          std_logic_vector (15 downto 0);
189   P1                   : out         std_logic_vector (15 downto 0);
190   P2                   : out         std_logic_vector (15 downto 0);
191   P3                   : out         std_logic_vector (7 downto 0);
192   P4                   : out         std_logic_vector (7 downto 0);
193   P5                   : out         std_logic_vector (7 downto 0);
194   P6                   : out         std_logic_vector (7 downto 0);
195   P7                   : out         std_logic_vector (15 downto 0);
196   P8                   : out         std_logic_vector (7 downto 0);
197   Y                    : out         std_logic_vector (7 downto 0);
198   data_out             : out         std_logic_vector (15 downto 0);
199   clk                  : in          std_logic);
200
201   end component;
202
203   component SFR
204   port ( clk    : in             std_logic;
205   wr                   : in      std_logic;
206   rd                   : in      std_logic;
207   ram_enable : in      std_logic;
208   address              : in      std_logic_vector (3 downto 0);
209   SFR_in               : in      std_logic_vector (15 downto 0);
210   SFR_out              : out     std_logic_vector (15 downto 0));
211   end component;
212
213   component fuzzifier
214   port ( CLK    : in               std_logic;
215   Write_RAM            : in        std_logic;
216   RAM_ADERR            : in        std_logic_vector (7 downto 0);
217   DATA_TO_FUZZ : in    std_logic_vector (15 downto 0);
218   DATA_IN              : in        std_logic_vector (7 downto 0);
219   FUZZY_DATA           : out       std_logic_vector (15 downto 0));
220   end component;
221
222   begin
223
224   Reconf_Inst: reconfigure
225   PORT MAP(
226   clk                    => clock,
227   enable                 => EN,
228   configure(17 downto 0)  => conf_LC(17 downto 0),
```

```
229    data(4 downto 0)        => data_fsm(4 downto 0),
230    rank_cmp(1 downto 0)    => bus_rank(1 downto 0),
231    norm_nul                => cnt_norm_nul,
232    add_sub                 => cnt_add_sub,
233    r_max_min               => cnt_r_max_min,
234    WR_SFR                  => cnt_WR_SFR,
235    RD_SFR                  => cnt_RD_SFR,
236    EN_SFR                  => cnt_EN_SFR,
237    order_inf               => cnt_order_inf,
238    EN_add                  => cnt_en_add,
239    clk_mul                 => cnt_clk_mul,
240    clk_add                 => cnt_clk_add,
241    clk_cmp                 => cnt_clk_cmp,
242    clk_div                 => cnt_clk_div,
243    clk_SFR                 => cnt_clk_SFR,
244    clk_con                 => cnt_clk_con,
245    clk_inf                 => cnt_clk_inf,
246    clk_fuz                 => cnt_clk_fuz,
247    clk_rul                 => cnt_clk_rul,
248    adr_SFR(3 downto 0)     => bus_address(3 downto 0),
249    config_column(9 downto 0)                          => bus_column(9
downto 0),
250    config_line(7 downto 0) => bus_line(7 downto 0));
251
252    CMP_Inst : max_min
253    port map (clk =>cnt_clk_cmp,
254    r_max_min               =>cnt_r_max_min,
255    X(7 downto 0)           =>Bus_P3(7 downto 0),
256    Y(7 downto 0)           =>Bus_P4(7 downto 0),
257    rank(1 downto 0)        =>bus_rank(1 downto 0),
258    Result(7 downto 0)      =>Bus_Q2(7 downto 0));
259
260    Add_Inst : add
261    port map (A(15 downto 0)                    =>Bus_P1(15 downto 0),
262    add_sub                 =>cnt_add_sub,
263    B(15 downto 0)          =>Bus_P2(15 downto 0),
264    clk                     =>cnt_clk_add,
265    enable                  =>cnt_en_add,
266    C(15 downto 0)          =>Bus_Q1(15 downto 0));
267
268    Mult_Inst : mult1
269    port map (A(7 downto 0)                        =>Bus_P5(7 downto 0),
270    B(7 downto 0)           =>Bus_P6(7 downto 0),
271    clk                     =>cnt_clk_mul,
272    norm                    =>cnt_norm_nul,
273    C(15 downto 0)          =>Bus_Q3(15 downto 0));
274
275    Div_Inst : division
276    port map (clk =>cnt_clk_div,
277    dividend(15 downto 0)   =>Bus_P7(15 downto 0),
278    divisor(7 downto 0)     =>Bus_P8(7 downto 0),
279    result(7 downto 0)      =>Bus_Q4(7 downto 0));
280
281    Rule_Inst : fuzzy_Rules
282    port map (clk =>cnt_clk_rul,
283    Mf11(2 downto 0)        =>Bus_fuzzy1(15 downto 13),
284    Mf12(2 downto 0)        =>Bus_fuzzy1(7 downto 5),
285    Mf21(2 downto 0)        =>Bus_fuzzy2(15 downto 13),
```

```
286   Mf22(2 downto 0)         =>Bus_fuzzy2(7 downto 5),
287   Rules_ADERR(5 downto 0)=>rule_add(5 downto 0),
288   Rule_in(2 downto 0)      =>rule_in(2 downto 0),
289   Write_Rules              =>WR,
290   Out_ling1(7 downto 0)    =>Bus_R(7 downto 0));
291
292   Infer_Inst : inference
293   port map (clk =>cnt_clk_inf,
294   in_degre1(4 downto 0)    =>Bus_fuzzy1(12 downto 8),
295   in_degre2(4 downto 0)    =>Bus_fuzzy1(4 downto 0),
296   in_degre3(4 downto 0)    =>Bus_fuzzy2(12 downto 8),
297   in_degre4(4 downto 0)    =>Bus_fuzzy2(4 downto 0),
298   order                    =>cnt_order_inf,
299   alpha1(4 downto 0)       =>Bus_alpha1(4 downto 0),
300   alpha2(4 downto 0)       =>Bus_alpha2(4 downto 0));
301
302   --      Connect_Inst : interconnexion_02
303   Connect_Inst : interconnect
304   port map (alpha1(4 downto 0)                        =>Bus_alpha1(4
downto 0),
305   alpha2(4 downto 0)       =>Bus_alpha2(4 downto 0),
306   clk                      =>cnt_clk_con,
307   cconfig_column(9 downto 0)=>bus_column(9 downto 0),
308   cconfig_line(7 downto 0) =>bus_line(7 downto 0),
309   data_in(15 downto 0)     =>Bus_data_in(15 downto 0),
310   Q1(15 downto 0)          =>Bus_Q1(15 downto 0),
311   Q2(7 downto 0)           =>Bus_Q2(7 downto 0),
312   Q3(15 downto 0)          =>Bus_Q3(15 downto 0),
313   Q4(7 downto 0)           =>Bus_Q4(7 downto 0),
314   R(7 downto 0)            =>Bus_R(7 downto 0),
315   data_out(15 downto 0)    =>Bus_data_out(15 downto 0),
316   P1(15 downto 0)          =>Bus_P1(15 downto 0),
317   P2(15 downto 0)          =>Bus_P2(15 downto 0),
318   P3(7 downto 0)           =>Bus_P3(7 downto 0),
319   P4(7 downto 0)           =>Bus_P4(7 downto 0),
320   P5(7 downto 0)           =>Bus_P5(7 downto 0),
321   P6(7 downto 0)           =>Bus_P6(7 downto 0),
322   P7(15 downto 0)          =>Bus_P7(15 downto 0),
323   P8(7 downto 0)           =>Bus_P8(7 downto 0),
324   Y(7 downto 0)            =>Y(7 downto 0));
325
326   SFR_Inst : SFR
327   port map (address(3 downto 0)                    =>bus_address(3 downto
0),
328   clk                      =>cnt_clk_SFR,
329   ram_enable               =>cnt_EN_SFR,
330   rd                       =>cnt_RD_SFR,
331   SFR_in(15 downto 0)      =>Bus_data_out(15 downto 0),
332   wr                       =>cnt_WR_SFR,
333   SFR_out(15 downto 0)     =>Bus_data_in(15 downto 0));
334
335   fuzzifier_X1 : fuzzifier
336   port map (CLK =>cnt_clk_fuz,
337   DATA_IN(7 downto 0)      =>X1(7 downto 0),
338   DATA_TO_FUZZ(15 downto 0) =>data_fuz1(15 downto 0),
339   RAM_ADERR(7 downto 0)    =>RAM_add1(7 downto 0),
340   Write_RAM                =>WR,
341   FUZZY_DATA(15 downto 0)  =>Bus_fuzzy1(15 downto 0));
```

```
342
343   fuzzifier_X2 : fuzzifier
344   port map (CLK =>cnt_clk_fuz,
345   DATA_IN(7 downto 0)      =>X2(7 downto 0),
346   DATA_TO_FUZZ(15 downto 0) =>data_fuz2(15 downto 0),
347   RAM_ADERR(7 downto 0)    =>RAM_add2(7 downto 0),
348   Write_RAM                =>WR,
349   FUZZY_DATA(15 downto 0)  =>Bus_fuzzy2(15 downto 0));
350
351   end BEHAVIORAL;
352
```

ملخص:

إن التطورات الحاصلة في الأنظمة المدمجة داخل رقاقات الكترونية أحدثت الكثير من التعقيدات في طرق التصميم و الإنجاز، وذالك ازداد صعوبتا بعد دمج أنظمة معادة التهيئة لزيادة عمر هذه الأنظمة و إضافة الكثير من المرونة للتطبيقات الممكنة.

في هذه الأطروحة قمنا بدراسة حول الأنظمة المعادة التهيئة، كذلك قمنا بالتفكير و محاولة دراسة المشاكل المتعلقة أساسا بإنجاز الأنظمة الغامضة، حيث قمنا بتصميم نظام غامض معاد التهيئة و هذا باستعمال الدارات المبرمجة بالبوابات المصفوفة (FPGA)، هذا النظام يحتوي على خصائص مميزة مقارنة مع الأنظمة التقليدية.

وأخيرا، قمنا بإنجاز تطبيق للنظام المنجز وهذا من خلال استخدامه كمراقب غامض لإدارة وتنظيم نشاط وحرارة نظام مدمج و هذا باستعمال عدة طرقه تمت مناقشتها وتطبيقها عمليا.

Abstract:

A VHDL based platform of Xilinx and a "Mentor Graphics" Modelim simulator are used to design a fuzzy reconfigurable system. The system is implemented on "Virtex II Pro" FPGA families and used to control the highly difficult thermal dissipation problem in a SoC via reconfiguring IP functional modes and using solely the SoC's SMBus communication channel.

Hardware and software reconfigurable techniques are thoroughly studied in order to design our fuzzy system with highly reconfigurable blocks, yielding to global functional system optimization.

Analysis of block's reconfiguration performance showed that not only reconfiguration has a powerful impact in system design but may lead to a new interesting design vision. In fact, the designer is no more fixed with a unique solution, but is faced with selecting multiple alternatives which generally depend on active functional structures and targeted aims.

Behavioural reconfiguration saves intellectual property and assures reliability in integrated system design.