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'Electronique
Pour l'obtention du diplôme de

MAGISTER

Option : contrôle

Par

MOUSSAOUI MOURAD

Thème:

CONTROLEUR FLOU TESTABLE

Soutenu le: 26/01/2009, devant la commission d'examen:

JURY :

Mr. Zegadi Ameur	**Professeur à l'université de Sétif**	**Président**
Mr. Khellaf Abdelhafid	**Professeur à l'université de Sétif**	**Rapporteur**
Mr. Amardjia Nourredine	**Maître de conférence à l'université de Sétif**	**Examinateur**
Mr. Bourouba Naceredine	**Maître de conférence à l'université de Sétif**	**Examinateur**
Mr. Chemali Hamimi	**Maître de conférence à l'université de Sétif**	**Invité**

Remerciements

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

Tout d'abord, j'adresse ma gratitude et mes profonds remerciements à Mr A. KHELLAF et Mr H. CHEMALI d'avoir accepter de diriger ce travail et de m'avoir bien orienté.

Je remercie Vivement Mr A. ZEGADI d'avoir accepter de présider le jury de cette thèse.

Je tiens à exprimer ma profonde reconnaissance à Mr N. BOUROUBA et Mr N. AMARDJIA de m'avoir fait l'honneur d'accepter d'être membres du jury de ce mémoire.

Je tiens enfin à remercier également tous les enseignants, collègues et amis, au près des quels j'ai trouvé un excellent esprit de coopération.

DEDICACES

A mes très chers parents pour leurs sacrifices et leurs soutiens, enfin pour tout,

A toute ma famille,

A tous ceux qui me sont chers,

A tous mes amis(es),

Enfin à tous qui de prés ou de loin ont contribué à la réalisation de ce travail

A vous tous j'offre ce modeste travail,

Moussaoui.M

Résumé

La conception d'un contrôleur flou testable 2 entrées – 1 sortie sur FPGA est effectuée, en utilisant le langage de description de matériel "VHDL", sur une plateforme de conception de Xilinx ISE7.1i couplée directement avec le simulateur ModelSim de Mentor Graphic.

Ce contrôleur est synthétisé sous forme d'un "IP Core" (Intellectual Property) où les exigences de rapidité de traitement et contraintes de testabilité sont fortement considérées. En plus, notre architecture prend en charge la reconfiguration très convoitée et ciblée dans les applications modernes. Les résultats de synthèse et de simulation obtenus et consignés dans ce mémoire montrent la bonne stratégie de développement suivie d'une part et attestent le degré élevé de maîtrise des techniques de conception d'autre part.

L'implantation du contrôleur sur différentes "FPGA" a montré son bon comportement global et une nette satisfaction des exigences de vitesse et de souplesse d'exploitation imposées et recherchées dans les applications industrielles modernes.

La validation du contrôleur développé a nécessité un passage par une phase de vérification exhaustive des différents modules qu'ils le constituent d'une part et d'une expérimentation d'autre part. En effet, une application très complexe du type freinage d'un métro a fait l'objet de cette expérimentation.

Le contrôleur développé pourrait représenter un embryon intelligent autour duquel il est possible de bâtir une bonne stratégie de commande de processus dans les systèmes embarqués.

Mots clés : Logique floue, Test, VHDL, IP.

GLOSSAIRE

FLC	*Fuzzy Logic Controller.*
FLIPS	*Fuzzy Logic Inference Per Second.*
WARP	*Weight Associative Rule processor.*
CAD	*Computer aided design.*
TTM	*Time To market.*
DUT	*Device Under Test.*
ATE	*Automatic Test Equipment.*
ATPG	*Automatic Test Pattern Generation.*
DFT	*Design For Testability.*
BIST	*Built In Self Test.*
LFSR	*Linear Feedback Shift Register.*
MISA	*Multiple Input Signature Analyser.*
MISR	*Multiple Input Shift Register.*
BS	*Boundary Scan.*
BSR	*Boundary Scan Register.*
TAP	*Test Access Port.*
IP	*Intellectual Property.*
VHDL	*Very High speed integrated circuits Hardware Description Language.*
FPGA	*Field Programmable Gate Array.*
ASIC	*Application Specific Integrated Circuit.*
CLB	*Configurable Logic Bloc.*
IOB	*Input Output Bloc.*
LUT	*Look-Up Tables.*

SOMMAIRE

Chapitre I
Logique et processeurs flous

Chapitre II
Test des systèmes électroniques

Chapitre III
Conception d’un contrôleur flou testable

Chapitre IV
Application, Contrôle de freinage d'un métro

INTRODUCTION GENERALE

Les avancées technologiques, de ces dernières années, ont permis la mise au point de produits très adaptés aux différentes applications qui contribuent au bien être de l"homme et qui ciblent essentiellement sa sécurité. Néanmoins, les coûts de développement restent très élevés vis-à-vis des produits à large consommation. Les plateformes de développement des cœurs IP constituent la nouvelle stratégie qui devrait optimiser les méthodes de conception. En effet, cette ouverture contribuera inévitablement à apporter de nouvelles solutions et maintenir des évolutions continuelles de l'électronique. Ces tâches désormais, ne sont plus confinées entre les mains des grandes compagnies uniquement, offrent des alternatives de conception de valeur.

Aujourd'hui, il est possible de réaliser des circuits intégrés numériques extrêmement complexes à base de plusieurs centaines de milliers de portes logiques grâce à ces plateformes de conception assistée par ordinateur (CAD) **[1].**

La plupart des circuits de par leur complexité, peuvent être affectés par des défauts liés à la technologie et aux processus de fabrication. Ces défauts peuvent présenter des mal-fonctionnalités permanentes ou temporaires.

La complexité croissante des circuits intégrés, associée à une augmentation du rapport entre le nombre de composants logiques et le nombre de broches, a fait augmenter de manière importante la difficulté de tester correctement ces circuits. Par conséquent, le test des circuits intégrés devient de plus en plus complexe.

Après l'élimination de toutes les erreurs de conception, le bon fonctionnement du circuit dépend alors essentiellement de la qualité du procédé technologique de fabrication. Pour s'assurer du bon fonctionnement, il est nécessaire de tester le circuit. Compte tenu de la complexité toujours croissante des circuits intégrés actuels, des techniques de conception en vue du test (DFT) sont utilisées pour augmenter la testabilité d'un circuit intégré et réduire son coût et son temps de conception "*Time To Market*" **[2].**

Les applications modernes sont complexes et difficiles à modéliser. Le caractère non linéaire et le manque de prédiction juste des événements ont poussé les concepteurs à exploiter la logique floue qui est de naissance développée dans ce sens.

Le système flou peut résoudre un problème indépendamment de la connaissance de la relation entrées-sorties **[3]**. Nous avons étudié dans ce travail la possibilité d'implémenter un processeur flou très rapide et capable de prendre en charge non seulement la non-modélisation du système à contrôler mais aussi les exigences de testabilité, de reconfiguration et l'implantation sous format d'un IP soft, IP hard ou mixte.

Comme dans certains champs d'application, les dispositifs électroniques utilisés pour la détection, la reconnaissance, l'identification et le stockage des événements doivent être aussi rapides que possibles, et doivent être conçus avec une forte immunité au bruit, avec faible consommation d'énergie et de hautes performances en termes de fiabilité, de robustesse et de flexibilité.

Nous proposons dans ce travail une alternative pour la prise en compte des contraintes fonctionnelles citées ci dessus par une conception d'un contrôleur où il est question de rechercher des solutions à travers la souplesse architecturale d'un côté et l'optimisation des fonctions à traiter d'autre part. Une réduction substantielle du temps de traitement est obtenue grâce à une identification intelligente des règles actives.

En ce qui concerne l'implémentation technologique du contrôleur flou présenté dans ce mémoire, les circuits programmables FPGA (Field programmable Gate Arrays) sont utilisés pour la phase préliminaire de développement. Les technologies numériques ont été choisies au lieu d'analogiques car elles laissent aux concepteurs un choix ouvert d'utilisation des cellules standard. Avec ces cellules numériques, le concepteur de l'électronique se sent beaucoup plus capable de concevoir de grandes et complexes architectures **[4].**

La conception et la synthèse du contrôleur flou ont été effectuées sous une plateforme de Xilinx ISE7.1i utilisant VHDL (**V**ery High Speed Integrated Circuit, **H**ardware **D**escription **L**angage) et un simulateur ModelSim de Mentor Graphic.

Cette thèse traite essentiellement le développement d'un produit numérique testable sur une plateforme travaillant à haut niveau. Le premier chapitre est dédié à une présentation générale de la logique floue et de l'implémentation des systèmes flous.

Le deuxième chapitre donne une description de la testabilité et des différentes techniques de test des systèmes électroniques.

Le troisième chapitre met l'accent sur la conception et la réalisation d'un produit capable d'apporter des décisions dans un délai très court, répondant aux contraintes de vitesse et test sous forme d'un contrôleur flou 2 entrées-1 sortie. On notera que la vitesse maximale du circuit flou à développer devrait répondre aux exigences des applications à traiter moyennant une optimisation de fréquence et un traitement parallèle de différentes tâches.

Les résultats de la synthèse du contrôleur flou et ses tests appropriés sont aussi présentés dans ce chapitre ainsi qu'une explication détaillée de l'architecture utilisée et des différentes méthodes impliquées.

Dans le dernier chapitre, notre contrôleur flou est appliqué à un système de freinage d'un métro (train) pour améliorer et apporter plus de souplesse à l'opération de 'freinage'.

Une conclusion situe notre contribution dans le domaine de l'implantation de contrôleurs flous et présente les perspectives pour une meilleure prise en charge des opérations de test.

CHAPITRE I

Logique et processeurs flous

I.1. Introduction

La théorie des sous-ensembles flous est assez récente, en 1965 quand L.A.ZADEH a proposé pour la première fois le concept de sous-ensembles flous pour pallier de la modélisation aux incertitudes des modèles classiques. Cette théorie permet la graduation dans l'appartenance d'un élément à une classe, c'est à dire qu'un élément peut appartenir plus ou moins fortement à cette classe **[5].**

Le champ d'application de la logique floue est vaste et on le retrouve dans :

- Aide à la décision, au diagnostic (domaine médical, orientation professionnelle…)
- Base de données (objets flous et/ou requêtes floues)
- Reconnaissance de formes.
- Agrégation multicritère et optimisation
- Commande floue de systèmes

I.2. Concept de base de la logique floue

I.2.1. Ensembles flous

Soient U: L'univers du discours.

A: un sous-ensemble de U.

Soit X un espace de points x, un ensemble flou A dans X est un ensemble défini par sa fonction d'appartenance μA(x) qui associe chaque élément x à X un nombre réel appartenant à l'intervalle [0,1] :

$$\mu A(x) : X \rightarrow [0,1]$$

Remarque : un sous ensemble classique est un cas particulier d'un sous ensemble flou.

I.2.2. Fonction d'appartenance

Dans un domaine discret ou continu X = {xi / i=0,1,….n}, un ensemble flou A peut être défini par un ensemble de pairs : degrés d'appartenance/ élément :

- cas discret : $A=\sum \mu A(xi)/ xi, (x \in X)$
- cas continu: $A= [\int \mu A(x)]/x$

Dans le domaine continu, les ensembles flous sont définis analytiquement par leurs fonctions d'appartenance **[6].**

Un exemple d'un sous-ensemble flou est montré Figure I.1

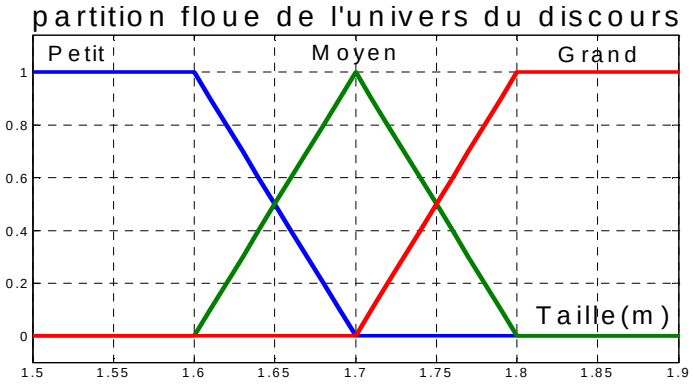


Figure I.1 : *Sous–ensembles flous et fonction d'appartenance*

Si μ(x) =0,30 on dit que **'x'** appartient à l'ensemble flou 'A' avec un degré d'appartenance (valeur de vérité) de 30% degré, Un ensemble flou est totalement déterminé par sa fonction d'appartenance.

I.2.3. Variables linguistiques

Le concept de variable linguistique joue un rôle très important dans le domaine de la logique floue. Une variable linguistique comme son nom le suggère, est une variable dont les valeurs sont des mots ou des phrases au lieu de nombre ; par exemple, «distance » est une variable linguistique et ses valeurs sont : proche, moyenne, éloignée, etc. l'ensemble des valeurs qu'elle peut prendre est appelée l'ensemble de termes. Considérons par exemple la variable distance définie sur l'ensemble des entiers positifs et caractérisée par les ensembles flous : proche, moyen et éloigné **[7]**.

- Logique floue est basée sur des variables floues dites **variables linguistiques** à valeurs linguistiques dans l'univers du discours U.
- Chaque valeur linguistique constitue alors un ensemble flou de l'univers du discours.

Exemple: considérons le cas de la température comme variable linguistique (figure I.2).

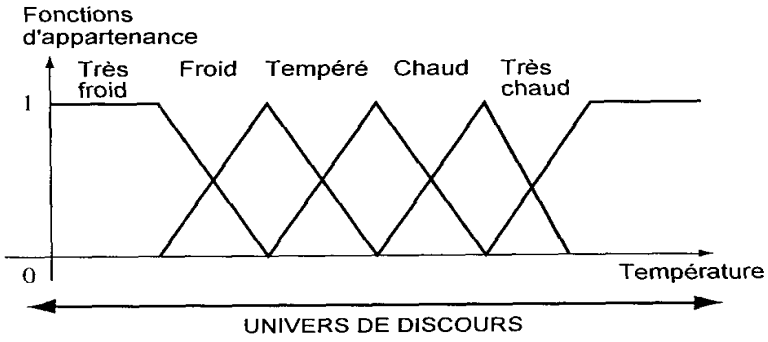


Figure I.2 *: Variables linguistiques (Froid, tempéré …)*

Univers du discours : Gamme de température de 0°C à 200°C.
Variable linguistique : La température.
Valeurs linguistiques : 'Très froid', 'Froid', 'Tempéré', 'Chaud' et 'Très Chaud'.

Exemple : considérons le cas où la vitesse d'un train est une variable (figure 1.3).

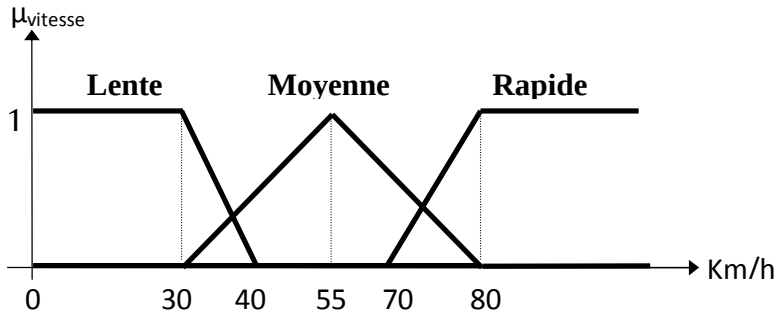


***Figure I.3** : Exemple de fonctions d'appartenance.*

- L'univers de discours : est de 0 à 100 km/h.
- La variable linguistique : La vitesse
- Les valeurs linguistiques : « lente », « moyenne », « rapide ».
  'Lente' a été choisie pour les vitesses inférieures à 30 km/h, 'rapide' pour les vitesses supérieures à 80 km/h et 'moyenne' est une solution intermédiaire.

I.3. commande floue

Un système flou ne traite pas une relation mathématique bien définie, mais utilise des inférences avec plusieurs règles se basant sur des variables linguistiques.

Il apprécie les variables d'entrées de façon approximative (faible, élevée, loin, proche), fait de même pour les variables de sorties (freinage léger ou fort) et édicte un ensemble de règles permettant de déterminer les sorties en fonction des entrées.

I.3.1. Structure du contrôleur flou

La structure conventionnelle d'un contrôleur flou est présentée Figure I.4, elle est composée de quatre blocs distincts dont les définitions sont données ci-dessous **[6] [8]**.

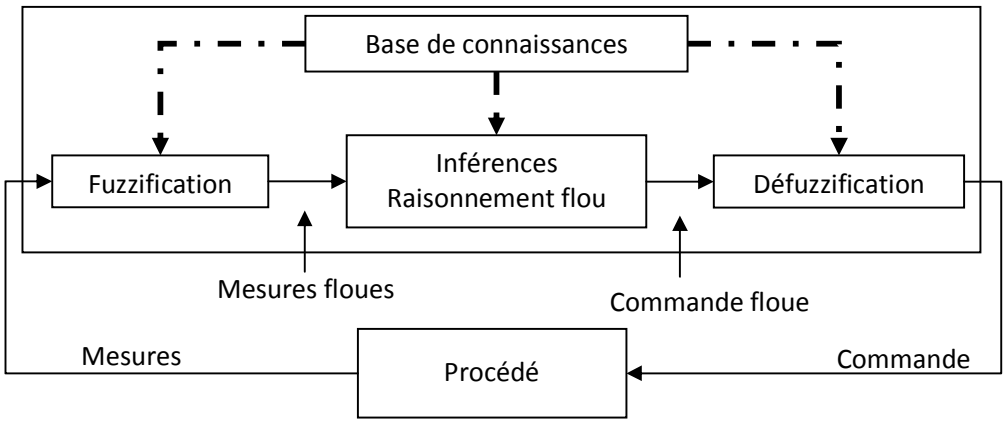


***Figure I.4 :** Structure de base d'un système flou.*

Principe

L'approche des problèmes par la logique floue est différente de celle adoptée, a priori, dans une démarche scientifique, elle est beaucoup plus pragmatique que déterministe. La décision en logique floue est basée sur la notion d'expertise, qui permet de quantifier le flou à partir de connaissances acquises antérieurement.

Il n'est pas nécessaire d'avoir un modèle entrées/sorties d'une voiture pour pouvoir la conduire de manière satisfaisante.

La commande floue a pour but de traiter des problèmes de commande de processus à partir uniquement de connaissances de comportement que les spécialistes du procédé doivent formuler sous forme linguistique (floue). En commande floue, la connaissance des diverses fonctions de transferts n'est pas nécessaire

Exemple - Commande de véhicule autonome (Figure I.5).
- Commande de température d'une serre
- Régulation de niveau complexe
- Commande avec capteur imprécis (Camera…)

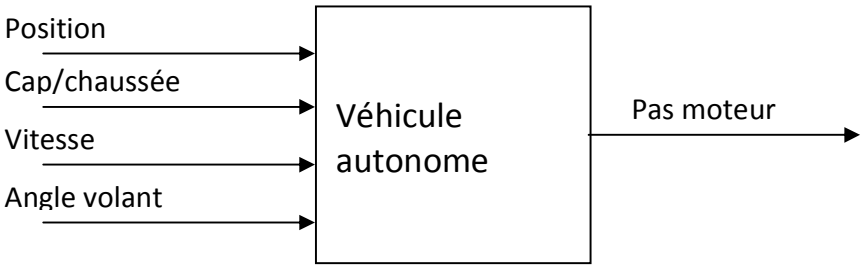


Figure I.5 : *Contrôleur de véhicule autonome.*

I.3.2. Avantages et inconvénients de la commande floue

Les différents avantages et inconvénients sont :

Avantages

1. La théorie est simple et s'applique à des systèmes complexes.
2. Pas de modèles mathématiques requis du procédé à asservir.
3. Robustesse de la commande floue vis à vis des incertitudes.
4. Possibilités de commande auto-adaptative aux variations du procédé.

Inconvénients

1. Technique de réglage essentiellement empirique.
2. Performances dépendent de l'expertise.
3. Il n'existe pas de théorie générale qui caractérise rigoureusement la stabilité, la robustesse (Difficultés de certification dans le transport, espace…).

I.3.3. Approche méthodologique

3 modules pour un régulateur à logique floue (RLF) (figure I.6). :

1. La fuzzification des entrées et sorties.

2. L'inférence floue selon une base de règle.
3. La défuzzification des sorties.

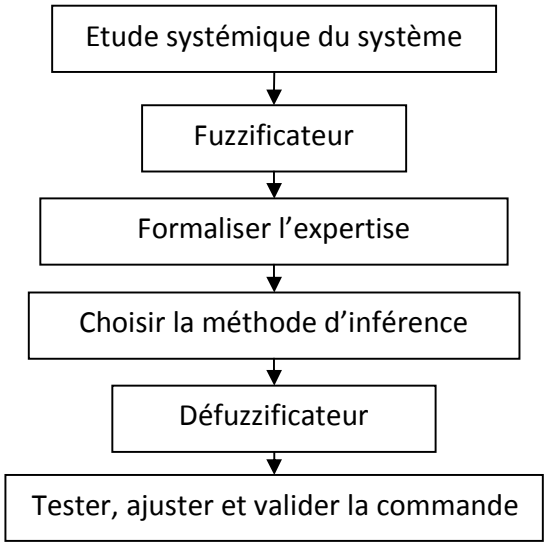


Figure I.6: *Approche méthodologique d'un régulateur à logique floue*

I.3.3.1. La fuzzification

Elle transforme les variables physiques en valeurs floues au travers de variables linguistiques (grand, petit…) **[6]**. La fuzzification est la procédure qui interface l'état non flou du système au domaine flou, dans cette étape la variable non floue x est conditionnée par les fonctions d'appartenance pour obtenir les valeurs floues µA (x) Correspondantes.

Les systèmes à logique floue traitent des variables floues et fournissent des résultats et des variables de sorties elles-mêmes floues, la fuzzification est l'étape qui consiste en la quantification floue des valeurs réelles d'une variable (figure I.7).

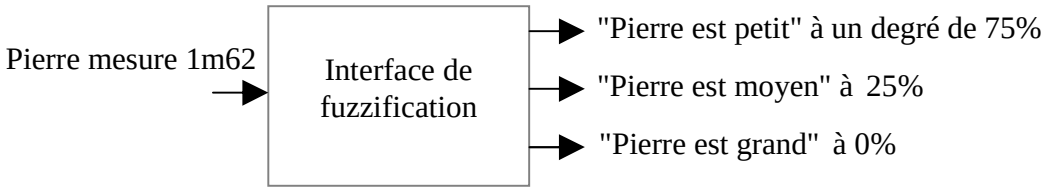


Figure I.7 : *Bloc de fuzzification*

Comment fuzzifier ?

Pour fuzzifier, il faut donner :

1. L'univers du discours : Plage de variations possibles de l'entrée considérée.
2. Une partition en classes floues de cet univers.
3. Les fonctions d'appartenances de chacune de ces classes.

Il faut fuzzifier les entrées et les sorties du processus flou. La fuzzification des variables est une phase délicate du processus mis en oeuvre par la logique floue. Elle est souvent réalisée de manière itérative et requiert de l'expérience.

Tableau I. 1 : *Ensemble flou standard.*

GN	Grand Négatif
MN	Moyen Négatif
PN	Petit Négatif
EZ	Environ Zéro
PP	Petit Positif
MP	Moyen Positif
GP	Grand Positif

Exemple : Un exemple de fuzzification continue est illustré sur la figure I.8, pour une seule variable x, avec des fonctions d'appartenance trapézoïdales. Ainsi x1 = 0,2 devient après fuzzification le sous-ensemble flou x1 = (0 0,8 0,2).

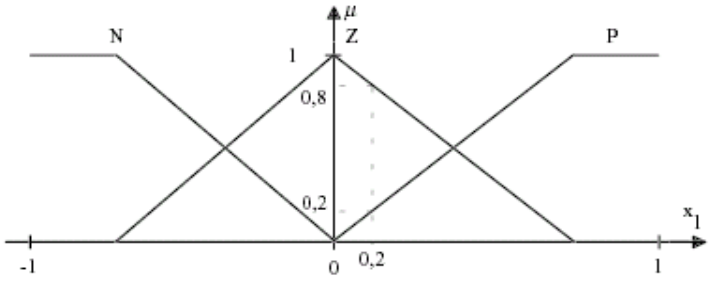


Figure I.8: *Fuzzification continue avec trois fonctions d'appartenance*

Cependant la forme de ces sous-ensembles est définie par :

- Les fonctions d'appartenance MF
- Les positions des fonctions d'appartenance
- Les chevauchements des fonctions d'appartenance

Il convient aussi de respecter certaines règles :

- forme des MF doit être convexe.
- Les sous-ensembles d'entrées (souvent des trapèzes) doivent se chevaucher partiellement afin qu'il n'y ait pas des zones indéterminées, tandis que pour ceux de sorties (souvent des triangles) cette précaution n'est pas nécessaire.

Eviter d'imbriquer plus de deux sous-ensembles, sinon il y'a des problèmes d'instabilité.

I.3.3.2. Base de connaissance de l'expert

La base des connaissances analysée et traitée par l'expert comprend :

- Les données réparties en partitions floues orchestrées par des fonctions d'appartenance.
- La base des règles qui régit la stratégie de commande de l'expert (SI- ALORS …).
- Dans cette partie seront abordés successivement la partition floue, la base des règles et leurs propriétés **[5] [9].**

A. Base de règles

Les systèmes à logique floue utilisent une expertise exprimée sous forme d'une base de règles du type: Si….Alors…

Si (Condition sur entrées) **Alors** (Action sur sortie)

Exemple : Si 'la pression est élevée' **alors** 'ouvrir un peu la vanne'

Ces règles sont énoncées à partir des connaissances du procédé par l'expert.

Définition

Si l'on considère **n** univers de discours U_i pour les entrées du RLF et que chaque univers U_i est partitionné en m_i classes floues, alors le nombre maximal de règles est :

$$r_{max} = \prod_{i=1}^{n} m_i$$

Très vite, le nombre de règles peut devenir important.

On peut ne pas considérer certaines configurations de sous-ensembles flous impossibles à obtenir par le processus.

Exemple: Commande automatique de freinage

La règle (**Si** Vitesse importante **ET** Distance à l'obstacle est nulle) n'est pas à considérer. L'augmentation de la sensibilité obtenue par une partition plus fine des entrées aboutit à un accroissement important du nombre de règles à définir par l'expert.

Lorsque toutes les règles sont du type: Si ( ) ET ( ) ET ( ) …Alors ( ), la base de règles s'écrit sous forme d'une matrice d'inférence.

Tableau I. 2: *Exemple de matrice d'inférence (5x5)*

Sortie		**Entrée e_1**				
		GN	**MN**	**EZ**	**MP**	**GP**
e_2	**GN**			*GP*	*MP*	
	MN			*MP*	*EZ*	*MN*
	EZ	*GP*	*MP*	*EZ*	*MN*	*GN*
	MP	*MP*	*EZ*	*M*		
	GP		*MN*	*GN*		

B. Inférence floue

Définition

C'est le mécanisme qui applique les opérations d'inférence à partir des règles, le moteur d'inférence est le véritable cerveau du contrôleur flou, l'opération consiste à admettre une

proposition en vertu de sa liaison avec d'autres propositions tenues pour vraies à partir de la base des règles fournie par l'expert et des sous-ensembles flous (figure I.9).

Exemple :

Si Temps est beau **ET** Moment est début matinée **Alors** Moral est haut.

Principe du raisonnement approximatif

Plus la condition sur les entrées est vraie plus l'action préconisée pour les sorties doit être respectée.

SI la température est très basse **ALORS** Chauffer fort

La conclusion d'une règle floue est l'appartenance d'une variable floue de sortie « Chauffer » à une classe floue « fort ».

Cette appartenance dépend de :

1. La classe floue de sortie considérée.
2. Le degré de validité de la prémisse (x0).
3. La méthode d'implication choisie.

Méthodes d'implication

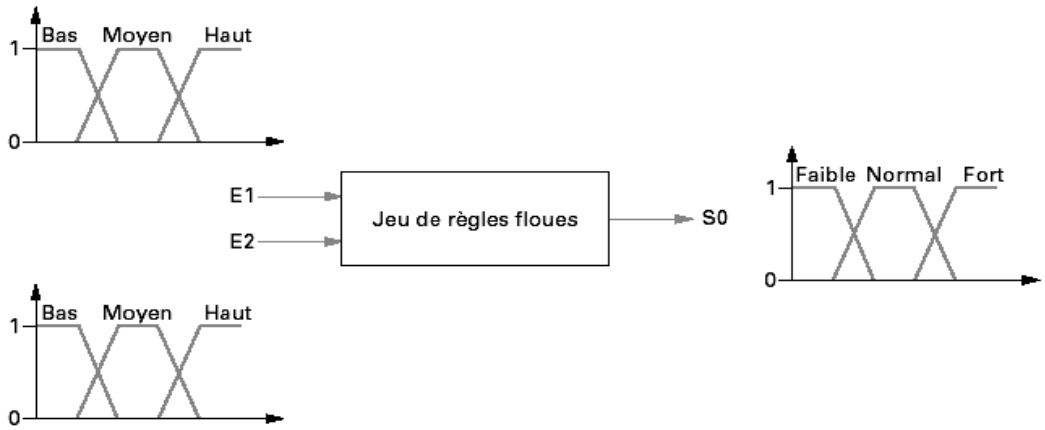


Figure I.9: *Inférence floue*

Activation des règles

R1 : Si (X1 est A11) **et** (X2 est A12) **alors** Y est B1
R2 : Si (X1 est A21) **et** (X2 est A22) **alors** Y est B2
R3 : Si (X1 est A31) **et** (X2 est A32) **et** (X2 est A33) **alors** Y est B3
…….

Une règle est activée dès qu'elle a une prémisse ayant une valeur de vérité non nulle.

Plusieurs règles peuvent être activées simultanément et préconisant des actions avec différents degrés de validités; ces actions peuvent être contradictoires.

Il convient d'agréger les règles pour fournir une appartenance de la variable floue de sortie à une classe floue consolidée

Composition de règles

On considère que les règles sont liées par un opérateur OU.

$$\mu_B(y) = MAX\left[\mu_{B_i}(y)\right] \qquad i \in \{indices\ des\ règles\ activées\}$$

On considère un moteur d'inférence (figure I.10) à 4 règles qui fournit pour sa sortie la tension S1, les résultats suivants :

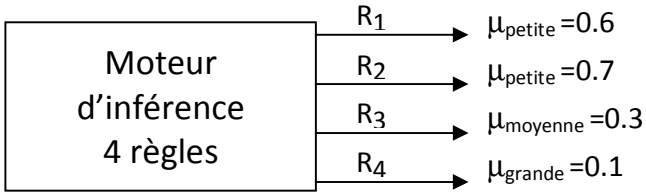


Figure I.10 : *Moteur d'inférence*

C. Méthodes d'inférences

Parmi les principales méthodes d'inférence, on peut citer :

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

I.3.3.3. La défuzzification

La défuzzification transforme les résultats du mécanisme d'inférence en grandeurs physiques, l'interface de défuzzification a pour objectif de transformer la sortie en valeurs non floues permettant ainsi la commande effective du système **[8] [10].** Plusieurs méthodes peuvent être envisagées dont la méthode de la hauteur et ses variantes ou celle de barycentre qui sont couramment les plus utilisées.

Les trois principales méthodes sont:

A. Centre de gravité de la surface (COG)

Toutes les valeurs de l'univers du discours de la sortie interviennent dans le calcul, où ils sont pondérés par leur degré de vraisemblance, 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 Ucg de la courbe présentée sur la figure I.11.

L'abscisse **U** du centre de gravité peut être calculée par la fonction $\frac{\sum_{i=1}^{m} u_i \mu_i(u_i)}{\sum_{i=1}^{m} \mu_i(u_i)}$ dans le cas discret (Méthode de Sugeno). Le calcul du centre de gravité permet bien d'obtenir une seule valeur pour la grandeur de sortie mais ça nécessite un calcul important.

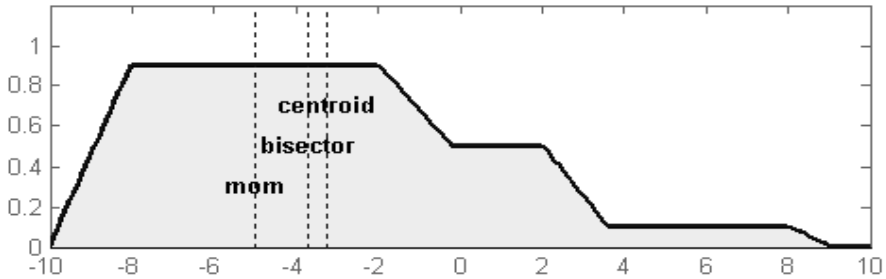


Figure I.11 : *Méthodes de Défuzzification*

B. Moyenne des maxima

C'est la moyenne des valeurs de sorties les plus préconisées, cette approche est intuitive car elle choisit le point avec la plus forte possibilité, c.-à-d. fonction d'appartenance maximale (figure I.11). 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.

C. Bissection de la surface.

Le résultat est donné par l'abscisse qui coupe la surface en 2 parties égales (figure I.11).

En commande floue, la méthode de défuzzification par COG est généralement utilisée, elle prend en compte l'influence de l'ensemble des valeurs données par la solution floue. La défuzzification MM est plutôt utilisée lorsqu'il s'agit de discriminer une valeur de sortie (Ex: reconnaissance de formes).

I.3.4. Surface de décision floue [5]

Cette forme de nonlinéarité en trois dimensions, mise en œuvre par le contrôleur flou, est appelée parfois "la Surface de Contrôle," elle est affectée par tous les paramètres principaux du contrôleur flou.

Exemple *:* Système de notation flou (figure I.12).
On choisit :

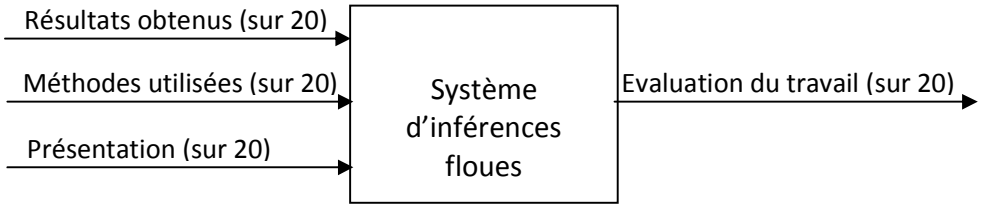


Figure I.12: *Exemple d'un système de notation floue.*

Exemple de base de règles

If (Résultat est excellent) ***then*** (Evaluation est excellente)
If (Résultat est moyen) ***then*** (Evaluation est moyenne)
If (Résultat est médiocre) ***then*** (Evaluation est médiocre)
If (Résultat est moyen) ***and*** (Méthode est médiocre) ***then*** (Evaluation est mauvaise)

Considérons, par exemple, la surface de contrôle pour ce système de notation flou qui a abouti à la réponse montrée dans la Figure I.13

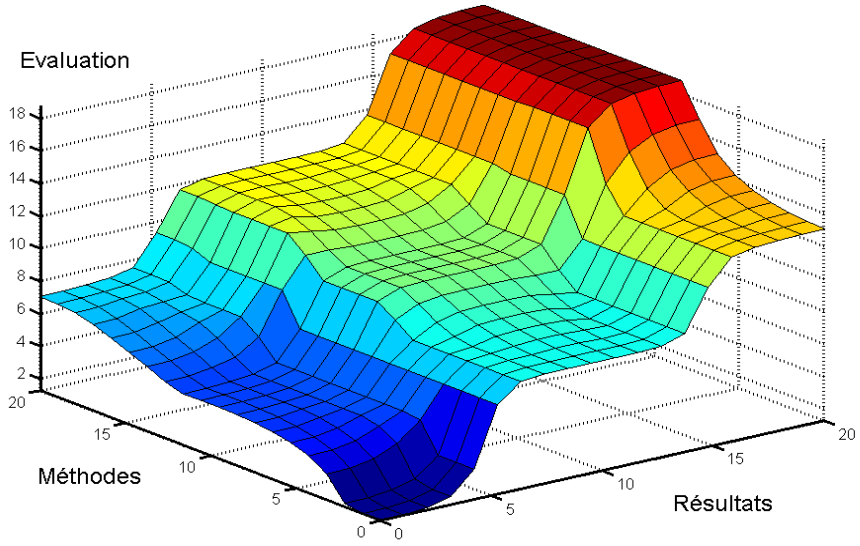


Figure I.13 : *Surface de décision floue*

Caractéristique de la surface floue

- Décision selon un système d'inférence flou.
- Non linéaire (figure I.13).
- Plus proche du comportement humain.

I.4. Implémentation des Contrôleurs Flous

I.4.1. Introduction

On envisage les descriptions générales et les recommandations ainsi que les instructions sur la façon dont on peut construire de vrais contrôleurs flous, d'ailleurs le processus de conception ainsi que l'implémentation des contrôleurs flous constitue la partie la plus passionnante du développement **[11].**

La manière la plus simple et la plus habituelle pour l'implémentation d'un contrôleur flou est de le réaliser sous forme d'un programme exécuté par un processeur. Cependant, un grand nombre d'applications de contrôle flou exigent le traitement de données en temps réel pour poursuivre l'évolution du dispositif externe. Par exemple, le contrôle de la vitesse d'une automobile ou la commande d'un moteur électrique sont caractérisés par leurs très grandes vitesses, de ce fait l'implémentation du contrôleur flou sous forme logiciel ne peut pas être considérée comme une solution appropriée à ce type d'application.

Les conditions d'implémentation matérielle sont :

- **Rapidité** : Exécution à grande vitesse (Réponse du contrôleur flou)
- **Faible complexité** : La complexité faible signifie que les algorithmes pour le traitement flou, la fuzzification et le défuzzification doivent être très simples et exigent l'allocation d'espace mémoire réduit.

- **Flexibilité élevée** : La flexibilité signifie la capacité du matériel à être employé avec succès dans différentes applications et configurations.

Ces conditions s'opposent l'une contre l'autre **[12]**. Ainsi il n'est pas facile de choisir la bonne manière, il faut faire un compromis entre ces facteurs, tels que le coût de fabrication (très important pour les contrôleurs flous de consommation) ou bien le coût de conception (important dans la recherche et le développement).

Ces dernières années, il y a eu un intérêt croissant pour le développement d'un matériel efficace, pour contrôleur flou, capable de faire face aux conditions des applications en temps réel. Togai et Watanabe ont développé à leur tour le premier processeur flou en 1985 **[13]**, plus tard Yamakawa a développé son processeur flou en utilisant pour la première fois des techniques analogiques.

Depuis, plusieurs processeurs utilisant différentes techniques d'implantation ont été proposées **[14]**. Les classes d'implémentation matérielle sont résumées au tableau I.3 :

Tableau I. 3 : *Avantages et inconvénients des différentes techniques*

CLASSE D'IMPLEMENTATION MATERIELLE	**AVANTAGE**	**INCONVENIENT**
Processeur numérique.	– Flexibilité concernant des outils Hardware et software.	– Basse performance.
Processeur numérique spécialisé.	– Croissance des performances.	– Coût élevé. – Complexité élevée. – Manque de flexibilité
Processeur analogique.	– Coût Bas. – Consommation d'énergie Basse. (haute performance)	– Basse précision. – Manque de flexibilité

On peut voir que n'importe quel type de processeur flou a ses côtés positifs et négatifs, c'est pourquoi nous discuterons brièvement tous**.**

I.4.2. Processeur Flou Analogique

Au cours de ces dernières années, les circuits analogiques ont attiré une attention particulière Comme étant une bonne solution pour l'implémentation des contrôleurs flous. Pour des applications spécifiques dans lesquelles le nombre d'entrées et de sorties est faible, il peut s'avérer intéressant d'effectuer le traitement de façon analogique. En effet, cette façon de procéder permet une solution qui nécessite un nombre de transistors nettement moins élevé que la solution numérique. Pour de petits systèmes quelques milliers de transistors suffisent.

L'un des inconvénients de cette méthode est son manque de flexibilité (voir Tableau I.3). Elle convient aux applications avec des règles d'inférence fixes, et le processeur ne peut s'adapter à son environnement par des méthodes d'apprentissage.

Cette implémentation est caractérisée par la plus grande vitesse de fonctionnement ainsi que la petite consommation d'énergie. Son efficacité fonctionnelle est beaucoup plus importantes que celle du numérique en raison de la possibilité d'exploitation des petites gammes d'appareils analogiques pour une grande variété de traitement linéaire et non linéaire, mais généralement le traitement flou ne nécessite pas une grande précision, d'où l'utilisation des circuits analogiques reste aussi un bon choix.

L'ensemble du système flou est divisé en deux parties en fonction de leurs activités, qui sont : la puce «rule» pour les inférences floues (FP9000), et la puce «defuzzifier **»** pour la défuzzification (FP9001), ces deux divisions fonctionnelles facilitent la configuration du système **[15] [16].**

Le moteur d'inférence flou est implémenté selon une architecture parallèle où les conséquents de toutes les règles sont définis et programmés en interne, agrégées par un "ou" analogique et combinées pour produire une valeur de sortie à défuzzifier. Le traitement interne à la fois dans FP9000 et FP9001 est effectué dans un mode analogique, mais une interface numérique a été introduite dans la dernière version afin de définir, modifier (écriture) et lire les paramètres de chaque règle floue rapidement.

La puce «Rule» se compose d'un bloc d'antécédents, d'un bloc de conséquent et d'un bloc mémoire de règles pour stocker l'ensemble des règles floues, jusqu'à quatre règles floues peuvent être stockées et traitées simultanément.

Pour décrire les variables d'entrée floues, uniquement les fonctions S et Z sont permises comme fonctions d'appartenance (figure I.14).

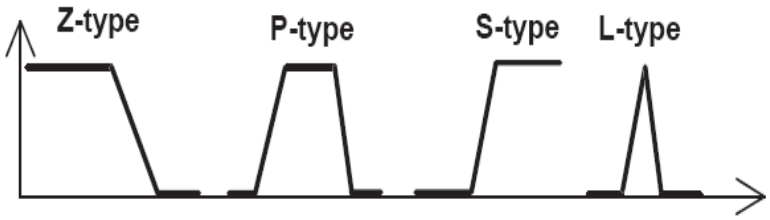


Figure I.14: *Formes standards des Fonctions d'appartenances.*

Un codage de trois bits peut fournir sept combinaisons possibles (le code 000 n'est pas assigné) NL, NM, NS, Z, PS, PM et PL pour la définition des valeurs des conséquents dans l'univers du discours. Le bloc conséquent effectue une opération maximum (t-conorm). La mémoire des règles comporte une interface numérique pour la définition et l'application des règles floues, c'est une mémoire à deux étages se composant de 24 registres de 8 bits.

Le bloc de defuzzification accepte des sous-ensemble flous de forme “singletons” , un de chaque conséquent. Ces singletons ont pour but d’augmenter la vitesse de calcule et de diminuer la complexité des circuits. Ils calculent la valeur de sortie “crisp output “ par la méthode du centre de gravité. Le contrôleur implémenté avec 13 règles floues est conçu avec ses circuits analogiques fonctionnant en mode de tension. Le circuit final est enfin implémenté en technologie CMOS de 2.4 microns (figure I.15). **[12].**

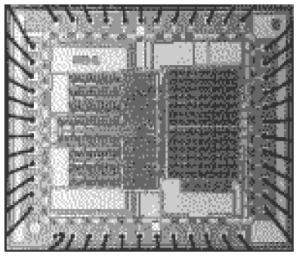


Figure I.15: *Contrôleur flou reconfigurable.*

I.4.3. Processeur Numérique

De nos jours, la plupart des contrôleurs flous sont implémentés comme programmes sous forme Software sur les processeurs et les microprocesseurs **[16][17].**

Les processeurs numériques sont aujourd’hui les plus utilisés, il remplace facilement ceux conçu sous forme software où le besoin d’un traitement rapide est nécessaire. L’exemple d'un tel processeur est le FC110-DFP (Digital Fuzzy Processor) de Togai, le FC110 est un processeur flou VLSI (Very Large Scale Integration) qui a été fabriqué par Togai InfraLogic en technologie CMOS pour les applications temps réel. L’architecture du FC110-DFP ressemble à celle des micro-contrôleurs, ce qui lui permet d’être programmé de la même façon. Cela signifie que sa programmation repose sur un jeu d’instructions en assembleur, la partie qui avantage ce processeur par apport aux autres processeurs est la puissance de son unité arithmétique et logique. Le schéma bloc du FC110-DFP est donné à la figure (I.16) **[12] [18]**.

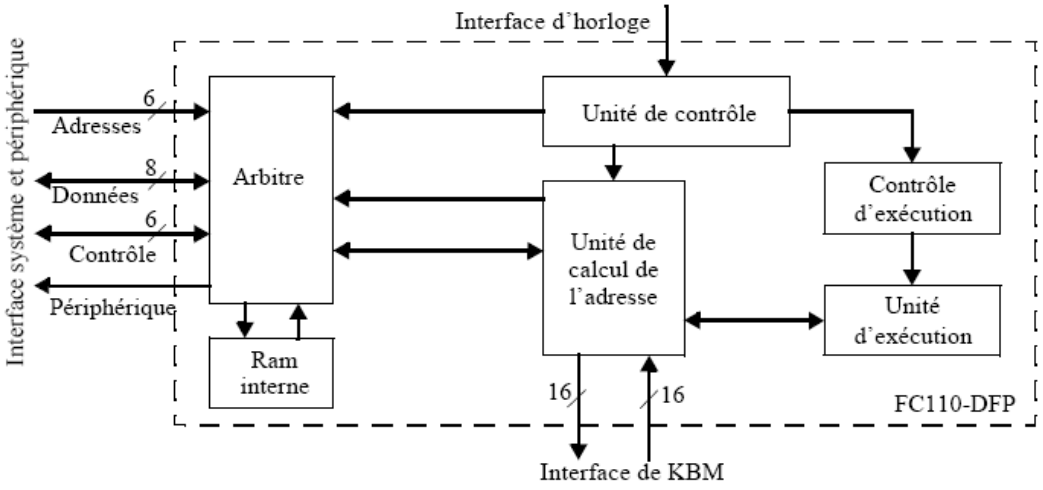


Figure I.16 : *Schéma bloc de FC110-DFP.*

I.4.4. Processeur Numérique Spécialisé (Type Co- Processeur)

La recherche sur l'équilibre entre les performances et le coût a mené a développer différentes architectures avec un appui spécifique, et plusieurs coprocesseurs d'accélération dédiés à la commande floue ont été proposés. Le matériel dédié peut être considéré comme une meilleure solution en termes de performance, mais il peut seulement couvrir une partie limitée des applications. Malgré la basse flexibilité, le matériel spécialisé peut représenter un choix efficace, en particulier pour les applications qui exigent un grand nombre de règles. Le matériel flou spécifique nous permet dans beaucoup de cas d'atteindre un meilleur rapport coût-performances en raison de l'exploitation du parallélisme dans le traitement flou et l'introduction des unités spéciales **[18] [19].**

Un processeur flou spécialisé est habituellement consédéré comme un Coprocesseur, L'exemple d'un tel processeur est le VY86C570, ce dernier est un co-processeur flou fabriqué par Togai InfraLogic en technologie "Advanced CMOS". Il est basé sur un coeur 12 bits de technologie FCA (Fuzzy Computational Acceleration). Le nombre des entrées, le nombre des sorties et le nombre des règles floues (jusqu'à 200 règles) sont limités par la taille de 4 Kmots de 12 bits d'une mémoire interne RB (Rule Base). La mémoire RB est accessible par le processeur hôte pour le chargement du projet flou. Pour des projets flous plus importants (plus de 200 règles), une mémoire externe (de taille 64 K mots de 12 bits maximum) peut être ajoutée. Cela permet d'augmenter le nombre des entrées et sorties jusqu'à 1024 et le nombre de règles à plus de 1000. Le VY86C570 fonctionne jusqu'à 20 Mhz ce qui permet d'effectuer l'évaluation d'un nombre important de règles pouvant atteindre 800,000 règles par seconde.

Le VY86C570 est capable d'exécuter de simples calculs flous aux très complexes systèmes avec des vitesses très élevées, La figure I.17 donne le schéma bloc du VY86C570.

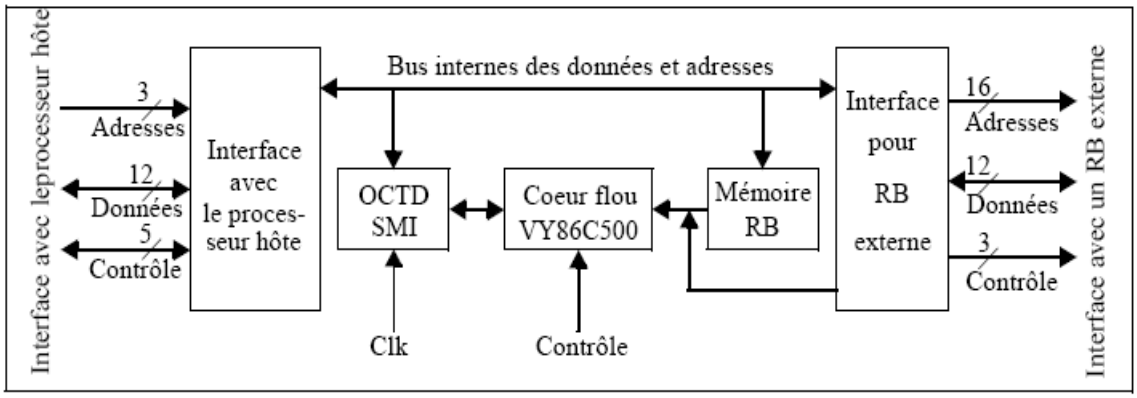


Figure I.17: *schéma bloc du VY86C570.*

Exemple :

Le tableau I.4 présente quelques caractéristiques réelles des deux processeurs flous, un d'une conception japonaise qui est le FP-3000 (processeur flou de nouvelle génération de grande vitesse,

appliqué dans différents produits d'Omron), et un d'une conception européenne qui est le WARP (Weight Associative Rule Processor) par SGS-Thomson.

Tableau I.4 : *Caractéristique principales de certains processeurs flous.*

Caractéristiques principales	***FP-3000, Omron***	***WARP, SGS-Thompson***
- Nb de règles traitant les Entrées	8	16
- Nb de règles traitant les Sorties	4	16
- Nb de Fonctions d'appartenance possible pour chaque entrée.	7	16
- Nb de Fonctions d'appartenance possible pour chaque sortie	7	128
- Nb de Forme de Fonction d'appartenance.	4 Formes	Toutes les formes
- Nb de règles	- Single mode : 29 - Expanded mode: 128/groupe avec 3 groupes.	Jusqu'à 256
- Temps de traitement	- 20 Règles avec 5 E et 2 S avec la méthode du défuzzification du centre de gravité, le temps est évalué à 650 µs	32 Règles avec 5 E et 1 S, le temps est évalué à 1.85 µs (1.5 MFLIPS)
- Résolution de données	- Unsigned /12 bits	8 bits

I.5. Performances des contrôleurs flous

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

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

I.6. Choix d'implémentation

L'implémentation matérielle de la 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 spéciales concernant la logique floue.
- Les composants flous de type ASIC (Application Specific IC) capables de fonctionner sans un processeur hôte.

- Les processeurs ou coprocesseurs flous numériques.
- Les processeurs ou coprocesseurs flous analogiques.

I.7. Contraintes d'implémentation

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 repose généralement sur un compromis entre les paramètres précédents. Par exemple, pour une application floue ne nécessitant pas une fréquence élevée de commande, une solution basée sur un micro-contrôleur classique est idéale vu le coût bas de sa réalisation. Par contre, si cette application nécessite une fréquence élevée de commande, cette solution n'est plus valable. Dans ce cas, c'est la simplicité de l'application floue qui joue le rôle important. Si l'application est simple, une solution de type «tables de transcodage» est possible, sinon l'utilisation d'un processeur flou dédié ou un composant ASIC devient inévitable. C'est alors le coût et la possibilité de la réalisation qui vont guider le choix final.

I.8.Conclusion

Dans ce chapitre, nous avons exploré les solutions matérielles possibles pour réaliser un composant flou.

Selon les techniques de conception, les contrôleurs peuvent être classés en deux catégories : numérique et analogique, l'approche numérique fournit une grande facilité de conception où les processeurs flous numériques offrent une plus grande flexibilité de configuration et de programmation. Leur coût est encore assez élevé et leur rapidité est encore limitée par la défuzzification. Les processeurs flous analogiques exploitent directement les propriétés des éléments de base de l'électronique. Ils sont de ce fait plus compacts et très rapides, mais manquent de flexibilité (inconvénient majeur).

CHAPITRE II

Test des systèmes électroniques

II.1. Introduction

II.1.1. Définitions

Le test est le processus qui permet de déterminer si le circuit est correct ou défectueux. Un circuit peut être défectueux parce que soit il ne répond pas aux spécifications du cahier des charges suite aux erreurs de conception (test fonctionnel), soit il ne correspond pas au circuit conçu donc il y'a erreur de fabrication dû à un défaut physique et on parlera alors de test structurel **[20]**.

Test Fonctionnel : Son but est de déterminer si le circuit fabriqué réalise bien les fonctions définies dans le cahier des charges. Ce test permet de vérifier le comportement du circuit en présence d'erreur afin d'en modifier la conception du circuit.

Test Structurel : Le but est de déterminer si le circuit fabriqué ne présente pas de défauts physiques **[21]**. On distingue trois types de ce test :

- Test de Continuité : Pour détecter les défauts grossiers liés à la continuité.
- Test Logique : Test numérique des fonctions du circuit.
  (Ce test exploite les modèles de faute).
- Test paramétrique (Caractérisation) : il est utile pour la détermination de paramètres électriques et permet de déterminer les limites de fonctionnement du circuit. Le caractère analogique de ce test est délicat, lent et couteux.

II.1.2. Les différentes phases de test :

Les différentes phases de test sont données ci-dessous

- Test sur Wafer (Défauts dus au processus de fabrication).
- Test du circuit encapsulé (Défaut dus aux processus d'encapsulation).
- Test du circuit dans la carte.
- Test de la carte dans le système.

II.1.3. Coût d'un circuit défectueux [22]

La figure II.1 montre le coût des opérations de test durant la vie d'un circuit :

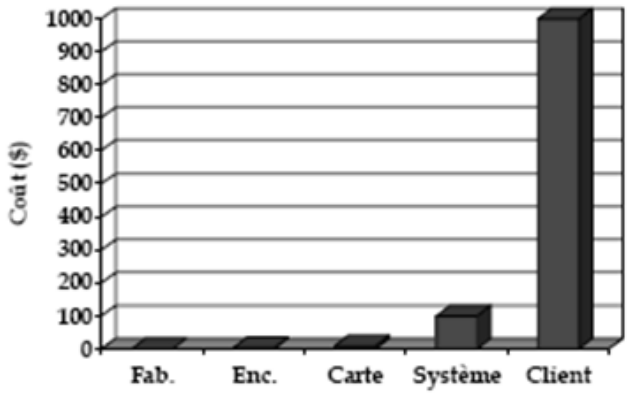


Figure II.1: *Coût d'un circuit défectueux*

II.1.4. Choix des Vecteurs de Test

La production des vecteurs de test est une étape importante du processus du test. Les techniques de génération sont :

- Vecteurs Exhaustifs (Tous les tests possibles).
- Vecteurs Aléatoires.
- Vecteurs Fonctionnels (Les mêmes vecteurs appliqués dans la phase de conception).
- Vecteurs de Test dépendant d'un modèle de fautes.

II.1.5. Techniques de Test Utilisées [23]

Elles se divisent en plusieurs groupes :

- Utilisation de la simulation de fautes avec les vecteurs de test fonctionnels
- Utilisation d'un générateur automatique de vecteurs de test
- Utilisation des techniques de conception en vue de test DFT, Afin d'améliorer la testabilité du circuit.

La technique de test dépend essentiellement de la phase d'application du test proprement dit.

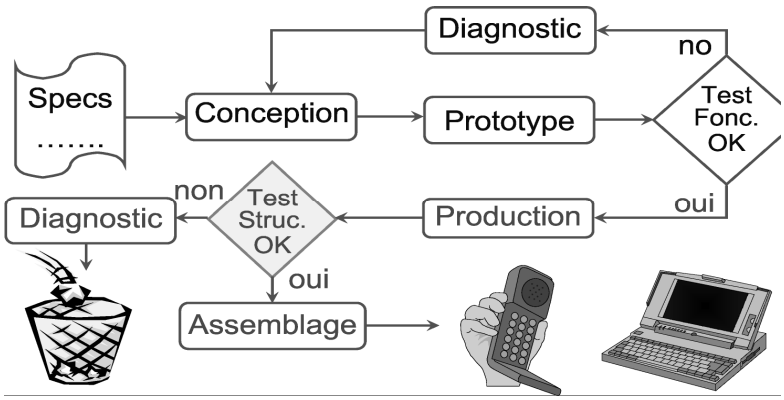


Figure II.2: *Fabrication des Circuits.*

II.2. Test Industriel

II.2.1. Définition

Ce test consiste en l'application de vecteurs de test sur les entrées d'un circuit sous test (DUT) et d'en analyser les sorties, afin de détecter les défauts éventuels de fabrication. Les types de test industriel sont:

- Test Fonctionnel: ce test vérifie les fonctionnalités logiques du circuit.
- Test de production (Fabrication ou Structurel): test des défauts physiques du circuit.
- Test de caractérisation: pour déterminer les limites fonctionnelles du circuit, il doit être effectué sur plusieurs circuits et plusieurs lots pour considérer les résultats statiquement valides.

II.2.2. Phases de test

Trois principales phases sont généralement considérées :

Phase de conception

Cette étape comprend :

1. Modélisation.
2. Simulation.
3. Synthèse.
4. Analyse des délais.
5. Placement routage.
6. Génération de test de production (DFT).

Phase de prototypage

Pendant cette phase, les tests ciblés sont :

1. Test de spécification (Test fonctionnel).
2. Test de caractérisation.
3. Test Pire-Cas "Worst-case".

Phase de production :

Les défauts de fabrication sont détectés pendant cette phase en effectuant les tests suivants :

1. Test de continuité.
2. Test structurel.
3. Test de caractérisation (Des échantillons).
4. Test sur Wafer.
5. Test après encapsulation.

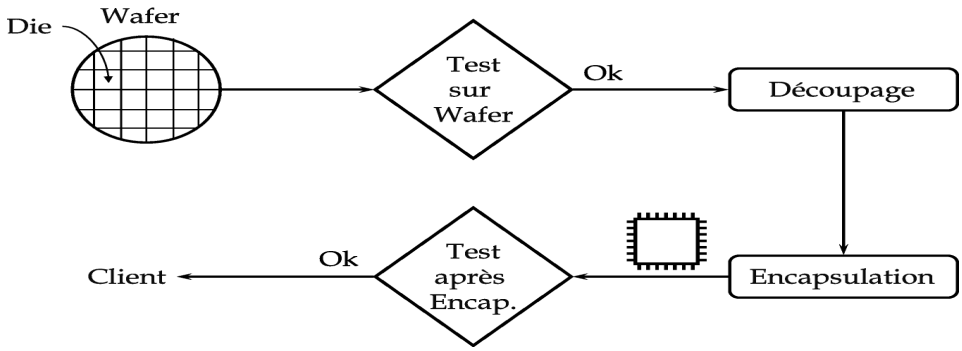


Figure II.3*: Test de production.*

II.2.3. Plan de Test

Le plan de test est la liste des tests et des procédures à effectuer pour vérifier la qualité du circuit sous le Test (DUT).

II.2.4. Equipement de test [20]

Les éléments à exploiter pour appliquer un test sont :

- **Testeur:** C'est la machine qui permet de générer les vecteurs et de comparer les résultats de test.
- **Wafer Prober:** C'est la machine robotisée qui permet de manipuler et de placer les "dies" sous des sondes connectées au testeur.
- **Handler:** C'est la machine robotisée qui manipule les circuits encapsulés pour le testeur.
- **PIB (Probe Interface Board):** C'est la carte interface entre le testeur et les sondes.
- **DIB (Device Interface Board):** C'est la carte interface entre le testeur et le circuit intégré encapsulé.

II.2.5. Analyse des données de test

C'est le processus par lequel on examine le résultat de test. C'est une opération fondamentale car sans l'analyse des données relatives au test il serait difficile de considérer toute évolution

II.3. Modélisation des défauts physiques

II.3.1. Introduction

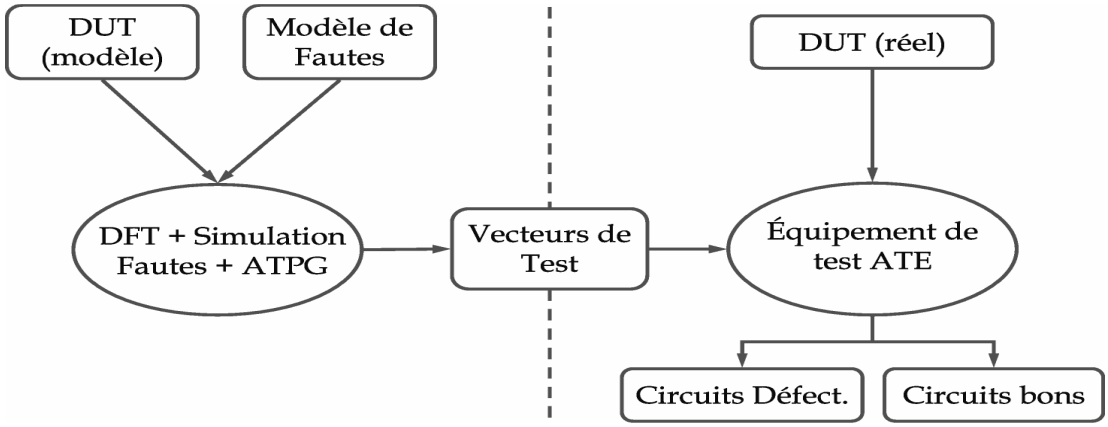


Figure II.4: *Test des Circuits.*

Pourquoi les modèles de fautes :

La modélisation des défauts physiques en états logiques sert à réduire la complexité du test. Modéliser les défauts physiques au niveau comportemental permettra aussi d'effectuer des tests très tôt dans la phase de conception d'un circuit (réduire le Time To Market) (Figure II.4). Il y'a une indépendance vis à vis des technologies utilisées **[1][2].**

Définitions

Défaut : pannes physiques qui affectent le circuit

Fautes (Pannes) : représentation des défauts au niveau logique selon un modèle de fautes.

Modèle de Fautes : Modèles utilisés pour représenter les défauts physiques au niveau logique dans le but de faire des simulations.

II.3.2. Types de défauts Physiques

Il existe plusieurs types de défauts :

- Défauts catastrophiques : (CC, CO).
- Défauts paramétriques : Déviation d'un paramètre en d'hors des spécifications.
- Défauts permanents : défauts toujours présents.
- Défauts temporaires : défauts présents sous certaines conditions.

II.3.3. Modèles de fautes utilisés

Les modèles les plus utilisés sont :

- Modèle de collages "Stuck-at".
- Modèle de collages au niveau de transistor.
- Modèle de court-circuit "Bridging Faults".
- Modèle de circuit-Ouvert "Stuck open Faults".
- Modèle des défauts de délais.
- Modèle de fautes spécifique aux mémoires.
- Modèle de fautes pour les circuits configurables.

II.3.4. Les fautes de collages multiples

C'est le même modèle que pour les collages, deux ou plusieurs signaux peuvent être collés au même temps, la technique dite IFA «Inductive faulte Analysis» ou « extraction Inductive des fautes » est la technique qui permet d'extraire les fautes les plus probables d'un circuit à partir de son Layout.

II.4. Simulation de Fautes

II.4.1. Description [2]

La simulation de fautes permet de modéliser et de simuler le circuit avec les fautes (Circuit fautif) dans le but de déterminer le taux de couverture (TC), la génération du dictionnaire des fautes et de réduire le nombre de vecteurs de test nécessaire pour la génération automatique du test.

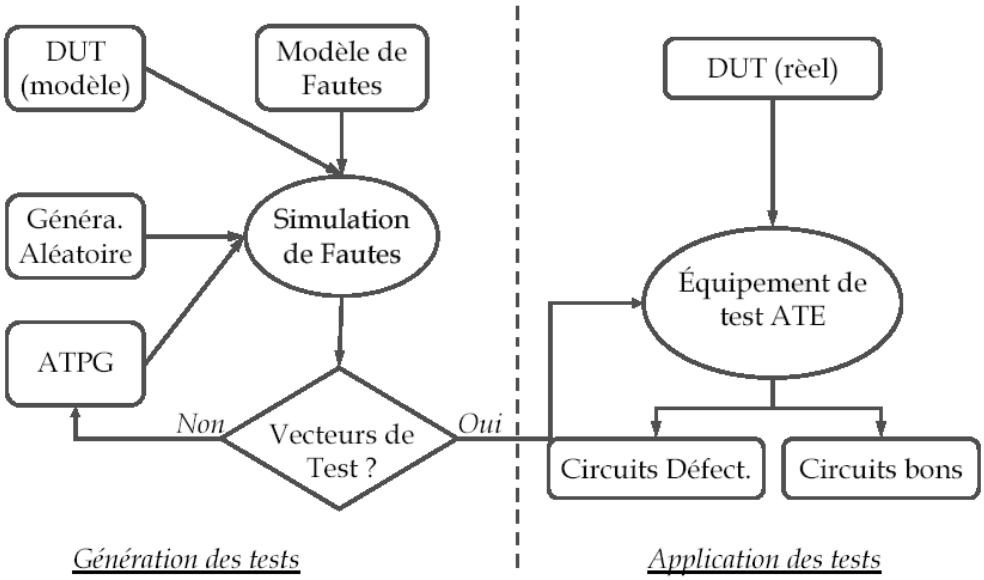


Figure II.5: *Test des circuits - Simulation de fautes.*

II.4.2. Simulation logique

C'est le programme qui permet de modéliser et de simuler le circuit à tester (DUT : Divice Under Test), il permet donc de vérifier les spécifications du circuit, et d'évaluer d'autres alternatives de conception, et il permet aussi de réduire le temps de conception (figure II.5).

a. Niveaux de simulation

Plusieurs niveaux de simulation sont possibles, le niveau comportemental ou fonctionnel et le niveau structurel sont les plus exploités.

b. Modélisation fonctionnelle des portes logiques

Cette opération utilise les tables de vérité, les BDD (Binary Decision Diagram), la table d'états et les fonctions.

II.4.3. Technique de Simulation Logique

Plusieurs techniques sont développées. Les plus utilisées sont :

- Simulation compilée
- Simulation a événements dirigés

II.4.4. Algorithmes de simulation de fautes

Ces algorithmes permettent de déterminer toutes les fautes détectables par chaque vecteur de test. [1]

II.5. Génération automatique des vecteurs de test « ATPG - Automatic Test Pattern Generation »

II.5.1. Description

C'est le programme qui permet de déterminer des vecteurs de test spécifiques pour détecter des fautes données, donc une qualité de vecteur de test améliorée et un coût de test réduit (voir figure II.5) [1][20].

II.5.2. Méthodes de génération

Plusieurs méthodes de génération de vecteurs de test peuvent être citées :

1. Génération manuelle: Les vecteurs de test sont écrits manuellement (expert).
2. Génération pseudo-aléatoire: les vecteurs de test sont choisis aléatoirement.
3. Génération exhaustive: Utilisation de tous les vecteurs de test possibles.
4. Génération déterministe (automatique): pour chaque faute non détectée on génère un vecteur de test qui la détecte.

On calcule le taux de couverture avec un simulateur de fautes. L'avantage c'est que le taux de couverture est maximum avec un coût minimum, par contre son inconvénient réside dans la complexité de son algorithme.

II.5.3. Détection de fautes

Trouver un vecteur de test T qui détecte la faute F revient à trouver le vecteur de test T qui:

1. Active la faute F à partir des entrées primaires (PIs).
2. Propage la faute F jusqu'aux sorties primaires (POs).

a. Activation des fautes

C'est le processus qui permet de ramener le circuit dans l'état où la faute produit une erreur dans le circuit, pour le modèle des collages, la valeur opposée du collage sur le signal en question est considérée.

b. Propagation des fautes

C'est le processus qui permet de propager la faute à travers toutes les portes du circuit jusqu'aux sorties primaires.

Détection de fautes = activation + propagation

II.5.4. Algorithme pour la génération déterministe [24]

Il existe plusieurs algorithmes d'ATPG :

- *Algorithme D.*
- *Algorithme PODEM "**P**ath **O**riented **DE**cision **M**aking".*
- *Algorithme FAN.*

II.5.5. Mesure de testabilité

Les coefficients qu'on affecte à un nœud pour quantifier la difficulté à le tester sont de trois types :

a) Contrôlabilité : Quantifie la difficulté à contrôler le signal.
b) Observabilité : Quantifie la difficulté à propager le signal.
c) Testabilité : Quantifie la difficulté à tester le signal.

II.6. Conception en Vue de Test (DFT)

II.6.1. Descriptions

C'est le processus qui intègre la testabilité du circuit dans la phase de conception, par l'ajout de matériel et la modification de la conception du circuit dans le but d'améliorer la testabilité (méthodes : ad hoc, Scan, Bist, JTAG) [2][24].

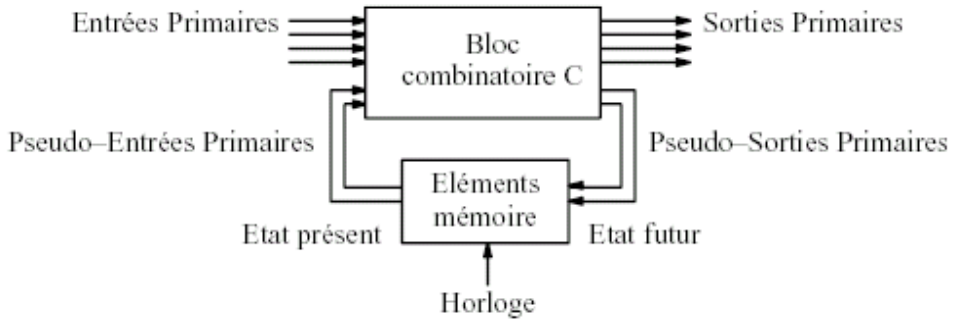


Figure II.6: *Représentation d'un circuit séquentiel.*

La DFT occupe une place importante dans la conception des circuits intégrés pour les raisons suivantes :

- Améliore le niveau de qualité des circuits (Augmenter le TC).
- Facilite le test des circuits complexes.
- Diminue le coût test en diminuant le coût de génération et d'application du test.
- Diminue le temps de développement des tests (Réduire le TTM).

II.6.2. Méthode de DFT [25]

Il n'existe pas une méthodologie unique pour résoudre tous les problèmes de test, comme il n'existe pas de technique valable pour tous les circuits, par contre il existe deux familles de techniques de DFT à savoir les méthodes ad hoc et les méthodes structurées (Chemin de Scan, Boundary-Scan, BIST), la description de ces dernières méthodes est donnée ci-dessous.

II.6.2.1. Chemin de test "Scan Path"

a. Description

C'est la méthode qui permet de rendre une machine à états finies « FSM » testable en contrôlant et observant les états internes de la machine.

La technique du **Scan path** consiste à transformer un circuit séquentiel (Figure II.9) en un circuit pseudo-combinatoire(Figure II.10). Le circuit est donc reconfiguré afin de le rendre pseudo combinatoire, c'est à dire combinatoire lorsque nécessaire. Le test des circuits combinatoires est bien moins complexe que celui des circuits séquentiels. Cette technique permet donc de transformer le problème afin de le simplifier. Ainsi un circuit scanné présente deux modes de fonctionnement : un mode de fonctionnement normal et un mode de test. Dans le mode fonctionnel, le circuit fonctionne normalement alors qu'en mode de test toutes les bascules du circuit sont connectées pour former un long registre à décalage **[1][2][24].**

Pourquoi le Scan ?

- Limitation de la technique des points de test (Nombre élevés de Pins additionnels d'entrées sorties).
- Complexité des circuits séquentiels.
- Nécessite d'initialiser les circuits.
- ATPG complexe et coûteux.
- Faible taux de couverture.

Comment ?

- Chaque élément mémoire du circuit est modifié pour pouvoir le configurer en mode décalage « shift register ».
- Le mode décalage est utilisé pour contrôler et observer le contenu des éléments mémoires « contrôler et observer les états ».

Avantages :

- Le problème de test des circuits séquentiels devient combinatoire (utilisation de l'ATPG combinatoire).
- Amélioration de la testabilité des circuits séquentiels (Qualité : TC).

Inconvénients :

Les additifs sont pondérés en termes de:

- Coût.
- Surface additionnelle (Mux et routage).
- Pins additionnels (Pis/Pos).
- Dégradation de performances (Délai de propagation).
- Augmentation du temps d'application des tests (Plusieurs cycle d'horloge pour un vecteur de test).

b. Fonctionnement du Scan

La structure générale d'un circuit séquentiel sous forme de machine d'états **[26]**.

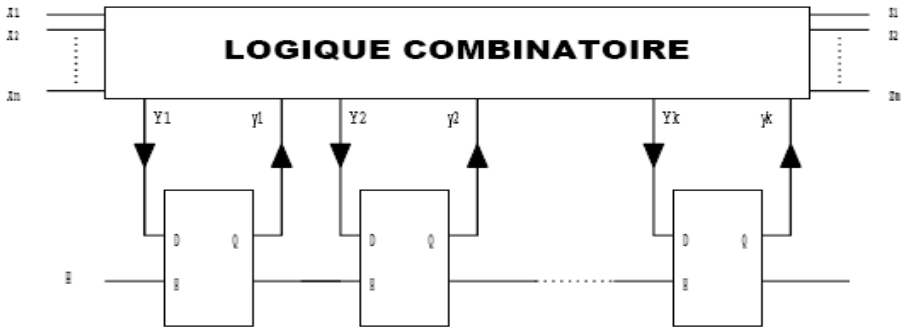


Figure II.7: *Structure générale d'un circuit séquentiel.*

➨ Le circuit séquentiel est transformé en circuit combinatoire :

- Chaînage des bascules (Eléments mémoires).
- Chaque entrée de bascule est considérée comme une sortie primaire.
- Chaque sortie de bascule est considérée comme une entrée primaire.

➨ Le test du circuit est effectué en deux phases :

- Test de bascules chaînées.
- Test de la partie combinatoire.

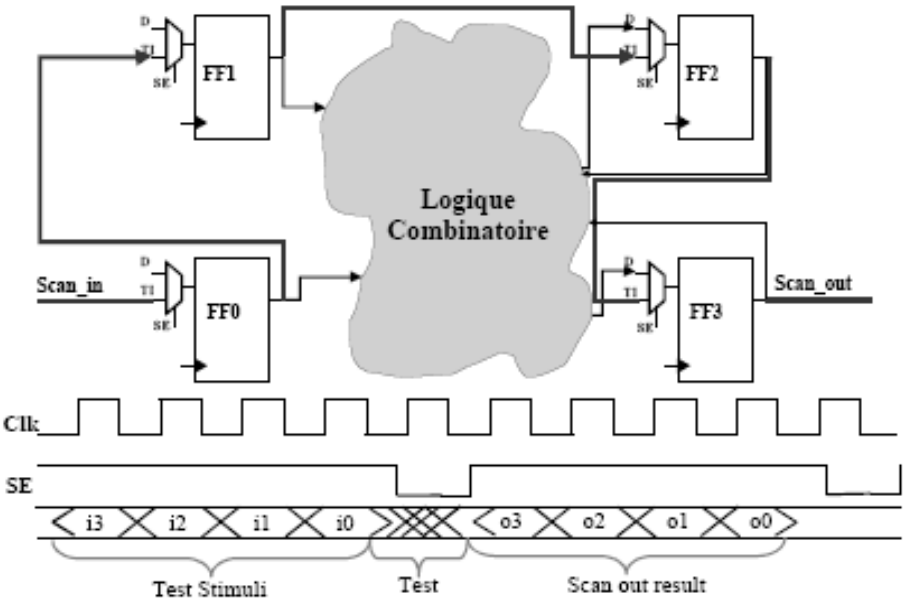


Figure II.8: *Structure de la technique "Scan Path"*.

c. Modes de Scan

Les différents modes de Scan sont

- Scan complet.
- Scan parallèle.
- Scan partiel.
- Scan en mode de reconfiguration.

II.6.2.2. Boundary scan

a. description

La technique du « Scan path » étant à l'origine de la méthode du « boundary scan » (BS), utilise la notion de chemin de balayage tant pour l'observabilité que pour la contrôlabilité des points internes d'un circuit séquentiel. C'est à dure que les bascules du circuit sont reliées en registre à décalage pour permettre l'opération de décalage des données en vue de l'injection de vecteurs et de la récupération de réponses aux séquences de test appliquées **[1][4]**.

La complexité des circuits intégrés, et la miniaturisation des cartes électroniques limitent l'accès aux nœuds internes des circuits et il y a eu recours à la technique Boundary scan. Le standard IEEE 1149.1 explique la technique Boundary scan pour le test de cartes **[24][27].**

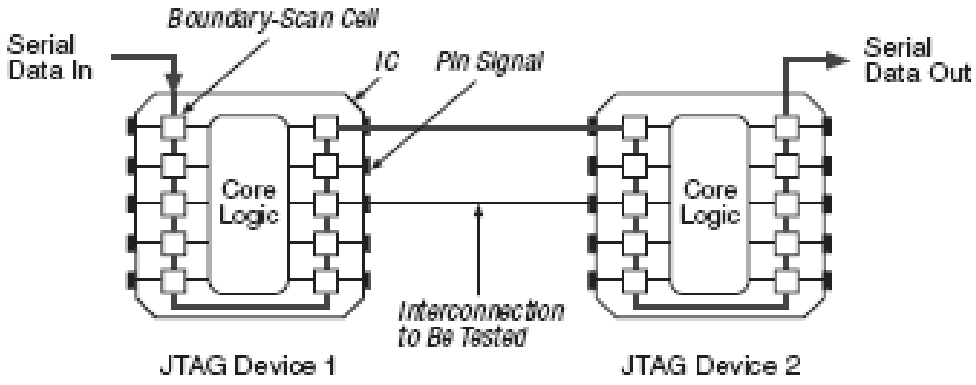


Figure II.9: *Carte "boundary scan".*

La figure II.9 montre l'utilisation de la technique du "boundary scan", on remarquera que pour chaque broche une cellule "boundary scan" est rajoutée. De manière semblable à la technique du "scan path", toutes les cellules BS seront connectées en série pour former dans un premier temps, le chemin de balayage du circuit et, ensuite, le chemin de balayage de la carte. Il devient donc possible d'accéder aux données du test par l'intermédiaire de broches d'entrées et de sorties BS tant au niveau des circuits qu'au niveau de la carte.

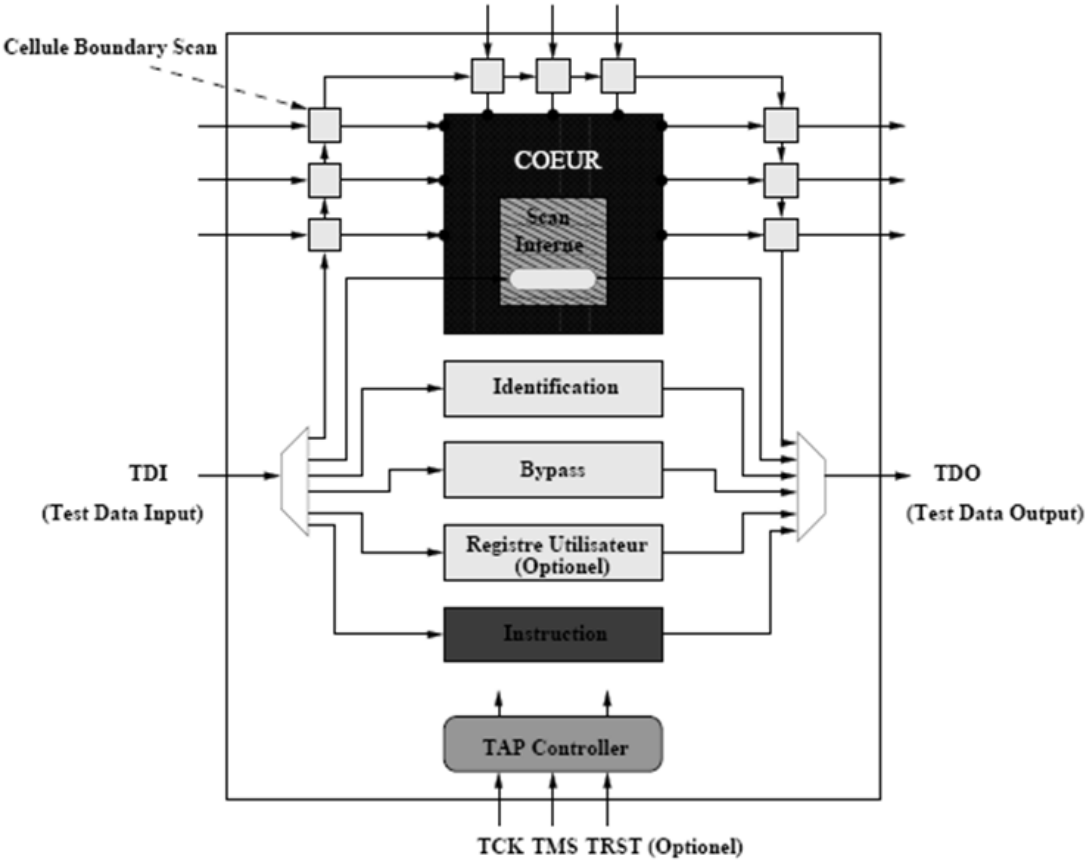


Figure II.10 : *Architecture générale d'un circuit Boundary Scan.*

Les figures II.10 et figure II.11 montrent l'architecture générale de la norme IEEE 1149 **[28].** Cette architecture comprend un registre d'instruction, un registre bypass un registre boundary scan BSR (en gris), des registres de données optionnels et une interface de test connue sous le nom de TAP. Le BSR (Boundary Scan Register) est une série de cellules reliant l'entrée à la sortie du circuit. Un chemin de balayage est structuré entre TDI et TDO. Cette architecture autorise le TAP à choisir de décaler les données à travers les deux types de scan selon qu'elles soient instructions ou données.

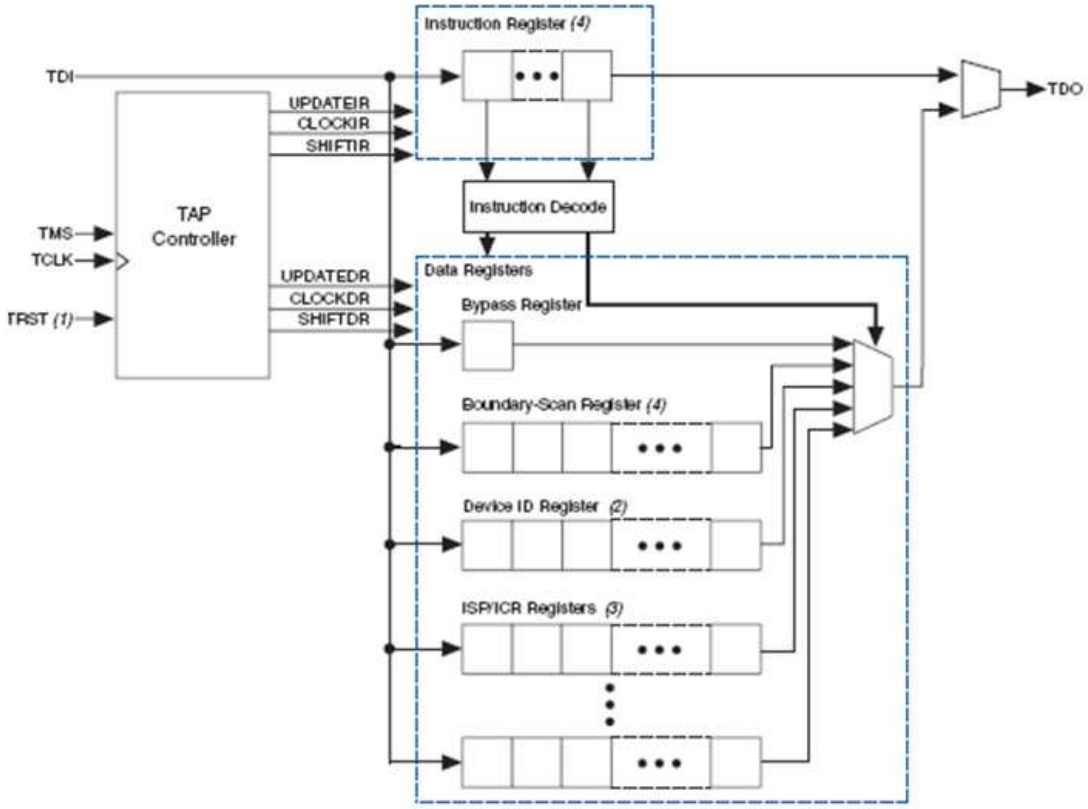


Figure II.11 : *La circuiterie du Boundary Scan*.

b. Cellule boundary scan :

La figure II.12 montre un exemple de cellule BS ; cette cellule opère selon différents modes :

- *Mode normal :* Mode_control = 0 et les valeurs passent de IN à OUT.
- *Mode de Scan :* Les cellules du boundary scan sont connectées pour réaliser le registre de boundary scan. La première cellule est alimentée par TDI et la dernière cellule fournit les informations à TDO. Dans ce mode ShiftDR = 1 et les impulsions d'horloges sont appliquées à clockDR.
- *Mode de capture :* Avec ShiftDR = 0, les données présentes sur IN peuvent être capturées dans le registre scan à chaque top d'horloge sur ClockDR.
- Mode de mise à jour : Une fois une valeur est stockée dans QA par scan ou par capture, elle peut être appliquée sur OUT en positionnant Mode_control à 1 et en appliquant un top d'horloge sur UpdateDR (Voir figureII.12).

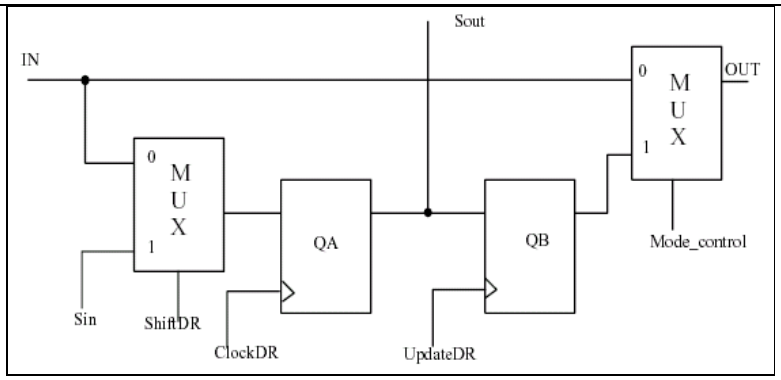


Figure II.12 : *Exemple de cellule boundary scan*

c. Les différents modes de test au niveau carte [29] :

La disponibilité de ces cellules au long du périmètre du circuit fait qu'il est possible de réaliser trois types de test différents au niveau de la carte: le test externe, le test interne et le test d'échantillonnage. Ces trois tests sont décrits ci-dessous:

Le test externe permet de tester les interconnexions entre les circuits supportant la norme 1149 et éventuellement de la logique externe. Dans ce mode, les vecteurs de test sont fournis par les cellules BS de l'interface de sortie des circuits et capturés par les cellules BS de l'interface d'entrée des circuits.

Le test interne permet d'obtenir une contrôlabilité et une observabilité quasi complètes des circuits supportant la norme IEEE 1149. La logique interne du circuit est commandée par les cellules d'entrée du BS et les réponses sont observées par les cellules de sortie BS. Le circuit peut aussi contenir des dispositifs de test interne et/ou des possibilités de test intégré ; et dans ce cas, les opérations de test interne sont effectuées durant ce mode.

Le troisième type de test est Le test par échantillonnage. Il permet de capter à la volée des données en entrée et en sortie des circuits. Ce test n'affecte pas le fonctionnement du système, il permet de vérifier à la vitesse d'opération normale du système les interactions et les performances dynamiques des diverses parties qui composent la carte.

d. Le port TAP et son contrôleur [30]:

Une carte conçue à base du standard BS IEEE 1149.1 contient un bus de test "TAP : test access port" (figure II.11) constitué de 4 signaux au minimum (un cinquième signal peut être utilisé pour initialiser la circuiterie de test).

TDI *(Serial Test Data Input)* : entrée série d'instructions et de données de test;

TDO *(Serial Test Data Output)* : sortie série d'instructions et de données de test. Il s'agit d'une broche à trois états commandée par le signal ENABLE;

TCK *(Test ClocK)* : horloge de test, indépendante de l'horloge du système (SCK);

TMS *(Test Mode Select)* : signal d'ordonnancement d'état du contrôleur TAP.

***TRST** (Test ReSeT input)* : signal de reset asynchrone (optionnel).

Le TAP est une machine à états finis dont les signaux TMS et TCK sont les entrées et les signaux de contrôle internes sont les sorties (Figure II.13)(Figure II.14). Le diagramme d'états de ce contrôleur comporte 16 états comme le montre la Figure II.13.

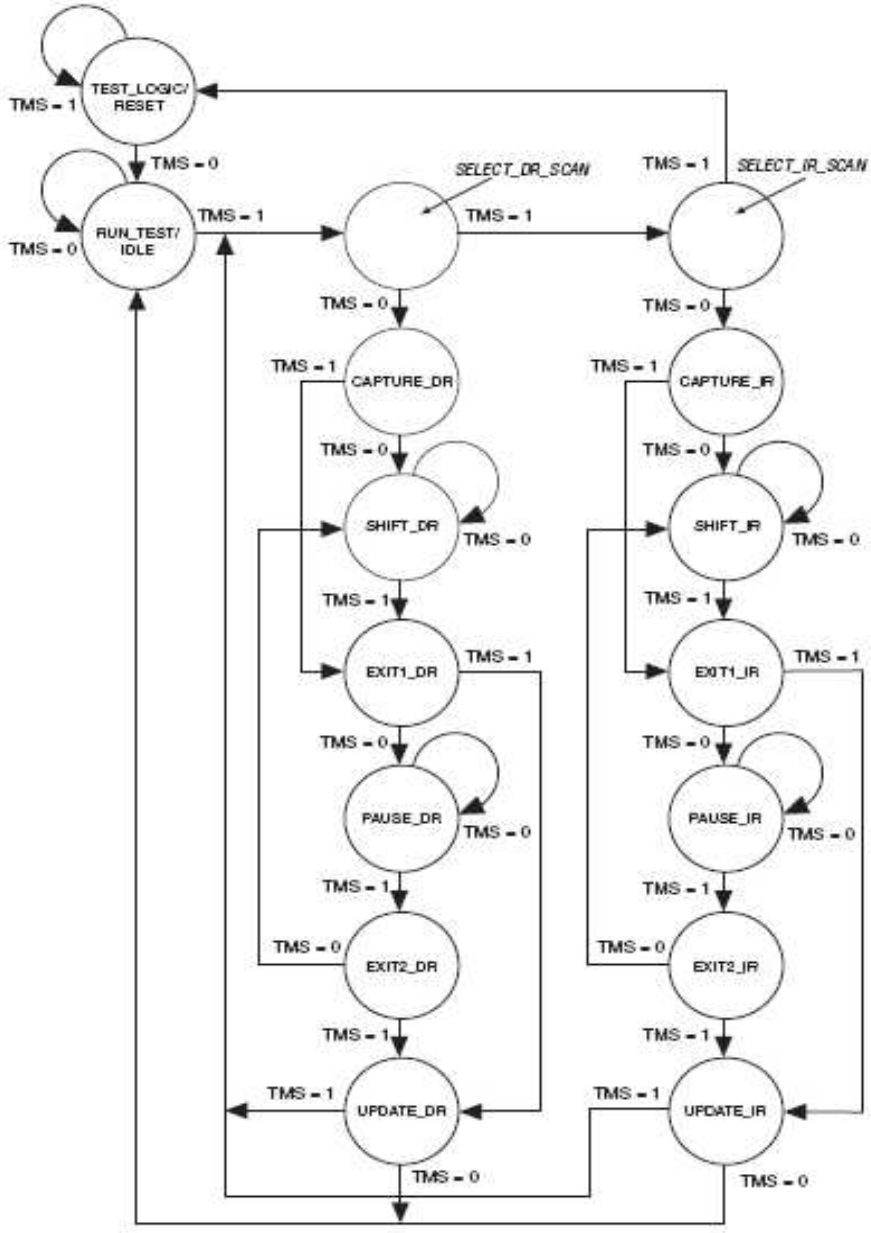


Figure II.13 : *Diagramme d'états du contrôleur TAP.*

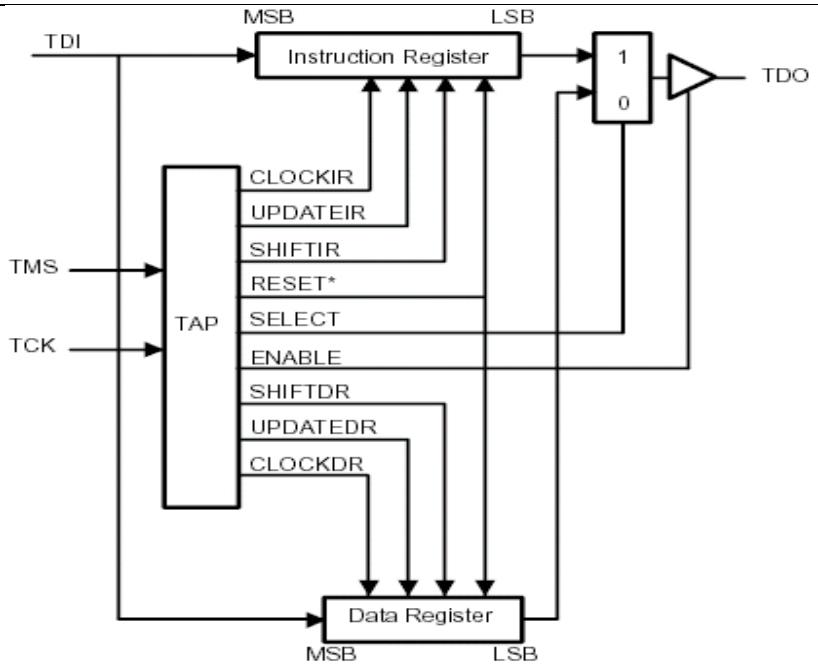


Figure II.14: *Interconnexions du contrôleur TAP*

e. Les registres :

Le registre d'instruction :

Ce registre est utilisé pour spécifier les opérations à exécuter (figure II.15), et la sélection des registres de données. Au début du cycle de décalage d'une instruction, il sera chargé avec l'état du test précédent. Ensuite, cet état sera décalé vers TDO en même temps qu'arrivera la nouvelle instruction par TDI.

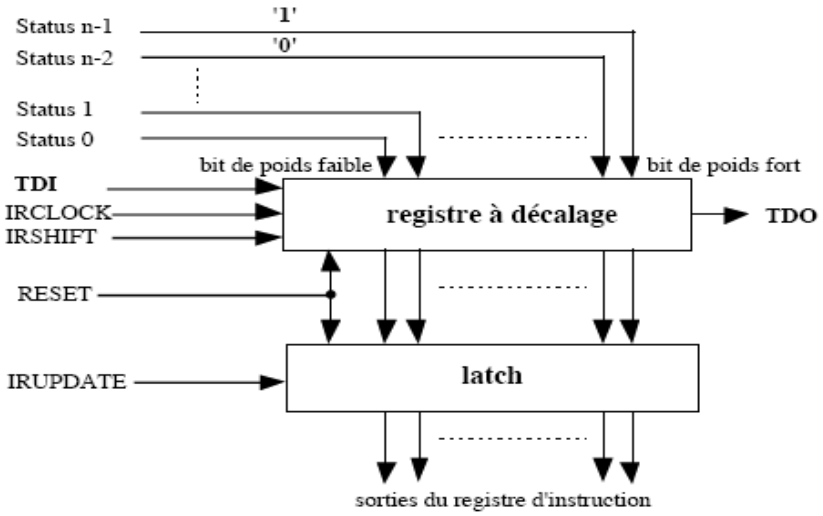


Figure II.15: *Le registre d'instruction.*

Il est composé au moins de deux bits, les deux premiers bits qui sont à "01"exploités pour détecter des défauts lors du décalage dans le chemin de balayage de la carte.

Les registres de données :

Registre boundary scan : il est constitué d'un 'enchaînement de plusieurs cellules BS chacune connectée à une broche de l'interface d'entrée ou de sortie du circuit. Ces cellules BS peuvent être connectées à des broches de données, de contrôle ou même d'horloge.

Registre bypass : C'est un registre de déviation composé d'un bit, qui relie directement TDI à TDO si on veut exclure un circuit de la carte lors du test.

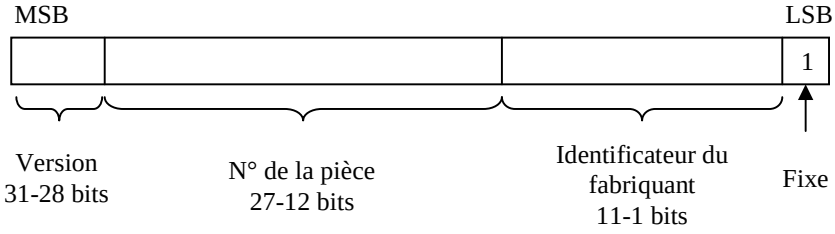


Figure II.16: *Structure du registre d'identification*

Registre d'identification (optionnel) : il est défini par la norme IEEE Std 1149.1 pour identifier le fabriquant et d'autres informations spécifiques au circuit (figure II.18).

Les "registres utilisateurs" (optionnels) : Les registres de l'utilisateur sont des registres de données spécifiques permettant l'accès à toutes les ressources de test et de logique interne du circuit. Ces ressources peuvent être un "scan path" interne ou des registres du type autotest intégré ("built-in self-test"). Mis à part les décalages, toute autre opération sur ces registres doit passer par l'état "Run-Test/Idle" du contrôleur TAP.

II.6.2.3. La technique BIST "Built In Self Test"

a. Description

Le test d'un circuit nécessite l'application de séquences de test à sa logique interne et à l'observation et l'analyse des réponses obtenues. Des techniques connues sous le nom de test intégré ont étés proposées afin d'inclure directement dans le circuit toutes ou parties des fonctions réalisées par le testeur (figure II.17).

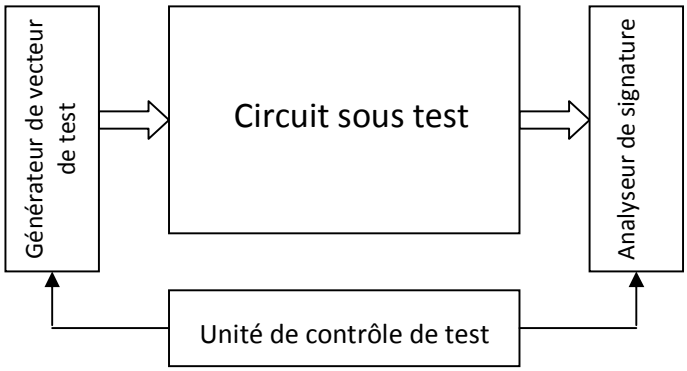


Figure II.17: *Schéma de principe d'une structure à test intégré*

D'une manière générale, les différentes architectures de test intégré mettent en œuvre un générateur de vecteurs de test (GVT), un analyseur de la réponse (subdivisé en un compacteur (AS) et un comparateur), et de la logique de contrôle. Ces différents modules sont ajoutés à la logique fonctionnelle comme éléments additionnels qui ne contribuent pas à l'exécution de sa fonction mais exclusivement pour des fins de test **[1] [24].**

Deux formes de structure de BIST existent l'une série, et l'autre parallèle (figure II.18).

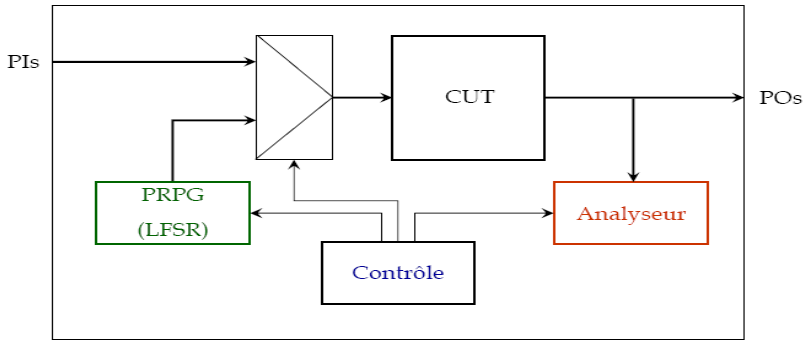


Figure II.18: *Le test intégré parallèle*

b. Génération de vecteurs de test [2]

Les techniques de génération intégrées de vecteurs dépendent naturellement de l'approche adoptée pour le test du CI : pseudo–aléatoire, exhaustive (pseudo–exhaustive), déterministe ou mixte (combinant aléatoire et déterministe).

Avec le test pseudo–aléatoire, le taux de couverture est fonction de la longueur de la séquence appliquée. Plus la séquence appliquée est longue, plus son taux de couverture s'approche du maximum, la longueur nécessaire pour l'atteindre est liée au circuit sous test.

c. Le LFSR *(Linear Feedback Shift Register)*

Un LFSR est un registre à décalage, dont quelques sorties sont combinées en une configuration de XOR pour former un mécanisme de rétroaction. Un LFSR peut être formé en plaçant des OU-exclusifs aux sorties de deux ou plusieurs bascules D, et de réinjecter le signal dans l'entrée des bascules (figure II.19).

De tels circuits sont cycliques du fait que si on agit sur le signal d'horloge à plusieurs reprises ils parcourent une séquence d'états fixe.

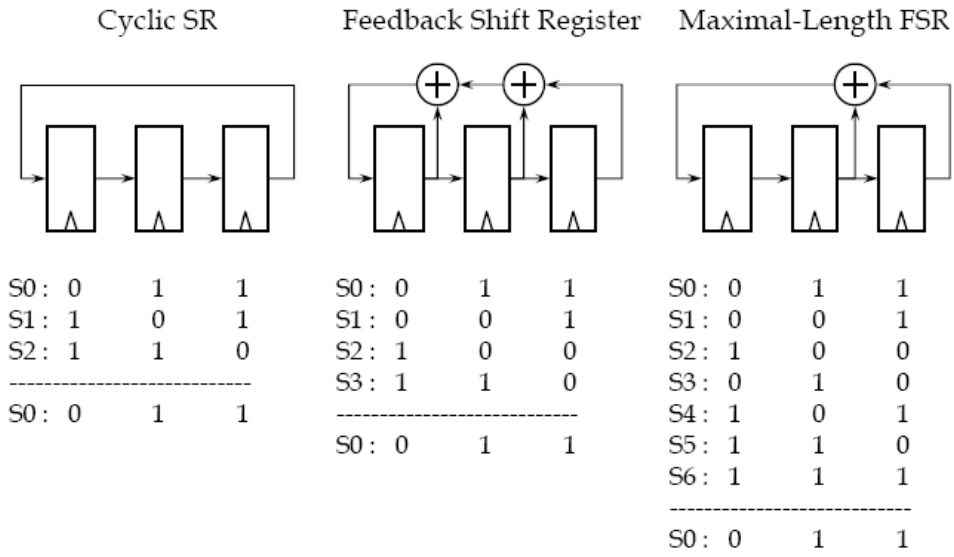


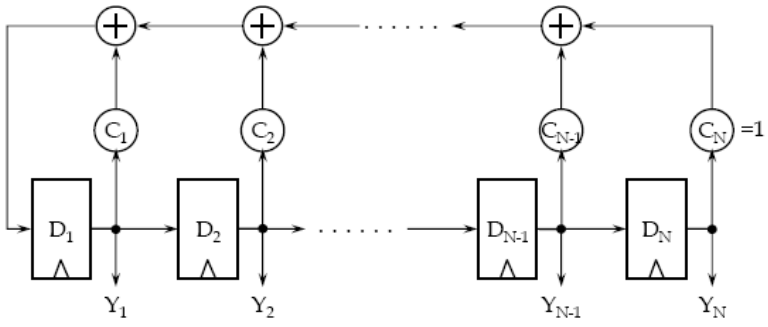
Figure II.19 : *Exemples de LFSR.*

La génération de séquences de vecteurs aléatoires s'effectue en utilisant des LFSR autonomes dont la forme la plus générale est représentée à la figureII.20, les constantes binaires "ci" impliquent la présence (ci = 1) ou l'absence (ci = 0) de connexions.

Une séquence de nombre a0, a1, ..., am, ... peut être associée à un polynôme appelé fonction génératrice G(x) par la relation:

$$G(x) = a0 + a1x + a2x2 + \ldots\ldots + amxm + \ldots\ldots$$

a. LFSR à XOR externe.



b. LFSR à XOR interne.

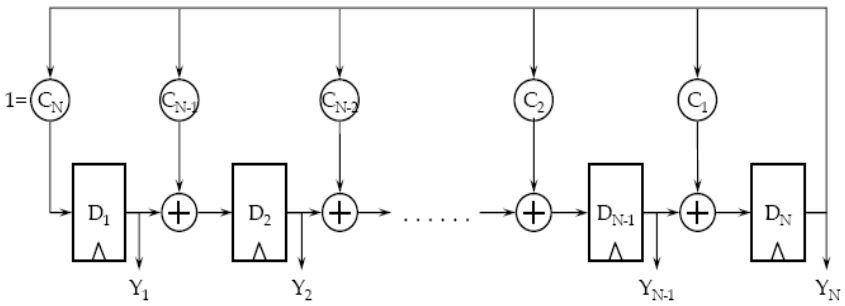


Figure II.20: *Les différents types de LFSR*

Exemple :

	V_1	V_2	v_3	V_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
Q_0	1	0	0	0	1	0	0	1	1	0	1	0	1	1	1
Q_1	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1
Q_2	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0
Q_3	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0

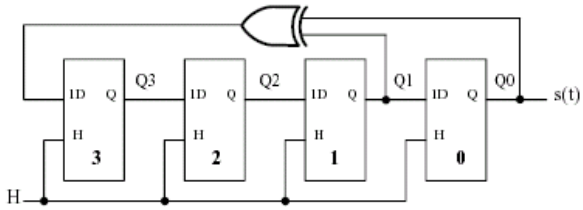


Figure II.21: *Exemple de LFSR à quatre étages*

Caractéristique d'un LFSR :

- Si l'état initial d'un LFSR est $a_{-1} = a_{-2} = ... = a_{-(n-1)} = 0$ et $a_{-n} = 1$, alors la séquence {am} du LFSR est périodique avec une période égale k tel que P(x) divise $(1-x^k)$.
- Si la séquence générée par un LFSR à n étages est de longueur 2^n-1, elle est appelée séquence de longueur maximum.
- Le polynôme caractéristique associé à une séquence de longueur maximum est appelé polynôme primitif.
- Un polynôme irréductible est un polynôme qui ne peut être factorisé.
- Un polynôme irréductible satisfait les deux conditions suivantes :
  - il a un nombre impair de termes (le terme 1 inclus).
  - si il est de degré supérieur à 3 alors P(x) doit se diviser en $(1+x^k)$, avec $k=2^n-1$.
- Un polynôme primitif est irréductible, si le plus petit entier positif k qui permet au polynôme de se diviser en $1+x^k$ est tel que k=2n–1 avec n degré du polynôme.

LFSR modifié :

On peut générer les vecteurs de test exhaustivement en utilisant un LFSR de période maximum modifié de manière à pouvoir atteindre l'état nul (figure II.22).

	V_1	V_2	v_3	V_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}
Q_0	1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1
Q_1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1
Q_2	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0
Q_3	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0

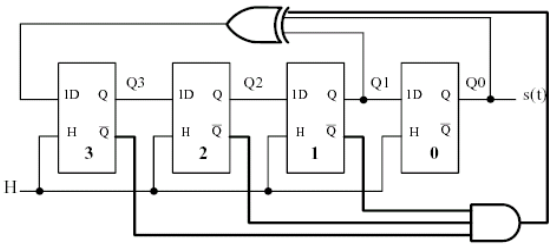


Figure II.22: *LFSR modifié*

d. Analyse de signature [20]:

La détection d'une faute nécessite la comparaison des sorties du circuit sous test avec les réponses de circuit exempt d'erreurs, pour cela il faut réduire l'information à traiter. Une multitude de techniques de compression existe, dont une seule peut satisfaire les contraintes

imposées par le test avec une probabilité de masquage $\frac{2^{m-n}-1}{2^m-1}$ (n nombre de bascule et m-1 le degré du polynôme caractéristique).Cette technique est basée sur la division polynomiale. Une séquence d'information correspondant à un polynôme Z(x) est comprimée grâce à un autre polynôme P(x). Le résultat (le quotient ou le reste) de cette division est appelé signature.

Compression série :

C'est un LFSR, auquel on ajoute une entrée qui représente la sortie d'un circuit sous test (figure II.23).

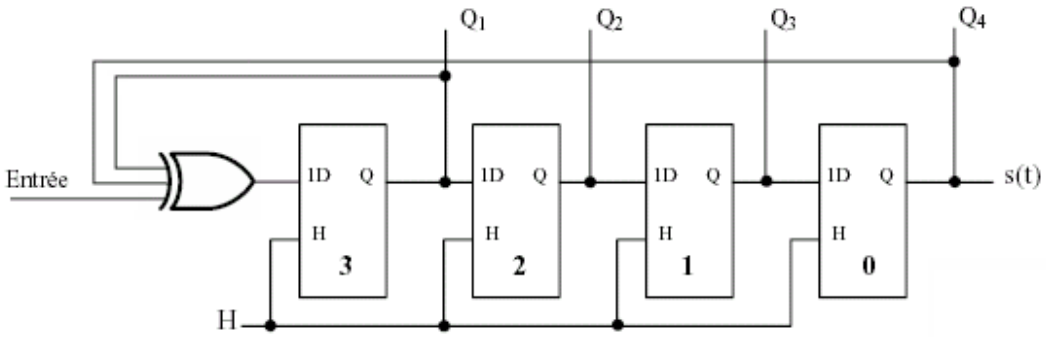


Figure II.23: *Exemple d'un analyseur de signature*

Compression parallèle MISA *(Multiple Input Signature Analyser)*

Lorsqu'on veut analyser plusieurs sorties ou entrées d'un circuit en même temps, il est intéressant de faire de la compaction parallèle, dans ce cas le circuit utilisé est un LSFR à entrées parallèles (MISR : Multiple Input Shift Register), (figure II.24).

La structure du MISR est identique à celle du LSFR sauf qu'elle accepte plusieurs entrées à analyser. Cette structure permet la compression simultanée de plusieurs séquences de sortie différentes du circuit sous test.

L'application d'une fenêtre de signature permet de limiter le nombre de bits de la donnée qu'il faut prendre en considération durant la compaction. En effet deux signaux "START" et "stop"(départ et arrêt) permettent respectivement l'ouverture et la fermeture de cette fenêtre.

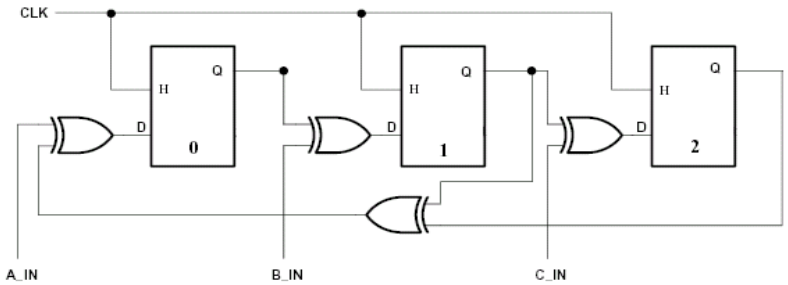


Figure II.24: *Analyseur de signature parallèle.*

Association Boundary scan BIST

Dans cette technique les cellules BS sont remplacées par des éléments appelés "template" capables d'être configurés en cellules BS, en plus les éléments adjacents aux PINs d'entrées forment un registre pouvant être configuré en PRPG (Générateur de vecteurs de test pseudo-aléatoire) et Les éléments de sortie peuvent être configurés en MISR ou en LFSR (figure II.25).

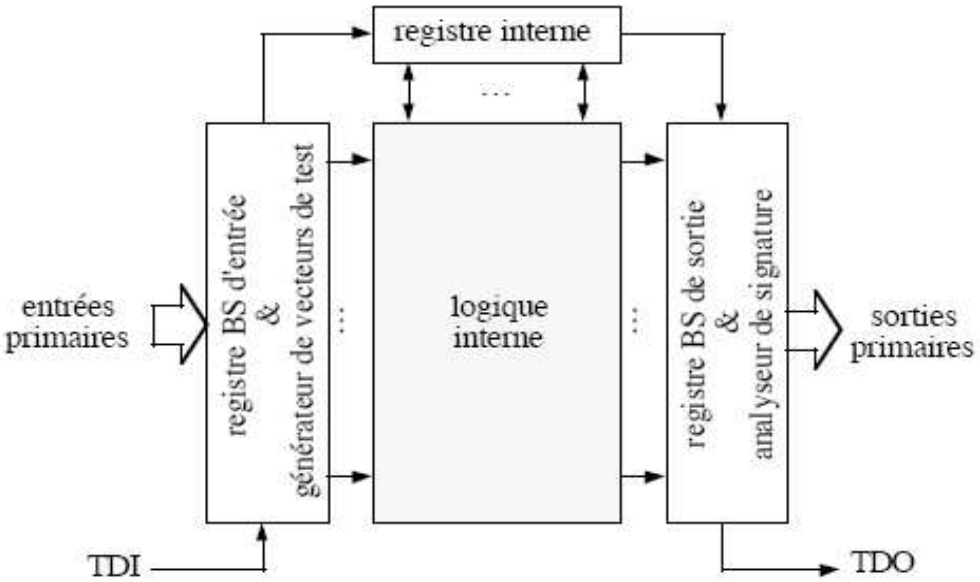


Figure II.25 : *Modèle structurel pour BIST avec BS*

La première étape dans la procédure du test est de configurer le registre d'entrée comme PRPG capable de générer 2^{s-1} séquences de s bits (où s est le nombre de cellules d'entrée).La séquence de test à chaque moment est de r = (n + m) bits, les n premiers bits sont les entrées primaires du circuit tandis que les m derniers bits sont le résultat du SCAN-IN du contenu des dernières bascules du générateur aux m bits du registre scan. Les réponses des sorties primaires sont appliquées au registre de sortie qui est configuré en MISR. Le contenu du registre scan sera décalé vers le registre de sortie configuré cette fois en LFSR, dans le même temps le générateur décale les nouveaux m bits dans le registre scan, après ceci, un nouveau cycle de test commence. Finalement la signature est observée au niveau du connecteur pour comparaison et diagnostic.

II.7. conclusion

Nous avons présenté dans ce chapitre un bref aperçu sur les différentes techniques de conception en vue de test DFT. Les méthodes les plus employées actuellement sont celles de balayage avec le Scan path, le boundary Scan et l'auto test Bist. L'intégration toujours poussée a fait que la tendance est focalisée sur l'intégration du test au sein des circuits pour accomplir de meilleures testabilités.

CHAPITRE III

Conception d'un contrôleur flou testable

III. 1. Introduction

Il s'agit de concevoir un contrôleur flou testable implanté sur un FPGA en utilisant une plateforme de développement exploitant le langage de description matériel VHDL **[31] [32]**.

Le contrôleur étudié dans ce travail est basé en partie sur l'architecture développée par A. Gabrielli dans **[33][34][35]** où nous avons apporté les modifications conformément aux consignes inscrites dans le cahier des charges.

Les étapes suivies pour la concrétisation de notre circuit sont définies ci-dessous:

- Développement d'un programme VHDL répondant aux exigences du cahier de charge sous une plate forme de Xilinx synthétisant le contrôleur et vérification de son fonctionnement.
- Etude et méthode d'insertion d'un mécanisme de test au contrôleur au niveau comportemental.

Les spécifications du contrôleur à concevoir sont :

- Le contrôleur doit satisfaire les règles de "Design for test" **[2]**.
- Il doit assurer une flexibilité et une lisibilité de sa description fonctionnelle, et son architecture doit être adaptable et compatible aux applications du contrôle par la logique floue.

III. 2. Conception d'un Contrôleur flou - IP Soft

III. 2. 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 allons choisir doit nécessairement permettre de faire des descriptions de haut niveau en plus des descriptions structurelles. Ainsi, 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 **[36][37][38].** La sélection de VHDL par rapport aux autres langages de modélisation tels que Verilog, C++ est liée a sa disponibilité dans notre laboratoire.

Les outils de conception et de simulation "Xilinx web pack ise7.1" de la société "Xilinx" et "Model Sim 6.0" de "Mentor Graphics" sont exploités pour la mise en œuvre de notre contrôleur flou (FLC - fuzzy logic controller). Ainsi le flot de conception basé sur VHDL est représenté sur la figure III.1 **[39][40]**.

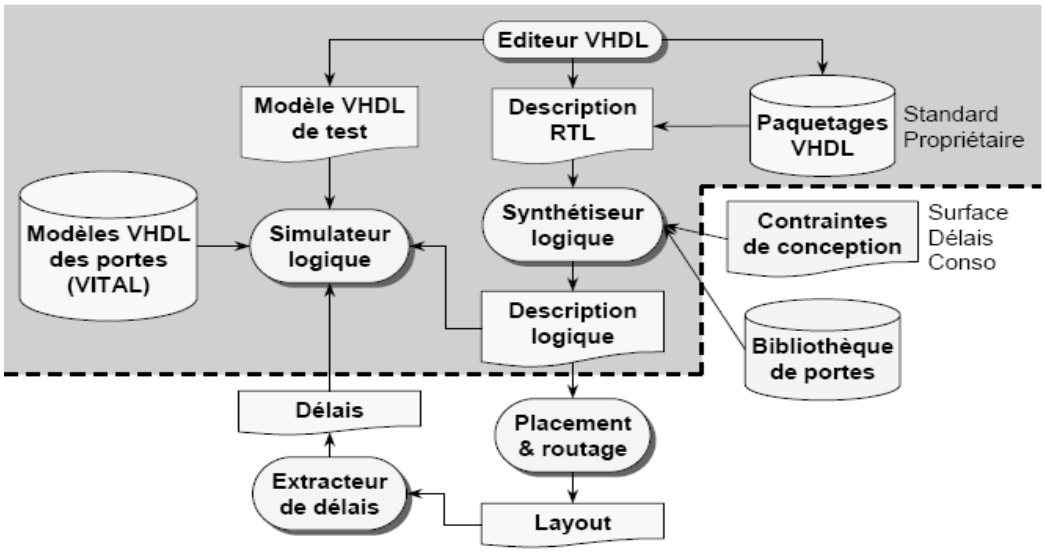


Figure III.1: *Flot de conception basé sur VHDL.*

Objectifs :

- Processeur flou de taille très réduite.
- Processeur flou très rapide (Grande vitesse).

Pour cela on utilise donc une architecture Parallèle/Pipeline où on sélectionne uniquement quatre règles actives et l'implémentation se fera sur des FPGA dans une plateforme Xilinx en exploitant le langage de description VHDL.

III. 2. 2. Modèle et partitionnement du contrôleur

III. 2. 2. 1. Description

Dans certains domaines d'applications, les composants électroniques utilisés pour la détection, l'identification et la mémorisation des évènements doivent être très rapides que possible, ainsi ils doivent avoir des bonnes performances (Robustesse, flexibilité …) afin qu'ils résistent contre les bruits, avec une consommation d'énergie très basse.

Ces exigences peuvent être satisfaites par l'utilisation d'une architecture parallèle/pipeline du processeur flou et par l'élimination du temps d'identification des règles actives, donc uniquement quelques règles floues sont sélectionnées (*Active rules*) et exécutées permettant de réduire donc le temps d'exécution.

Dans le but d'atteindre de bonnes performances on a divisé la conception et la synthèse du processeur flou en plusieurs phases :

- Le processeur flou (Pipeline) travaille avec la plus grande fréquence possible.
- Le processeur doit être implémenté par Hardware (matériel) 'HW' simple algorithme tel que la méthode de Sugeno d'ordre zéro pour l'inférence et la défuzzification. **[41]**
- Le processeur procède uniquement par quatre règles actives afin de réduire la consommation et d'accélérer le processus.

- On utilise 49 règles floue de base, toutes les combinaisons possibles pour un système flou de deux entrés X_0 et X_1 avec 7 sous-ensembles flous pour chacune.
- Un seul chevauchement permis entres deux fonctions d'appartenance voisines.
- Les sous-ensembles des antécédents servent a adresser la mémoire des règles.
- Un bloc « Active interval selector » permet d'identifier et de mettre en jeu uniquement les règles actives qui correspondent aux données des entrées, donc consommation réduite en temps.

III. 2. 2. 2. Architecture du processeur flou

Les principales caractéristiques du processeur flou sont résumées comme suit :

- Deux variables d'entrées floues numériques (6-bits).
- Une variable de sortie floue numérique (6-bits).
- 3-bits pour chacun des antécédents (alpha) et les degrés de vérité des prémisses thêta).
- 7 fonctions d'appartenance pour chaque variable d'entrées.
- 6-bits de sortie => 64 'crisp' (Valeurs de sortie).
- (7 X 7) : 49 règles floues.
- Application de la t-norme pour le calcul du minimum (Alpha).
- 1 seul chevauchement permis entre deux fonctions d'appartenance adjacentes.
- Caractéristiques des l'FPGA (Voir annexe (B)).

Avec ces caractéristiques, le processeur aurait les performances suivantes :

- La fuzzification est exécutée en un top d'horloge.
- Le module d'inférence est exécuté en deux tops d'horloge.
- La defuzzification est exécutée en deux tops d'horloge.
- Le controleur donne une réponse tous les 5 tops d'horloge.

III. 2. 2. 3. Description des différents blocs du contrôleur

L'architecture du processeur flou est présentée sur la figure III.2.

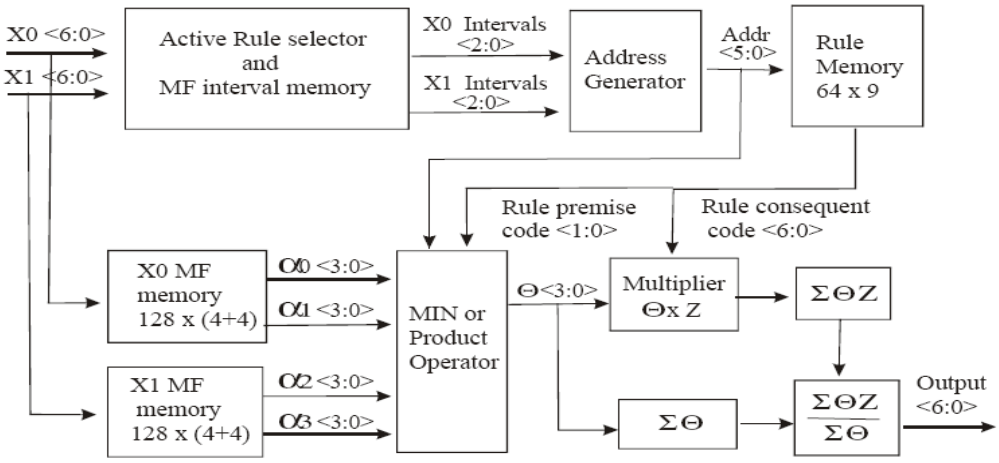


Figure III.2: *Schéma bloc du processeur flou*

a. Bloc de fuzzification

a. 1. Sélection des intervalles actifs

Ce bloc (figure III.2) contient les points d'abscisses finales des 7 fonctions d'appartenance (MFs), les 7 MFs définissent 6 intervalles, donc uniquement 5 données sont suffisantes pour la représentation des fonctions d'appartenance et par conséquent l'identification et la sélection des intervalles des sous-ensembles Fs auxquels la variable Xi appartient. (Figure III.3)

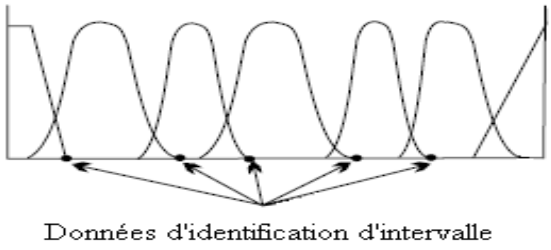


Figure III.3: *Identification d'intervalle*

Si la valeur de la variable d'entrée est inférieure au point final de la première fonction d'appartenance, alors les fonctions d'appartenance actives seront la première et la deuxième (figure III.4). Si la valeur de la variable présentée sur l'entrée se situe entre les deux points finaux des deux MFs alors les MFs actives seront la deuxième et la troisième et ainsi de suite.

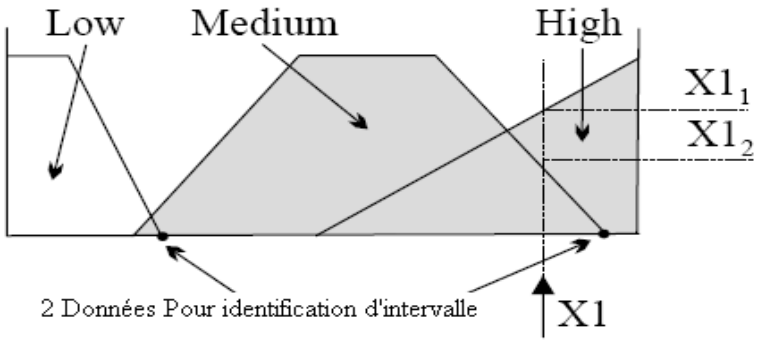


Figure III.4: *Identification d'intervalle et de degrés d'appartenance*

a. 2. Générateur d'adresses

Ce bloc permet de générer les 4 adresses des 4 règles actives après identification des intervalles actifs, et cela dépend des valeurs des variables d'entrées Xo et X1. Pour identifier un intervalle parmi les 8 sous-ensembles Fs d'entrées, un mot de 3 bits est utilisé pour coder ces intervalles.

Le générateur d'adresse concatène ces paires d'intervalles de 3bits pour former une adresse sur 6 bits (6 bits permettent l'adressage de nos 49 règles).

Chaque intervalle sollicité est adjacent à l'autre intervalle successif, donc avec n'importe quel code d'intervalle, le suivant sera facile à déduire (figure III.5).

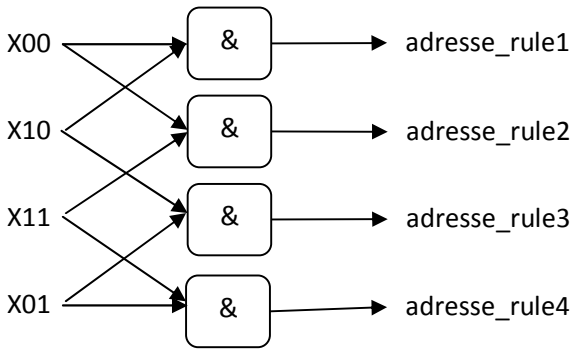


Figure III.5: *Générateur d'adresses*

Exemple :

Si la paire des codes d'intervalle sont : X1 : 001 et pour X0 : 101, les adresses générées qui correspondent aux 4 règles floue actives seront :

Adresse0 := 001–101
Adresse1 := 001–110
Adresse2 := 010–101
Adresse3 := 010–110

a. 3. Générateur des alphas

Ce bloc sert à générer les degrés de vérité des antécédents (alpha), l'opération peut se faire en deux manières :

- Par une opération arithmétique
- Par des look-up tables.

Comme le calcul arithmétique consomme beaucoup de temps et de surface de l'FPGA alors on a eu recours à l'implémentation par des look-up tables ayant un temps d'accès rapide (RAM). Ceci nous mène à un bloc mémoire (moins de calcul) où chaque degré d'appartenance est codé sur 3 bits, donc pour chaque entrée on a besoin des deux degrés relatifs à chaque intervalle MF.

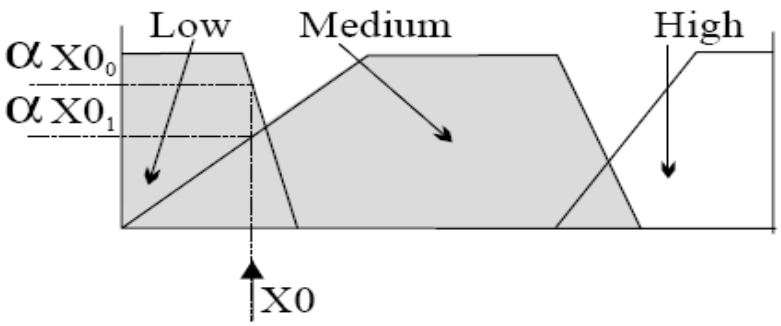


Figure III.6 : *Génération de degrés d'appartenance (alpha)*

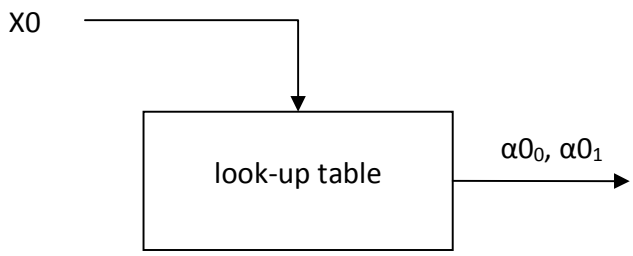


Figure III.6 : *Générateur de degrés d'appartenance (alpha)*

Les variables d'entrées X0 et X1 servent à adresser un bloc mémoire de 64 mots de (6-bits) (figure III.6). Donc il s'agit d'une conversion des 64 valeurs d'entrée de 6bits en 64 mots de 6 bits (valeurs alpha) (figure III.7).

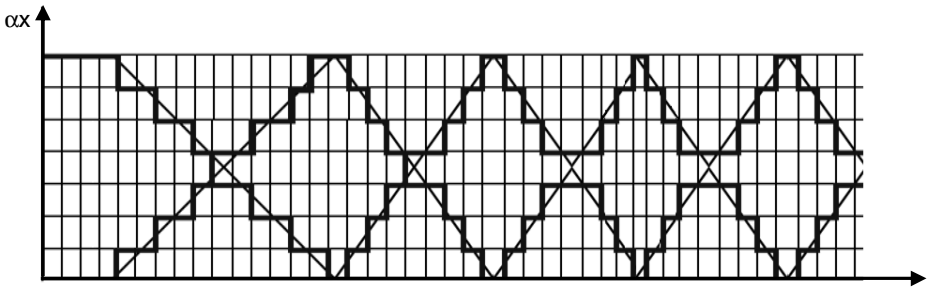


Figure III.7: *Données normalisées et numérisé.*

Format du mot : [degrès1 (3bits)-degrès2 (3bits)], (figure III.8)

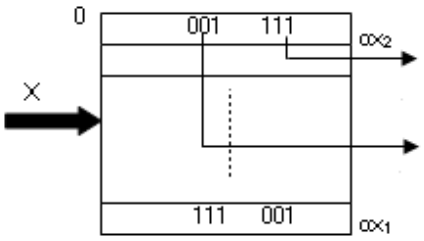


Figure III.8: *RAM des codes alpha*

Exemple :

Xo = 110010 : c'est l'adresse N°: 50 => son contenu est : 100011

Ainsi : alpha0 := 011, alpha1 :=100

b. Bloc d'inférence

b. 1. Opérateur Inférence

Dans ce processus l'opération t-norme a été implémentée pour obtenir les valeurs du minimum Θi (i appartient à l'intervalle [0,3]) parmi les deux valeurs α (3 bits) des deux variables d'entrées X0, X1.

Ce bloc reçoit, comme entrées, les 4 valeurs (αX00, αX01, αX11, αX12), les 4α sont groupés deux a deux : (αX00 avec αX11, αX00 avec αX12, αX01 avec αX11 et αX01 avec αX12), d'où on aura quatre combinaisons qui correspondent aux quatre règles actives (figure III.9).

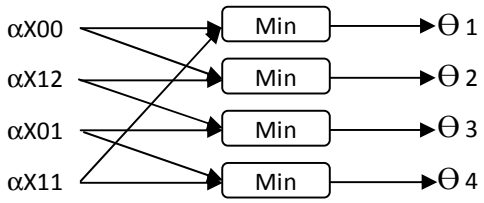


Figure III.9: *Calcul du minimum*

b. 2. Mémoire des règles

Pour ces deux entrées floues X0 et X1 de 7 Fonctions d'appartenance, on a 7 X 7 combinaisons de X0 et X1 d'où 49 règles (Tableau III.1).

Pour n'importe quelle valeur d'entrées du contrôleur flou, on aura 4 possibilités de combinaisons entre X0 et X1, ces 4 combinaisons impliquent 4 règles actives uniquement parmi toutes les règles possibles, et cela donc pour réduire le temps d'exécution.

Les sorties floues de la mémoire des règles ou bien les conséquents des règles [Zj] sont codés aussi sur 3 bits, le bloc est constitué donc de 49 mots de 3 bits.

Tableau III. 1: *Mémoire des règles* (7X7)

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

c. Bloc de défuzzification

Ce bloc reçoit du circuit Opérateur Θ les valeurs des Θi et de la mémoire des règles le code du conséquent de la règle Zj et il effectue la somme ∑ Θi et ∑ Zj* Θi :

Une multiplication rapide est faite entre Zj et Θi, et opérations d'additions (figure III.10).

Finalement on effectue la dernière opération de division de la formule :
Fuzz output = ∑ Zj* Θi /∑ Θi, pour obtenir la valeur de sortie (figure III.11)
La codification du numérateur « ∑ Zj* Θi » est sur 11 bits alors que le dénominateur «∑ Θi » est sur 5 bits.

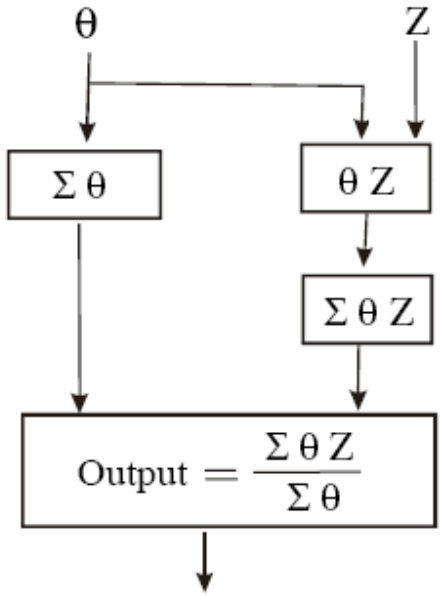


Figure III.10: *Bloc de defuzzification*

L'algorithme de division consiste à soustraire et à comparer chaque fois le dénominateur avec les 6 bits MSB du numérateur, une boucle est chargée à répéter la même opération pour tous les bits du numérateur.

```
Begin
   divtemp:=numérateur(10 downto 5);
        for i in 0 to 4 loop
   if(divtemp >= ('0' & dénominateur)) then
      divout(5-i) := '1';
      divtemp := divtemp - dénominateur;
   else
      divout(5-i) := '0';
   end if;
```

A la fin si le reste est supérieur au 'dénominateur /2' on fait alors une approximation supérieure:

```
«   if(divtemp) >= ('0' & denomérateur(4 downto 1))then
            divout := divout + "000001";
      end if; »
```

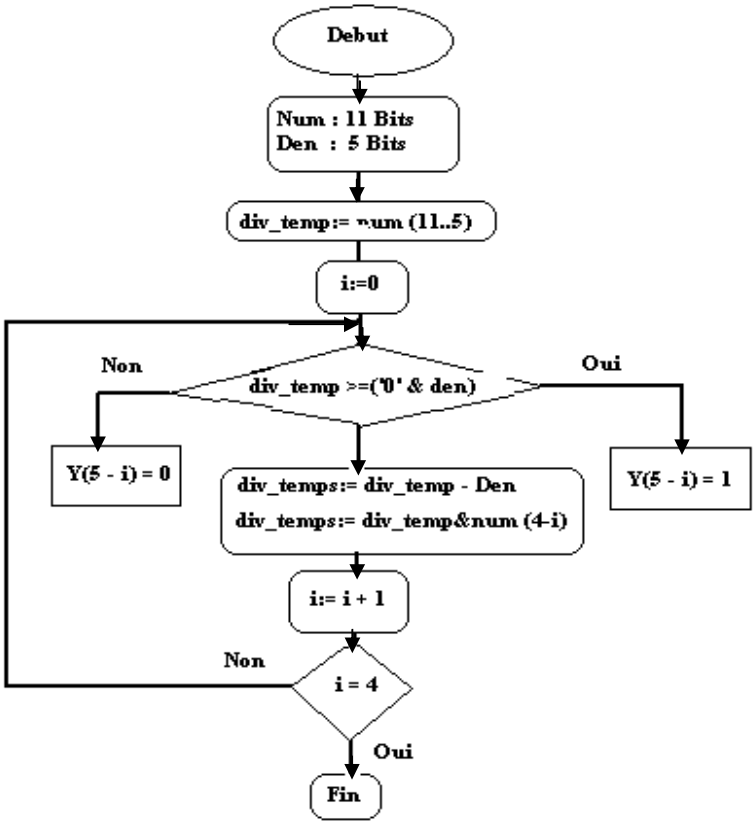


Figure III.11: *Algorithme de la division*

III. 3. Implémentation du contrôleur flou

III. 3.1. Introduction

On a utilisé l'FPGA de la famille Virtex de type « 4vlx25ff668 » (Voir annexe (B)) pour la synthèse de notre contrôleur flou, son comportement est modélisé à l'aide de la programmation VHDL. Le programme VHDL comporte déjà 3 grandes parties à savoir la fuzzification, l'inférence et la defuzzification.

Bloc de La fuzzification : comprend deux modules

- Un module appelé : « **degrés** » : sert à identifier les deux degrés de vérité alpha correspondants à chaque entrée Xi.
- Un module appelé : « **interval_selector** » : sert à identifier les deux intervalles M et M+1 pour toute entrée X quelconque.

Bloc d'inférence : comprend trois modules

- Un module appelé « **memRegles** » : c'est une mémoire où on stocke toutes les règles floues possibles (7x7), elle est adressable par les codes d'intervalles d'appartenance des entrées.

- Un module appelé « **MINN** » : sert pour la comparaison des alphas et déduire les alphas Min Figure III.9. Ce module est constitue de 4 sous modules appelés « MINIMUM » pour effectuer les 4 opérations de comparaison.
- Un module appelé « **PROD** » : c'est une deuxième alternance par rapport a celle du minimum, Ce module comporte un sous module « Productor » pour effectuer les multiplications de 3 bits X 3 bits, ce dernier utilise un composant « additionneur » pour toutes les opérations d'additions nécessaires dont le principe est défini ci-dessous :

  On normalise les produits : 111 X 111 = 110_001 : '49', deux cas sont possibles :
  - Si le résultat du produit est égal à zéro alors : α_i ***PROD*** α_j donne zéro.
  - Si le résultat du produit est différent de zéro alors on se sert de sa partie MSB de trois bits [110] et on l'ajoute un 1 (110 + 001 = 111) donc : 111(1) ***PROD*** 111(1) correspond à 111 (1).

    Exemple : 111 X 001 = 000_111, donc α_i ***PROD*** α_j = 000 + 1 = 001.

On peut utiliser donc deux façons de raisonnement (Inférence) selon l'entrée MIN_PROD du contrôleur implémenté, si elle est à 0 donc la t-norme 'MIN' est choisie, sinon la t-norme 'PROD' est activée.

Bloc de deffuzification : comprend deux modules

- Un module appelé « multip » : sert au calcul de la formule $\sum Zj^* \Theta i$.
- Un module appelé « divider » : sert au calcul de la formule :

$$\text{Fuzz output} = \sum Zj^* \Theta i / \sum \Theta i$$

La structure globale du contrôleur flou est donnée par l'assemblage de ces derniers modules, où la figure III.12 représente le contrôleur flou dans sa construction globale.

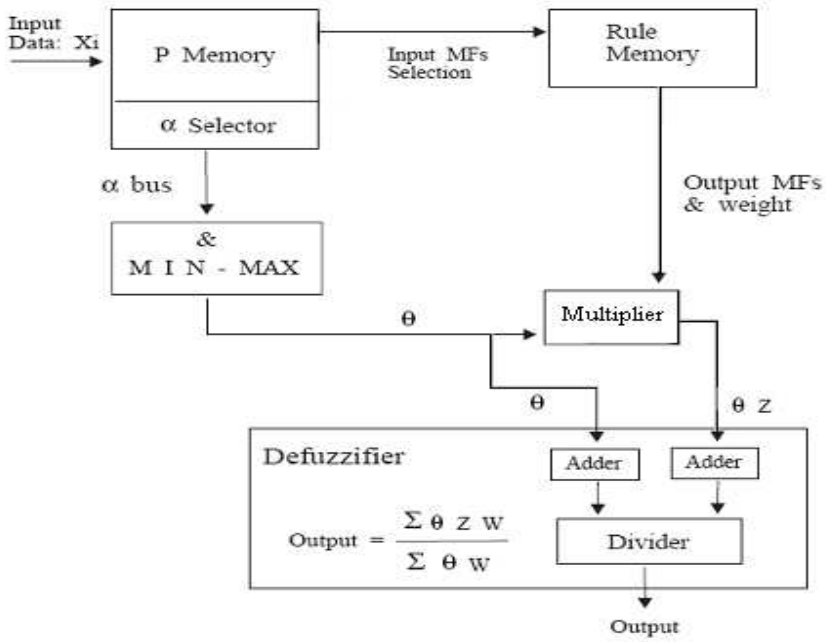


Figure III.12: *Structure du contrôleur flou*

Pour atteindre de bonnes performances, un compromis vitesse précision est considéré, d'où on a opté de coder les entrées et la sortie sur 6 bits pour satisfaire la majorité des applications connues **[42]**. Ainsi que le choix du FPGA est lié à plusieurs facteurs de type technologiques, coût, dimension (taille) et plus particulièrement aux possibilités d'intégration dans d'autres composants virtuels IP **[43][44].**

Les caractéristiques du circuit FPGA Virtex (4vlx25ff668) utilisé sont résumées dans le tableau III.2, (pour plus d'information sur les circuits FPGA, voir annexe (B))

Tableau III. 2: *Caractéristiques du circuit FPGA Virtex (4vlx25ff668)*

Elément	Nombre
Nombre de Slices	10752
Nombre de Flip Flops	21504
Nombre de LUTs	21504
Nombre de IOBs	450
Nombre de FIFO16/RAMB16s	72
Nombre de GCLKs	32

III. 3.2. Bloc de fuzzification

a. Module « degrés d'appartenance »

Les fonctions d'appartenance et leurs degrés respectifs sont codés sur 3 bits. Pour cela la longueur totale de la RAM des degrés d'appartenance pour les deux entrées est de $2^6 = 64$. Ainsi, un bloc de RAM de 64 bits est implanté pour stocker tous les degrés d'appartenance des deux entrés X0 et X1 après discrétisation des entrées du système, on définit les valeurs à enregistrer dans les RAM (figure III.13).

La figure III.13 représente la RAM des « degrés d'appartenance **»** avec les entrées/sorties correspondantes.

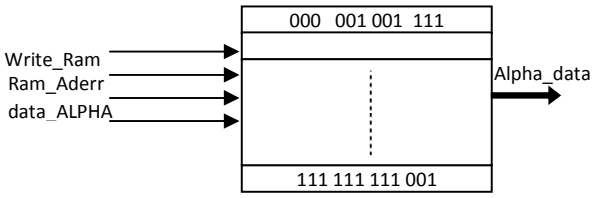


Figure III.13: *Degrés d'appartenance*

Où les entrées : Write_Ram, Ram_Aderr, Data_ALPHA.
Les sorties: Alpha_Data (données pour le bloc d'inférence).

Modes de lecture et d'écriture :

Mode Ecriture : Il faut charger la RAM par les valeurs obtenues après discrétisation comme suit : -

Mettre le signal "Write_Ram" = 1.

- Envoi de l'adresse sur le bus d'adresse de la RAM "Ram_Aderr".

Envoi de la valeur de alpha sur le bus "Data_ALPHA" (figure III.14).

Mode Lecture : La lecture de la RAM se fait lorsque le système se met en marche comme suit:

- En positionnant le signal "Write_Ram" à zéro.
- Lecture du contenu d'une adresse (figure III.14).

Dans ce module l'entrée représente est utilisée directement comme pour l'adressage des degrés d'appartenance dans la mémoire, cela nous permet de gagner beaucoup de temps et réduit considérablement les procédures d'affectation et donc nous évite ainsi d'utiliser un module supplémentaire de génération d'adresses.

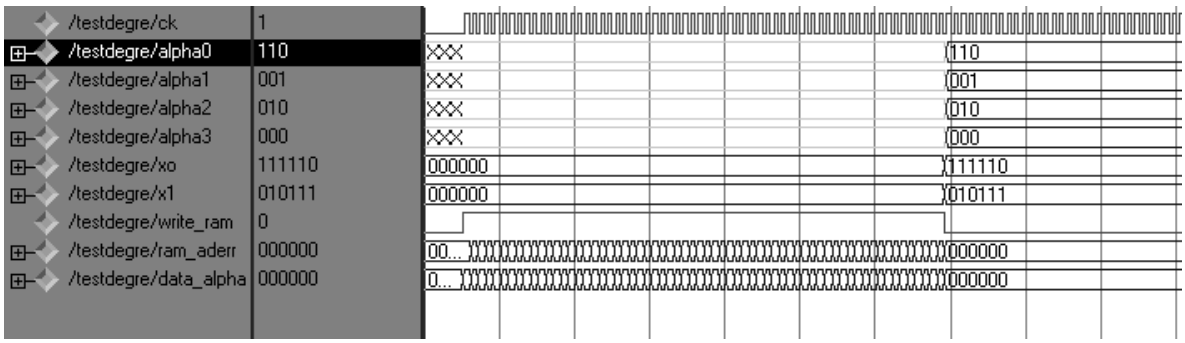


Figure III.14: *Chargement de la RAM*

b. Module « interval_selector »

On a utilisé aussi un petit bloc mémoire RAM pour contenir les points finaux pour les 7 MFS pour chaque variable d'entrée Xo et X1, les 7 MFs définissent 6 intervalles qui sont codés sur 3 bits (000, 101) donc uniquement 5 données sont suffisantes pour l'identification et la sélection des intervalles et par conséquent les sous-ensembles Fs auxquels la variable Xi appartient. Ces 5 données de 6 bits exigent un espace de RAM : 5 X 6 =30 bits pour chaque entrée Xi (figure III.16).

Après identification et spécification du système (fuzzification), on définit les valeurs à enregistrer dans la RAM (figure III.15), la figure représente la RAM de « degrés d'appartenance » avec les entrées/sorties correspondantes.

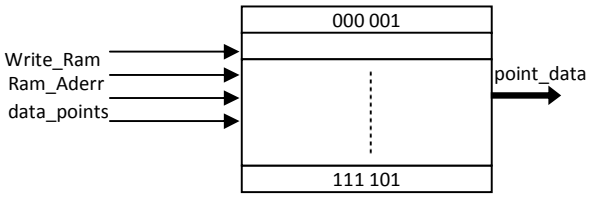


Figure III.15: *Les limites des intervalles*

Où les entrées : Write_Ram, Ram_Aderr, Data_points.

La sortie : point_Data (données pour l'analyse et la comparaison).

Modes de lecture et d'écriture :

Mode Ecriture : On charge la RAM par les points finaux des intervalles comme suit:

- Mettre le signal "Write_Ram" = 1.

- Envoi de l'adresse sur le bus d'adresse de la RAM "Ram_Aderr"(figure III.17)
- Envoi de la valeur du point final de l'intervalle M sur le bus "Data_ALPHA".

Mode Lecture : La lecture de la RAM se fait lorsque le système se met en marche comme suit :

- On positionne le signal "Write_Ram" à zéro.
- Lecture du contenu d'une adresse (figure III.17)

Identification d'intervalle

L'identification des intervalles mises en jeu se fait par comparaison de la valeur d'entrée présente X_i avec les différentes valeurs de données (a, b, c, d, ….) stockées dans la RAM.

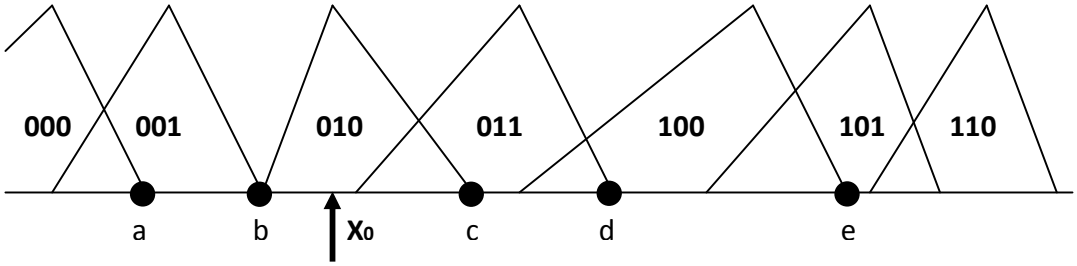


Figure III.16: *Les points (données) d'identification d'intervalle*

Comparaison SI Xo < c Alors

SI Xo < a Alors intout0 <="**000**" -- 1er interval
SINON SI Xo < b Alors intout0 <="**001**" -- 2eme interval
SINON intout0<="**010**"-- 3eme interval

SINON SI Xo >= e Alors intout0 <="**101**"-- 6me interval
SINON SI Xo >= d Alors intout0 <="**100**"-- 5eme interval
SINON intout0<= "**011**"-- 4eme interval
(Voir figure III.16) .

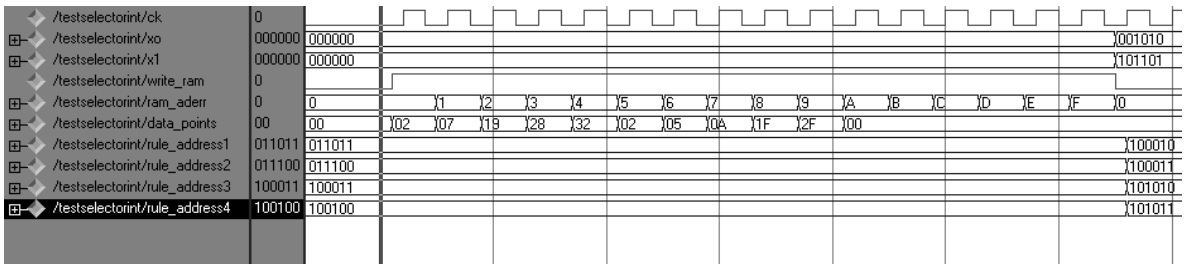


Figure III.17: *Chargement de la RAM (intervalle) et simulation de la génération d'adresses*

Identification des règles actives

Les codes de 3 bits des deux intervalles de chacune des entrées X0 et X1 forment entre eux 4 mots de 6 bits par simple concaténation, ces derniers représentent les adresses des 4 règles actives.

La simulation du module de fuzzification (programme VHDL relatif à la fuzzification nommée « Fuzzification » dans le programme principal de notre contrôleur) a produit les résultats consignés dans la figure III.18.

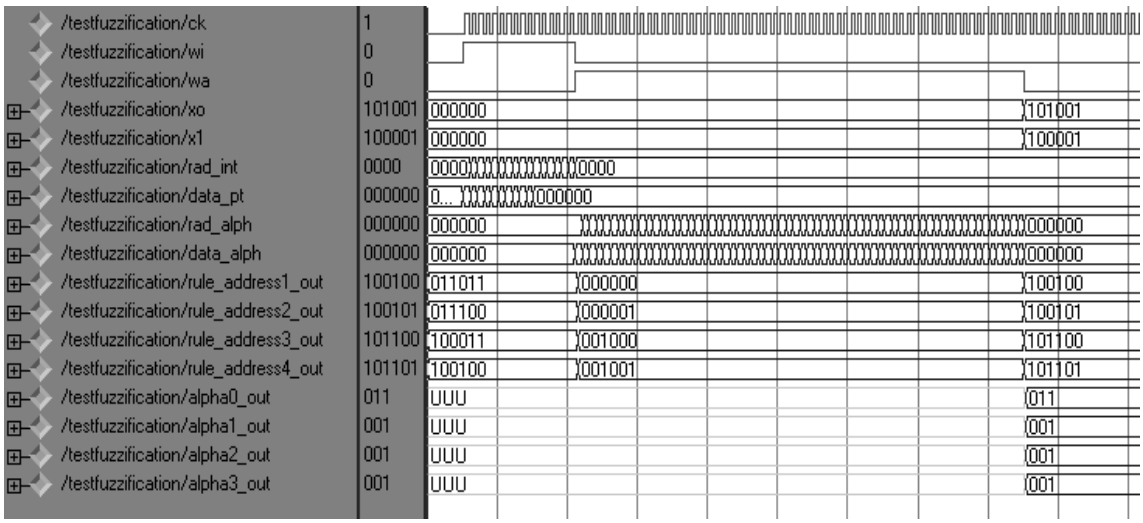


Figure III.18: *Simulation du module de fuzzification*

On note que la fuzzification est effectuée en un top d'horloge (figure III.18.).

Résultat de synthèse du bloc de fuzzification :

Après la synthèse, la proportion d'utilisation des éléments du circuit 4vlx25ff668-12, est donnée dans le tableau III.3, aussi la fréquence maximale obtenue est de **1167.679MHz**).

Tableau III. 3 : *Pourcentage d'utilisation d'éléments d'FPGA Virtex (4vlx25ff668)*

Elément	Utilisation %
Nombre d'IOBs	16 %
Nombre de FIFO16/RAMB16	2%
Nombre de GCLKs	3%

D'où, on peut conclure que deux blocs RAM de type RAMB16s sont utilisés pour implanter ce module. Chaque RAM utilise des entrées sorties de type FIFO16/RAMB16.

Après synthèse, le schéma RTL du bloc de fuzzification est montré sur la figure III.20 :

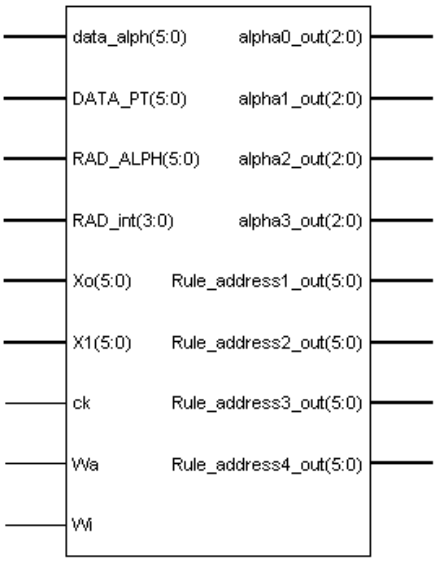


Figure III.19: *Bloc de Fuzzification*

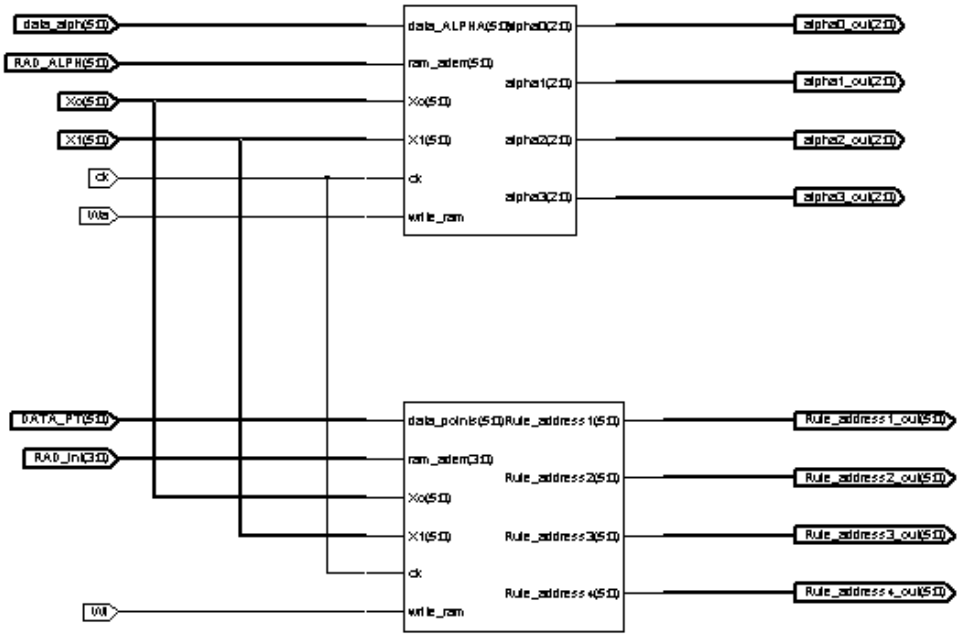


Figure III.20: *Schéma(RTL) interne du bloc de Fuzzification*

III. 3.3. Bloc d'inférence et règles

a. Module « mémoire des règles »

Les **conséquents** (les conclusions) dans notre étude sont stockés dans la RAM. L'adressage de notre RAM des règles floues se fait par un générateur d'adressage. Après identification des intervalles d'appartenance de chaque entrée, le générateur d'adresse concatène le code de l'intervalle concerné M (3 bits) (voir figure) avec le code de l'intervalle adjacent M+1 (3bits) pour former un mot de 6 bits qui est l'adresse de la règle active concernée par la combinaison de X0 et X1 (Tableau III.4). Chaque adresse contient une conclusion d'une règle floue codée sur 3 bits comme le montre le tableau III.4:

Exemple :

Tableau III. 4: *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

Les règles sont codées sur 3 bits, et comme on a 49 règles possibles alors la longueur totale de la RAM est de 64 X 3 bits, mais les 15 adresses restantes ne peuvent pas être générées par notre générateur d'adresse car elles sont inutilisables dans notre système (figure III.21).

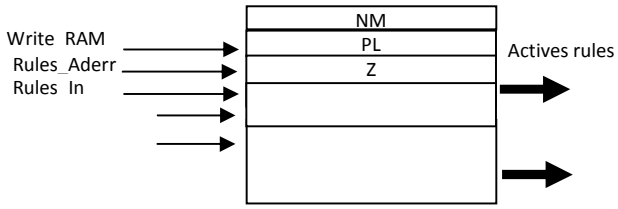


Figure III.21: *Bloc Mémoire des règles*

Les entrées : Write_RAM, Rules_ADERR

Les sorties : Actives Rules, (entrée du bloc défuzzification).

Modes de lecture et d'écriture : (figure III.22)

Mode Ecriture : Les conclusions des règles sont chargées dans la RAM des règles selon la procédure suivante :

- Mettre le signal "Write_Ram" = 1.
- Mettre le signal : "Write_RAM" = 1 (Validation de l'écriture).
- Envoi de l'adresse de la règle sur le bus : "Rules_Aderr".
- Envoi de l'antécédent sur le bus : "Rules_In" (Donnée à enregistrer).

Mode Lecture : La lecture de la RAM des règles étant automatique commence juste au démarrage du système, où la RAM est directement adressée par le Générateur d'adresses correspondantes aux intervalles d'appartenance des variables d'entrée tout en mettant le signal "Write_Rules" à zéro. D'où on récupère les conséquents correspondants (Z1, Z2, Z3, Z4).

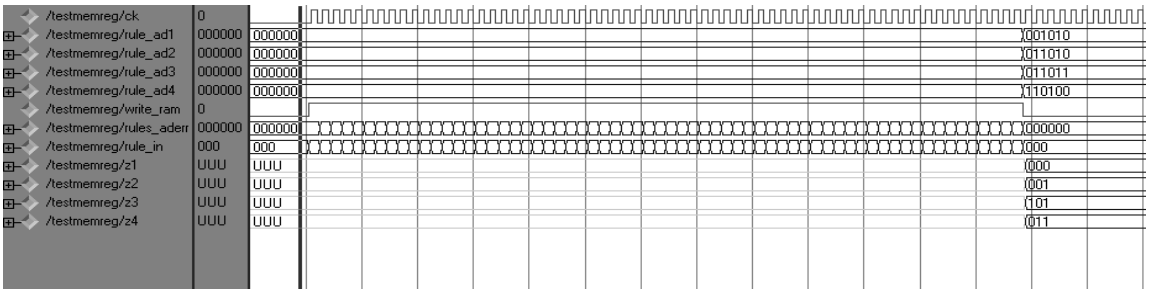


Figure III.22: *Simulation du module des règles.*

Après la synthèse de ce module « **mémoire de règles »,** la proportion d'utilisation des éléments de l'FPGA (4vlx25ff668-12), est donnée dans le tableau III.5 comme suit :

Tableau III. 5: *Pourcentage d'utilisation d'éléments d'FPGA Virtex (4vlx25ff668)*

Elément	Utilisation %
Nombre d'IOBs	10 %
Nombre de FIFO16/RAMB16	5 %
Nombre de GCLKs	3%

Ce qui correspond à l'utilisation de 4 blocs RAM de type RAMB16s.

b. Module « inférence MIN_PROD »

Selon l'entrée **MIN_PROD** 0/1 **:** on peut sélectionner deux modèles de raisonnement flou, et donc choisir soit la t-norme MIN ou bien la t-norme PROD (MAX), (figure III.23)

Ce bloc reçoit les 4 valeurs (αXo0, αXo1, αX11, αX12), les 4α sont groupés deux à deux : (αXo0 avec αX11, αXo0 avec αX12, αXo1 avec αX11 et αXo1 avec αX12), d'où on aura quatre combinaisons qui correspondent aux quatre règles actives, on pourra selon le signal MIN_PROD faire la comparaison et déduire soit le minimum pour chaque combinaison ou bien le produit normalisé pour chaque combinaison.

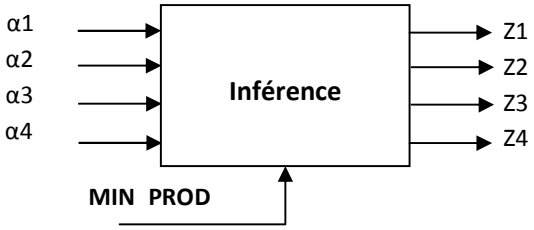


Figure III.23: *Le bloc d'inférence*

Après simulation :

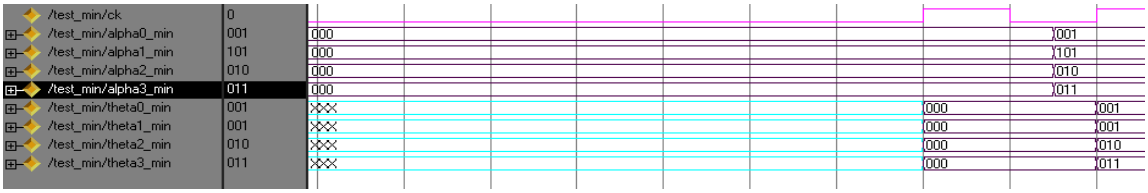


Figure III.24: *Simulation du bloc « Min »*

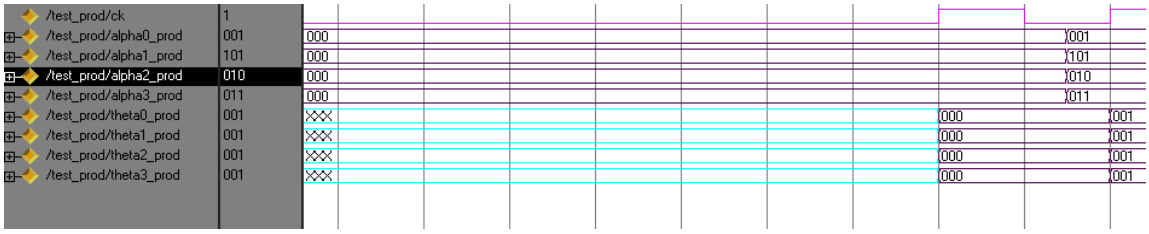


Figure III.25: *Simulation du bloc « Prod »*

On constate que les deux modules MIN et PROD ont le même pourcentage d'utilisation des éléments de l'FPGA :

Tableau III. 6: Pourcentage *d'utilisation d'éléments d'FPGA*

Elément	Utilisation %
Nombre d'IOBs	5 %
Nombre de GCLKs	3%

Résultat de synthèse du bloc d'Inférence et Règles :

Après synthèse, la proportion d'utilisation des éléments du circuit 4vlx25ff668-12, est donnée dans le tableau, la fréquence maximale obtenue est de **846.525MHz**.

Tableau III. 7: Pourcentage d'utilisation d'éléments d'FPGA Virtex (4vlx25ff668)

Elément	Utilisation %
Nombre d'IOBs	16 %
Nombre de FIFO16/RAMB16	5 %
Nombre de GCLKs	3 %

4 blocs RAM de type RAMB16s sont utilisés.

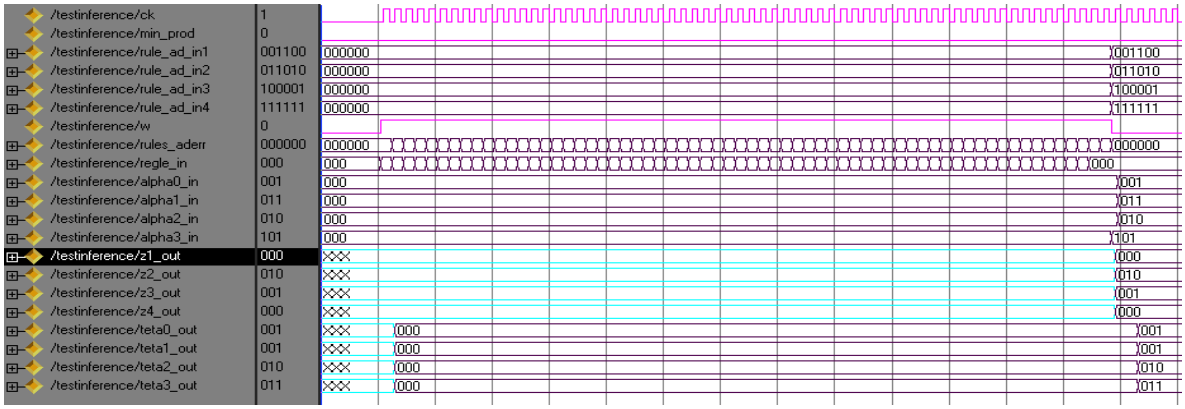


Figure III.26: *La simulation du module d'inférence et Règles.*

On remarque que le module d'inférence donne une réponse à chaque deux top d'horloge (figure III.26).

Après la synthèse le schéma RTL du bloc d'inférence est montré sur la figure III.27 :

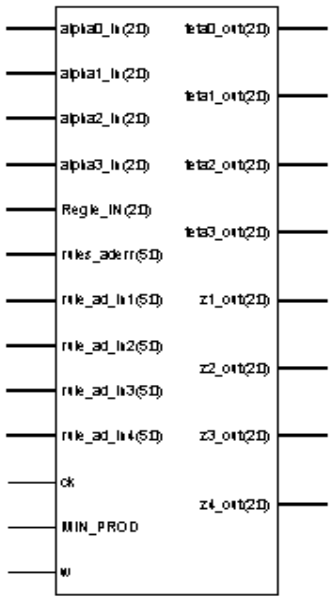


Figure III.27: *Schéma RTL du bloc d'inférence*

III. 3.4. Bloc de défuzzification

Ce bloc reçoit les valeurs des Θi (i appartient [0,3]) et reçoit à partir de la mémoire des règles les code des conséquents de la règle Zj, le but est d'obtenir la valeur de sortie en utilisant la formule :

$$\text{Fuzz output} = \sum Zj^* \Theta i \,/ \sum \Theta i$$

D'où trois opérations mathématiques élémentaires (Figure III.28) sont nécessaires pour l'obtention du résultat en sortie qui sont :

1) La somme des Θi **: ∑ Θi.**
2) La multiplication **: Zj* Θi** et puis donc la somme **∑ Zj* Θi.**
3) La division **: ∑ Zj* Θi /∑ Θi.**

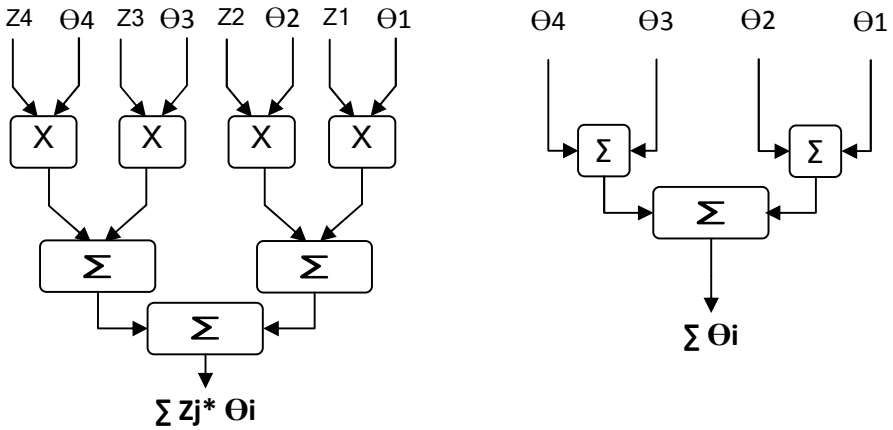


Figure III.28: *Multiplication et addition*

Notre module de défuzzification se compose de plusieurs étapes de calcule :

- Calcule de la somme des degrés **∑ Θi**.
- Calcule de la multiplication rapide (**Zj* Θi**) par des unités nommées « **multip** ».
- Calcule de la somme des multiplications **∑ Zj* Θi**.
- Calcule de la division (**∑ Zj* Θi /∑ Θi**) par une unité nommée « **divider** ».

Figure III.29: *Bloc « Multip » [Zj* Θi]*

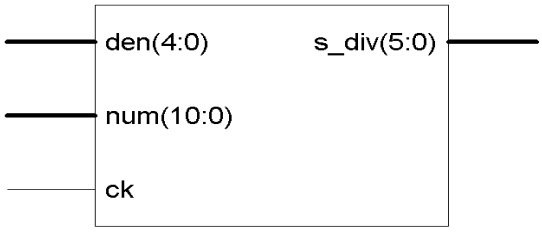


Figure III.30: *Bloc « Divider » [∑ Zj* Θi /∑ Θi]*

Les différentes phases de traitement associées à ce bloc de défuzzification sont représentées dans la figure III.31 :

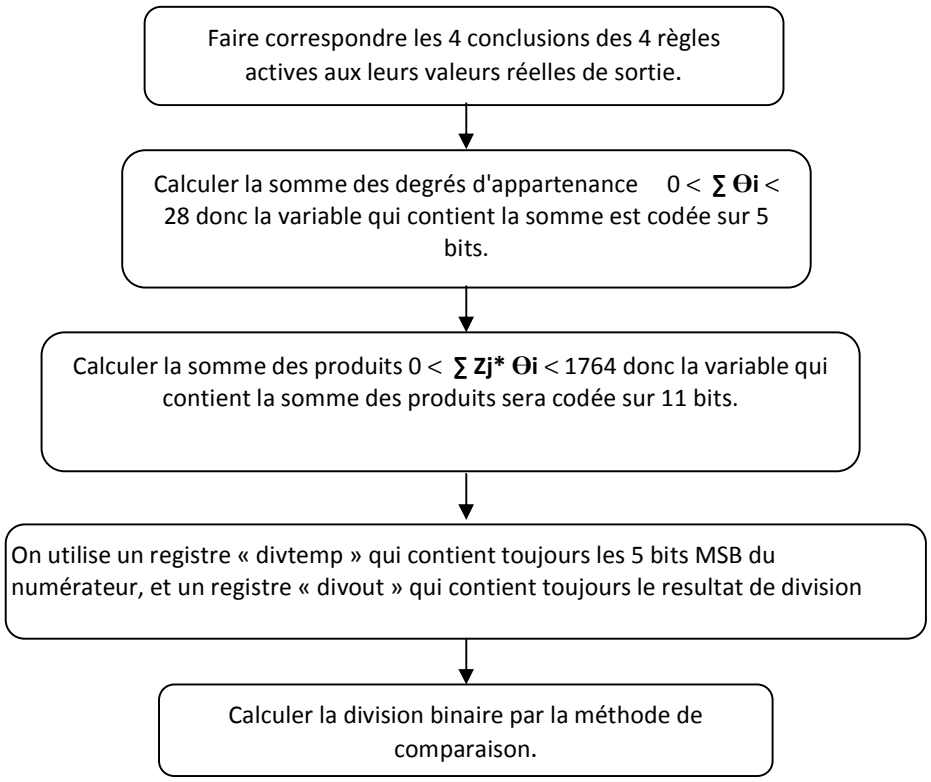


Figure III.31: *Les différentes phases de traitement de la défuzzification*

Les premières phases de calcul (addition et produit) peuvent être effectuées en un seul cycle d'horloge. En ce qui concerne la division où nous avons utilisé la méthode de comparaison et soustraction, elle est effectuée aussi en un seul top d'horloge et elle n'engendre qu'une faible surface par rapport aux autres méthodes de division usuelles.

Tableau III. 8: *Pourcentage d'utilisation d'éléments d'FPGA du module de la division.*

Elément	Utilisation %
Nombre de IOBs	12 %
Nombre of GCLKs	3 %

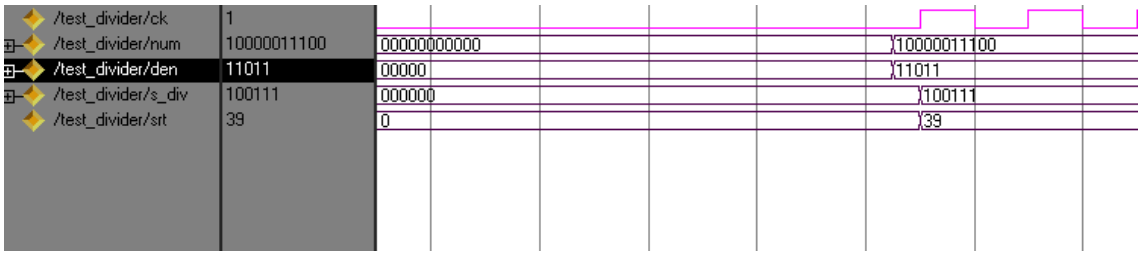


Figure III.32 : *Simulation de la division*

La simulation du module de division est représentée sur la figure III.32.

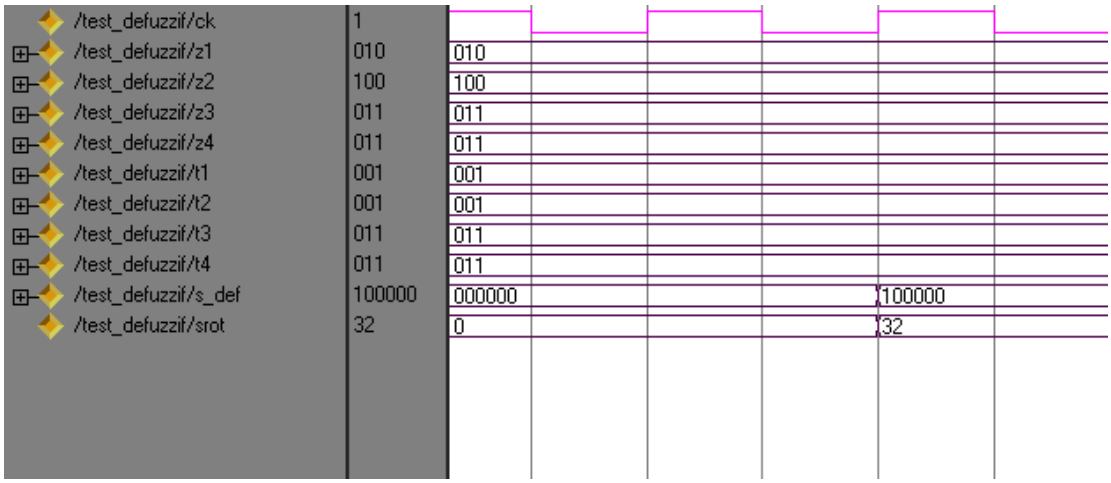


Figure III.33: *Simulation du module de défuzzification*

Le résultat de défuzzification est obtenu après deux tops d'horloge (figure.III.33). .

Résultats de synthèse du bloc de defuzzification :

La fréquence maximale obtenue est de 214.675MHz et les détails d'utilisation des blocs des FPGA sont donnés dans le tableau III.9.

Tableau III.9: *Pourcentage d'utilisation des éléments du FPGA*

Elément	**Utilisation %**
Nombre de Slices	1 %
Nombre de Flip Flops	0 %
Nombre de LUTs	0 %
Nombre de IOBs	14 %
Nombre de GCLKs	3 %

Pour la génération des différentes fonctions de calcul, ce module de défuzzification n'utilise ni bloc RAM, ni LUT (Look up table).

Une étude comparative concernant les surfaces et les fréquences obtenues pour chacun des modules du contrôleur a permis de conclure qu'effectivement la défuzzification occupe une grande partie de la surface du contrôleur, et sa fréquence reste très inférieure aux fréquences de travail des autres modules, de ce fait ce module impose sa fréquence à tous les autres modules lors de leur assemblage. Car on effectue des opérations synchronisées et sans latence.

III. 3.5. Module "contrôleur"

C'est le programme principal qui assemble tous les modules précédemment décrits pour l'implémentation finale du contrôleur flou (figure III.34). Le programme principal du contrôleur développé est donné en Annexe (A).

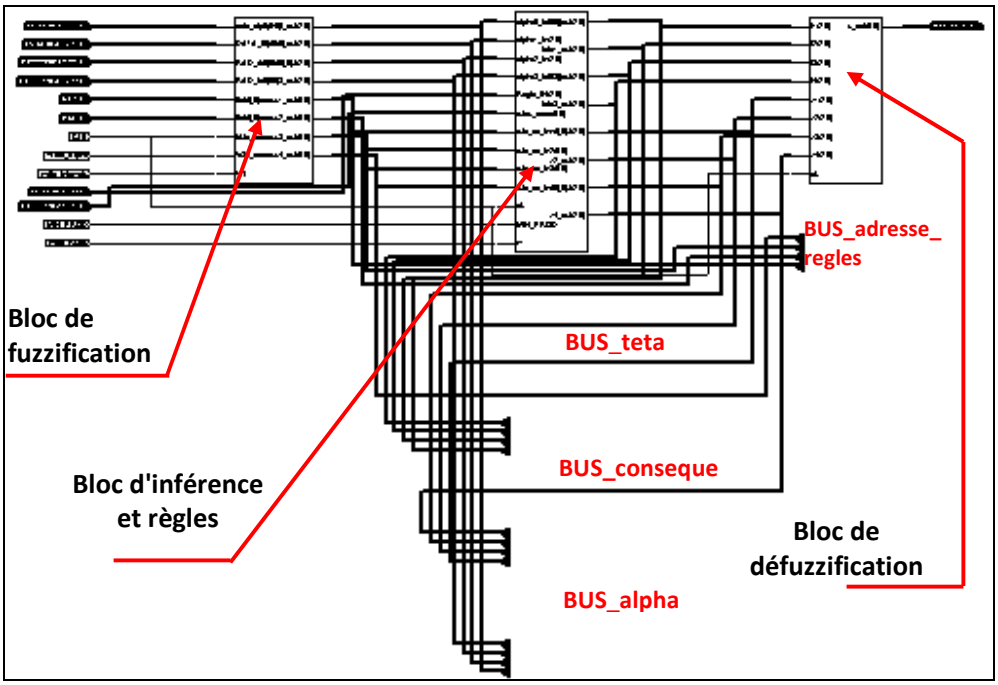


Figure III.34: *Schéma du contrôleur produit par l'outil (Web-Pack. de Xilinx).*

Après l'implémentation du contrôleur développé, nous avons pu consigner les résultats ci-dessous :

Résultats de synthèse :

La fréquence maximale obtenue sur le FPGA Virtex (4vlx25ff668) est de 165.780MHz.

Tableau III. 10: *Pourcentage d'utilisation des éléments du FPGA.*

Elément	Utilisation %
Nombre de Slices:	2%
Nombre de Flip Flops	1%
Nombre de LUTs:	1%
Nombre de IOBs:	19%
Nombre de FIFO16/RAMB16s:	8%
Nombre de RAMB16 utilisé:	**6**
Nombre de GCLKs:	3%

Les entrées du contrôleur : CLK, MIN_PROD, Write_Alpha, write_Rules, write_intervals, Address_Alpha, Address_Rules, Address_Points, DATA_Alpha, DATA_rules, DATA_Points, X0, X1.

Les bus intérieurs : BUS_adresse_regles, BUS_alpha, BUS_teta, BUS_consequent.

Les sorties du contrôleur : SORTIE.

Simulation : Le diagramme temporel sur la figure III.35 montre le fonctionnement du contrôleur.

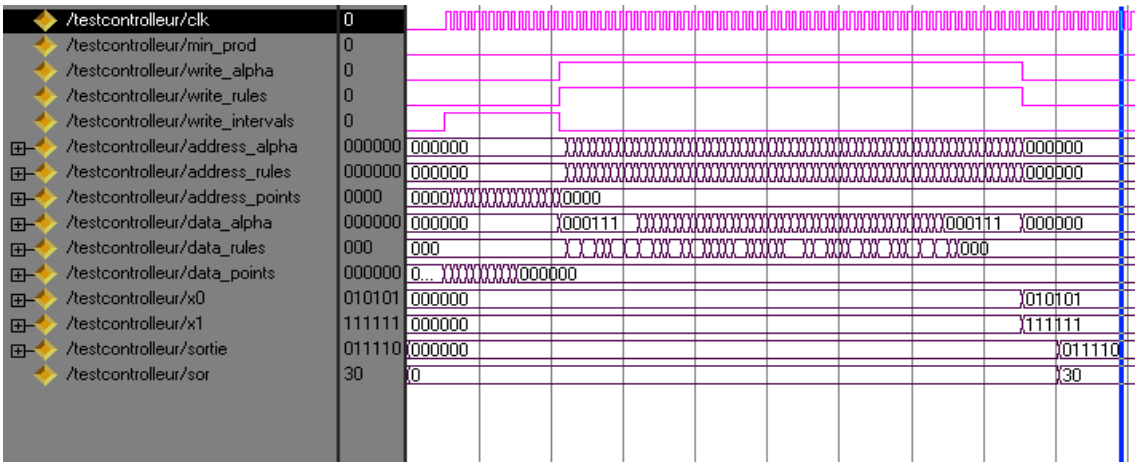


Figure III.35 : *Le fonctionnement du contrôleur*

Implantation physique : La phase d'implantation passe par 3 étapes

- Traduction (translation).
- Mappage (Mapping).
- Placement et routage.

La structure interne du Virtex (4vlx25ff668-12) après implantation des différents modules du contrôleur flou, est représentée sur la figure III.36.

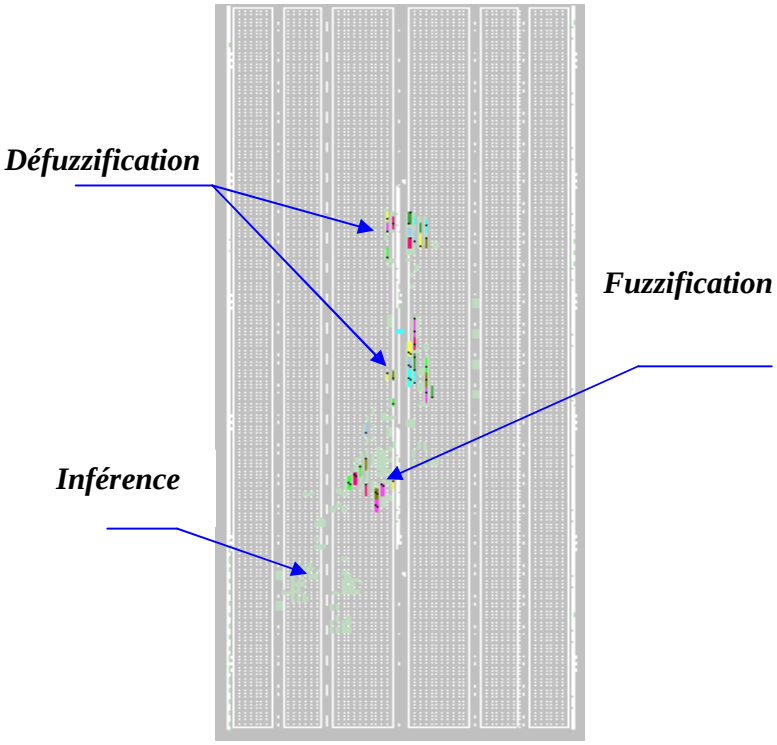


Figure III.36: *Implantation physique du contrôleur sur le FPGA*

Propriétés :

Après implémentation le récapitulatif des éléments utilisé dans l'FPGA est comme suit :

- 86 IOBs
- 6 Blocks RAMs
- 417 Générateurs de Fonctions.
- 223 Carry Symbols
- 248 Flip Flops
- 1 Global Buffer

Performances :

Le bloc de défuzzification représente 51.17% des cellules du contrôleur, ce qui montre la nécessité de continuer à rechercher les meilleures optimisations possibles même si c'est aux dépends de la précision.

Le contrôleur donne une réponse tous les cinq tops d'horloge, un temps de réponse estimé à 30 ns, et sa vitesse est de 165.780MHz soit plus de 165 MFIPS (Méga Fuzzy Inference Per Second).

III. 4. Synthèse du contrôleur avec d'autres types de FPGA

Nous avons essayé d'utiliser d'autres types de FPGA afin d'amélioré les performances (espace utilisé, Fréquence etc…), quelques exemples sont cités ci-dessous, mais pour plus de détails on trouve une étude en annexe (C) où on a implémenté notre contrôleur sur toutes les familles FPGA disponibles dans l'application ISE 7.1i :

a. FPGA qui permettent l'intégration de notre contrôleur :

- FPGA virtexE (v50efg256-6):

Tableau III. 11: résultats de synthèse

Elément	Utilisation %
Number of Slices:	273 out of 768 35%
Number of Slice Flip Flops:	236 out of 1536 15%
Number of 4 input LUTs:	408 out of 1536 26%
Number of bonded IOBs:	86 out of 180 47%
Number of BRAMs:	6 out of 16 37%
Number of GCLKs:	1 out of 4 25%

(Fréquence maximale: 78.839MHz)

- FPGA virtexE (v50ecs144-8):

Tableau III. 12: résultats de synthèse

Elément	Utilisation %
Number of Slices:	273 out of 768 35%
Number of Slice Flip Flops:	236 out of 1536 15%
Number of 4 input LUTs:	408 out of 1536 26%
Number of bonded IOBs:	86 out of 98 87%
Number of BRAMs:	6 out of 16 37%
Number of GCLKs:	1 out of 4 25%

(Fréquence maximale: 103.929MHz)

- FPGA Spartan2E (2s50eft256-6) :

Tableau III. 13: résultats de synthèse

Elément	Utilisation %
Number of Slices:	273 out of 768 35%
Number of Slice Flip Flops:	236 out of 1536 15%
Number of 4 input LUTs:	408 out of 1536 26%
Number of bonded IOBs:	86 out of 182 47%
Number of BRAMs:	6 out of 8 75%
Number of GCLKs:	1 out of 4 25%

(Fréquence maximale: 78.839MHz)

- FPGA Spartan3 (3s200pq208-5):

Tableau III. 14: résultats de synthèse

Elément	Utilisation %
Number of Slices:	251 out of 1920 13%
Number of Slice Flip Flops:	200 out of 3840 5%
Number of 4 input LUTs:	374 out of 3840 9%
Number of bonded IOBs:	86 out of 141 60%
Number of BRAMs:	6 out of 12 50%
Number of GCLKs:	1 out of 8 12%
Number of MULT18X18s	4 out of 12 33%

(Fréquence maximale: 111.211MHz)

Notre choix porté sur le FPGA Virtex4 (4vlx25ff668) est bien justifié, si on remarque la dégradation des performances du notre contrôleur dans les exemples expérimentés (tableaux III. 11, 12, 13, 14):

- Augmentation d'utilisation des éléments de le FPGA.
- Diminution de la fréquence maximale.

b. FPGA ne permettant pas l'intégration du contrôleur :

- FPGA Spartan2 (2s15cs144-6):

Tableau III. 15: résultats de synthèse

Elément	Utilisation %
Number of Slices:	273 out of 192 142% (*)
Number of Slice Flip Flops:	236 out of 384 61%
Number of 4 input LUTs:	406 out of 384 105% (*)
Number of bonded IOBs:	86 out of 90 95%
Number of BRAMs:	6 out of 4 150% (*)
Number of GCLKs:	1 out of 4 25%

- FPGA Spartan3 (3s50pq208-5):

Tableau III. 16: résultats de synthèse

Elément	Utilisation %
Number of Slices:	251 out of 768 32%
Number of Slice Flip Flops:	200 out of 1536 13%
Number of 4 input LUTs:	374 out of 1536 24%
Number of bonded IOBs:	86 out of 124 69%
Number of BRAMs:	6 out of 4 150% (*)
Number of GCLKs:	1 out of 8 12%
Number of MULT18X18s	4 out of 4 100%

(*) On voit d'après les tableaux III. 15 et III.16 que les FPGAs qui ne permettent pas l'intégration du contrôleur, elles ont un dépassement soit dans le nombre de blocks RAMs ou dans le nombre des IOB utilisés et parfois dans le nombre des entrées LUTs.

III.5. Comment tester le fonctionnement d'un contrôleur flou [12]

a. Comment voir si le contrôleur conçu répond vraiment au raisonnement flou espéré ?

La méthode d'essai la plus largement répandue est « la simulation ».

Le premier et le plus important critère que le contrôleur flou doit satisfaire est sa réponse au bon sens des connaissances professionnelles et la connaissance de l'expert en matière de domaine d'application.

Certaines recommandations sont comme suit :

- Il est préférable d'examiner le fonctionnement du contrôleur flou tout en appuyant sur son plan ou sur son modèle, s'il est disponible.
- Pendant la simulation, essai des différentes combinaisons d'entrées et observation les sorties correspondantes
- Essai des valeurs d'entrées suspectées à gâcher le fonctionnement du contrôleur flou synthèse, telles que :
  - des valeurs d'entrée se trouvent aux extrémités de l'univers du discours.
  - des valeurs d'entrée se trouvent aux extrémités de différentes fonctions d'appartenances.
  - des valeurs d'entrée correspondantes aux chevauchements de fonctions d'appartenances.

On peut également examiner le fonctionnement du contrôleur flou sur la base de sa « surface de contrôle ».où on doit vérifier que la surface du contrôle est conforme aux espérances ou bien à la table de règles autrement dit (au raisonnement voulu). Voir figure III.38, figure III.39, figure III.40. Le manque de conformité est habituellement dû à un de trois problèmes suivants :

- Règles indiquées fausses.
- Fonctions d'appartenance définies fausses.

- Opérateurs flous inadéquats.

D'où on doit vérifier les changements de la surface et sa concentration sur les différentes parties. Ceci permettra de voir si la surface a des anomalies inattendues.

b. Testbench du contrôleur

Le testbench **[40]** est écrit de façon à générer toutes les combinaisons d'entrées (X0, X1) possibles qui permettent d'expérimenter le système (contrôleur flou) et analyser les différentes réponses du contrôleur afin d'avoir une idée globale sur le comportement (contrôle) du contrôleur. La matrice des règles du tableau III.17 est développée :

Tableau III. 17 : *Table des règles (7X7)*

U		**X1**						
		NL	**NM**	**NS**	**Z**	**PS**	**PM**	**PL**
X0	**NL**	NL	NL	NL	NL	NL	NL	NL
	NM	NM	NM	NM	NL	NL	NL	NL
	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
	PL	PL	PL	PL	PL	PL	PL	PL

Après simulation :

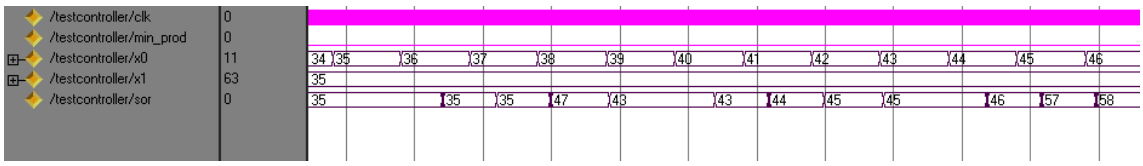


Figure III.37: *Simulation « Test Bench »*

La surface de commande donnée par le testbench est illustrée dans la figure III.38, elle est élaborée avec l'outil de visualisation de MATLAB 7.1.

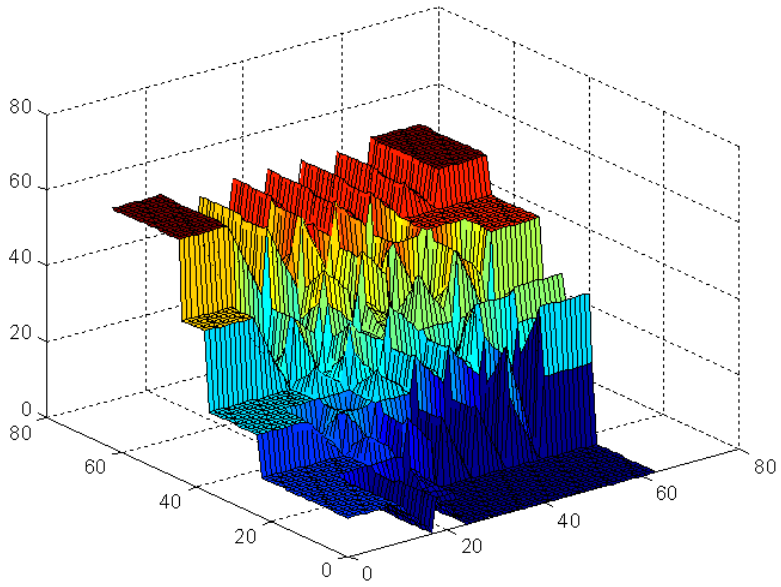


Figure III.38: *Surface du contrôleur flou de la table des règles (Tableau III.17)*

Pour plus d'assurance que le contrôleur fait bien sa tâche et qu'il reproduise bien le raisonnement flou implémenté dans sa matrice de règles, on a simulé différents cas (quelques matrices) où on a donc mis tous les alphas à 1 avec des intervalles égaux pour produire une certaine symétrie dans la surface générée et faciliter ainsi les calculs.

Exemple 1 :

Tableau III. 18: ***Table des règles* (7x7)**

U								
			NM					
		NL	NM	NS	Z	PS	PM	PL
	NM	NL	NM	NS	Z	PS	PM	PL
X 0		NL	NM	NS	Z	PS	PM	PL
		NL	NM	NS			PM	PL
		NL	NM	NS			PM	PL
		NL	NM	NS	Z	PS	PM	PL
		NL	NM	NS	Z	PS	PM	PL

Le testbench a produit la surface de contrôle de la figure III.39 :

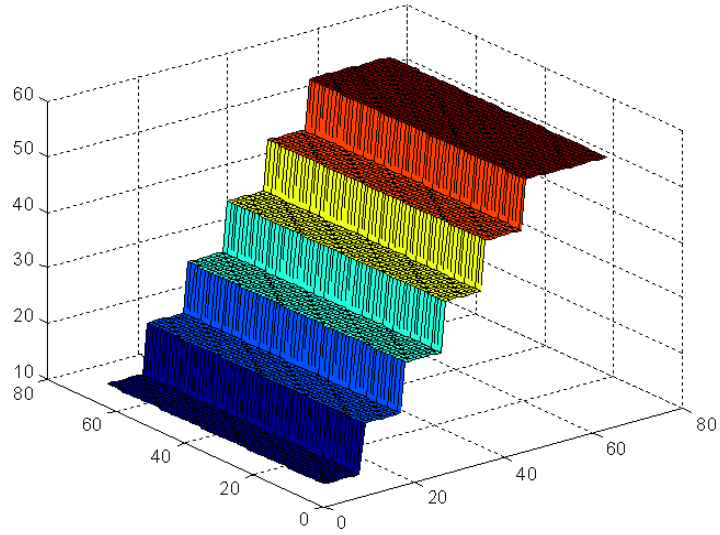


Figure III.39: ***Surface du contrôleur flou (αi =1) de la table des règles (Tableau III.18)***

Exemple 2 : **Alpha = 1, avec la matrice où PL (110) partout.**

Tableau III. 19: *Table des règles (7X7)*

U							
		PL	PL	PL	PL	PL	PL	PL
		PL	PL	PL	PL	PL	PL	PL
		PL	PL	PL	PL	PL	PL	PL
		PL	PL	PL			PL	PL
		PL	PL	PL			PL	PL
		PL	PL	PL	PL	PL	PL	PL
		PL	PL	PL	PL	PL	PL	PL

Le testbench a donné la surface de contrôle de la figure III.40 :

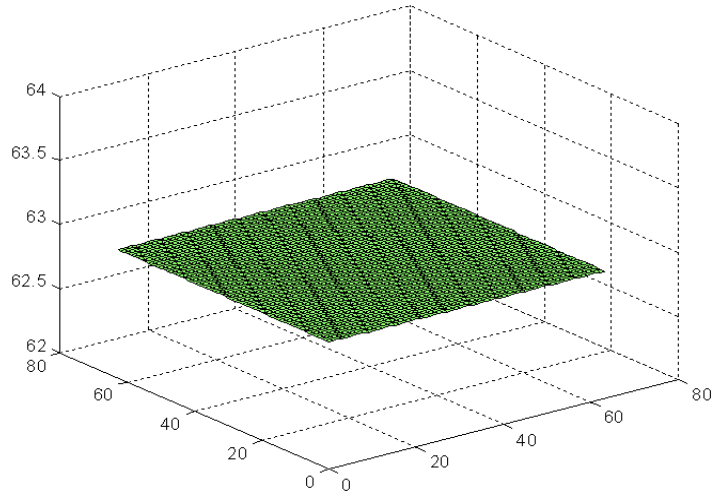


Figure III.40 : *Surface du contrôleur flou (αi =1) de la table des règles (Tableau III.19)*

III. 6. Test du contrôleur flou- DFT

La structure du contrôleur est modifiée afin d'insérer un mécanisme de test basé sur la compaction où les éléments additifs dûs à la DFT (Design For Testability) sont : un générateur de vecteurs de test, un analyseur de signatures, un registre pour stocker la signature correcte du contrôleur, une entrée d'initialisation du test et une sortie pour vérifier le bon ou le mauvais fonctionnement du contrôleur.

Pour tester les mémoires embarquées du bloc de fuzzification et des règles, on a eu recours au " *Memory Bist*" dans la structure du système de test "Bist global" et qui comprend les éléments suivants **[45] [46]** :

- LFSR : Un *contrôleur Bist* qui génèrera les vecteurs de test, les données et les signaux de contrôle.
- MISA : qui permet la compression simultanée de plusieurs séquences.
- Le comparateur : qui permet l'analyse de signature. (Voir chapitre II).

La figure III.41 représente l'architecture proposée pour le test du contrôleur.

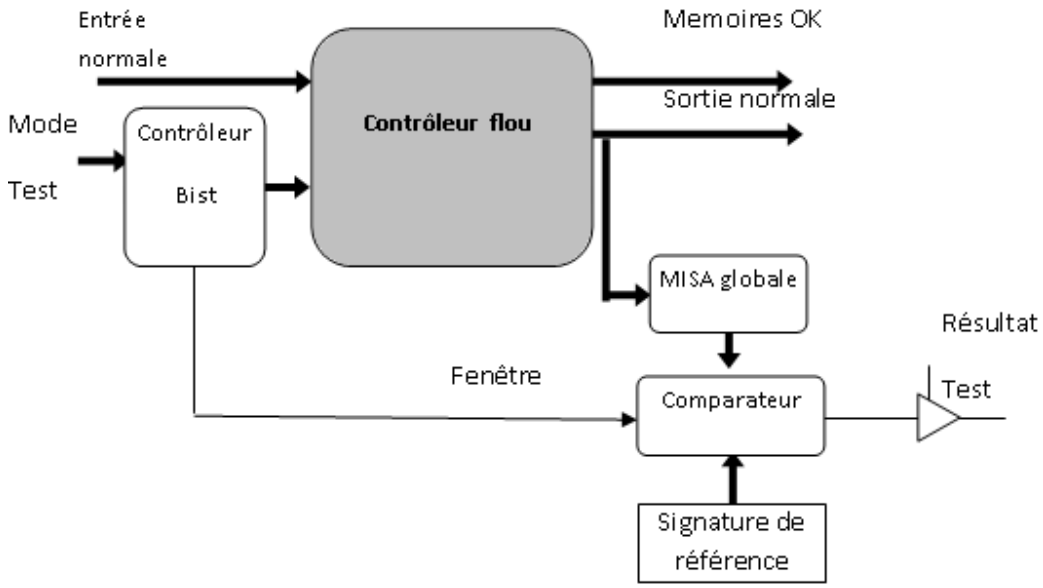


Figure III.41: *Contrôleur flou avec test intégré*

La figure III.42 représente la structure du "*Memory Bist* :

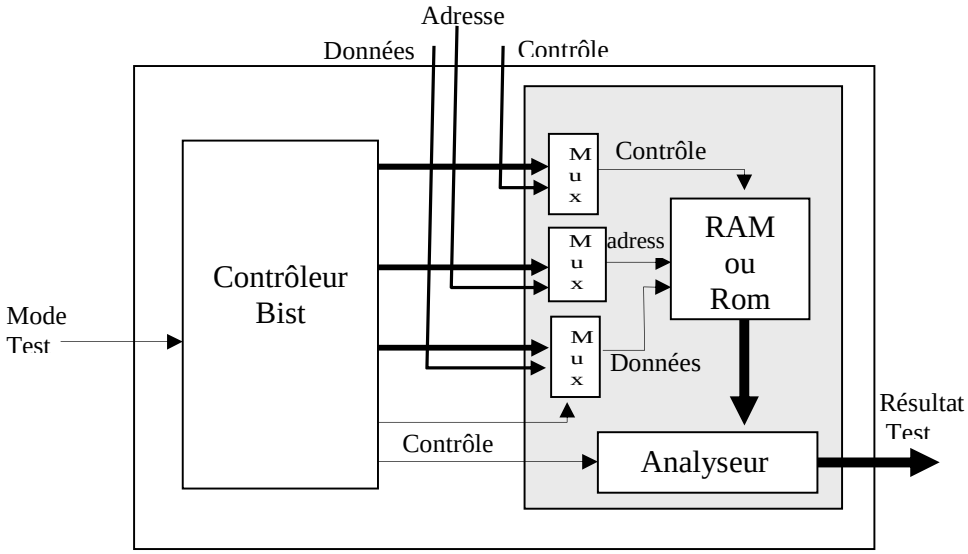


Figure III.42 : *Memory Bist*

L'architecture globale du contrôleur munie de son circuit de test est représentée sur la figureIII.43.

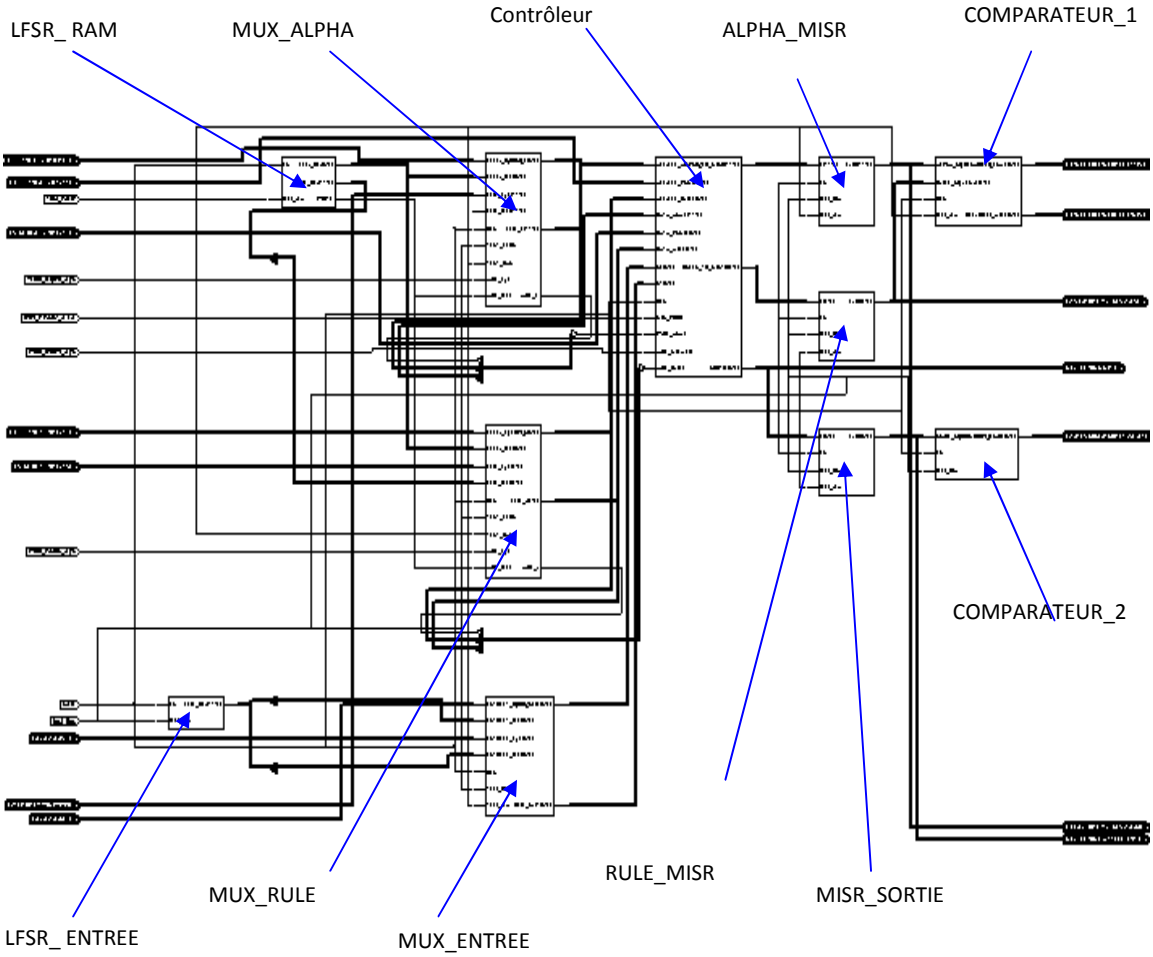


Figure III.43: *Architecture globale du contrôleur (Schéma de Bloc)*

Le LFSR utilisé (est un registre de 6 bits), il génère de manière pseudo aléatoire les 64 vecteurs possibles. Le LFSR1 délivre les vecteurs d'adresse au module de fuzzification et au module d'inférence (où les données des RAM sont adressées sur 6 bits). Un autre registre LFSR2 de 12 bits, subdivisé en deux fois 6 bits, de manière à ce qu'il soit tout le temps synchronisé avec le LFSR1 (pour générer une donnée pour chaque adresse).

Il faut noter que la partie MSB du LFSR2 est dépendante de la partie LSB pour préserver la synchronisation.

Le circuit de compaction utilisé est de type MISA (Multiple Input Signature Analyser), il fait la compression des sorties du bloc de fuzzification, les sorties du bloc d'inférence et la sortie du contrôleur.

Un signal « Test_flou » ou « Test_RAM » est ajouté pour réaliser le basculement entre le mode de test et le mode fonctionnement normal du contrôleur, tout en veillant à ne pas avoir d'effet sur les autres circuits connectés.

Deux types de test :

- ***Test-flou :*** Test du fonctionnement du contrôleur et ces différents blocs.
- ***Test-RAM :*** Test des mémoires RAM.

Pour le «test-flou », Les signatures obtenues sont collectées sur les bus "sortie_signature'', La signature finale du contrôleur est comparée à la fin avec une signature de référence.

Pour le « test-RAM », les signatures obtenues sont collectées sur les bus "Alpha_signature'' pour le module de fuzzification, et sur le bus "Rule_Signature" pour le module d'inférence. Les signatures finales sont comparées avec des signatures de référence des deux blocs .

Simulation :

Après simulation des ensembles LFSR, contrôleur et MISA (figureIII.44), la signature finale obtenue est intégrée dans le circuit afin de la comparer avec le circuit après injection de fautes. Le résultat de test est donné sur le bus "Result_Test" (largeur 4 bits), ce qui nous permet désormais de faire un diagnostic et cibler le module défaillant.

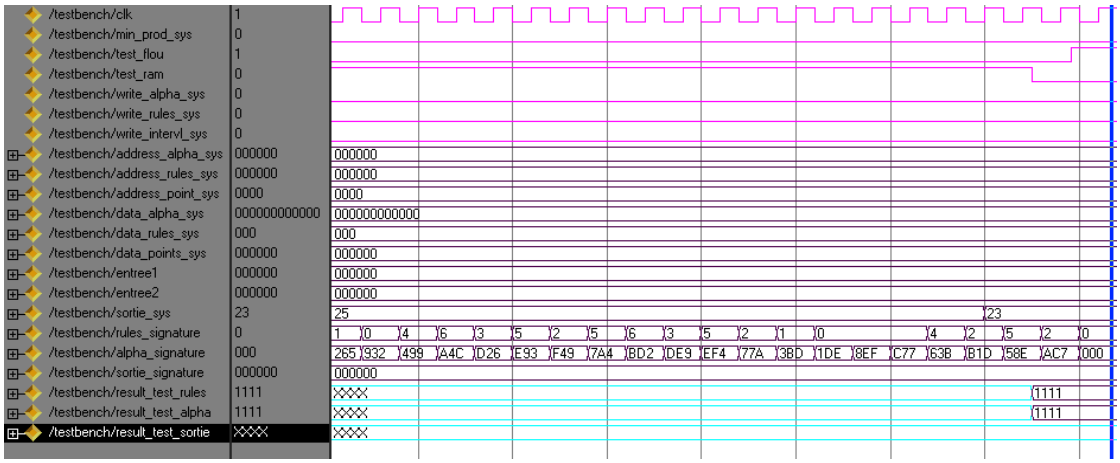


Figure III.44.a: *Bist RAM*

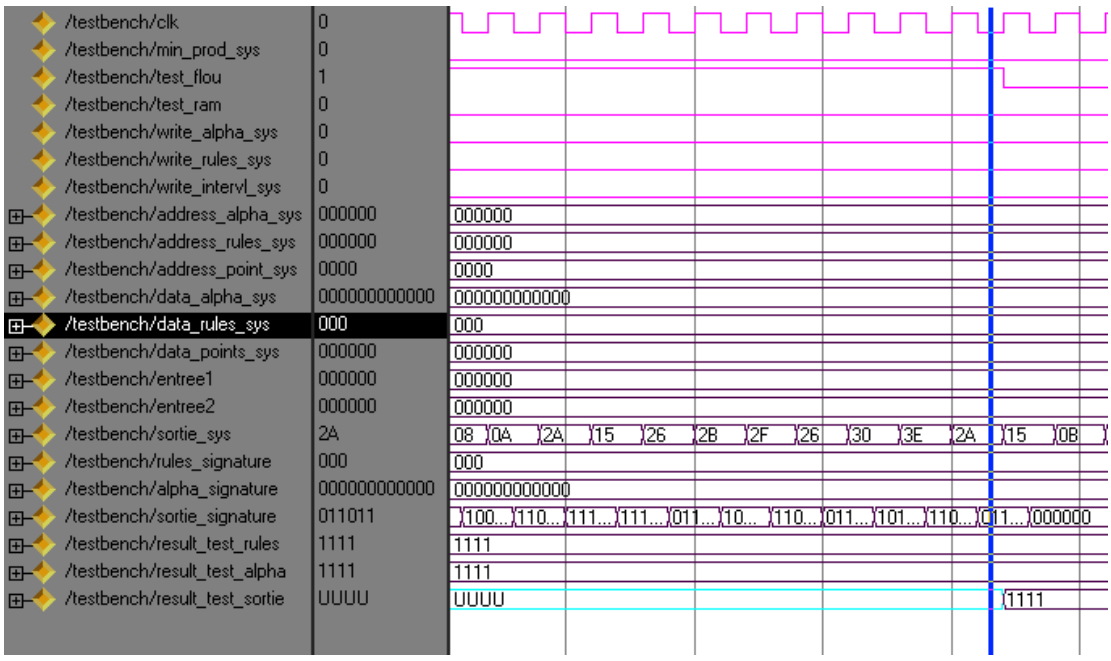


Figure III.44.b : *Bist Sortie*

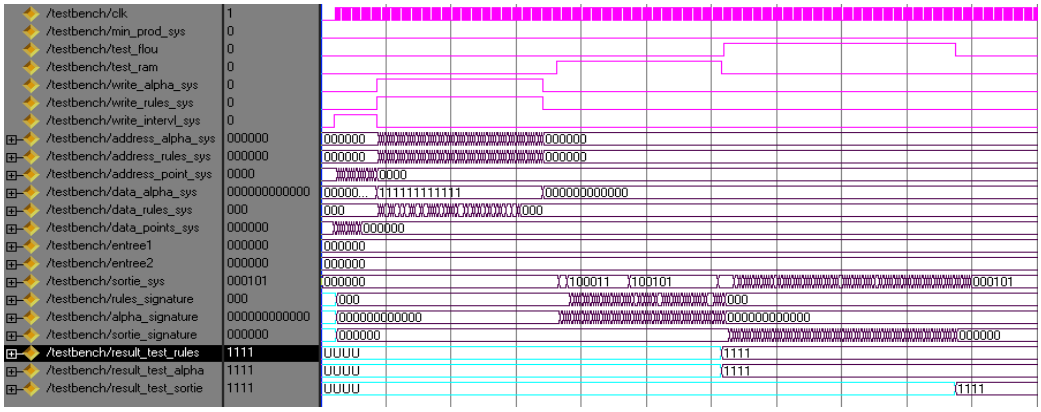


Figure III.44.c: *Bist globale*

Figure III.44: *Simulation de l'architecture Bist*

Après expérimentation du système de test, les signatures obtenues après 32 cycles d'horloge sont : **"20h, "AAAh"et "7h"** pour les modules : contrôleur, les deux blocs RAM fuzzification (Alpha) et le bloc de règles respectivement.

La surface additionnelle due au test est représentée sur la figure III.45

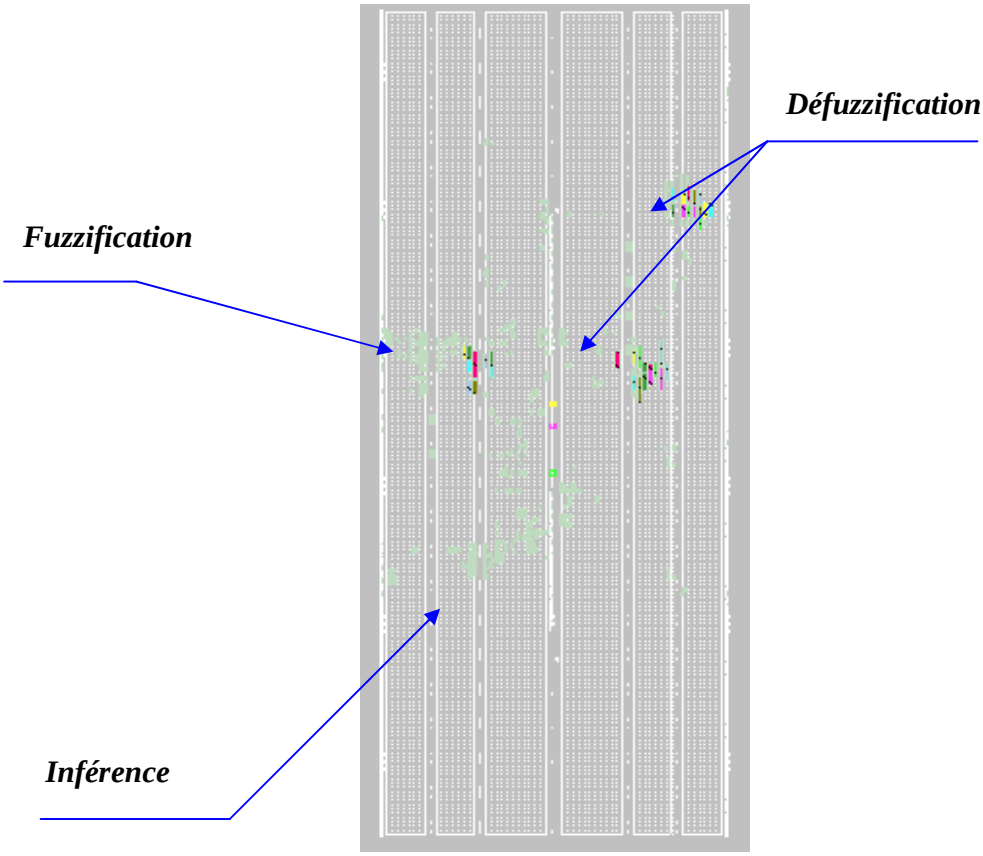


Figure III.45: *Implantation physique sur FPGA (4vlx25ff668) du circuit contrôleur.*

Résultats de synthèse **:**

La fréquence maximale obtenue est de 73.956MHz, les détails d'utilisation du FPGA sont consignés dans le tableau III.20. (Pour d'autres exemples d'implantation, voir annexe (D)).

Tableau III. 20. *Pourcentage d'utilisation des éléments du FPGA*

Elément	Utilisation %
Nombre total de Slices:	3%
Nombre total de Flip Flops	1%
Nombre total de LUTs:	2%
Nombre total d'IOBs:	21%
Nombre total de FIFO16/RAMB16s:	8%
Nombre de RAMB16s utilisées	6
Nombre total de GCLKs:	9%

Après comparaison des tableaux (III.20) et (III.10), après et avant l'intégration du 'TEST', on remarque que la fréquence maximale a baissé, elle est à 73.956 MHz, alors que le nombre de 'SLICES' et tables LUT ainsi que le nombre d'IOB ont augmenté d'une manière significative.

III. 7. Injection de fautes

Différentes approches ont été proposées pour l'injection de fautes dans les descriptions VHDL. Certaines méthodes reposent sur l'utilisation des commandes spécifiques à l'injection de fautes incluses dans le simulateur, tandis que d'autres consistent en la modification des descriptions VHDL.

Comme les techniques d'injection via les commandes du simulateur sont généralement orientées pour le débugging des programmes et donc ne ciblant pas directement le test fonctionnel, alors nous avons exploité deux techniques d'injection de fautes à l'échelle comportementale**.**

a. Première méthode

L'occurrence d'une faute dans cette méthode est gérée en considérant les différents cas suivants **[47] [48]**:

1. changer la valeur de n'importe quelle constante ou modifier l'occurrence de n'importe quelle variable.

2. changer une valeur retournée par une fonction.

3. modifier les conditions de boucle telle qu'un cycle ne sera jamais exécuté.

4. modifier les instructions de test logique (*if* C1 *then* T1; *elseif* C2 *then* T2; *end if*;) de deux façons :

a. quelle que soit C la condition vraie (C1) est validée.

b. quelle que soit C la condition fausse (C2) est validée.

5. modifier les instructions de sélection (*case* ID *is when* ID1=> T1; *when others* Tn+1; *end case*;) de trois façons :

a. la valeur de ID est modifiée sur un bit.

b. l'une des valeurs (ID1… IDn) est modifiée sur un bit.

c. l'instruction *case* sera modifiée comme pour l'instruction *if* dans 4.

6. modifier une expression (x = y + z) telle que sa branche droite modifie un bit.

Les erreurs (ou fautes) peuvent être injectées dans la description VHDL à trois niveaux : les objets, les déclarations et les expressions. Les classes d'erreurs sont: les erreurs de collage, les erreurs d'échange et les erreurs mortes (figure III.46).

Le collage fixe une valeur d'affectation d'un élément quelconque (figure III.46 a), l'échange se fait par des éléments du même type (figure III.46. b) et les erreurs mortes figent l'exécution d'une instruction en l'excluant du programme (figure III.46. c).

Elément VHDL	Exemple de code
Objets	A <= B *XOR* (C *AND* D); A <= '0'; A <= '1';
Expressions	A <= B *XOR* (C *AND* D); A <= B *XOR* '0'; A <= B *XOR* '1';

a. Erreurs de collage.

Elément VHDL	Exemple de code
Objets	A <= B *XOR* (C *AND* D); E <= B *XOR* (C *AND* D);
Expressions	A <= B *XOR* (C *AND* D); A <= B *XNOR* (C *AND* D);

b. Erreurs d'échange.

Elément VHDL	Exemple de code
Déclaration	A <= B *XOR* (C *AND* D); --A <= B *XOR* (C *AND* D);

c. Erreurs mortes.

Figure III.46: *Exemple d'erreurs.*

Après injection de fautes de collage dans les programmes VHDL du contrôleur flou (figure.III.47), la signature modifiée est représentée sur le tableau III.21.

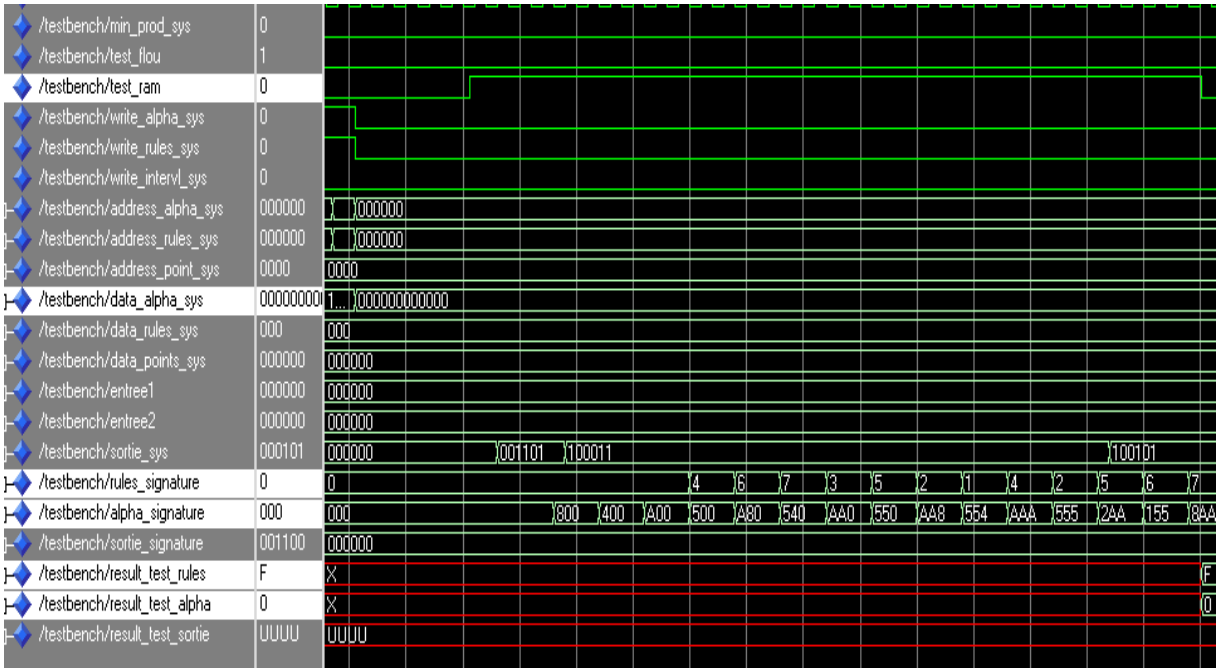


Figure III.47: *Exemples d'injection de fautes au niveau de la RAM des éléments ALPHA. (Result _test_alpha = 0)*

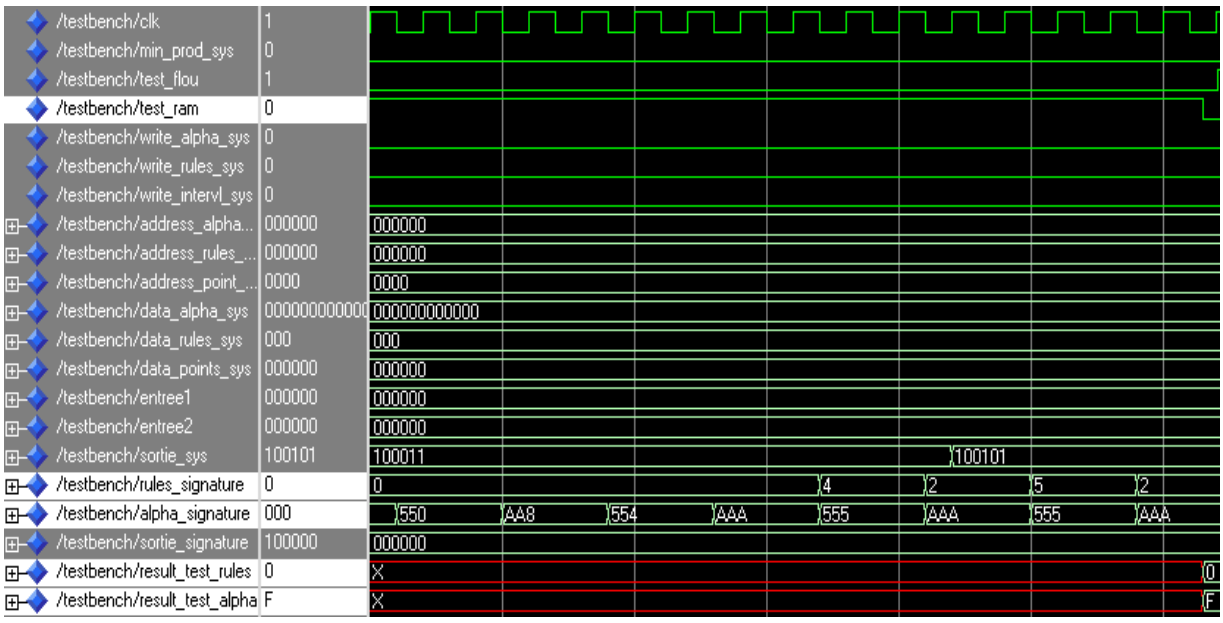


Figure III.48: *Exemples d'injection de fautes au niveau de la RAM règles, (Result _test_rules = 0)*

Tableau III. 21: *Injection de fautes.*

Fautes	Signature du module	Signatures après injection
Changement d'un bit dans ram des coefficients « Alpha »	**AAAh**	8AAh
Changement d'un bit dans la ram des règles.	**7h**	2h
Collage d'un bit interne dans le bloc de la fuzzification. (module de sélection d'intervalles)	**20h**	03h

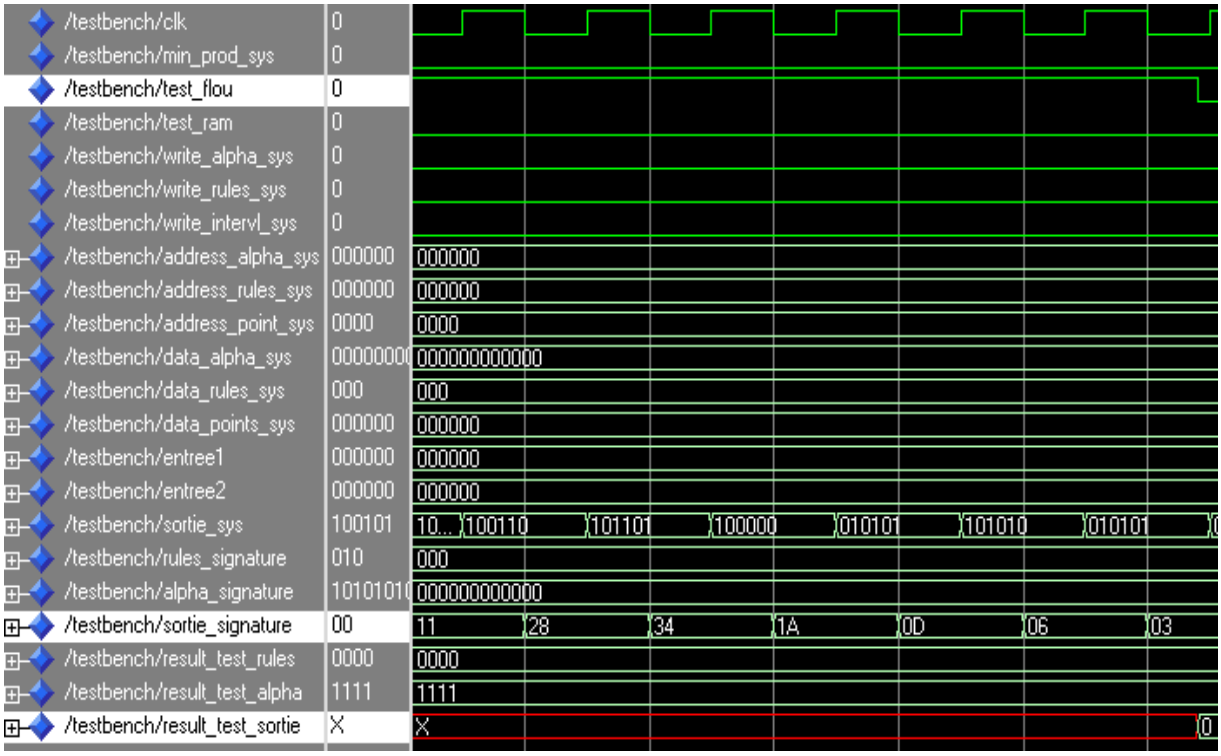


Figure III.49: *Exemples d'injection de fautes au niveau du contrôleur flou (fuzzification), (Result _test_sortie = 0)*

La signature de la sortie du contrôleur flou étant à 20h, après injection de faute au niveau du bloc de fuzzification dans le module « Interval_selector », cette dernière a changé et vaut '03'h.

On remarque qu'il est possible de suivre l'effet de chaque instruction, mais il est impératif de ne considérer que l'effet à l'echelle plus élevée, c'est à dire en mode sous-blocs fonctionnels, taches, procédures…etc, afin d'optimiser le temps de test.

Noter également que les signaux 'result_test' permettent le diagnostic et la localisation des defauts.

b. Deuxième méthode

Le système d'injection de fautes proposé dans **[49]** peut injecter des fautes à n'importe quel signal dans les parties combinatoires et séquentielles des descriptions VHDL structurelle ou comportementale. Il est composé de deux parties essentielles : le contrôleur d'injection et la logique d'insertion de fautes.

- **La logique d'insertion de fautes :**

Deux LFSR, fonctionnant en parallèle, sont utilisés pour générer des séquences différentes qui seront comparées pour déterminer le nombre de bits égaux. La valeur de vecteur de contrôle "**Ctrl**" (variant de 1 jusqu'à 15) détermine le nombre de bits des deux séquences qui doivent être égales pour que l'injection de faute soit autorisée (figure III.48).

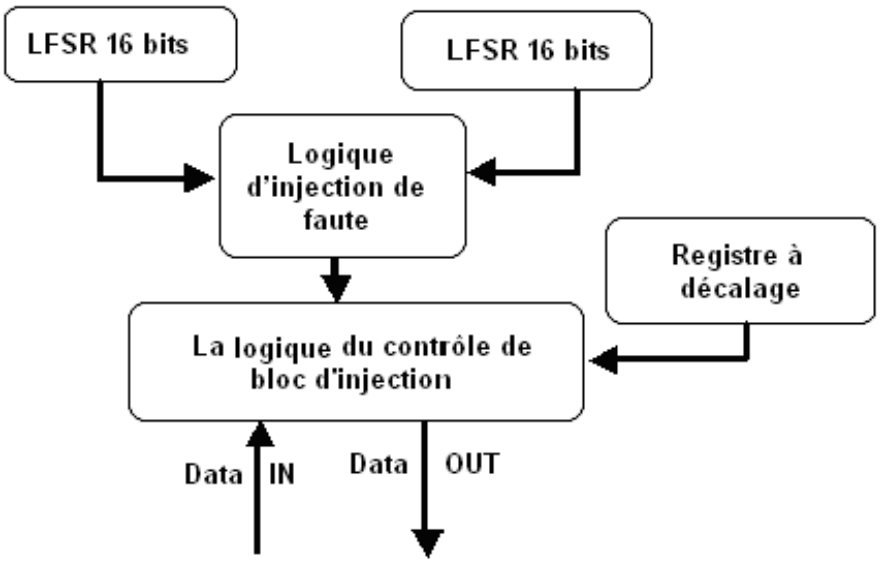


Figure III.48: *Architecture générale de bloc d'injection de faute*.

- **Le contrôleur d'injection :**

L'insertion de fautes est contrôlée par un vecteur de contrôle "**Ctrl**" fourni par l'utilisateur donnant ainsi deux modes d'injection :

- Mode d'injection de fautes permanentes fourni par l'utilisateur Ctrl = "0000".
- Mode d'injection de fautes transitoires fourni par le contrôleur Ctrl = "0001" Jusqu'à "1111".

1. Injection de fautes transitoires :

Quand le système reçoit l'un des codes **Ctrl** = "0001" jusqu'à "1111", le contrôleur active le mode d'insertion d'une faute transitoire et procède à la détermination des moments d'injection d'une manière pseudo-aléatoire basée sur l'utilisation de LFSR. Ainsi, pour connaître les positions des bits où il faut injecter la faute, on a eu recourt à un registre à décalage, dont les bits mis à 1 indiquent les emplacements d'injection de fautes.

La figure III.49 représente l'organigramme d'injection de fautes transitoires.

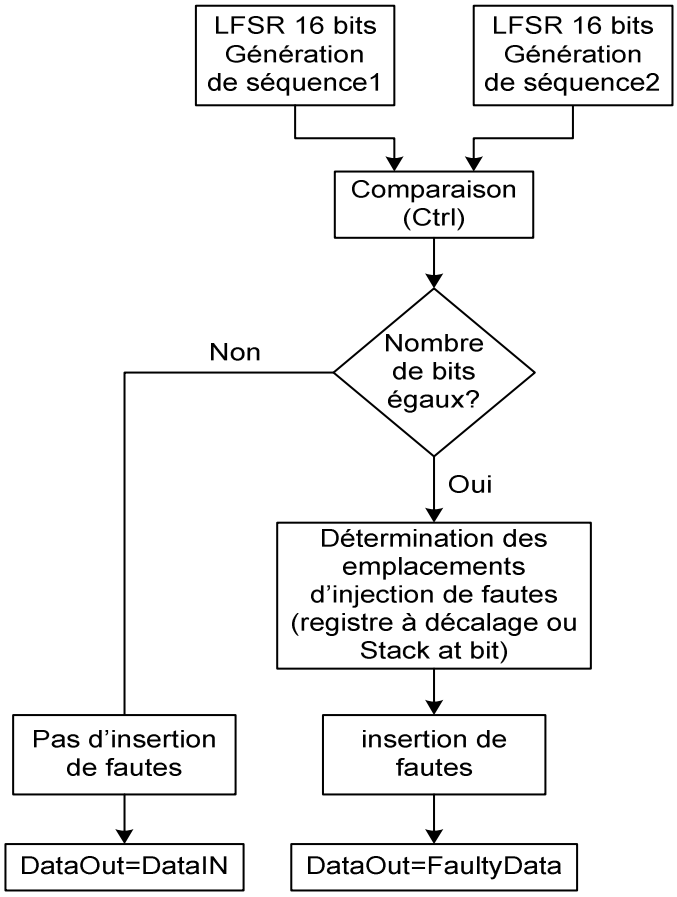


Figure III.49: *Flot d'injection de fautes transitoires.*

Les résultats de l'étude des pourcentages d'injections de fautes pseudo aléatoires en fonction du vecteur "Ctrl" sont donnés dans la Figure III.50.

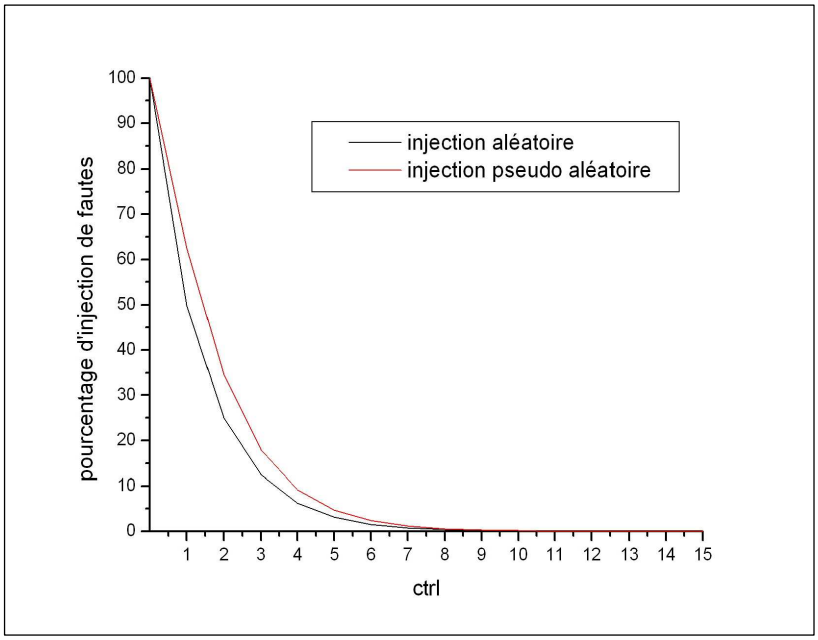


Figure III.50: Pourcentage d'injection de fautes en fonction de Ctrl.

2. Injection de fautes permanentes :

En positionnant le vecteur Ctrl = "0000", la logique d'insertion de fautes injecte un collage à '1' (StackAtValues = '1') ou à '0' (StackAtValues = '0') dans le bit sélectionné par FaultAtBits (figure III.51).

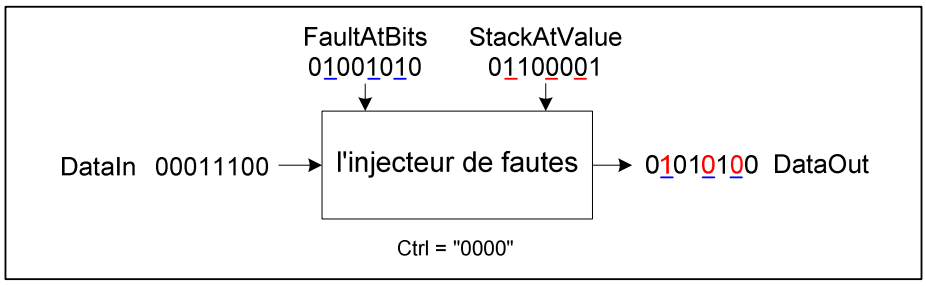


Figure III.51: *Injection de faute permanente*

Le programme d'injection selon ce mécanisme n'est pas explicité vu qu'une meilleure approche a été publiée **[50]**.

III. 8. Conclusion

Dans ce chapitre, une méthode de conception d'un contrôleur flou (2 entrées/1 sorties) sous forme d'un Soft IP est présentée. L'architecture est décrite en langage de description matériel VHDL basée sur l'utilisation de circuits FPGA.

La conception et les caractéristiques des différents blocs constituant le contrôleur sont explicités pour limiter la bonne maitrise de développement.

Les caractéristiques de notre contrôleur conçu sont : la fréquence de travail est de 165.780MHz, le pourcentage de blocs RAM utilisé est de 8% et le nombre de flip-flop est de 1% sur les virtex4.

L'intégration de la DFT au sein du contrôleur pour assurer sont bon fonctionnement a augmenté la surface et le nombre de bits des opérateurs logiques, ainsi que ces bus d'interconnexion et en conséquence a fait réduire sa fréquence de travail.

Afin d'évaluer l'approche de test proposée nous avons recouru à deux techniques d'injections de fautes à l'échelle comportementale :

- la première méthode a consisté en la modification du code source du programme VHDL.
- la seconde méthode a reposé sur l'utilisation d'un injecteur de fautes.

Les résultats de synthèse présentés et une évaluation de l'architecture utilisée nous permettent de conclure que notre contrôleur répond largement aux contraintes discutées dans le cahier des charges.

CHAPITRE IV

Contrôle de la phase de freinage d'un métro.

IV.1.Introduction

Un métro **[51]**, apocope du terme **métropolitain**, lui-même abréviation de chemin de fer métropolitain, est un chemin de fer urbain souterrain le plus souvent, sur viaduc quelquefois, au sol rarement. En 1981, le Comité des Métropolitains de l'UITP « International Association of Public Transport » définit « le chemin de fer métropolitain » comme un chemin de fer conçu pour constituer un réseau permettant le transport d'un grand nombre de voyageurs à l'intérieur d'une zone urbaine au moyen de véhicules sur rails avec contrôle externe, dans un espace totalement ou partiellement en tunnel et entièrement réservé à cet usage.

La définition nord-américaine du métro est plus synthétique : c'est un transport public urbain de masse en mode guidé sur site propre intégral, sans croisement avec tout autre mode de transport ni accès piétonnier.

IV.1.1. Le frein en chemin de fer

Un train n'est jamais facile à arrêter, du fait de sa longueur et son poids, les risques d'emballement sont bien réels même encore de nos jours. Au siècle dernier, il fallait mettre des préposés au freinage appelés gardes frein dans tous les wagons. Ceux-ci réagissaient sur ordre du machiniste qui sifflait aux freins.

On comprend aisément les risques que cela comprenais et les accidents n'étaient pas rares loin de là. Le freinage d'un train doit être assuré par la mise en œuvre de systèmes multiples permettant d'en assurer l'arrêt ou d'en maîtriser la vitesse. Il doit être performant, automatique, fiable et disponible en permanence, même dans des conditions critiques.

Aujourd'hui, le freinage d'un train est bien amélioré par l'apport de technologies modernes, le principe reste le même : celui d'une commande sécuritaire actionnant pneumatiquement les appareils de puissance (frein à semelles sur roue ou frein à disque).

IV.1.2. Principe du freinage [51]

Deux petites définitions sont à retenir :

Frein continu : Sa mise en action est à partir d'un point quelconque, il provoque le serrage de tous les véhicules (de la tête à la queue du train), tous les freins sont commandés d'un certain endroit (Distance).

Frein automatique : Le frein se met en action de lui-même, suite à une anomalie dès que quelque chose s'oppose à son fonctionnement, ou dès qu'on s'approche de l'arrivée planifiée.

IV.1.3. Actionnement des freins

La commande du freinage au niveau local se distingue par le type d'énergie utilisée pour assurer la mise en action des freins (au niveau des cylindres et étriers) :

- Energie pneumatique : l'actionnement est assuré à l'aide d'air comprimé.
- Energie hydraulique : l'actionnement est assuré à l'aide d'une pression hydraulique.
- Energie électrique : l'actionnement est assuré par un moteur électrique.

On rencontre sur les véhicules ferroviaires plusieurs sortes de freins **[52]**:

- Le frein à bloc, où des semelles en fonte ou composite agissent sur les bandages des roues.
- Le frein à disque où des garnitures s'appliquent sur les disques de freins, ces derniers étant disposés entre les roues ou dans le cas d'un essieu moteur, flasques des deux cotés des roues.
- Le frein à vis.
- Le frein éléctro-pneumatique.
- Le frein à récupération.
- Le frein rhéostatique.
- Le frein électromagnétique.
- Le frein direct.

  …Etc.

IV.2.Contrôle du freinage

La figure IV.1 représente le schéma bloc de régulation d'un système de freinage par logique floue. L'opération de freinage est expliquée ci-dessous :

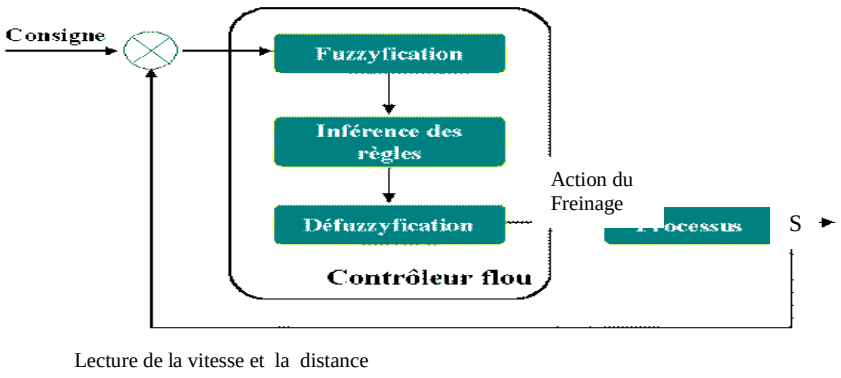


Figure IV.1: *Contrôle Flou*

IV.2.1. Présentation du contrôleur

Le contrôleur flou étudié dans ce travail est essentiellement destiné au contrôle de la phase de freinage d'un métro. Le but est de ralentir avec le moins d'à-coups possibles et de s'arrêter à l'endroit voulu, c'est à dire à une des stations de la ligne. Quand le métro reçoit la commande de

freinage, celui ci procède à un réglage de vitesse (diminution) en fonction de la distance pour assurer un freinage de bonne qualité (d'où on a introduit la notion d'un paramètre que nous avons appelé : **Indice de freinage**)

IV.2.2. Conception du contrôleur

Le contrôleur conçu dans ce travail est divisé en trois blocs : un bloc de fuzzification des entrées, un bloc d'inférence et un bloc de défuzzification.

IV.2.2.1. Bloc de fuzzification

Il y a deux choix à faire :

a. Choix des entrées et des sorties

Choix des variables d'entrée et de sortie : Ce choix dépend du contrôleur que l'on veut réaliser et des paramètres disponibles.

Dans notre cas, il y a deux ensembles d'entrée (correspondant à des capteurs):

- La distance D en km : « **DISTANCE**».
- La vitesse V en km/h : « **SPEED** ».
  (Jusqu'à la prochaine étape)

Et un ensemble de sorties (actionneur): il s'agit de la procédure de freinage

- La pression sur les freins F, ou degré de freinage : **« BRAKE »**

Figure IV.2: *Bloc Contrôleur à Logique Floue*

Les variables d'entrée SPEED et DISTANCE sont converties en variables floues pour y appliquer les règles de contrôle, donc pour chacune des variables d'entrée et de sortie on doit identifier les intervalles de variation convertis en 7 sous ensembles flous MFs. Les fonctions d'appartenance sont choisies de façon à obtenir un contrôle optimal du freinage du métro.

Pour définir le domaine des valeurs que les variables peuvent prendre, on partitionne le domaine en intervalles, auxquels on associe un label descriptif. Cette étape permet donc de distinguer les différentes variables linguistiques qui seront utilisées lors du contrôle flou. Après normalisation, nous avons obtenu les tableaux suivants :

Les entrées :

Tableau IV. 1 : *Ensemble flou spécifique à l'entrée SPEED.*

VF	VERY FAST
F	FAST
M	MIDIUM FAST
SW	SLOW
VS	VERY SLOW
VVS	VERY VERY SLOW
ST	STTOPED

Tableau IV. 2 : *partitionnement de l'intervalle de variation de la vitesse.*

SPPED (km /h)	**Borne** inf	**Borne** sup
VF	56	INFINIE
F	44	60
M	30	50
SL	15	40
VS	6	25
VVS	1	7
ST	0	2

Tableau IV. 3 : *Ensemble flou spécifique à l'entrée DISTANCE.*

VF	VERY FAR
F	FAR
M	MIDIUM FAR
N	NEAR
VN	VERY NEAR
VVN	VERY VERY NEAR
AT	AT PLACE

Tableau IV. 4 : *partitionnement de l'intervalle de variation de la distance.*

DISTANCE (m)	**Borne** inf.	**Borne** sup.
VF	60	INFINIE
F	40	63
M	12	47
N	7	31
VN	4	10
VVN	1	5
AT	0	2

La sortie :

Tableau IV. 5 : Ensemble flou spécifique à la sortie FREIN.

P	PLEIN
PP	PRESQUE PLEIN
M	MOYEN
L	LEGER
TL	TRES LEGER
TTL	TRES TRES LEGER
NO	NO

Pour les fonctions d'appartenance de la sortie on a choisi des formes de singletons.

Tableau IV. 6 : partitionnement de l'intervalle de variation.

FREIN (%)	**Borne**
P	60
PP	50
M	38
L	25
TL	13
TTL	6
NO	0

b. choix des formes de fonctions d'appartenance (surfaces de contrôle)

Les fonctions d'appartenance des différents sous-ensembles flous sont explicitées ci-dessous. Nous avons choisi une forme triangulaire pour toutes les fonctions d'appartenance. Pour notre application, on a choisi la forme triangulaire pour les fonctions d'appartenances comme c'est illustré sur les figures (IV.3 et IV.4).

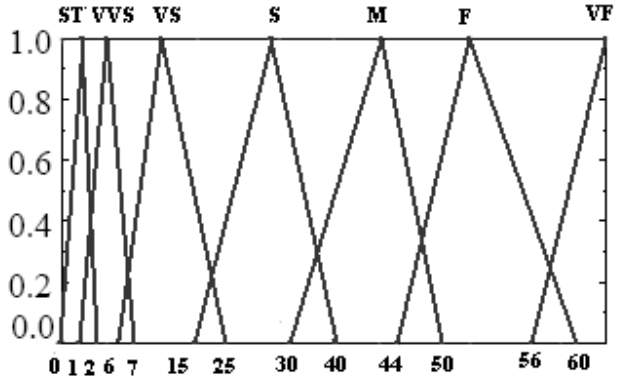


Figure IV.3: *Forme des fonctions d'appartenance de la variable d'entrée 'Vitesse'*

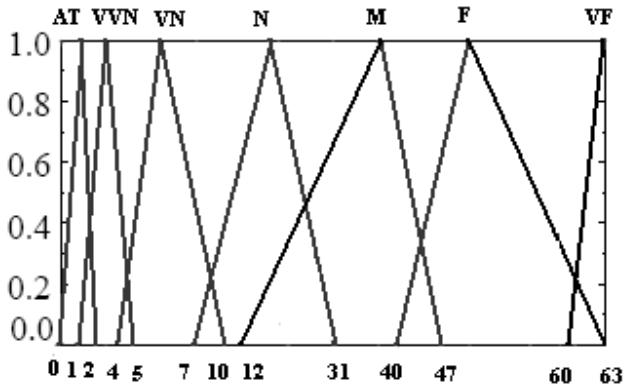


Figure IV.4: *Forme des fonctions d'appartenance de la variable d'entrée 'Distance'.*

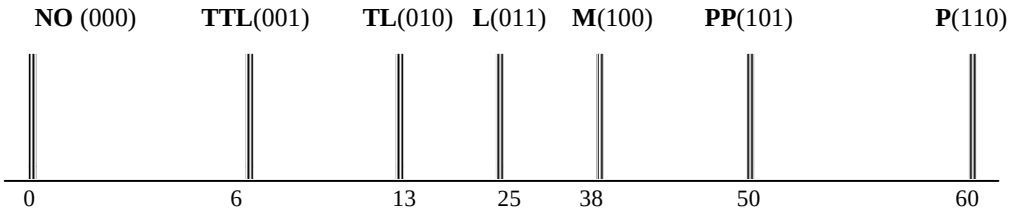


Figure IV.5: *Fonctions d'appartenance de la variable de sortie 'Action sur le frein'.*

IV.2.2.2. Bloc d'inférence

Le bloc d'inférence reçoit le résultat de la fuzzification des entrées qui comprend les fonctions d'appartenance MFX0, MFX1, et leurs degrés d'appartenance αXo0, αXo1, αX11, αX12. De ce fait on a alors seulement quatre règles qui seront activées **[34]**.

Le bloc d'inférence comprend deux processus qui travaillent en parallèle : calcul du minimum et décision.

Calcul du minimum :

$\Theta 1 = \min(\alpha Xo0, \alpha X11)$. $\Theta 2 = \min(\alpha Xo0, \alpha X12)$.

$\Theta 3 = \min(\alpha Xo1, \alpha X11)$. $\Theta 4 = \min(\alpha Xo1, \alpha X12)$.

Où u est la variable de sortie.

Construction de la matrice d'inférence « décision » :

Il s'agit de l'élaboration de la base de règles du contrôleur, en se basant sur la méthode de «Mamdani» pour la sélection des règles. Où nous avons élaboré un ensemble de règles qui semblent satisfaire a notre problème, la difficulté à élaborer cette table à conduit à une vérification exhaustive cas par cas des différentes situations de la 'distance' et la' vitesse' selon la règle **SI** CONDITION **ALORS** ACTION. Où nous avons 7 fonctions d'appartenance pour chaque variable d'entrée. Le nombre de règles est donc de 7 x 7 soit 49 règles, (voir tableau IV.7) :

Tableau IV. 7 : Table des règles (7 X 7)

Sortie		**Entrée DISTANCE (X0)**						
		AT	VVN	VN	N	M	F	VF
Entrée VITESSE (X1)	ST	NO	NO	NO	NO	NO	NO	NO
	VVS	TTL	NO	NO	NO	NO	NO	NO
	VS	TL	TTL	NO	NO	NO	NO	NO
	SL	L	TL	TTL	NO	NO	NO	NO
	M	M	L	TL	TTL	NO	NO	NO
	F	PP	M	L	TL	TTL	NO	NO
	VF	P	PP	M	L	TL	TTL	NO

IV. 2. 2. 3. Bloc de défuzzification

Finalement, il reste à choisir une des méthodes de défuzzification que nous devrons utiliser. La méthode du centre de gravité de la surface (COG) est implémentée dans notre programme VHDL.

L'abscisse frein du centre de gravité peut être calculée par la fonction : $\frac{\sum_{i=1}^{m} u_i \mu_i(u_i)}{\sum_{i=1}^{m} \mu_i(u_i)}$.

Le processus de défuzzification est réalisé en quatre opérations en parallèle :

- L'opération de codage des fonctions d'appartenance résultantes du bloc d'inférence en valeurs réelles (les règles actives). Où on a choisi de formes de singletons afin de faciliter le calcul et gagner la rapidité (Voir figure IV.5).
- L'opération d'addition : somme des Θi ($\sum \Theta i$).

- L'opération de multiplication des Zj* Θi.
- L'opération de division : ∑ Zj* Θi /∑ Θi.

IV. 2. 2. 4. Assemblage du contrôleur flou

Après l'élaboration des différents blocs du contrôleur, nous l'avons implémenté suivant la figure IV.6.

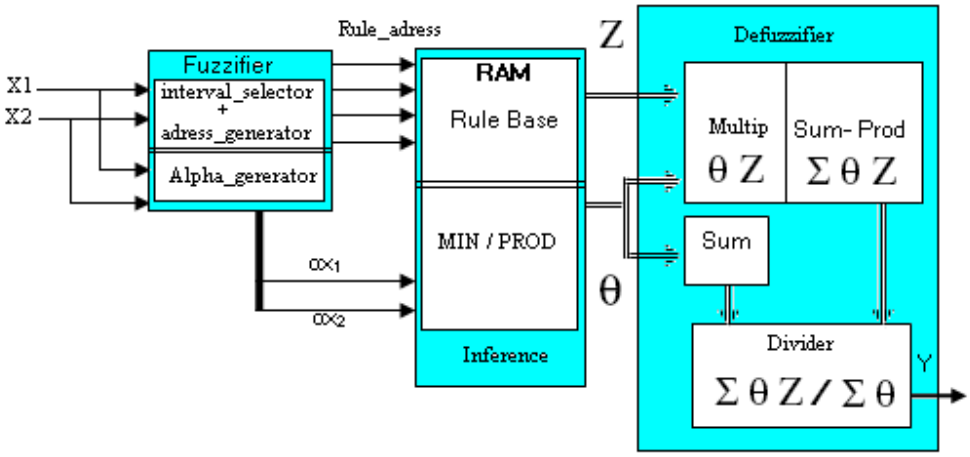


Figure IV.6 *: Architecture du contrôleur.*

IV.3.Types de freins

Le contrôleur agit selon les données qu'il reçoit des capteurs (distance, vitesse) pour cela on suppose que le contrôleur dispose de deux registres d'entrée (Vitesse, Distance).

Deux types de freins peuvent survenir

- **Frein d'arrivée :** La phase de freinage est la distance parcourue par le métro pendant que sa vitesse diminue, c'est-à-dire à partir du moment où l'expert procède intuitivement à la mise en action du système de freinage et l'arrêt complet du métro.

  La vitesse et la distance évoluent dynamiquement, leurs valeurs sont lues à chaque pas d'échantillonnage et stockées dans leurs registres de données respectivement Rv et Rd 'Entrées du contrôleur flou', le contrôleur procède au freinage par étape (Raisonnement flou) en dégradant la vitesse en fonction de la distance afin d'assurer le meilleur confort possible.

- **Frein d'urgence :** Dès que quelque chose s'oppose au fonctionnement du train (obstacles), cela provoque une chute de pression par exemple ce qui charge automatiquement le ¨registre Distance¨ du contrôleur par la distance de freinage, où automatiquement celui ci procède au freinage d'urgence.

IV.4.Qualité de freinage

Comme le freinage dépend des différentes situations de vitesse du métro et de la distance où il se trouve par rapport au point final où il devrait exécuter sa tâche de freinage définitif (vitesse =0, distance =0). Pour cela on a introduit, pour prendre en charge la qualité ou le confort, un indice de freinage comme défini ci dessous :

Tableau IV. 8 : *Ensembles Flous d'indice de freinage*

TTB	Très Très bon Freinage	(110)
TB	Très bon Freinage	(101)
B	Bon freinage	(100)
P	Passable	(011)
M	Mauvais freinage	(010)
TM	Très mauvais freinage	(001)
TTM	Très Très mauvais freinage	(000)

Dans la conception des métros modernes, on introduit un Indice de confort et sécurité du voyageur, IL est exigé de ralentir, stopper le métro dans un laps de temps donné et cela tout en garantissant la stabilité du comportement routier du métro.

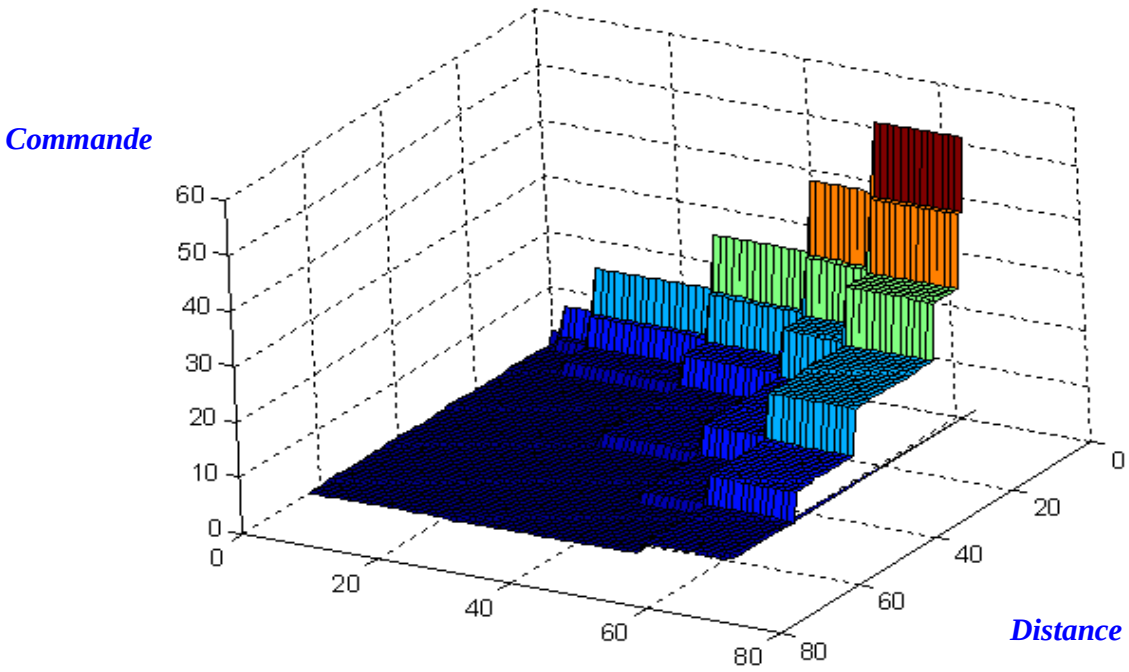
L'indice de freinage « If » est une interprétation de l'action du freinage où pour chaque type de freinage résulte un confort donné (voir figure IV.9).

Tableau IV. 9 : Ensemble flou spécifique au indice de freinage / freinage.

Indice de Freinage	Confort	Action sur Frein	Action
TTB	Très Trés Bon	**NO**	Pas de dégradation de vitesse (attente)
TB	Très Bon	**TTL**	Une seule dégradation de vitesse (Très Très léger)
B	Bon	**TL**	Deux dégradations de vitesse à la fois (Très léger)
P	Passable	**L**	Trois dégradations de vitesse à la fois (Léger)
M	Mauvais	**M**	Quatre dégradations de vitesse à la fois (Moyen)
TM	Très Mauvais	**PP**	Cinq dégradations de vitesse à la fois (presque plein)
TTM	Très Trés Mauvais	**P**	Six dégradations de vitesse à la fois (Plein)

Résultats :

Après simulation, on a obtenu la surface de contrôle illustrée dans la figure IV.6, elle correspond au raisonnement flou attendu qui satisfait la matrice d'inférence élaborée précédemment (voir tableau IV. 7) et dépendant des paramètres de fuzzification considérés.



re IV.6 : ***Surface du contrôleur Flou du Freinage***

IV.5.Exemples et résultats de simulation

IV.5.1. Exemple1 :

Distance := 48 (V.Far)

SPEED := 55 (V.Fast)

Les résultats de simulation sont présentés sur la figure IV.7.

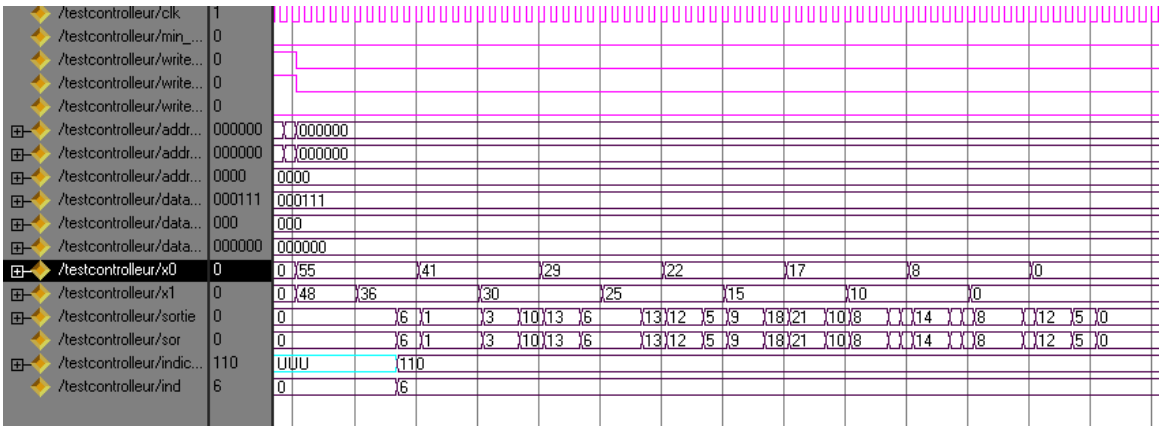


Figure IV.7: *Simulation du freinage à partir de (V.Far et V.Fast)*

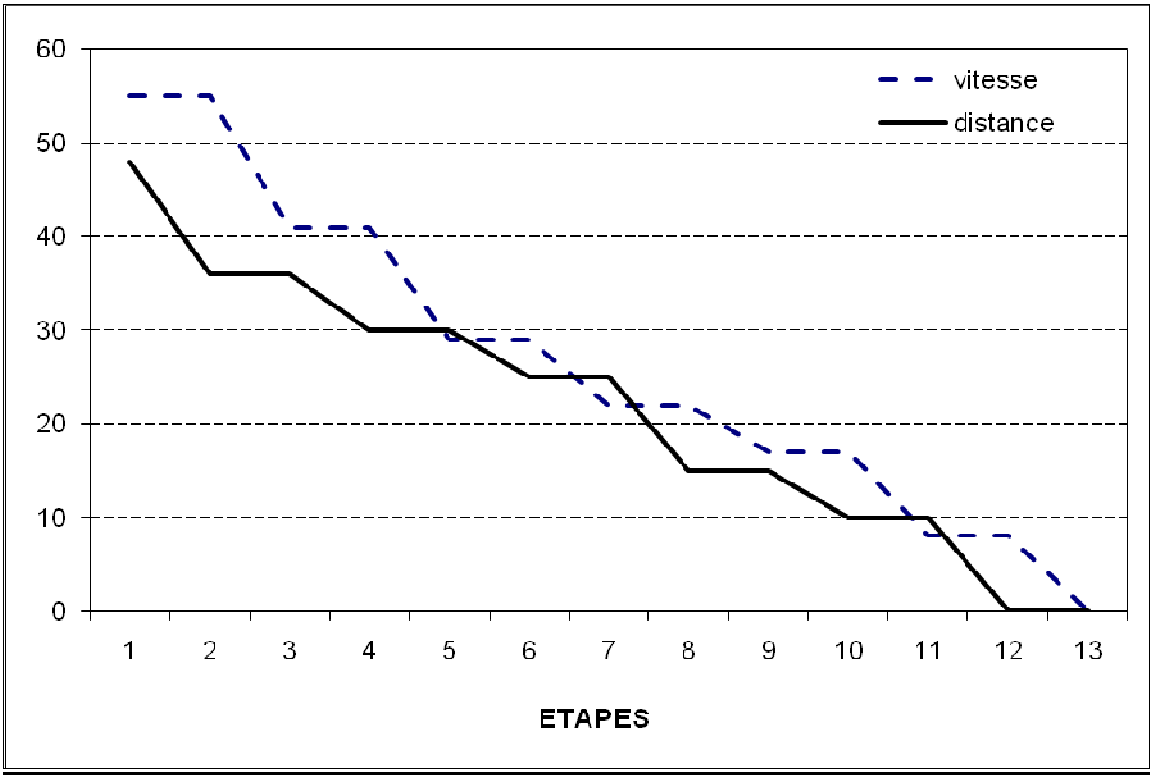


Figure IV.8 : *Freinage à partir de V.FAR /V.FAST en 13 étapes*

Un exemple de freinage est montré sur la figure IV.8 où il y'a eu freinage à partir de V.FAR /V.FAST en 13 étapes, donc selon une dégradation régulière et confortable acxomplissant ainsi un indice de freinage est de 6 « 110 », (TTB : 'très très bon freinage').

IV.5.2. Exemple2 :

Distance := 25 (M)

SPEED := 55 (V.Fast)

Les résultats de simulation sont affichés sur la figure IV.9.

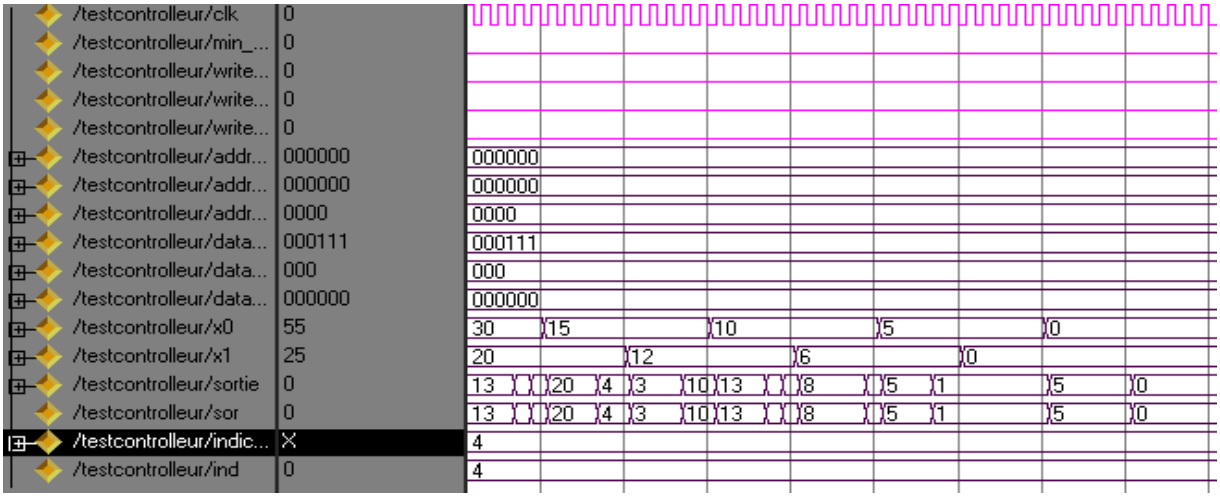


Figure IV.9: *Simulation du freinage à partir de (M et V.Fast)*

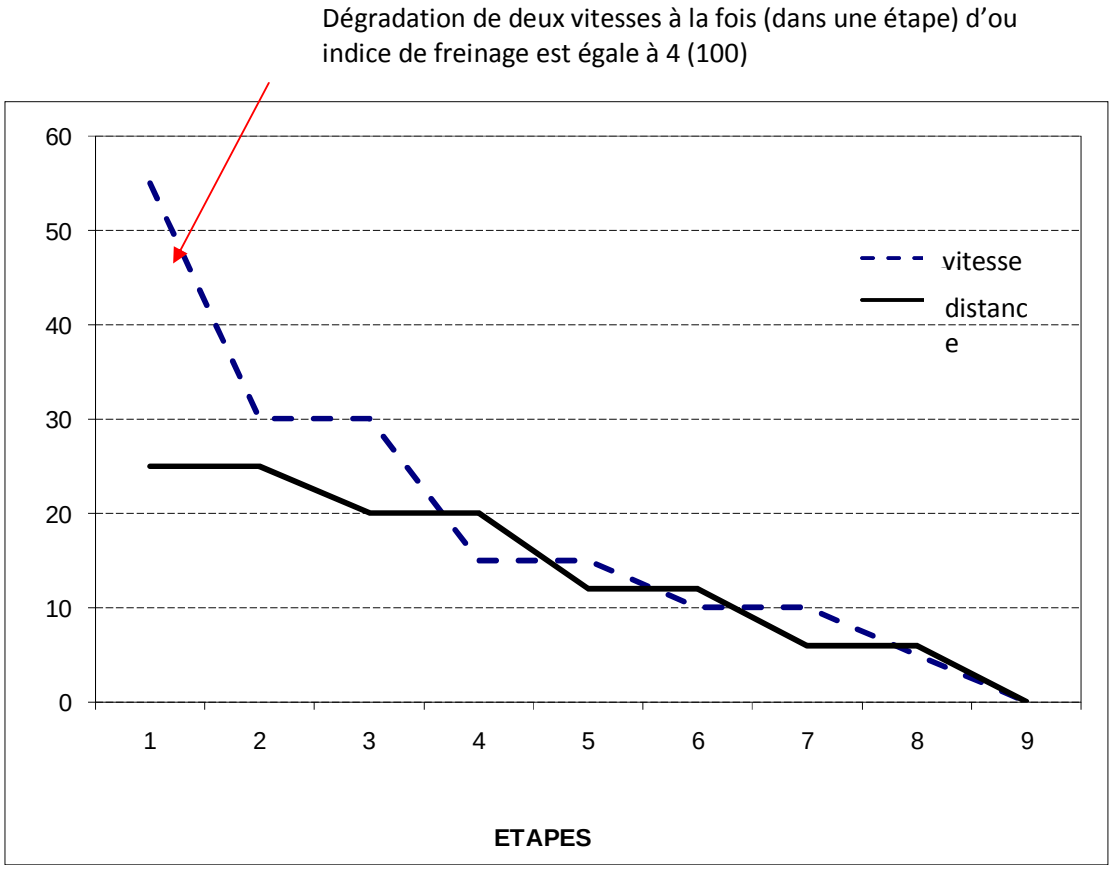


Figure IV.10**: *Freinage à partir de (V.FAST - MIDUIM)***

IV.6. Freinage linéaire du train

Dans ce cas on a introduit un capteur pour lecture des valeurs de la distance et la vitesse continuellement, il sert tout d'abord à la vérification du fonctionnement du contrôleur, une fois l'action du freinage est lancée il prédit le nouvel état de la distance et la vitesse en fonction de l'ancien état, cette prédiction est basée sur le respect de la notion du confort où la vitesse décroit proportionnellement à la décroissance de la distance.

Dans notre simulation, une fois on introduit les deux premières valeurs de la distance et la vitesse, le processus de dégradation de vitesse en fonction de la distance (freinage) commence et ne s'arrête jusqu'au l'arrêt finale du métro.

Il s'agit d'un contrôleur flou dédié au freinage, son schéma est montré sur la figure IV.11.

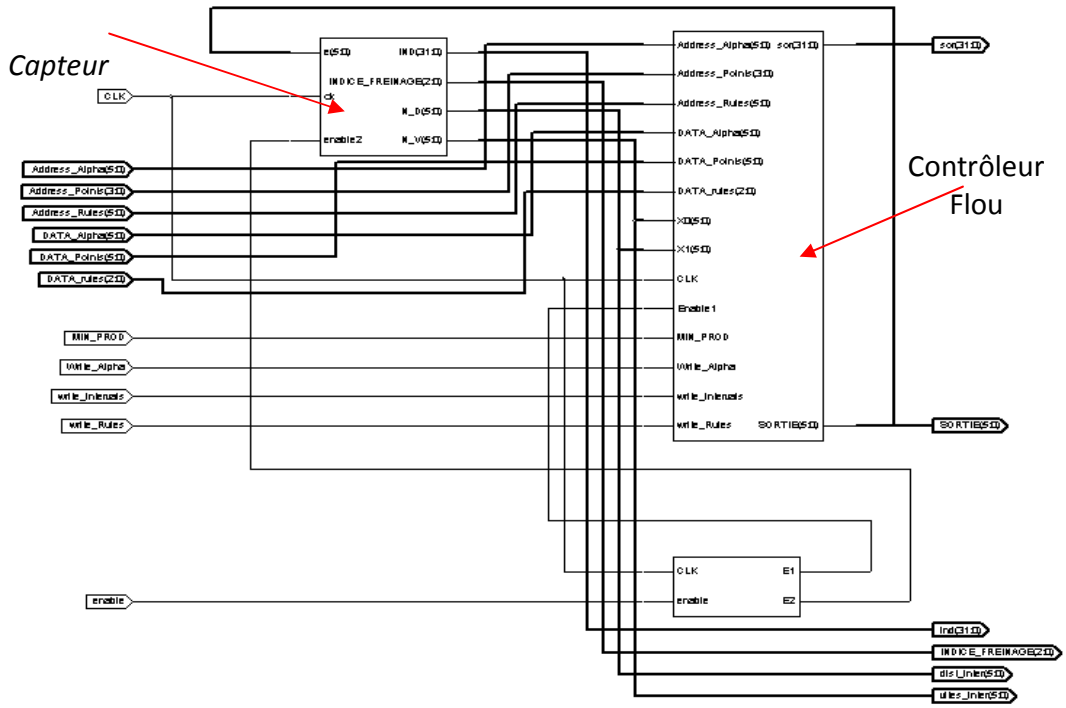


Figure IV.11: *Schéma bloc de l'asservissement de freinage*

Exemples et résultats de simulation

- ***Exemple1:***

V=50, D=50 (Fast, Far):

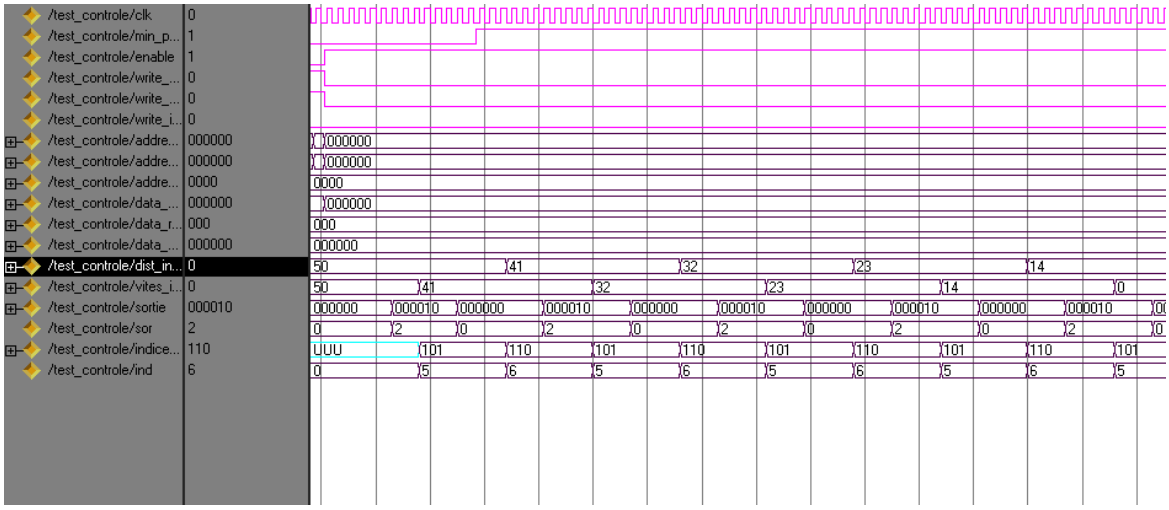


Figure IV.12: *Simulation du freinage à partir de (Far et Fast)*

L'action du freinage est entre 0 (NO) et 2 (TTL : dégradation d'une seule vitesse à chaque fois), l'indice de freinage est entre 5 et 6 (TB : Très bon freinage et TTB : très très bon freinage).

- ***Exemple2:***

V=50, D=0 (Fast, AT)

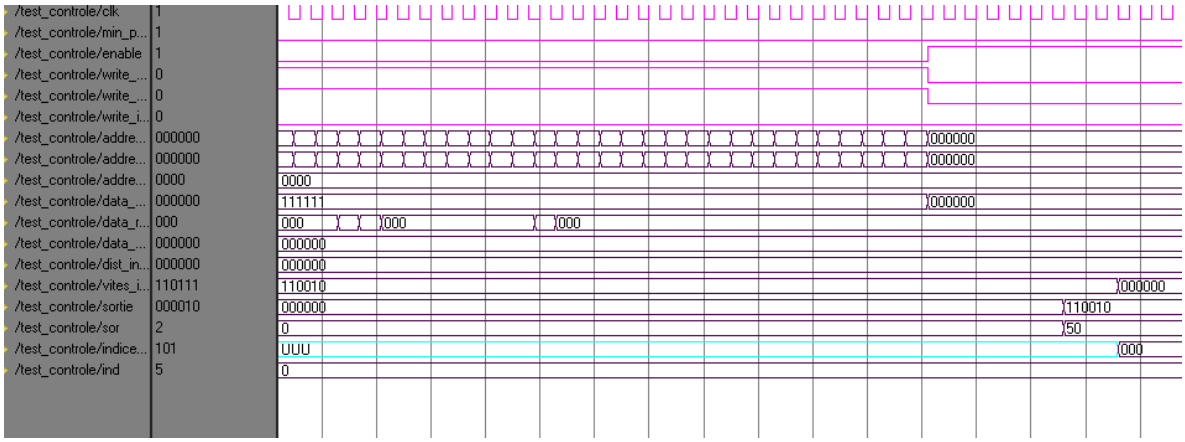


Figure IV.13 : *Simulation du freinage à partir de (AT et .Fast)*

L'action est 50 (frein plein), réellement à la distance AT 'Zéro' ***le freinage est impossible***, d'où il est interprété par un indice de freinage égale à 0 (TTM : Très très mauvais Freinage).

- ***Exemple3:***

V=0, D=50 (Stop, Far)

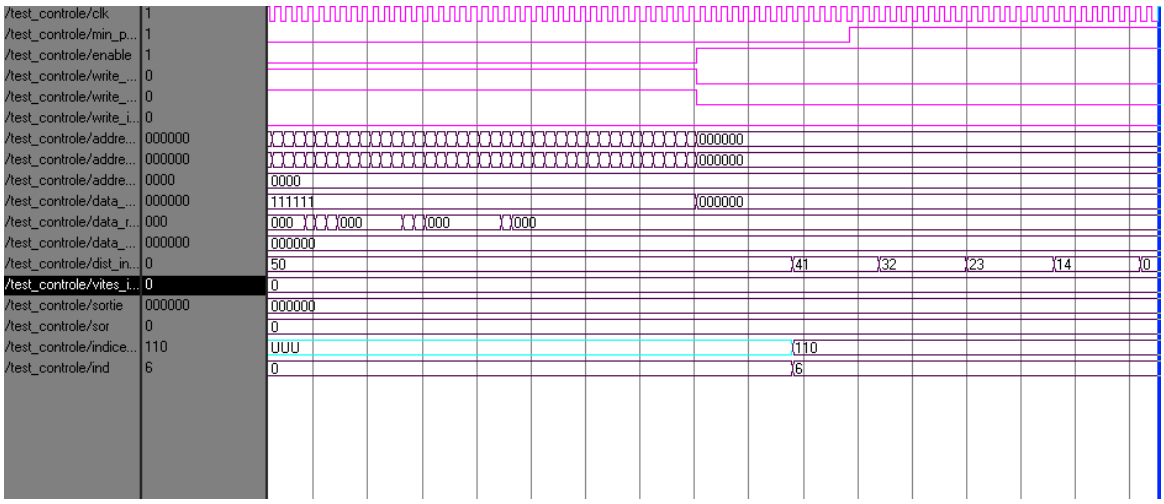


Figure IV.14: ***Simulation du freinage à partir de (Stop et Far)***

L'action est NO 'Zéro' => aucune décision de dégradation de vitesse du fait qu'elle est nulle, l'action du contrôleur flou ''Zéro'' est interprétée comme ***une action d'Attente*** => ***Pas de déclenchement de freinage*** d'où l'indice de freinage est égal à 6.

IV.7. Surface de freinage

Pour montrer le comportement global, on a représenté la surface de freinage comme l'illustre la figure IV.15, Les opérations de freinage acceptées sont d'indices supérieurs à 3 c'est à dire (If ≥ 3)

Limites dans la vitesse *(cas particuliers):*

- Quand on est à une grande vitesse ≥ M (moyenne) et à distance AT ' nulle ' c à d que le train se trouve a la station alors le freinage est impossible.

Limites dans la distance *:*

- Quand on est à une distance ≥ VVN *(très très proche)* à vitesse VVS *(très très faible)* 'Nulle', le freinage n'a pas de sens, et il y'aura pas de déclenchement de freinage donc uniquement AT qui est accepté.

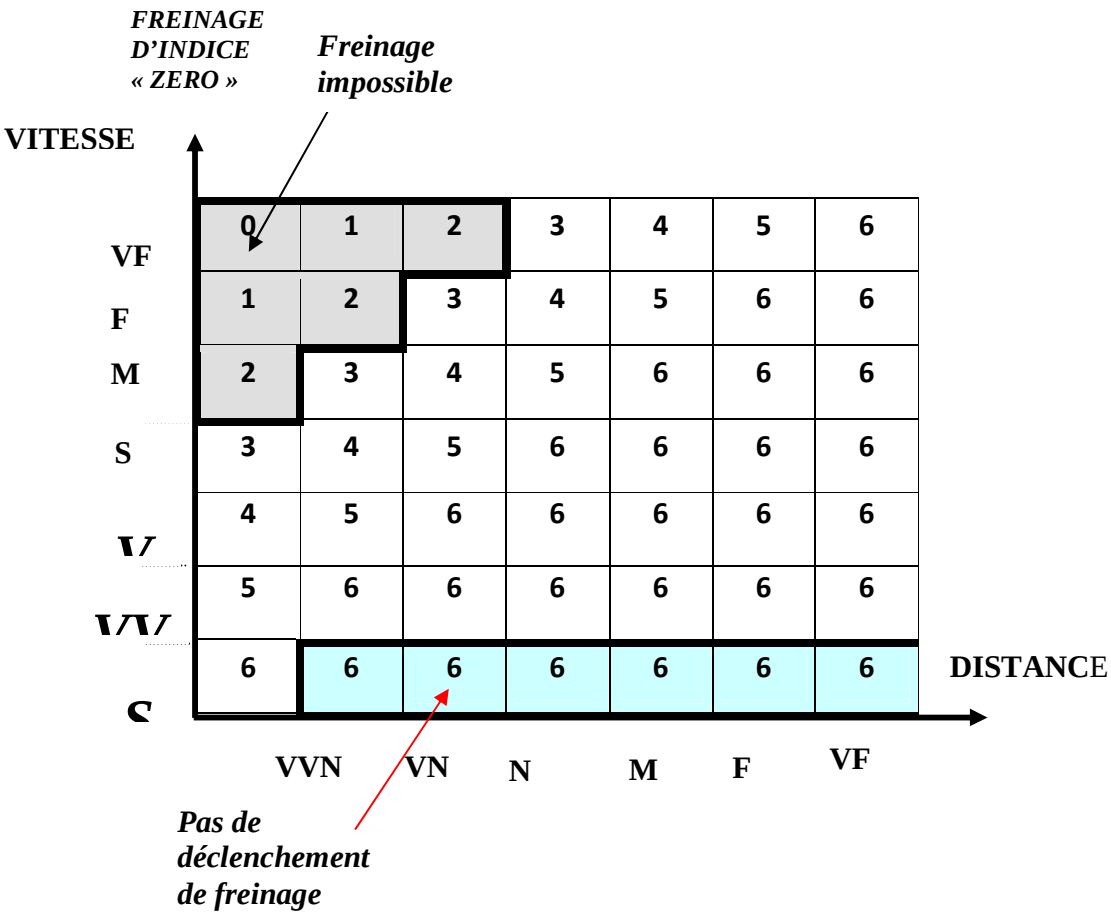


Figure IV.15: *Zones de freinages*

IV.8. Conclusion

Le but principal recherché est de présenter la stratégie de contrôle de la phase de freinage d'un métro par la logique floue en utilisant un contrôleur flou numérique (2 entrées / 1 sortie) conçu en VHDL et implémenté sur FPGA.

La justification du choix des fonctions d'appartenances de la distance, de la vitesse et du freinage, ainsi que l'élaboration intuitive des règles ont permis l'implémentation d'un circuit répondant aux contraintes formulées dans le cahier des charges.

La simulation de divers cas et situations a montré la complexité du processus de freinage. Alors nous avons introduit un indice de confort afin de faciliter l'implémentation et éviter le recours à la quantification complète des effets qui concourent durant la phase de freinage.

Ce contrôleur flou dédié aux applications rapides, est aussi flexible dû à l'architecture modulaire exploitée. Néanmoins il peut avoir besoin d'un processeur hôte pour la gestion des opérations courantes et ordinaires car il a été volontairement démuni du pouvoir de gestion au dépend de la rapidité. Parmi les opérations déléguées au processeur hôte, on cite :

- gestion de la phase de démarrage.
- Communication avec d'autres systèmes.
- Adressage et gestion de priorité
- ….etc.

CONCLUSION GENERALE

La conception d'un contrôleur flou présentée dans ce travail sous forme d'un IP soft et implantée sur FPGA a montré que les facilités de développement offertes par les moyens numériques modernes à de hauts niveaux d'abstraction permettent effectivement une bonne mise en œuvre des techniques floues pour résoudre des problèmes difficiles à modéliser et exigeant des traitements rapides.

L'organisation et la structure générale de notre contrôleur ont suivi les bases du modèle de A. Gabrielli mais où nous avons introduit les mécanismes de test à travers la DFT à l'échelle comportementale. La conception et le test ont été conduits en parallèle.

Les divers domaines d'études concernés par ce mémoire sont multiples, car il s'agit de maîtriser d'abord les notions de la logique floue, la conception de processeurs et contrôleurs, les techniques de conception en vue de test et particulièrement l'utilisation optimale d'une plateforme de développement telle que Xilinx 7 .1i et le simulateur de Mentor Graphic pour mettre au point un produit sous forme d'IP.

Les différents programmes développés, pour la conception de notre contrôleur, montrent que la stratégie suivie repose essentiellement sur la production d'une solution simple, modulaire et qui respecte les caractéristiques de portabilité et généricité imposées par la production des IPs.

Les simulations et les implantations multiples de notre contrôleur ont permis de dresser une vue claire sur les implantations possibles de circuits dans les FPGA.

Les nouvelles FPGA (Virtex, Spartan…etc) implémentent facilement notre contrôleur et présentent des fréquences de travail supérieures aux exigences initialement consignées dans le cahier de charge.

Notre première contribution scientifique peut se résumer à la mise au point effective d'un contrôleur de fréquence 167.780 MHz sur FPGA de type Virtex4 « 4vlx25ff668 », mais l'apprentissage cumulé de différents domaines de l'électronique moderne afin d'effectuer ce travail peut être considéré comme une bonne introduction à la recherche.

Comme principales perspectives, nous pensons qu'il serait utile d'adjoindre au contrôleur flou lui même une interface souple pour faciliter l'interaction et la communication avec des processeurs hôtes car notre contrôleur flou est volontairement démuni du pouvoir de gestion de traitements standards afin qu'il accomplisse d'une manière rapide le traitement flou.

Le développement de plusieurs noyaux IP spécifiques travaillant en parallèle est fortement souhaité pour accélérer les traitements en temps réel dans les applications multitâches et fortement exigeantes en circulation de données.

BIBLIOGRAPHIE

Bibliographie

[1] Michael L. Bushnell, Vishwani D. Agrawal,“ Essentials Of Electronic Testing For Digital, Memory And Mixed-Signal VLSI Circuits” Kluwer Academic Publishers, 2002.

[2] B. Wilkins "testing digital circuits: An introduction ", Van Nostand Reinheld (UK) Co. Ltd, 1986.

[3] Jan Jantzen, “Foundations of Fuzzy Control” , 2007 John Wiley & Sons

[4] Himanshu Bhatnagar, “Advanced ASIC chip synthesis”, Second edition, Kluwer Academic Publishers, 2002.

[5] Kevin M. Passino, Stephen Yurkovich, “Fuzzy Control”, An Imprint Of Addison-Wesley Longman, Inc.1998.

[6] B. Bouchon, "La logique floue et ces applications", Addison Wesley, France, 1995.

[7] José Galindo, Angélica Urrutia, Mario Piattini, “Fuzzy Databases: Modeling, Design and implementation”, IDEA Group Publishing, 2006.

[8] P. Borne, J. Rozinoer, J.-Y. Dieulot, L.Dubois, "Introduction à la Commande Floue", Editions TECHNIP, 1998.

[9] B. Robert, “Fuzzy and Neural Control” Delft, University of Technology, Netherlands, October 2001.

[10] N. Goléa, "Identification et Commande Adaptative Floues", Thèse d’état, Département Electronique, Sétif, 2002.

[11] J. Jantzen, "Design of Fuzzy Controllers", Technical University of Denmark. Tech. report N° 98-E 864, 1998.

[12] Leonid Reznik “Fuzzy controllers” Victoria university of technology, melbourne, Australia, 1997

[13] Togai M. And Watanabe H. “Expert Systems on a Chip: an Engine for Real- time Approximate reasoning”, IEEE Expert Systems Magazine, vol.1, 1986, pp. 55-62.

[14] G. Ascia , V. Catania“A Parallel Processor Architecture for Real-Time Fuzzy”, Kluwer Academic Publishers, Fuzzy hardware: Architectures and Applications, 1998

[15] Carlos Dualibe, Michel Verleysen, Paul G.A. Jespers, “Design of Analog Fuzzy Logic Controllers in CMOS Technologies, Implementation, Test and Application”, Kluwer Academic Publishers, 2003

[16] A. kandel, G. Langholz, “Fuzzy hardware: Architectures and Applications”, Kluwer Academic Publishers, Bostan, 1998.

[17] Market J. Patyra, Janos L. Grantner, “Digital Fuzzy Logic Controller: Design and Implementation”, IEEE Transactions on Fuzzy Systems, Vol. 4, NO. 4, November 1996.

[18] Hassan Elsalah "Utilisation classique et Extension des Fonctionnalités des processeurs flous " thèse de doctorat, université de Savoie, 1992.

[19] Chiu S, “ Software Tools for Fuzzy Control”, In: Industrial Applications of Fuzzy Logic and Intelligent Systems ed. J. Yen, R. Langari and L.A. Zadeh, IEEE Press,1995,

pp. 313-40

[20] M. Abramovici, M. Breur, A. Friedman, "Digital systems testing and testable design", Computer Science Press England, 1990.

[21] Benoît charlot, "Modélisation de fautes et conception en vue du test structurel des microsystèmes", thèse de doctorat à l’institut National Polytechnique de Grenoble, 2001

[22] R. Rajsuman, “System on Chip: Design and Test”, Artech House signal processin library, 2000.

[23] T. W. Williams, «VLSI Testing», North-Holland, 1986.

[24] Alexander Miczo, “Digital Logic Testing and Simulation”, second edition,

Wiley-Interscience, A John Wiley & Sons, INC., Publication, 2003.

[25] H. Fujowara "Logic Testing and Design for Testability", the MIT Press Cambridge, March 1986.

[26] David Hély, “Conception en vue du test de circuits sécurisés", thèse de doctorat à l’université Montpellier, 2005.

[27] C. M. Maunder, R. E. Tullos "An introduction to the boundary scan standard: ANSI/IEEE Std 1149.1", IEEE journal of electronic testing, 1991.

[28] Texas Instruments "IEEE std 1149.1 (JTAG) Boundary scan Testability primer", 1996.

[29] M.Lubaszewski "Le test unifie des cartes appliqué à la conception des systèmes fiables", thèse de doctorat à l’institut national polytechnique de Grenoble, 1994.

[30] “Standard Test Access Port and Boundary-Scan Architecture”. IEEE Std 1149.1-1990, 1990.

[31] V. Muresan, D.Crisu, X. Wang, "From VHDL to FPGA a case study of a fuzzy logic controller", proceeding of international conference of young lectures, p 83-90,11-17. August 1997.

[32] K. M. Deliparaschos, F. I. Nenidakis, S. G. Tzafestas, "A Fast Digital Fuzzy Logic Controller: FPGA Design and Implementation", IEEE 2005.-

[33] A. Gabrielli, A Gandolfi, M. Masetti, "Short Time Decision VLSI fuzzy Processor", Fuzzy hardware: Architectures and Applications, Kluwer Academic Publishers Bostan, 1998.

[34] D, Flachieri, A. Gabrielli, A Gandolfi, M. Masetti "A two input fuzzy chip running at a processing rate of 30 nanosecond realized in 0.35 micron CMOS technology", University of Bologna and Istituto Nazionale di Fisica Nucleare

[35] A. Gabrielli, A Gandolfi, M. Masetti "Design of a Very High Speed Fuzzy Processor by VHDL Language", Physics Department University of bologna, 1996.

[36] D. Pellerin "An Introduction to HDLs for Simulation and Synthesis", Protel Technology, Inc, 2000.

[37] A. Vachoux "Modélisation de Systèmes Intégrés Numériques Introduction à VHDL", Antrim Design Systems, Inc. 2001.

[38] J. Weber, M. Meaudre "VHDL du langage au circuit, du circuit au langage", Masson, 1997.

[39] Xilinx, “ ISE 7 Software Manuals and help”, 2005.

[40] Xilinx, “Synthesis and Simulation Design Guide 9.1i”, 1995-2007

[41] M. Sugeno – “Industrial Application of Fuzzy Control” – Elsevier Science Publishers Co., 1985

[42] A. kandel, G. Langholz, “Fuzzy hardware: Architectures and Applications”, Kluwer Academic Publishers, Bostan, 1998.

[43] A. Milenkovic, D. Fatzer “Teaching IP Core Development: An Example” Proceedings of the International Conference on Microelectronic Systems Education (MSE’03) IEEE 2003.

[44] L. Sekanina, “Towards Evolvable IP Cores for FPGAs”, Proceedings of The 2003 NASA/Dod Conference on Evolvable Hardware, IEEE, 2003.

[45] J. Theo, W. Tung, “BIST for Deep Submicron ASIC Memories with High Performance Application“, ITC, 2003.

[46] Slimane Boutobza,”Outils de génération de structures BIST/BISR pour mémoires ‘’, thèse de doctorat à l’Institut National Polytechnique de Grenoble , spécialité : microelectronique,2002.

[47] F. Celeiro, L. Dias, J. Ferreira, “VHDL Fault Simulation for Defect-Oriented Test and Diagnosis of Digital ICs“, IEEE. European Design Automation Conference with EURO-VHDL'96 and exhibition, Vol 26, N° 2, 1996.

[48] E. Jenn, J. Arlat, M. Rimén, "Fault Injection into VHDL Models: The MEFISTO Tool", Fault Tolerant Computer Symposium. (FTCS), IEEE, 1994.

[49] S. Seward, P. Lala, "Fault Injection for Verifying Testability at the VHDL Level", International test conference, ITC, 2003.

[50] I. Mezeh, L .Boudjlida, "Conception d'un microcontrôleur IP" thèse d'ingénieur, Dpt Electronique, Sétif, 2005.

[51] Christian Wolmar, “The Subterranean Railway: How the London Underground Was Built and How It Changed the City Forever”, Atlantic Books, London, 2005.

[52] http://www.foudurail.net/frein.html.

ANNEXES

ANNEXE (A)

Programme principal du contrôleur flou

```
----------------------------------------------------------------------------------
-- Company: UFAS ( Sétif)
-- Engineer: TEST LAB
-- Create Date: 12/03/2007
-- Design Name:
-- Module Name: CONTROLLEUR FLOU.VHD
-- Project Name:  FUZZY Logic CONTROLLER IP Cores.
-- Target Device:
-- Tool versions: Web_Pack 7.1
-- Description: STRUCTURAL PROGRAM OF A FUZZY Logic
--                    CONTROLLER IP Cores
-- CONTROLEUR FLOU SE COMPOSE DE 3 COMPOSANTES( fuzzification, inference et de defuzzification)
--
-- Dependencies:
-- Maximum Frequency: FEREQUENCE in 4vlx25ff676-12
--
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity controller is

   port (
        CLK                : IN std_logic;
        MIN_PROD       : IN std_logic;

        Write_Alpha     : IN std_logic;
        write_Rules       : IN std_logic;
                                write_intervals : IN std_logic;

        Address_Alpha   : IN std_logic_vector (5 downto 0);
        Address_Rules    : IN std_logic_vector (5 downto 0);
                                Address_Points  : IN std_logic_vector (3 downto 0);

        DATA_Alpha        : IN std_logic_vector(5 downto 0);
        DATA_rules         : IN std_logic_vector(2 downto 0);
        DATA_Points        : IN std_logic_vector(5 downto 0);

        X0                      : IN std_logic_vector (5 downto 0);
        X1                      : IN std_logic_vector (5 downto 0);


        SORTIE                 : OUT std_logic_vector(5 downto 0);
        Sor                      : out integer);
end controller;
```

```
architecture Structural of Controller is
--*****************************************************************************
COMPONENT fuzzification
port(                   ck,Wi,Wa: in std_logic;
                        Xo,X1 : in std_logic_vector(5 downto 0);

                        RAD_int: in std_logic_vector(3 downto 0);
                        DATA_PT: in std_logic_vector(5 downto 0);

                        RAD_ALPH: in std_logic_vector(5 downto 0);
                        data_alph : in std_logic_vector(5 downto 0);


                         Rule_address1_out : out std_logic_vector(5 downto 0);
                        Rule_address2_out : out std_logic_vector(5 downto 0);
                        Rule_address3_out : out std_logic_vector(5 downto 0);
                        Rule_address4_out : out std_logic_vector(5 downto 0);

                        alpha0_out : out std_logic_vector(2 downto 0);
                        alpha1_out : out std_logic_vector(2 downto 0);
                        alpha2_out : out std_logic_vector(2 downto 0);
                        alpha3_out : out std_logic_vector(2 downto 0));
END COMPONENT;
--*****************************************************************************
COMPONENT inference
port( ck:in std_logic;
      MIN_PROD: in std_logic;

                      rule_ad_in1 : in std_logic_vector(5 downto 0);
                      rule_ad_in2 : in std_logic_vector(5 downto 0);
                      rule_ad_in3 : in std_logic_vector(5 downto 0);
                      rule_ad_in4 : in std_logic_vector(5 downto 0);
                        w: in std_logic;
                      rules_aderr: in std_logic_vector(5 downto 0);
                      Regle_IN : in std_logic_vector(2 downto 0);
                      alpha0_in,alpha1_in:in std_logic_vector(2 downto 0);
                      alpha2_in,alpha3_in:in std_logic_vector(2 downto 0);
                      z1_out,z2_out: out std_logic_vector(2 downto 0);
                      z3_out,z4_out: out std_logic_vector(2 downto 0);
                      teta0_out,teta1_out:out std_logic_vector(2 downto 0);
                      teta2_out,teta3_out:out std_logic_vector(2 downto 0));
END COMPONENT;
--===================================================================
Component defuzzification
 port (                 ck : in std_logic;
                        z1,z2,z3,z4 : in std_logic_vector(2 downto 0);
                        t1,t2,t3,t4 : in std_logic_vector(2 downto 0);
                        s_def :out std_logic_vector(5 downto 0);
                        srot:out integer);
end component;
--===================================================================
  signal BUS_alpha              : std_logic_vector (11 downto 0);
  signal BUS_adresse_regles : std_logic_vector(23 downto 0);
  signal BUS_teta               : std_logic_vector(11 downto 0);
  signal BUS_consequent      : std_logic_vector(11 downto 0);
```

```
begin
Inst_fuzzification: fuzzification PORT MAP( CK => CLK,Xo => X0,X1=> X1,
                                        Wi => write_intervals, Wa => Write_Alpha,
                                        RAD_int => Address_Points,DATA_PT => DATA_Points,
                                        RAD_ALPH => Address_Alpha,data_alph=> DATA_Alpha,

                        Rule_address1_out =>BUS_adresse_regles(5 downto 0),
                        Rule_address2_out =>BUS_adresse_regles(11 downto 6),
                        Rule_address3_out =>BUS_adresse_regles(17 downto 12),
                        Rule_address4_out =>BUS_adresse_regles(23 downto 18),

                                        alpha0_out  =>BUS_alpha(2 downto 0)  ,

                                        alpha1_out  =>BUS_alpha(5 downto 3) ,

                                        alpha2_out   => BUS_alpha(8 downto 6),

                                        alpha3_out   => BUS_alpha(11 downto 9));
--*************************************************************************
Inst_inference: inference PORT MAP(          CK => CLK,
                        MIN_PROD => MIN_PROD,
                        rule_ad_in1 => BUS_adresse_regles(5 downto 0),
                        rule_ad_in2 => BUS_adresse_regles(11 downto 6),
                        rule_ad_in3 => BUS_adresse_regles(17 downto 12),
                        rule_ad_in4 =>BUS_adresse_regles(23 downto 18) ,

                        w => write_rules,
                        rules_aderr => Address_Rules,
                        Regle_IN    => DATA_rules ,

                        alpha0_in    => BUS_alpha(2 downto 0),
                        alpha1_in    => BUS_alpha(5 downto 3),
                        alpha2_in    =>BUS_alpha(8 downto 6),
                        alpha3_in   => BUS_alpha(11 downto 9),
                        z1_out   => BUS_consequent(2 downto 0),
                        z2_out   => BUS_consequent(5 downto 3),
                        z3_out   =>BUS_consequent(8 downto 6),
                       z4_out  => BUS_consequent(11 downto 9),
                       teta0_out=> BUS_teta(2 downto 0),
                       teta1_out=> BUS_teta(5 downto 3),
                       teta2_out=> BUS_teta(8 downto 6),
                       teta3_out=> BUS_teta(11 downto 9));

--*************************************************************************
inst_defuzzification: defuzzification port map (CK => CLK,
                        z1     =>BUS_consequent(2 downto 0),
                        z2     =>BUS_consequent(5 downto 3),
                        z3     => BUS_consequent(8 downto 6),
                        z4    =>BUS_consequent(11 downto 9),
                        t1   => BUS_teta(2 downto 0),
                        t2   => BUS_teta(5 downto 3),
                        t3   => BUS_teta(8 downto 6),
                        t4  => BUS_teta(11 downto 9),
                        s_def => SORTIE,
                        srot => sor );
End Structural;
```

ANNEXE (B)

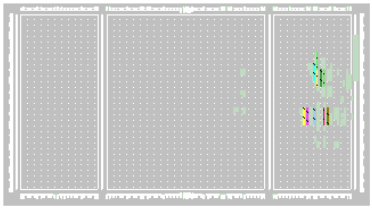
ANNEXE (C)

Synthèse du contrôleur flou avec différents types de FPGA

1. VirtexE (v300epq240-8)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 3072 3%
Number of Slice Flip Flops:	120 out of 6144 1%
Number of 4 input LUTs:	89 out of 6144 1%
Number of bonded IOBs:	73 out of 162 45%
Number of BRAMs:	2 out of 32 6%
Number of GCLKs:	1 out of 4 25%



(Maximum frequency :475.285 MHz)

Figure C.1: *Implantation physique du bloc d'inférence sur FPGA « VirtexE»*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	66 out of 3072 2%
Number of Slice Flip Flops:	48 out of 6144 0%
Number of 4 input LUTs:	108 out of 6144 1%
Number of bonded IOBs:	72 out of 162 44%
Number of BRAMs:	4 out of 32 12%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 288.184MHz)

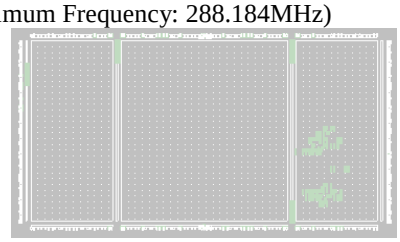


Figure C.2: *Implantation physique du bloc d'inférence sur FPGA « VirtexE»*

Bloc de défuzzification

Elément	Utilisation %
Number of Slices:	148 out of 3072 4%
Number of Slice Flip Flops:	68 out of 6144 1%
Number of 4 input LUTs:	209 out of 6144 3%
Number of bonded IOBs:	63 out of 162 38%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 168.435MHz)

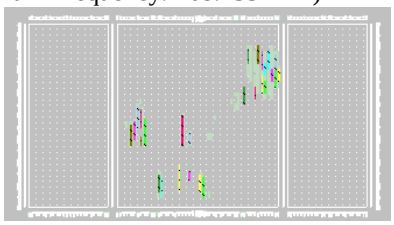


Figure C.3: *Implantation physique du bloc de defuzzification sur FPGA «VirtexE»*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	273 out of 3072 8%
Number of Slice Flip Flops:	236 out of 6144 3%
Number of 4 input LUTs:	408 out of 6144 6%
Number of bonded IOBs:	86 out of 162 53%
Number of BRAMs:	6 out of 32 18%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 103.929MHz)

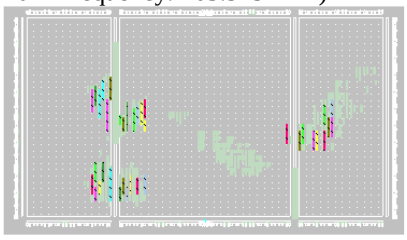


Figure C.4: *Implantation physique du module Contrôleur Flou sur FPGA «VirtexE»*

2. Virtex2 (2v250fg456-6)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 1536 6%
Number of Slice Flip Flops:	120 out of 3072 3%
Number of 4 input LUTs:	89 out of 3072 2%
Number of bonded IOBs:	73 out of 200 36%
Number of BRAMs:	2 out of 24 8%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 889.284MHz)

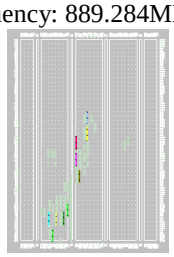


Figure C.5: *Implantation physique du bloc de Fuzzification sur FPGA « Vitex2 »*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	71 out of 1536 4%
Number of Slice Flip Flops:	48 out of 3072 1%
Number of 4 input LUTs:	118 out of 3072 3%
Number of bonded IOBs:	72 out of 200 36%
Number of BRAMs:	4 out of 24 16%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 450.450MHz)

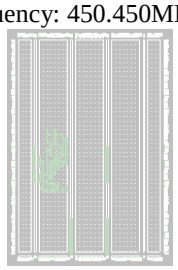


Figure C.6: *Implantation physique du bloc d'inférence sur FPGA « Vitex2 »*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	91 out of 1536 5%
Number of Slice Flip Flops:	32 out of 3072 1%
Number of 4 input LUTs:	157 out of 3072 5%
Number of bonded IOBs:	63 out of 200 31%
Number of MULT18X18s:	4 out of 24 16%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 166.459MHz)

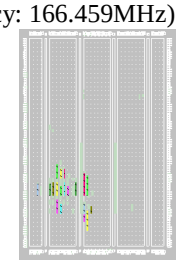


Figure C.7: *Implantation physique du bloc de defuzzification sur FPGA « Vitex2 »*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	217 out of 1536 14%
Number of Slice Flip Flops:	200 out of 3072 6%
Number of 4 input LUTs:	370 out of 3072 12%
Number of bonded IOBs:	86 out of 200 43%
Number of BRAMs:	6 out of 24 25%
Number of GCLKs:	1 out of 16 6%
Number of MULT18X18s:	4 out of 24 16%

(Maximum Frequency: 141.127MHz)

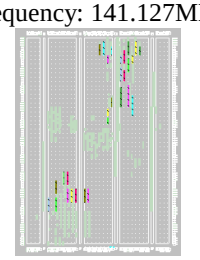


Figure C.8: *Implantation physique du module Contrôleur Flou sur FPGA « Vitex2 »*

3. Virtex2P (2vp2ff672-7)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 1408 6%
Number of Slice Flip Flops:	120 out of 2816 4%
Number of 4 input LUTs:	89 out of 2816 3%
Number of bonded IOBs:	73 out of 204 35%
Number of BRAMs:	2 out of 12 16%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 1099.505MHz)

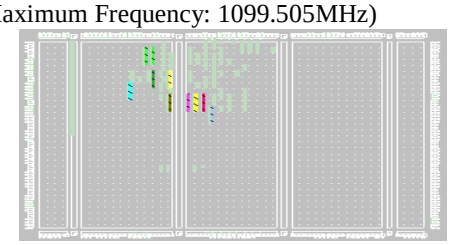


Figure C.9: *Implantation physique du bloc de Fuzzification sur FPGA « Vitex2P »*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	71 out of 1408 5%
Number of Slice Flip Flops:	48 out of 2816 1%
Number of 4 input LUTs:	118 out of 2816 4%
Number of bonded IOBs:	72 out of 204 35%
Number of BRAMs:	4 out of 12 33%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 482.276MHz)

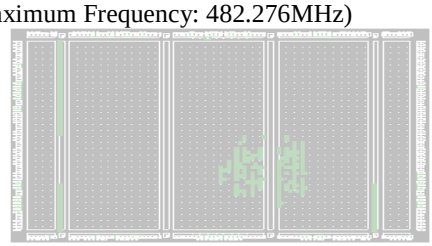


Figure C.10: *Implantation physique du bloc d'inférence sur FPGA« Vitex2P »*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	91 out of 1408 6%
Number of Slice Flip Flops:	32 out of 2816 1%
Number of 4 input LUTs:	157 out of 2816 5%
Number of bonded IOBs:	63 out of 204 30%
Number of MULT18X18s:	4 out of 12 33%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 204.489MHz)

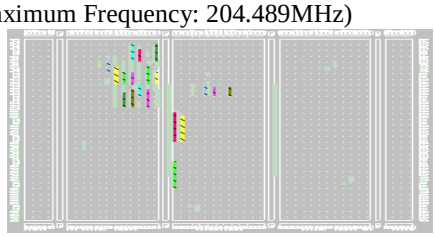


Figure C.11: *Implantation physique du bloc de defuzzification sur FPGA« Vitex2P »*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	217 out of 1408 15%
Number of Slice Flip Flops:	200 out of 2816 7%
Number of 4 input LUTs:	368 out of 2816 13%
Number of bonded IOBs:	86 out of 204 42%
Number of BRAMs:	6 out of 12 50%
Number of MULT18X18s:	4 out of 12 33%
Number of GCLKs:	1 out of 16 6%

(Maximum Frequency: 181.536MHz)

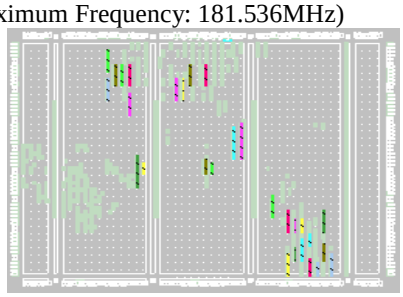


Figure C.12: *Implantation physique du module Contrôleur Flou sur FPGA« Vitex2P »*

4. Spartan3E (3s500epq208-4)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 4656 2%
Number of Slice Flip Flops:	120 out of 9312 1%
Number of 4 input LUTs:	89 out of 9312 0%
Number of bonded IOBs:	73 out of 158 46%
Number of BRAMs:	2 out of 20 10%
Number of GCLKs:	1 out of 24 4%

(Maximum Frequency: 542.005MHz)

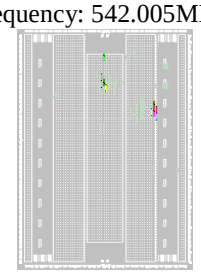


Figure C.13: *Implantation physique du bloc de fuzzification sur FPGA «Spartan3E»*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	71 out of 4656 1%
Number of Slice Flip Flops:	48 out of 9312 0%
Number of 4 input LUTs:	118 out of 9312 1%
Number of bonded IOBs:	72 out of 158 45%
Number of BRAMs:	4 out of 20 20%
Number of GCLKs:	1 out of 24 4%

(Maximum Frequency: 227.015MHz)

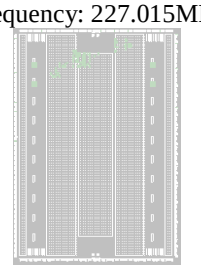


Figure C.14: *Implantation physique du bloc d'inférence sur FPGA «Spartan3E»*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	116 out of 4656 2%
Number of Slice Flip Flops:	32 out of 9312 0%
Number of 4 input LUTs:	157 out of 9312 1%
Number of bonded IOBs:	63 out of 158 39%
Number of MULT18X18s:	4 out of 20 20%
Number of GCLKs:	1 out of 24 4%

(Maximum Frequency: 111.309MHz)

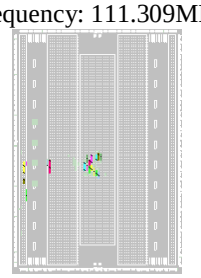


Figure C.15: *Implantation physique du bloc de defuzzification sur FPGA «Spartan3E»*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	249 out of 4656 5%
Number of Slice Flip Flops:	200 out of 9312 2%
Number of 4 input LUTs:	380 out of 9312 4%
Number of bonded IOBs:	86 out of 158 54%
Number of BRAMs:	6 out of 20 30%
Number of MULT18X18s:	4 out of 20 20%
Number of GCLKs:	1 out of 24 4%

(Maximum Frequency: 83.795MHz)

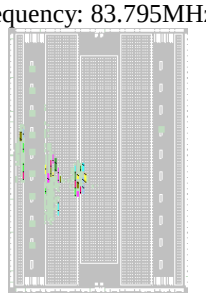


Figure C.16: *Implantation physique du module Contrôleur Flou sur FPGA «Spartan3E»*

5. Spartan3 (3s1500lfg676-4)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 13312 0%
Number of Slice Flip Flops:	120 out of 26624 0%
Number of 4 input LUTs:	89 out of 26624 0%
Number of bonded IOBs:	73 out of 487 14%
Number of BRAMs:	2 out of 32 6%
Number of GCLKs:	1 out of 8 12%

(Maximum Frequency: 580.046MHz)

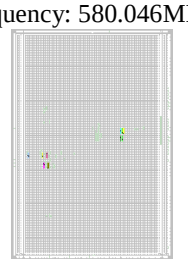


Figure C.17: *Implantation physique du bloc de fuzzification sur FPGA«Spartan3»*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	71 out of 13312 0%
Number of Slice Flip Flops:	48 out of 26624 0%
Number of 4 input LUTs:	118 out of 26624 0%
Number of bonded IOBs:	72 out of 487 14%
Number of BRAMs:	4 out of 32 12%
Number of GCLKs:	1 out of 8 12%

(Maximum Frequency: 231.803MHz)

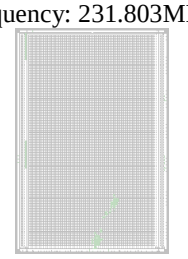


Figure C.18: *Implantation physique du bloc d'inférence sur FPGA«Spartan3»*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	116 out of 13312 0%
Number of Slice Flip Flops:	32 out of 26624 0%
Number of 4 input LUTs:	157 out of 26624 0%
Number of bonded IOBs:	63 out of 487 12%
Number of MULT18X18s:	4 out of 32 12%
Number of GCLKs:	1 out of 8 12%

(Maximum Frequency: 122.549MHz)

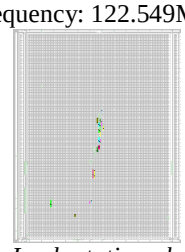


Figure C.19: *Implantation physique du bloc de defuzzification sur FPGA «Spartan3»*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	251 out of 13312 1%
Number of Slice Flip Flops:	200 out of 26624 0%
Number of 4 input LUTs:	374 out of 26624 1%
Number of bonded IOBs:	86 out of 487 17%
Number of BRAMs:	6 out of 32 18%
Number of MULT18X18s:	4 out of 32 12%
Number of GCLKs:	1 out of 8 12%

(Maximum Frequency: 95.793MHz)

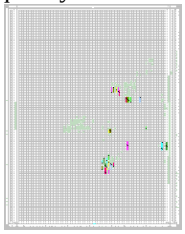


Figure C.20: *Implantation physique du module Contrôleur sur FPGA «Spartan3»*

6. Spartan2E(2s300epq208-7)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 3072 3%
Number of Slice Flip Flops:	120 out of 6144 1%
Number of 4 input LUTs:	89 out of 6144 1%
Number of bonded IOBs:	73 out of 146 50%
Number of BRAMs:	2 out of 16 12%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 423.370MHz)

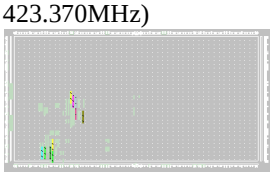


Figure C.21: *Implantation physique du bloc de Fuzzification sur FPGA «Spartan2E»*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	66 out of 3072 2%
Number of Slice Flip Flops:	48 out of 6144 0%
Number of 4 input LUTs:	108 out of 6144 1%
Number of bonded IOBs:	72 out of 146 49%
Number of BRAMs:	4 out of 16 25%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 257.069MHz)

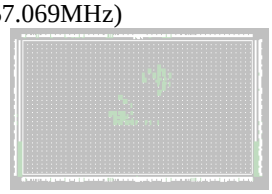


Figure C.22: *Implantation physique du bloc d'inférence sur FPGA «Spartan2E»*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	148 out of 3072 4%
Number of Slice Flip Flops:	68 out of 6144 1%
Number of 4 input LUTs:	209 out of 6144 3%
Number of bonded IOBs:	63 out of 146 43%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 143.308MHz)

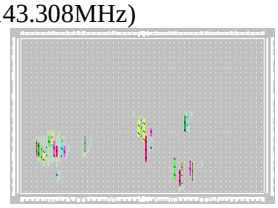


Figure C.23: *Implantation physique du bloc de defuzzification sur FPGA «Spartan2E»*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	273 out of 3072 8%
Number of Slice Flip Flops:	236 out of 6144 3%
Number of 4 input LUTs:	408 out of 6144 6%
Number of bonded IOBs:	86 out of 146 58%
Number of BRAMs:	6 out of 16 37%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 87.958MHz)

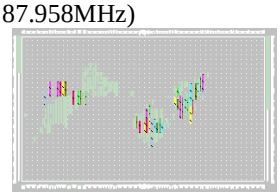


Figure C.24: *Implantation physique du module Contrôleur sur FPGA «Spartan2E»*

7. Spartan2 (2s200pq208-6)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 2352 4%
Number of Slice Flip Flops:	120 out of 4704 2%
Number of 4 input LUTs:	89 out of 4704 1%
Number of bonded IOBs:	73 out of 144 50%
Number of BRAMs:	2 out of 14 14%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 353.482MHz)

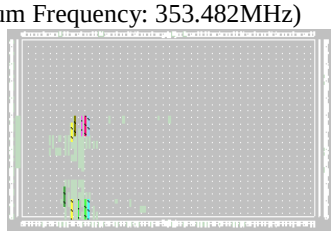


Figure C.25: *Implantation physique du bloc de Fuzzification sur FPGA «Spartan2»*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	66 out of 2352 2%
Number of Slice Flip Flops:	48 out of 4704 1%
Number of 4 input LUTs:	108 out of 4704 2%
Number of bonded IOBs:	72 out of 144 50%
Number of BRAMs:	4 out of 14 28%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 209.205MHz)

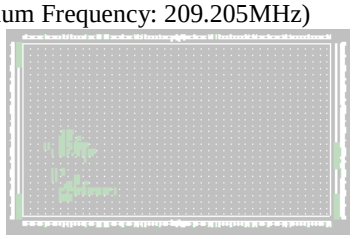


Figure C.26: *Implantation physique du bloc d'inférence sur FPGA «Spartan2»*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	148 out of 2352 6%
Number of Slice Flip Flops:	68 out of 4704 1%
Number of 4 input LUTs:	209 out of 4704 4%
Number of bonded IOBs:	63 out of 144 43%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 121.330MHz)

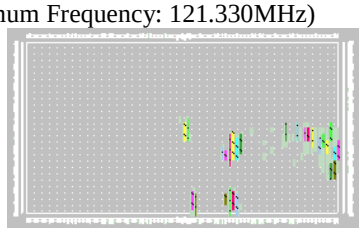


Figure C.27: *Implantation physique du bloc de defuzzification sur FPGA «Spartan2»*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	273 out of 2352 11%
Number of Slice Flip Flops:	236 out of 4704 5%
Number of 4 input LUTs:	406 out of 4704 8%
Number of bonded IOBs:	86 out of 144 59%
Number of BRAMs:	6 out of 14 42%
Number of GCLKs:	1 out of 4 25%

(Maximum Frequency: 77.967MHz)

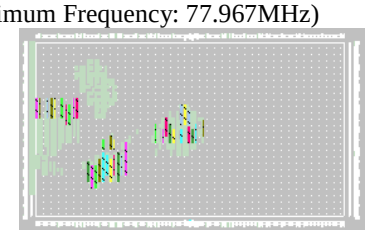


Figure C.28: *Implantation physique du module Contrôleur sur FPGA «Spartan2»*

8. Virtex 4 (4vlx25ff676-12)

Bloc de fuzzification

Elément	Utilisation %
Number of Slices:	98 out of 10752 0%
Number of Slice Flip Flops:	120 out of 21504 0%
Number of 4 input LUTs:	89 out of 21504 0%
Number of bonded IOBs:	73 out of 450 16%
Number of FIFO16/RAMB16s:	2 out of 72 2%
Number used as RAMB16s:	2
Number of GCLKs:	1 out of 32 3%

(Maximum Frequency: 1167.679MHz)

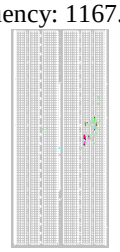


Figure C.29: *Implantation physique du bloc de fuzzification sur FPGA «Virtex4»*

Bloc d'inférence

Elément	Utilisation %
Number of Slices:	72 out of 10752 0%
Number of Slice Flip Flops:	48 out of 21504 0%
Number of 4 input LUTs:	120 out of 21504 0%
Number of bonded IOBs:	72 out of 450 16%
Number of FIFO16/RAMB16s:	4 out of 72 5%
Number used as RAMB16s:	4
Number of GCLKs:	1 out of 32 3%

(Maximum Frequency: 846.525MHz)

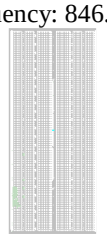


Figure C.30: *Implantation physique du bloc d'inférence sur FPGA «Virtex4»*

Bloc de defuzzification

Elément	Utilisation %
Number of Slices:	131 out of 10752 1%
Number of Slice Flip Flops:	80 out of 21504 0%
Number of 4 input LUTs:	209 out of 21504 0%
Number of bonded IOBs:	63 out of 450 14%
Number of GCLKs:	1 out of 32 3%

(Maximum Frequency: 214.675MHz)

Figure C.31: *Implantation physique du bloc de defuzzification sur FPGA «Virtex4»*

Module "Controleur"

Elément	Utilisation %
Number of Slices:	256 out of 10752 2%
Number of Slice Flip Flops:	248 out of 21504 1%
Number of 4 input LUTs:	428 out of 21504 1%
Number of bonded IOBs:	86 out of 450 19%
Number of FIFO16/RAMB16s:	6 out of 72 8%
Number used as RAMB16s:	6
Number of GCLKs:	1 out of 32 3%

(Maximum Frequency: 165.780MHz)

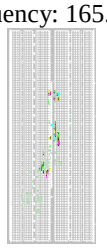


Figure C.32: *Implantation physique du Module Contrôleur sur FPGA «Virtex4»*

ANNEXE (D)

Synthèse du contrôleur flou Testable avec différents types de FPGA

Virtex2 (2v250fg456-6)

Elément	Utilisation %
Number of Slices:	325 out of 1536 21%
Number of Slice Flip Flops:	382 out of 3072 12%
Number of 4 input LUTs:	492 out of 3072 16%
Number of bonded IOBs:	95 out of 200 47%
Number of BRAMs:	6 out of 24 25%
Number of MULT18X18s:	4 out of 24 16%
Number of GCLKs:	3 out of 16 18%

(Maximum Frequency: 49.349MHz)

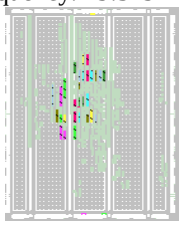


Figure D.1: *Implantation physique du Module Contrôleur Testable sur FPGA «Virtex2»*

Virtex2P (2vp2ff672-7)

Elément	Utilisation %
Number of Slices:	328 out of 1408 23%
Number of Slice Flip Flops:	382 out of 2816 13%
Number of 4 input LUTs:	496 out of 2816 17%
Number of bonded IOBs:	95 out of 204 46%
Number of BRAMs:	6 out of 12 50%
Number of MULT18X18s:	4 out of 12 33%
Number of GCLKs:	3 out of 16 18%

(Maximum Frequency: 57.002MHz)

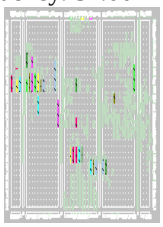


Figure D.2: *Implantation physique du Module Contrôleur Testable sur FPGA «Virtex2P»*

Spartan3 (3s1500lfg676-4)

Elément	Utilisation %
Number of Slices:	349 out of 13312 2%
Number of Slice Flip Flops:	382 out of 26624 1%
Number of 4 input LUTs:	495 out of 26624 1%
Number of bonded IOBs:	95 out of 487 19%
Number of BRAMs:	6 out of 32 18%
Numberof MULT18X18s:	4 out of 32 12%
Number of GCLKs:	3 out of 8 37%

(Maximum Frequency: 31.861MHz)

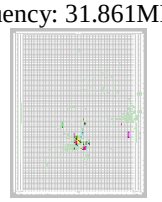


Figure D.3: *Implantation physique du Module Contrôleur Testable sur FPGA «Spartan3»*

Virtex4 (3s1500lfg676-4)

Elément	Utilisation %
Number of Slices:	366 out of 10752 3%
Number of Slice Flip Flops:	430 out of 21504 1%
Number of 4 input LUTs:	547 out of 21504 2%
Number of bonded IOBs:	95 out of 450 21%
Number of BRAMs:	6 out of 72 8%
Number used as RAMB16s:	6
Number of GCLKs:	3 out of 32 9%

(Maximum Frequency: 73.956MHz)

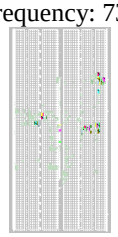


Figure D.4: *Implantation physique du odule Contrôleur Testable sur FPGA*