République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Abderrahmane Mira

Faculté de la Technologie

Département d'Automatique, Télécommunication et d'Electronique

Projet de Fin d'Etudes

En vue de l'obtention du diplôme de Master

Filière : Automatique

Spécialité : Automatique et informatique industrielle

Thème

Conception d'un contrôleur flou de type Mamdani en langage VHDL

Préparé par :

BERKANI Nawel

HAMZAOUI Nabil

Dirigé par :

M.F.YAHIAOUI

Mme F.ZAOUCHE

Examiné par :

M.B.MENDIL

M.L.TIGHZERT

Année universitaire : 2021/2022

Dédicace

Je dédie ce travail, avant tout le monde, au cœur et l'âme de toute mon existence :

Mes parents

La plus belles des mère pour les prières silencieuses et sincères , pour toutes les fois où tu t'es levé tôt afin de me préparer pour l'école et toutes les fois où tu t'es couché tard pour t'assurer que tu m'as bien enseigné... Maman ! Tu es et tu seras toujours un symbole d'espoir, de foi et d'amour dans ma vie. Et **mon merveilleux père** pour les dons sans fin et le soutien inconditionnel et pour toutes les fois où tu t'es réveillé les matins sombres et m'as attendu sous la pluie peu importe le froid qu'il faisait dehors juste pour t'assurer que j'arrivais à l'heure. Je chérirai toujours ce que vous avez fait pour moi et je vous aime également pour toujours et à jamais.

Mes frères

J'aimerais aussi dédier ce travail à mon grand frères **Fayçal** et mon petit frère **Haroune** ma vie serait vide sans vous

Je dédie ce travail à **mon petit oncle** qui nous a quitté si tôt que je n'oublierai jamais ,que dieux l'accueille dans son vaste paradis

A mes **grands-parents**, toutes **mes tantes** et **oncles**, **cousins** et **cousines**

Pour ma deuxième famille **mes amis** ! merci d'avoir créé les meilleurs moments de ma vie et le soutien que je reçois toujours de votre part !

A l'homme le plus sage que j'ai jamais rencontré, l'ex enseignant **M. MIRA Athmane** qui ma encourager à poursuivre mes études disant que la connaissance n'a jamais de fin, je voulais vous dire que ce n'est qu'un début pour moi

A tous mes enseignants du primaire à l'université, que serait le monde sans vous !

En fin j'aimerais faire ce dédicace a toutes les femmes qui voulait mais n'ont pas pu poursuivre leurs études

Nawel

Dédicace

Je dédie cet ouvrage.

A ma **mère** et mon **père** qui m'ont soutenu et encouragé durant ces années d'études.

Qu'ils trouvent ici le témoignage de ma profonde reconnaissance.

A mes **frères**, mes **amis** et ceux qui ont partagé avec moi tous les moments d'émotion lors de La réalisation de ce travail , ils m'ont chaleureusement supporté et encouragé tout au long de mon parcours.

A ma **famille**, mes **proches** et à ceux qui me donnent de l'amour et de la vivacité.

A tous mes **amis** qui m'ont toujours encouragé, et à qui je souhaite plus de succès.

A tous ceux que j'aime.

Nabil

Remerciements

Tout d'abord, nous remercions le bon Dieu de nous avoir aidé et donné le courage, la volonté et le savoir pour réaliser ce modeste travail

Nos profonds et sincères remerciements vont en premier lieu à nos encadrants **M.YAHIAOUI** et **Mme ZAOUCHE** de nous avoir aidé et guidé tout au long de ce travail, qu'ils trouvent ici l'expression de toutes nos reconnaissances et nos profonds respects pour la confiance qu'ils nous ont accordés, merci pour l'intérêt qu'ils ont portés à notre travail, merci pour leurs précieux conseils. Merci encore une fois pour votre disponibilité malgré vos multiples responsabilités. Que Dieu le tout-puissant vous protège, et vous garde pour nous tous inchallah.

Nous tiendrons également à exprimer toutes nos gratitudes aux membres de Jury, **M.MENDIL** et **M.TIGHZERT** pour leurs contributions scientifiques lors de l'évaluation de ce travail. Veuillez trouver ici l'expression de nos profonds respects et de nos sincères remerciements.

Nos vifs remerciements vont également à nos enseignants pour leurs qualités d'enseignement dispensés durant les cinq années qui furent les plus belles de notre vie.

Enfin, on adresse nos plus sincères remerciements à nos parents, sans eux on n'en serait pas là aujourd'hui, nos frères, nos sœurs et tous nos proches et amis, qui nous ont accompagnés, aidés, soutenus et encouragés tout au long de la réalisation de ce projet de fin d'étude.

Table des matières

III. Chapitre 3 : Conception d'un contrôleur flou en VHDL

Liste des figures

Liste des tableaux

Liste des abréviations et symboles

$\mu_A(x)$: Degré d'appartenance

TS : **T**akagi_**S**ugeno

COG : **C**entre **O**f **G**ravity

PLD : **P**rogramable **L**ogic **D**evice

CPLD :**C**omplex **P**rogramable **L**ogic **D**evice

SPLD :**S**imple **P**rogramable **L**ogic **D**evice

HCPLD : High **C**apacity **P**rogramable **L**ogic **D**evice

SRAM : **S**tatic **R**andom-**A**ccess **M**emory

FPGA **F**ield-**P**rogrammable **G**ate **A**rray

HDL **H**ardware **D**escription **L**anguage

VHSIC **V**ery **H**igh-**S**peed **I**ntegrated **C**ircuit

VHDL **V**ery **H**igh-**S**peed **I**ntegrated **C**ircuit **H**ardware **D**escription **L**anguage

CLB : **C**onfigurable logic **B**loc

IOB : **I**nput /**O**utput **B**loc

IEEE : **I**nstitute **O**f **E**lectrical **A**nd **E**lectronics **E**ngineers

Introduction générale

Introduction

L'incertitude du langage naturel et le mécanisme de raisonnement approximatif du cerveau humain ont été sujet de plusieurs recherches conduisant aux développements de nombreux concepts, parmi ceux on cite la logique floue. Vu la réussite de nombreuses applications à résoudre les problèmes des systèmes complexes [1,2], les commandes par logique floue ont connus un succès croissant depuis la fin du siècle dernier, notamment dans le domaine de l'automatique. La raison pour laquelle, elle est de plus en plus nécessaire dans la mise en œuvre matérielle des systèmes flous qui peuvent être facilement adaptés à diverses applications ou à des changements d'environnement dans lequel ce système est exploitée.

Actuellement, une grande variété d'outils se sont penchés à la modélisation des contrôleurs flous, principalement les dispositifs à haute vitesse de traitement tout comme les cartes FPGA. L'utilisation des FPGA dans les systèmes de contrôle industriels est d'un grand intérêt, en raison de leur haute capacité de traitement. Les FPGA ont été déjà utilisés avec succès dans différents systèmes de contrôle, soit dans la mise en œuvre des contrôleurs à logique floue réseaux de neurones, commande de moteurs asynchrones, commande de convertisseur de puissance, systèmes mécatroniques, etc.

Des recherches ont été menées dans la conception d'outils de modélisation et de la mise en œuvre du contrôleurs flous. L'implémentation FPGA de conjonctions paramétriques basées sur des fonctions génératrices a été proposée dans [5, 6]. Une étude sur l'implémentation matérielle des contrôleurs flous sur des dispositifs FPGA est présentée dans [3,4,7,8,9]. Dans les travaux [7, 8], une implémentation du contrôleur flou de type Mamdani sur une carte FPGA a été développé pour le suivi du point de puissance maximale dans un système de conversion d'énergie photovoltaïque.

Contexte du mémoire

L'objectif de ce travail est de concevoir un contrôleur flou en utilisant le langage VHDL et les principes de logique floue de type Mamdani.

Le but est de définir les variables d'entrées et de sortie, leurs univers de discours, puis les sous-ensembles flous du bloc de fuzzification. Une fois les variables floues définies, un ensemble de règles est établi en utilisant les connaissances d'un expert pour décrire le comportement du système, autrement dit un raisonnement flou est appliqué. Enfin, le modèle flou de Mamdani est choisi pour inférer sur les règles activées et les défuzzifier en utilisant la méthode du centre de gravité.

Méthodologie

Nous considérons un modèle flou de Mamdani avec deux variables d'entrées, des valeurs floues données par des fonctions d'appartenance triangulaires et trapézoïdales et une variable de sortie avec des valeurs données par des fonctions d'appartenance singletons

En langage VHDL, nous allons tout d'abord décrire les fonctions d'appartenance d'entrées et de sortie que consiste notre contrôleur, puis décrire la base de règles floues et inférer sur celles activées par la méthode d'inférence Mamdani pour pouvoir les défuzzifier en utilisant la méthode du centre de gravité. Enfin, la description du contrôleur sera simulée avec le logiciel de simulation ModelSim .

Présentation du mémoire

Le contenu de notre travail est divisé en trois chapitres dont :

Le premier chapitre présente une revue et une introduction à la logique floue, les opérations sur les ensembles flous, les principaux concepts des ensembles flous tels que les fonctions d'appartenance et les variables linguistiques, ainsi que les blocs d'un contrôleur à logique flou .

Le deuxième chapitre présente une introduction sur les FPGAs et le langage de description matérielle VHDL.

Le troisième chapitre présente la méthodologie de conception du contrôleur flou d'un système de freinage automatique en langage VHDL. En fin le contrôleur flou conçu sera vérifié en simulation à l'aide du logiciel ModelSim, et les résultats seront présentés.

Une conclusion générale viendra résumer le contenu de ce travail et mettre en avant les résultats obtenus ainsi que les perspectives de recherche.

Chapitre 1

Logique floue : Généralités et structure

I.1. Introduction

Le terme logique floue a été introduit avec la proposition de la théorie des ensembles flous par Lotfi Zadeh en 1965[10]. Des contrôles de processus par la logique floue ont été démontrés en 1975 par Ebrahim H. Mamdani et S. Asilian [11]. Les règles floues si / alors ont été utilisées pour réguler un modèle de machine à vapeur; et avec cela un grand nombre d'applications de contrôle flou ont été déployées.

Dans ce chapitre, la théorie des ensembles flous et le contrôleur de type Mamdani seront présentés.

I.2. Ensembles flous

Dans l'ensemble classique, la fonction caractéristique attribue une valeur 0 ou 1 à chaque individu de l'ensemble universel, en discriminant ainsi les membres et les non-membres de l'ensemble classique considéré [13].

Les valeurs attribuées aux éléments de l'ensemble universel se situent dans une plage spécifiée et indiquent le degré d'appartenance de ces éléments à l'ensemble. Des valeurs plus grandes indiquent des degrés plus élevés appartenant à un ensemble, une telle fonction est appelée une fonction d'appartenance et l'ensemble est défini par le fait qu'il s'agit d'un ensemble flou [13].

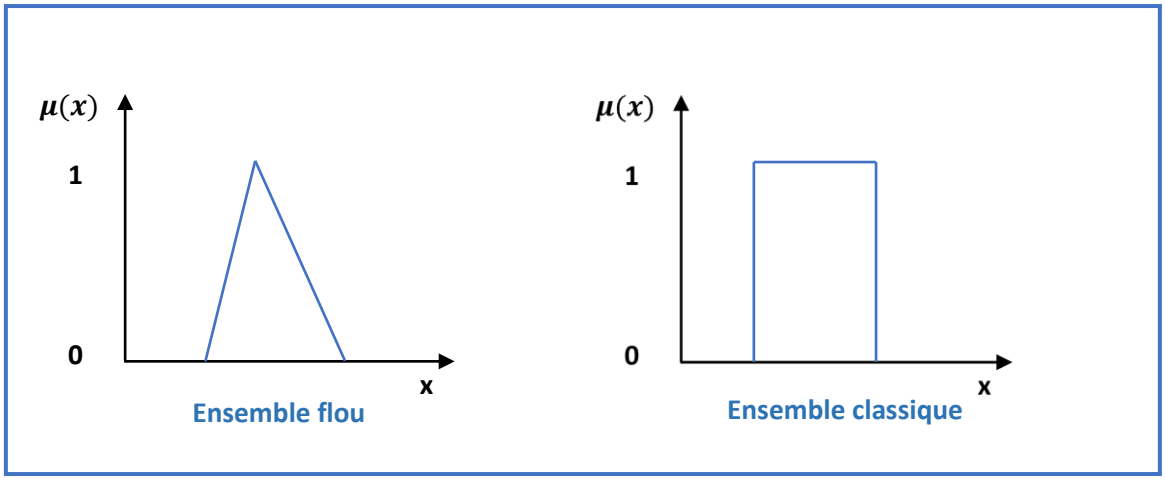

Figure I. 1 : Fonction d'appartenance de l'ensemble flou et de l'ensemble classique

Un ensemble flou est donc un ensemble contenant des éléments qui ont divers degrés d'appartenance à un ensemble .Cette idée est en contraste avec l'ensemble classique ou net car les membres d'un ensemble net ne seraient pas membres à moins que leur appartenance ne soit complète dans cet ensemble (c'est-à-dire que leur appartenance est attribuée une valeur de 1) [13].

L'appartenance des éléments d'un ensemble flou n'a pas besoin d'être complète et peuvent également être membres d'un autre ensemble flou sur le même univers.

L'ensemble flou est mappé sur une valeur numérique réelle dans l'intervalle 0 à 1. Si un élément de l'univers, disons x, est un membre de l'ensemble flou A, alors le mappage est donné par $\mu_A(x) \in [0,1]$. Donc l'ensemble flou A dans l'univers de discours U est défini comme :

$$\boldsymbol{\mu_A(x) : U \rightarrow [0,1]} \qquad \textbf{(I.1)}$$

Où μ_A est connu sous le nom de fonction d'appartenance et $\mu_A(x)$ est le degré d'appartenance de x. La fonction d'appartenance est le degré de vérité ou le degré de compatibilité, elle est le composant crucial d'un ensemble flou. Par conséquent, toutes les opérations sur les ensembles flous sont définies en fonction de leurs fonctions d'appartenance.[12]

I. 2.1. Univers de discours

Une fois que les choix des fonctions d'appartenance et de partitionnement sont faits, il reste à déterminer les univers de discours des variables du contrôleur flou. L'univers de discours est l'intervalle des valeurs des entrées/sortie du contrôleur.

I.2.2 Variables linguistiques

Une variable linguistique appelée aussi attribut linguistique. Tout comme une variable algébrique prend des nombres comme valeurs, une variable linguistique prend des mots ou des phrases comme valeurs. L'ensemble des valeurs qu'elle peut prendre est appelé un ensemble de termes [12] .

Une variable linguistique est représentée par le triplet $(x, T(x), U)$

Où,

x : est le nom de la variable linguistique.

$T(x)$: est l'ensemble de termes linguistiques de la variable x

U : est l'univers du discours qui se rapporte à la variable linguistique x

Exemple :

Si « Erreur » est une variable linguistique est définie dans l'univers de discours $U = [-1, 1]$, ses termes linguistiques peuvent être :

Grand négatif, (GN), Petit négatif (PN), Zéro (ZR), Petit positif (PP), Grand positif (GP)

L'ensemble de termes flous est donc :

T (Erreur) = {Grand négatif, (NB), Petit négatif (NS), Zéro (ZR), Petit positif (PS), Grand positif (PB)}

La figure ci-dessous illustre ces conceptions

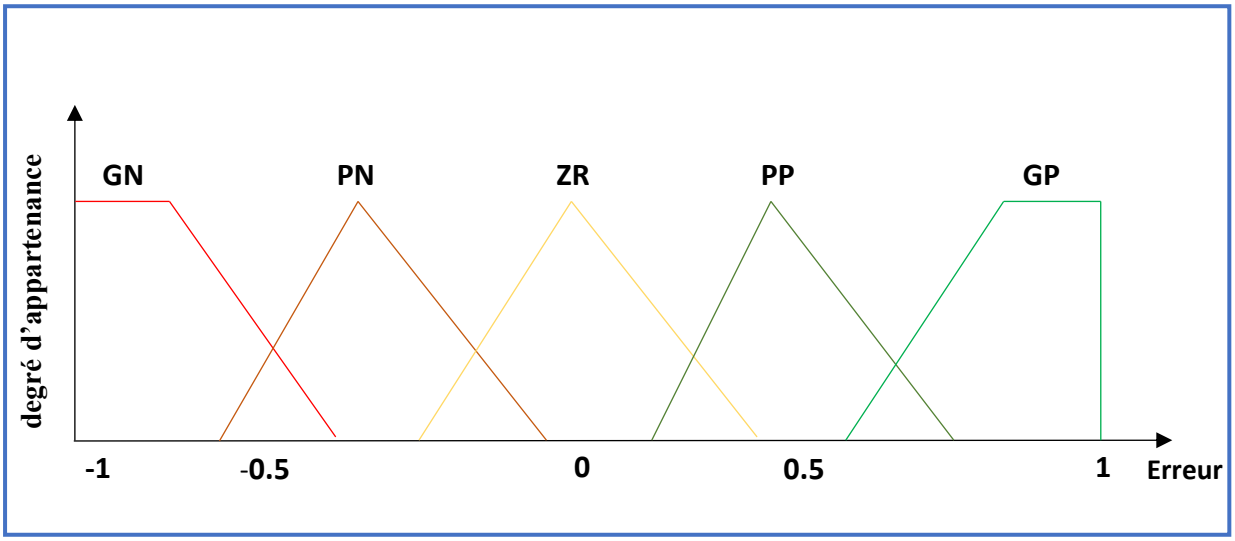

Figure I. 2:Variable linguistique "Erreur".

I.2.3. Opérations d'ensembles flous

Comme dans la logique classique, les opérations logiques AND, OR et NOT peuvent être appliquées avec des ensembles flous. La signification de ces opérateurs est comme suite :

- l'***intersection*** entre deux fonctions est représentée par l'opérateur ***AND*** qui correspond à l'opération ***minimum ;***
- l'***union*** de deux fonctions est souvent exprimée par l'opérateur ***OR*** qui est défini par l'opération ***maximum ;***
- le ***complément*** d'une fonction est représenté par l'opérateur ***NOT*** qui correspond à l'inverse de l'état logique de la valeur de la fonction.

Soient A et B deux ensembles flous dans l'univers de discours U avec deux degrés d'appartenance $\mu_A(u)$ et $\mu_B(u)$ respectivement.

Le tableau ci-dessous illustre les opérations d'ensembles flous dans la première colonne, les opérateurs logiques respectifs dans la deuxième colonne, et leurs formes respectives dans la troisième colonne.

Tableau I. 1: Résumé des opérations d'ensembles flous.

Opération	**Opérateurs logiques**	**Sa forme dans la logique floue**
Intersection	**AND**	$\mu_A(u) \cap \mu_B(u) = min(\mu_A(u), \mu_B(u)) ; \mu \in U$
Union	**OR**	$\mu_A(u) \cup \mu_B(u) = max(\mu_B(u), \mu_B(u)) ; u \in U$
Complément	**NOT**	$\mu_{\bar{A}}(u) = 1 - \mu_A(u) \quad ; u \in U$

I.3. Fonctions d'appartenance

Les termes, qui permettent de qualifier une variable linguistique, sont définis par l'intermédiaire de fonctions d'appartenance.

La fonction d'appartenance $\mu_A(x)$ décrit l'appartenance de l'élément *x* de l'ensemble de base *X* dans le sous-ensemble flou A, où pour $\mu_A(x)$une grande classe de fonctions peut être prise [12].

Exemple : La figure I.3 représente le degré d'appartenance de deux fonctions trapézoïdales. La variable x_0 peut appartenir au sous-ensemble flou A, et simultanément à un autre sous-ensemble flou B. Les degrés d'appartenance de la variable x_0 peuvent prendre deux valeurs dans les deux sous-ensembles (cf. Figure I.3):

$$degré\ d'appartance\ de\ x0 = \begin{cases} \mu_A(x_0) = 0.75 \\ \mu_B(x_0) = 0.20 \end{cases}$$

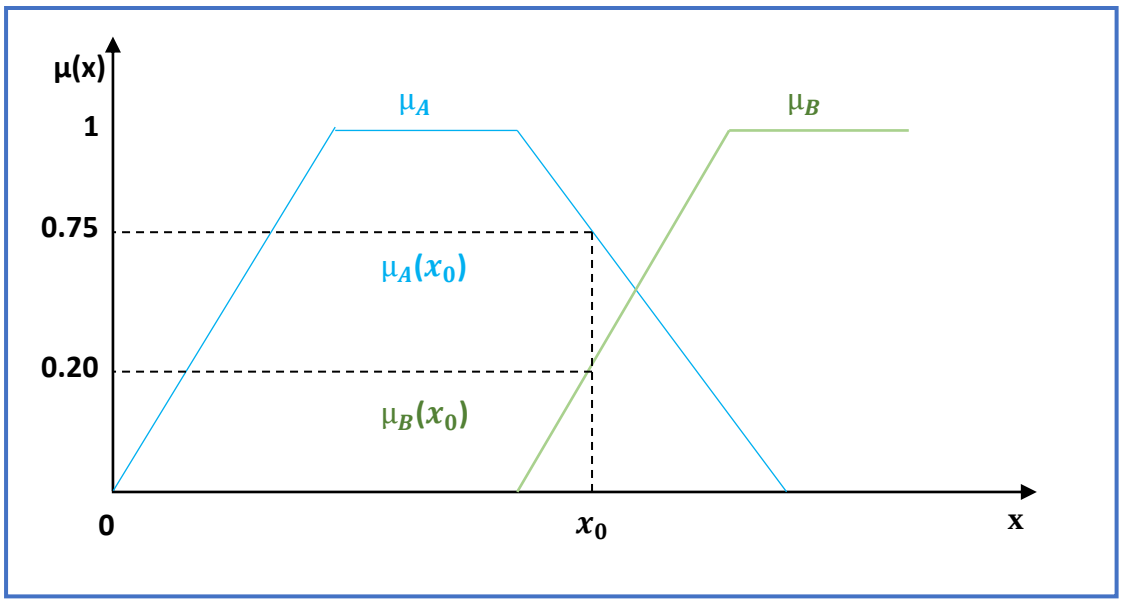

Figure I. 3: Degré d'appartenance de x_0 dans les sous-ensembles A et B.

I.3.1. Types de fonctions d'appartenance

Les fonctions d'appartenance sont définies par un ensemble de paramètres [15]. Les fonctions les plus couramment cités dans la littérature sont :

1. Fonction triangulaire ;
2. Fonction trapézoïdale ;
4. Fonction gaussienne ; et
5. Fonction singleton.

Dans les parties suivantes, les partitionnements des fonctions d'appartenances de type triangulaire, trapézoïdale et singleton sont représentés [15].

I.3.1.1. Partitionnement avec des fonctions triangulaires

La fonction d'appartenance triangulaire (cf. Figure I.4) peut être définie par trois paramètres {a,b,c}, qui divisent l'espace en trois comportements :

$$\mu_{triangulaire}(x)=\begin{cases}\frac{x-a}{b-a} & a \le x \le b \\ \frac{c-x}{c-b} & b \le x \le c \\ 0 & ailleurs\end{cases} \quad \text{(I.2)}$$

Une autre façon de définir la fonction triangulaire consiste à utiliser les opérations maximum et minimum comme suit :

$$\mu_{triangulaire}(x)=max\left(min\left(\frac{x-a}{b-a},\frac{c-x}{c-b}\right),0\right) \quad \textbf{(I.3)}$$

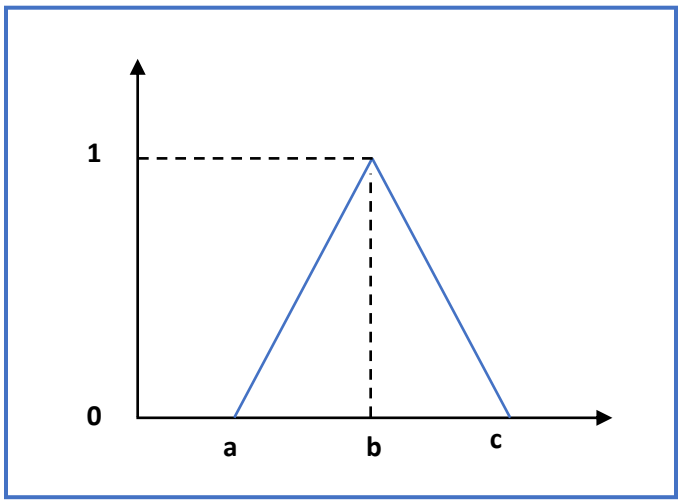

Figure I. 4:Fonction d'appartenance triangulaire

I.3.1.2. Partitionnement avec des fonctions trapézoïdales

La fonction d'appartenance trapézoïdale (cf. Figure I.5) peut être définie par quatre paramètres {a,b,c,d}, qui divisent l'espace en quatre comportements :

$$\mu_{\text{trapézoïdale}}(x)=\begin{cases}\frac{x-a}{b-a} & a \le x \le b \\ 1 & b \le x \le c \\ \frac{c-x}{c-b} & c \le x \le d \\ 0 & ailleurs\end{cases} \quad \text{(I.4)}$$

Alternativement, la fonction trapézoïdale peut être définie avec les opérations maximum et minimum comme suit :

$$\mu_{\text{trapézoïdale}}(x)=max(min\left(\frac{x-a}{b-a},1,\frac{d-x}{d-c}\right),0) \quad \text{(I.5)}$$

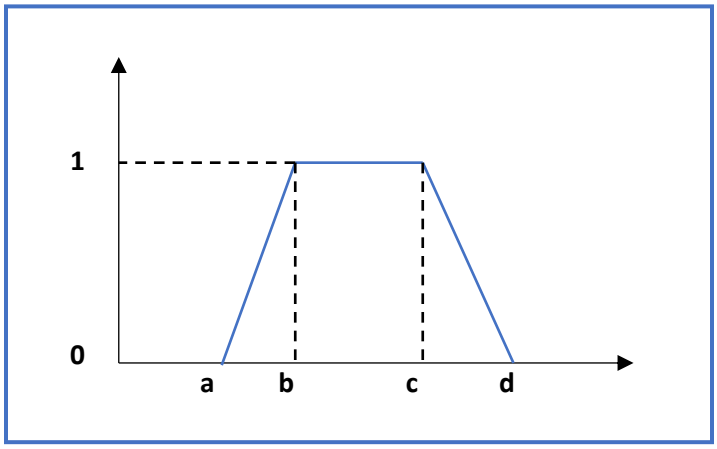

Figure I. 5:Fonction d'appartenance trapézoïdale.

I.3.1.3. Partitionnement avec des fonctions singletons

La fonction d'appartenance singleton (cf. Figure I.6) attribue la valeur d'appartenance 1 à une valeur particulière de x et attribue la valeur 0 pour le reste .

$$\mu_{singleton}(x) = \begin{cases} 1 & x = a \\ 0 & ailleurs \end{cases} \quad \text{(I.6)}$$

Elle est représentée par la fonction d'impulsion comme indiqué.

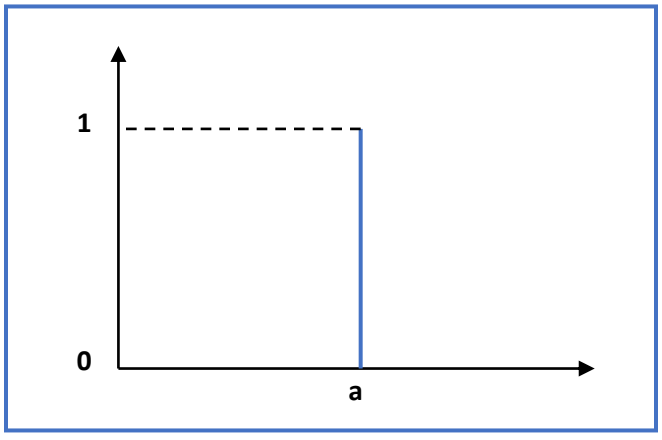

Figure I. 6:Fonction d'appartenance singleton.

I.4. Contrôleurs flous

Les systèmes contrôlés par des contrôleurs flous sont basés sur des règles plutôt que sur des équations mathématiques. Les entrées et sorties d'un système sont converties en une forme appropriée par le bloc de fuzzification. Une fois que toutes les règles ont été définies en fonction de l'application, le processus de contrôle commence par le calcul des conséquences des règles. Enfin, l'ensemble flou est défuzzifié en une action de contrôle précise à l'aide du module de défuzzification. Les blocs constituants l'architecture d'un contrôleur flou sont illustrés dans la Figure suivantes [12] :

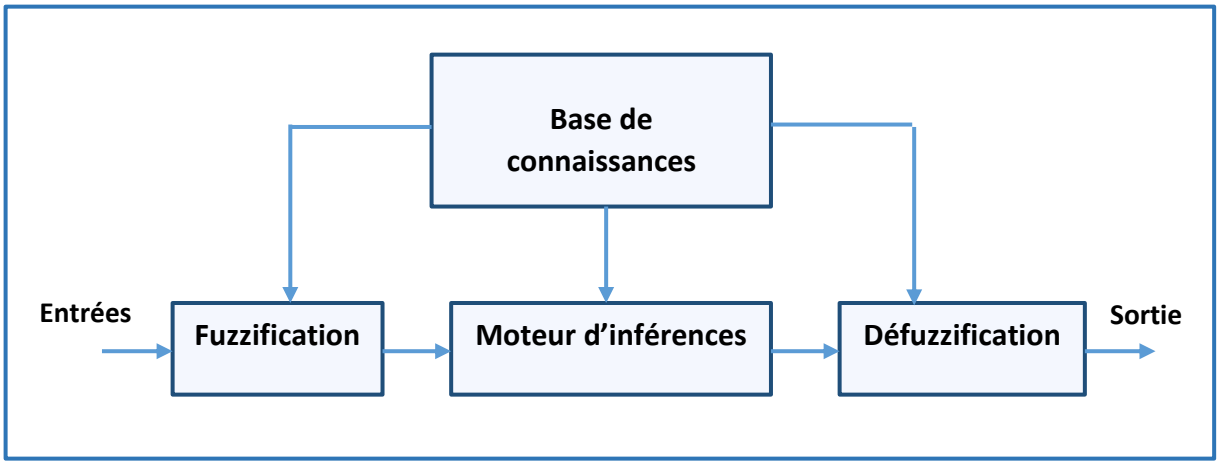

Figure I. 7: Architecture d'un contrôleur flou.

I.4.1. Fuzzification

Dans cette étape l'ensemble classique avec des données d'entrée scalaires est transformée en un ensemble flou correspondant, c'est le processus dans lequel le concepteur définit ou détermine les degrés d'appartenance pour chacune des variables d'entrées.

Chaque entrée réelle est attribuée un terme linguistique dans l'univers du discours. Par exemple, pour le paramètre d'entrée « hauteur », les termes flous peuvent être « grand », « court », « normal », « très grand » et « très court ». Chaque entrée réelle peut être attribuée plusieurs étiquettes. Lorsque le nombre d'étiquettes augmente, la résolution du processus est meilleure. Dans certains cas, l'attribution d'un grand nombre d'étiquettes conduit à un temps de calcul important et rend ainsi le système flou instable. Par conséquent, en général, le nombre d'étiquettes pour un système est limité à un nombre impair dans la plage [11 , 12], de sorte que la surface est équilibrée et symétrique.[12]

La plage de la valeur d'entrée qui correspond à une étiquette spécifique peut être identifiée par une fonction d'appartenance.[12]

Pour illustrer ce processus, supposons que la variable d'entrée ***e***, qui peut être l'angle d'oscillation du pendule inversé, est une variable linguistique et que sa plage autrement dit , l'univers du discours est [-a , a]. Supposons en outre que les sept valeurs linguistiques suivantes (étiquettes floues) soient sélectionnées [12]

NG ---négatif grand NM ---négatif moyen NP ---négatif petit PG --- positif grand
PM --- positif moyen PP --- positif petit EZ --- environ zéro

Cet exemple est illustré dans la Figure I. 8.

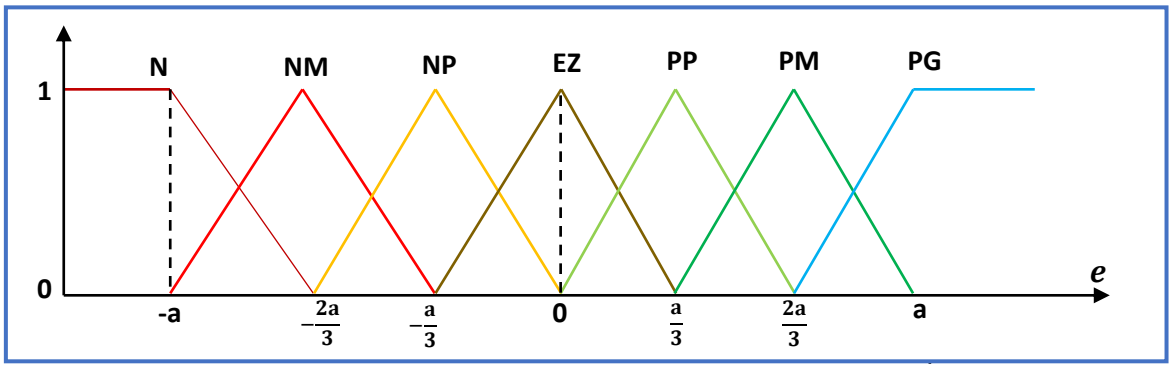

Figure I. 8: Quantification floue possible de la gamme [-a , a] des nombres flous

Les formes n'ont pas besoin d'être symétriques et n'ont pas besoin d'être également réparties sur les plages données.[14]

I.4.2. Règles et opérateurs flous

Pour toute opération de logique floue, la sortie est obtenue à partir de l'entrée réelle par le processus de fuzzification et de défuzzification. Ces processus impliquent l'utilisation de règles, qui forment la base pour obtenir la sortie floue. Une règle floue IF-THEN , également appelée instruction conditionnelle floue ou implication floue, est généralement de la forme : [12]

$$\boldsymbol{IF}\ (\ Antécédent\ )\ \boldsymbol{THEN}\ (\ Conséquent\ )$$

Exemple d'une règle floue IF-THEN :

$$\boldsymbol{IF}\ (x\ is\ A)\ \boldsymbol{THEN}\ (z\ is\ Z) \qquad \text{(I.7)}$$

Où x, z représentent les variables floues et A, Z sont les sous-ensembles dans l'univers du discours.

Les règles floues sont le plus souvent appliquées aux systèmes de contrôle. Les types courants de règles sont les règles floues de Mamdani et les règles floues de Takagi – Sugeno (TS) [12].

La plupart des applications pratiques impliquent une structure de règles composées. Ces dernières ont plus d'une partie antécédente reliées par des connecteurs de conjonction et de disjonction [12].

L'opérateur de conjonction utilise l'intersection impliquant l'opérateur "AND" comme suit :

$$\boldsymbol{IF}\ (\ antécédent1\ \boldsymbol{AND}\ antécédent2\ \boldsymbol{AND}\ \ldots\ldots\boldsymbol{AND}\ antécédentn\ )\ \boldsymbol{THEN}\ (\ consequent\ )$$

De même, l'opérateur de disjonction utilise l'opération d'union impliquant l'opérateur "OR" comme suit :

$$\boldsymbol{IF}\ (antécédent1\ \boldsymbol{OR}\ antécédent2\ \boldsymbol{OR}\ldots\ldots\ldots\boldsymbol{OR}\ antécédentn)\ \boldsymbol{THEN}\ (conséquent)$$

De même, les règles complexes peuvent être décomposées en formes plus simples et reliées à l'aide des connecteurs "AND" ou "OR" comme indiqué dans les règles suivantes. [12]

$$\boldsymbol{IF}\ (X1\ =\ A1\ \boldsymbol{AND}\ X2\ =\ A2)\ \boldsymbol{THEN}\ (Y\ =\ B) \qquad \text{(I.8)}$$

$$\boldsymbol{IF}(\ X1\ =\ A1\ \boldsymbol{AND}\ X2\ =\ A3)\ \boldsymbol{THEN}\ (Y\ =\ B) \qquad \text{(I.9)}$$

I.4.3. Inférences floues

La procédure d'inférence reçoit comme entrées les variables d'entrée floues et un ensemble de règles pour calculer leurs dégrées de vérité, et la sortie de cette étape est une variable floue.

Les méthodes d'inférence floue couramment utilisées sont la méthode d'inférence de Mamdani, la méthode de Takagi-Sugeno (TS) et celle de Tsukamoto. Toutes ces méthodes se ressemblent mais ne diffèrent que par leurs conséquents. La méthode d'inférence floue Mamdani utilise des ensembles flous comme conséquence de la règle, tandis que la méthode TS utilise des fonctions de variables d'entrée et la méthode d'inférence Tsukamoto utilise un ensemble flou avec une fonction d'appartenance monotone comme conséquence de la règle[12] .

I.4.3.1. Méthodes d'inférence floue de Mamdani

le modèle de Mamdani est la méthode couramment utilisée en raison de sa structure simple min-max. Le modèle a été proposé par Mamdani en 1975 [13]. cette inférence floue a d'abord été

introduite comme méthode pour créer un système de contrôle en synthétisant un ensemble de règles de contrôle linguistique obtenues auprès d'opérateurs humains expérimentés.[11]

I.4.3.1.1 Agrégation de règles

Le système basé sur des règles implique plusieurs règles et chacune fournit une sortie ou un résultat. La partie conséquente également connue sous le nom de conclusion est unique pour chaque règle qui a été exécutée en fonction des paramètres d'entrée. Une conclusion globale doit être tirée des conséquences individuelles. Cette méthode d'obtention de la conclusion globale à partir de l'ensemble de règles est appelée agrégation de règles. Les règles floues peuvent être agrégées en utilisant les opérateurs "AND" ou "OR". Le processus d'agrégation des règles à l'aide de l'opérateur "AND" est appelé agrégation conjonctive et le processus d'agrégation des règles à l'aide de l'opérateur "OR" est appelé agrégation disjonctive. [12]

- Agrégation conjonctive :

Conséquent = Conséquent1 **AND** Conséquent2 **AND** ... **AND** Conséquent_n

- Agrégation disjonctive :

Conséquent = Conséquent1 **OR** Conséquent2 **OR** ... **OR** Conséquent_n

En utilisant ces opérateurs, une décision finale est prise sur la sortie de l'ensemble flou.

Les règles floues de Mamdani sont sous la forme :

- Règle 1:

$$\boldsymbol{IF}\ (x\ is\ A3\ \boldsymbol{OR}\ y\ is\ B1)\ \boldsymbol{THEN}\ (z\ is\ C1) \qquad \text{(I.10)}$$

- Règle 2:

$$\boldsymbol{IF}\ (x\ is\ A2\ \boldsymbol{AND}\ y\ is\ B2\ )\boldsymbol{THEN}\ (z\ is\ C2) \qquad \text{(I.11)}$$

- Règle 3:

$$\boldsymbol{IF}\ (x\ is\ A1)\ \boldsymbol{THEN}\ (\ z\ is\ C3) \qquad \text{(I.12)}$$

Exemple :

$$\boldsymbol{IF}\ (x\ IS\ A1\ \boldsymbol{AND}\ y\ IS\ B1\ )\boldsymbol{THEN}\ (\ z\ IS\ C1) \qquad \text{(I.13)}$$

$$\boldsymbol{IF}\ (x\ IS\ A2\ \boldsymbol{AND}\ y\ IS\ B2\ )\boldsymbol{THEN}\ (z\ IS\ C2) \qquad \text{(I.14)}$$

Pour cet exemple, un système d'inférence Mamdani est illustré à la figure I.9.

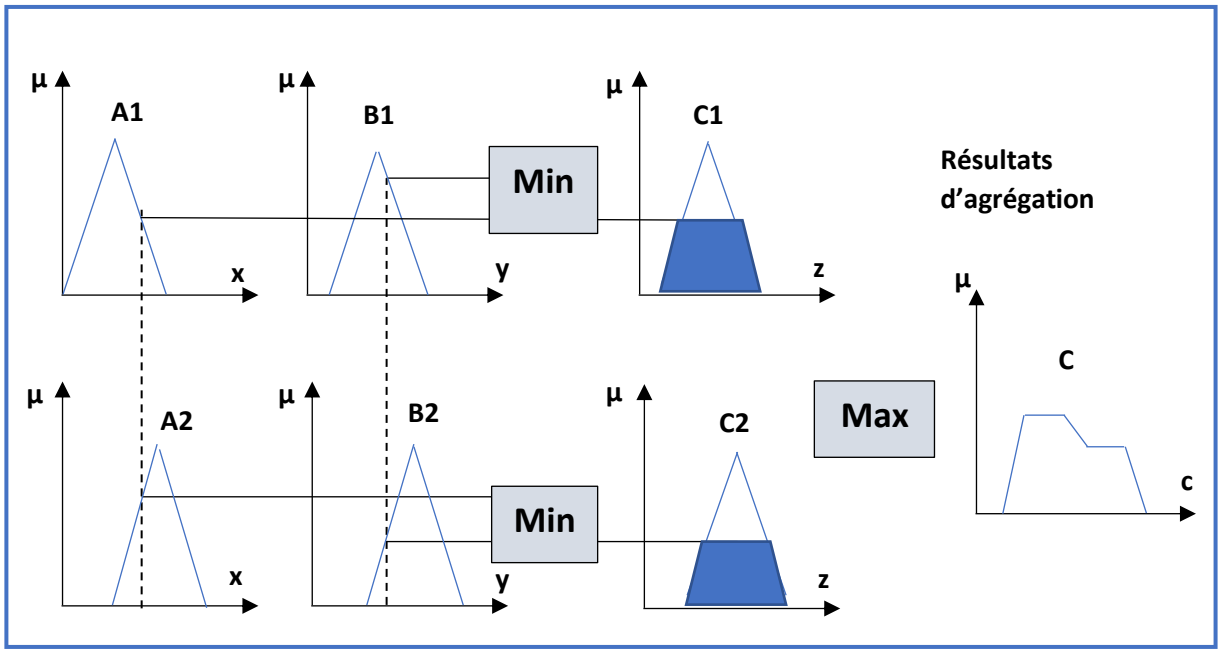

Figure I. 9: Exemple de système d'inférence Mamdani (Min-Max)

I.4.4. Défuzzification

Une donnée floue, est inappropriée pour les applications temps réel et doit être convertie en une valeur nette. Ce processus de conversion du domaine flou au domaine net est la défuzzification. La défuzzification se fait en utilisant de nombreuses méthodes : [15,16]

1. Méthode du centre de gravité.
2. Méthode de la moyenne pondérée.
3. Adhésion moyenne-max.
4. Centre des sommes.
5. Principe d'adhésion maximale.
6. Centre de la plus grande zone.
7. Premier des maxima ou dernier des maximas

Les plus utilisées sont les suivantes :

I.4.4.1. Principe d'appartenance Max

La méthode du principe d'appartenance Max trouve la valeur défuzzifiée dans laquelle la fonction d'appartenance est un maximum [12]. Cette méthode est également appelée la méthode de la hauteur. La valeur défuzzifiée peut être déterminée à partir de l'expression suivante :

$$\mu A(Z*) \geq \mu A(Z)$$

Elle calcule la valeur défuzzifiée à un rythme très rapide, elle est très précise uniquement pour les fonctions d'appartenance à sortie maximale.

La représentation graphique de la défuzzification d'appartenance max illustrée à la figure I.10

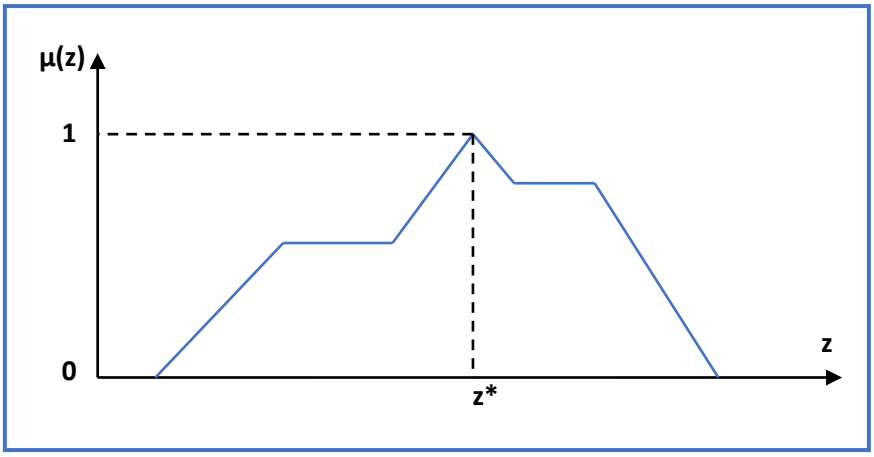

Figure I. 10: La méthode du principe d'appartenance max

I.4.4.2. Méthode du centre de gravité (COG)

La méthode de défuzzification COG a été développée par Sugeno en 1985. Cette méthode est également connue sous le nom de méthode du centre de surface ou du centroïde elle est la méthode la plus utilisée. Elle définit z* où z*= $\int \mu A(z)zdz / \int \mu A(z)zdz$ est la sortie défuzzifiée, μA(z) est la fonction d'appartenance agrégée et z est la variable de sortie . La méthode COG est capable de produire des résultats très précis par contre les calculs sont difficiles pour les fonctions d'appartenance complexes [12]

La représentation graphique de la méthode COG est illustrée à la figure I.11.

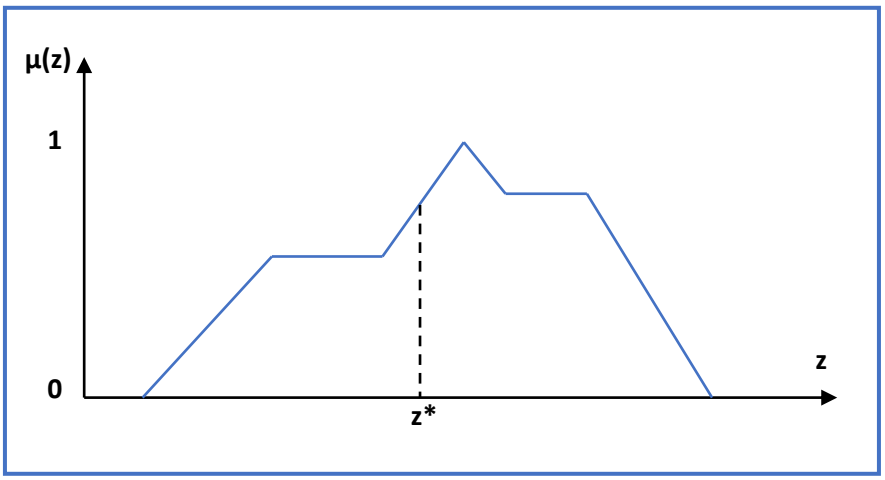

Figure I. 11: Méthode de défuzzification du centroïde

I.4.4.3. Méthode de la moyenne pondérée

Dans la méthode de la moyenne pondérée, la sortie est obtenue par la moyenne pondérée de la sortie de chaque fonction d'appartenance du système.[12] Cette méthode ne peut être appliquée que pour les fonctions d'appartenance de sortie symétriques. Elle définit z* où z* =∑ **μA(z)z**/∑ **μA(z)** est la sortie défuzzifiée, A(z) est la fonction d'appartenance agrégée z et z* est le poids associé à la fonction d'appartenance.

La valeur défuzzifiée obtenue dans cette méthode est très proche de celle obtenue par la méthode COG, comme cette dernière, la méthode de la moyenne pondérée est moins intensive en calcul

La représentation graphique de la méthode de la moyenne pondérée est illustrée à la figure I.12

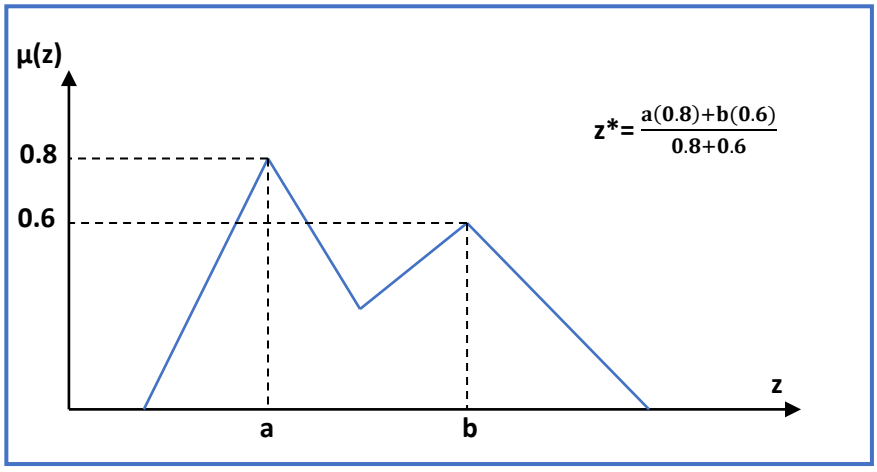

Figure I. 12:Méthode de la moyenne pondérée

I.5. Conclusion

Dans ce chapitre, nous avons illustré la théorie de base du contrôleur de logique floue qui est les ensembles flous et la logique floue qui peut être comprise comme une extension des ensembles classiques. Nous avons également expliqué les différentes méthodes de base de règles, de moteur d'inférence et de défuzzification pour obtenir la sortie numérique du contrôleur, ainsi que des notions de variables linguistiques et de valeurs floues inhérentes à la fonction d'appartenance et ses types pour l'antécédent et le conséquent.

Chapitre 2

FPGA et le langage de description VHDL

II.1. Introduction

Les dispositifs logiques programmables (PLD) sont tous ces éléments de circuit qui permettent aux ingénieurs de simuler, de concevoir et de traiter des informations du monde réel sous forme numérique. Actuellement, on distingue deux grands dispositifs de PLD (Programmable Logic Device): CPLD (Complex Programmable Logic Device) et FPGA (Field Programmable Gate Array). [18]

Les FPGA d'aujourd'hui, conçus principalement par Xilinx et Altera intel, leaders des dispositifs logiques programmables. Les FPGA peuvent permettre au développeur d'exécuter une conception et ainsi d'afficher son état pendant l'exécution[19] en utilisant des langages de description connus sous le nom de HDL (Hardware Description Languages). Les HDL les plus utilisés sont :VHDL, Verilog et Abel.[20]

VHDL (VHSI Hardware Description Language), est l'acronyme de VHSI (Very High Speed Integrated circuit) HDL, cela signifie que VHDL permet d'accélérer le processus de conception..[21]

II.2. Circuits logiques programmables

Le nom de PLD (Programmable Logic Device) est une signification générique établie pour tout système numérique dont le fonctionnement est déterminé par l'utilisateur. Actuellement, ces circuits intégrés sont utilisés pour réaliser toutes sortes de circuits numériques . On peut alors dire, de manière générale, que ces dispositifs permettent de programmer toutes sortes de composants logiques booléens, des portes les plus simples aux plus complexes, en passant par les circuits logiques combinatoires et séquentielles [22].

Les PLD sont généralement classés en dispositifs logiques programmables simples (SPLD) et complexes(HCPLD). En outre, ils sont regroupés comme montré dans la figure II.1 : [23] [24].

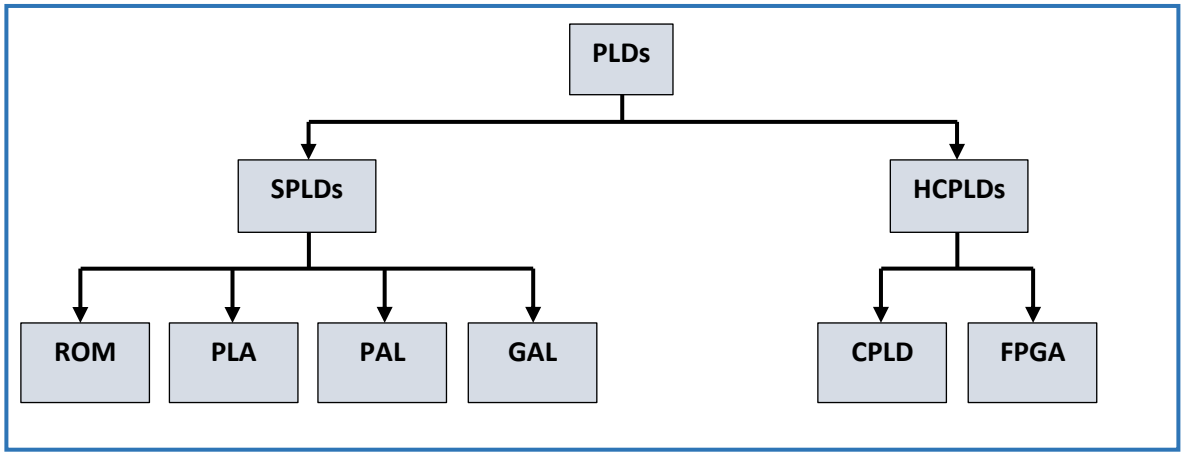

Figure II. 1: Classification des PLDs

II.3. Circuit FPGA

Les FPGA sont des circuits intégrés reprogrammables, basés sur des réseaux de blocs logiques configurables dont les connexions peuvent être modifiées en fonction des besoins du concepteur. [25] Cela veut dire que les FPGA sont programmables, Ce qui leur donne une grande souplesse d'utilisation, puisque leur programmation peut être modifiée pour les améliorer ou corriger des bugs.[26].

La figure II.2 montre la carte SPARTAN-3 de XILINX

.

Figure II. 2:Carte FPGA SPARTAN-3 de XILINX

II.3.1. Architecture interne des FPGA

L'architecture d'un FPGA varie d'un fabricant à l'autre et aussi selon la technologie utilisée. De manière générique et comme on peut le voir sur la Figure II.3, un FPGA est composé de blocs logiques configurables (CLB), entourée de cellules IOB (Input/Output Blocks) , interconnectées entre eux via des canaux de communication verticaux et horizontaux [27].

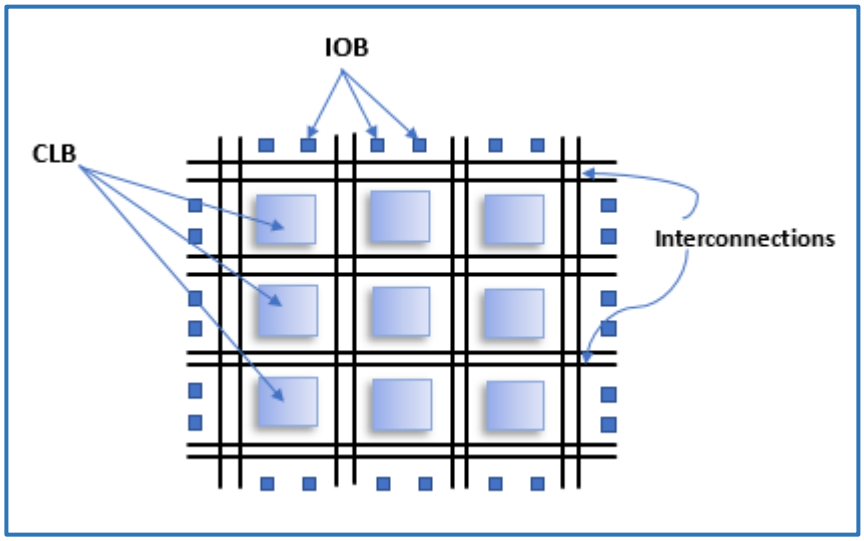

Figure II. 3: Architecture d'une carte FPGA

II.3.1.1. Les blocs logiques configurables (CLB)

Ce sont des blocs de base qui sont utilisés dans la réalisation d'un circuit numérique. Tous les FPGA ont des blocs logiques configurables. C'est le cœur du FPGA, qui permet de mettre en œuvre les différentes fonctions logiques. La figure II.4 montre un CLB [27].

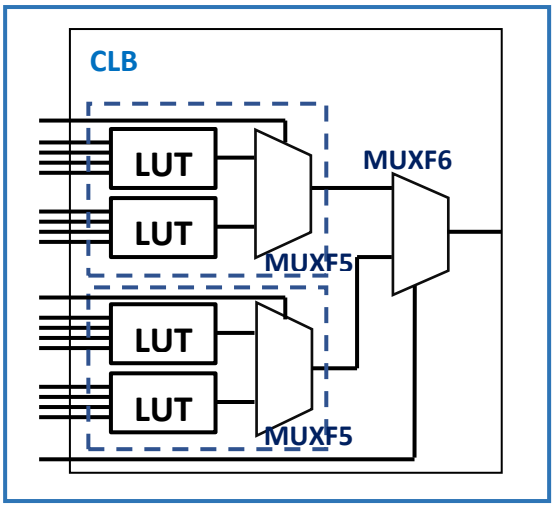

Figure II. 4:Bloc logique configurable

II.3.1.2. Les blocs d'entrées/sorties (IOB)

Les IOB fournissent l'interface entre les broches du composant FPGA et la logique interne. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels .Ceux-ci sont divisés en banques configurées pour interconnecter la logique de différentes normes électriques de manière indépendante. [27]

La Figure II.5 montre la configuration des blocs d'entrée et de sortie

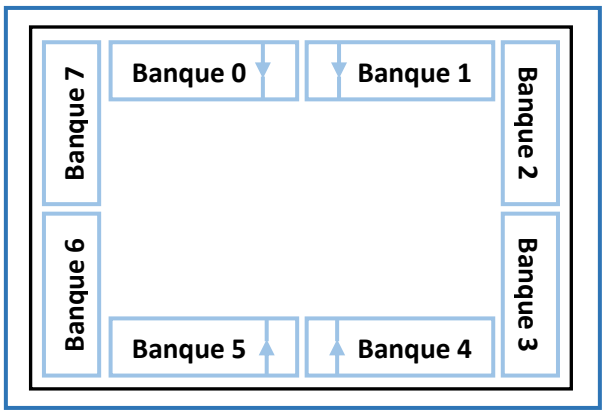

Figure II. 5: Configuration des blocs I/O regroupées en banques

II.3.1.3. Les interconnections

Il s'agit d'une structure d'interconnexion polyvalente et multiniveaux entre les composants du FPGA.

Afin de mettre en œuvre les éléments logiques présentés dans les sections précédentes, ils doivent non seulement être correctement configurés, mais également connectés les uns aux autres. La structure d'interconnexion interne d'un FPGA consiste en un ensemble de fils reliés par des éléments programmables.[27]

II.4. A propos du VHDL

Une conception d'un système numérique peut être capturée soit sous forme de schéma, soit en utilisant des langages de description spéciaux [20] HDL (Hardware Description Languages), parmi ces langages le VHDL.

Le langage de description des circuit électroniques numériques VHDL (VHSIC Hardware Description Language) est une norme définie par IEEE 1076 par laquelle le comportement d'un circuit électronique peut être décrit. Il autorise plusieurs méthodologies de conception (comportemental, flot de données, structurel) tout en étant d'un très haut niveau d'abstraction en électronique. Utiliser ce langage évite de dériver des tables de vérité des spécifications, que l'on simplifie alors en équations booléennes à implémenter. Il permet d'expliciter la fonction à implémenter sans se soucier de ces détails, bien que l'implémentation en pratique se fera à l'aide de portes logiques (le compilateur se chargera de passer du code aux portes).

Un projet VHDL peut contenir de nombreux fichiers, où le code VHDL se trouve généralement dans des fichiers avec l'extension *.vhd[28].

II.5. Structure de code VHDL simple

Un système numérique est décrit par ses entrées et ses sorties et la relation entre elles.

Le VHDL, d'une part, décrit l'aspect extérieur du circuit : entrées et sorties ; et d'autre part relie les entrées aux sorties. La description des ports d'entrées/sorties est dans la partie entité, et la description du comportement du circuit est dans la partie architecture, chaque architecture doit être associée à une entité.

De plus, bien que ce ne soit pas strictement nécessaire, les bibliothèques et les packages peuvent également être définis, ce qui indiquera quels types de ports et d'opérateurs qu'on peut utiliser. La définition des bibliothèques et des packages doit toujours apparaître avant la définition de l'entité[28].

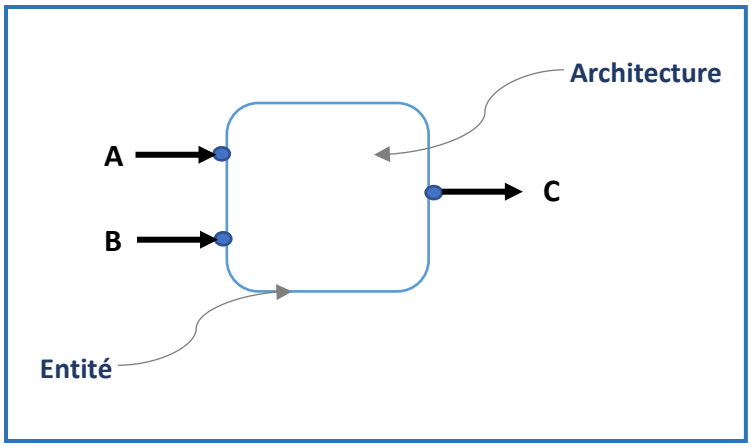

Figure II. 6: Entité et architecture

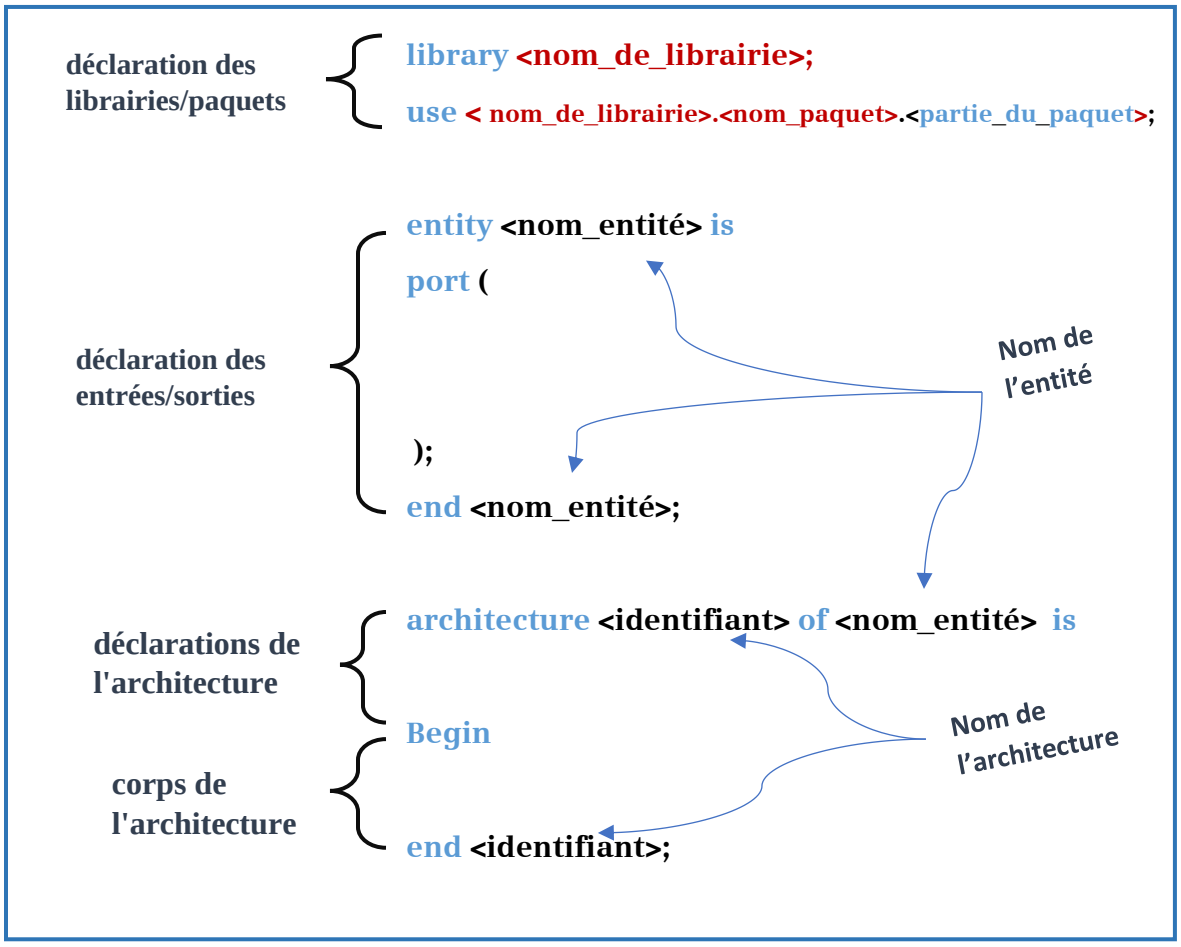

Figure II. 7: Structure d'un code VHDL

II.5.1. Déclaration de Librairie (Library)

Une librairie est une bibliothèque où sont stockées des entités de conception précompilées. Elles sont déclarées en en-tête d'une description VHDL par l'instruction 'library'. Les descriptions sont compilées dans la librairie 'work' par défaut à moins qu'une autre librairie soit couramment activée dans l'outil de compilation[29].

II.5.1.1. Paquets VHDL fondamentaux (Package)

Apres avoir déclarer les librairie, l'instruction 'use' est utilisée afin d'accéder aux objets qui peuvent être utilisés dans la suite de la description VHDL[29].

Le tableau ci-dessus montre les paquetages développés et normalisés par l'IEEE(Institut of Electrical and Electronics Engineers)

Tableau II. 1: Paquets associés à la librairie IEEE

Paquets	**Leurs fonction**
std_logic_1164	Pour définir les types std_logic .
std_logic_arith	Pour utiliser les opérateurs arithmétiques '+','-','*','/' pour les types std_logic.
numeric_std	Utilisation des valeurs décimales pour le type std_logic.
std_logic_signed	Les type std_logic avec valeur signée.
std_logic_unsigned	Le type std_logic avec valeur non signée.
math_complex	fournit un calcul numérique avec des nombres complexes.
math_real	fournit un calcul numérique avec des nombres réels.
std_logic_textio	Utilisation des valeurs ASCII pour le type std_logic.
numeric_bit	Utilisation des valeurs décimales pour le type std_logic.

II.5.2. Déclaration de l'entité (Entity)

Une déclaration d'entité définie les ports d'interface entre une entité de conception et son environnement. Une déclaration d'entité peut être partagée par plusieurs entités de conception possédante une architecture différente. Une entité est l'abstraction d'un circuit, qu'il provienne d'un système électronique complexe ou d'une simple porte logique elle ne décrit que la forme extérieure

du circuit, elle définit le nom et nombre de ports, types de données d'entrée et de sortie. Une entité est analogue à un symbole schématique sur les schémas électroniques, qui décrit les connexions de l'appareil au reste de la conception.[28]

La figure II.8 montre la syntaxe de déclaration d'entité

```
Entity nombre is
Generic (cte1: type := valeur1; cte2: type:= valeur2;…);
port (entee1, entree2, … : in type;
      sortie1, sortie2, …: out type;
       nom_port : mod_port type);
 end nombre;
```

Figure II. 8 : Déclaration d'entité

Les ports peuvent être 'IN', 'OUT', 'INOUT' ou 'BUFFER'. Les ports d'entrée ne peuvent être qu'en lecture et leur valeur ne peut pas être modifiée dans la description du comportement du circuit (architecture), les ports de sortie ne peuvent qu'être écrits mais ne prennent jamais de décisions en fonction de leur valeur. S'il est strictement nécessaire d'écrire sur un port en même temps que sa valeur doit être prise en compte, le type serait 'INOUT' ou 'BUFFER'.[28]

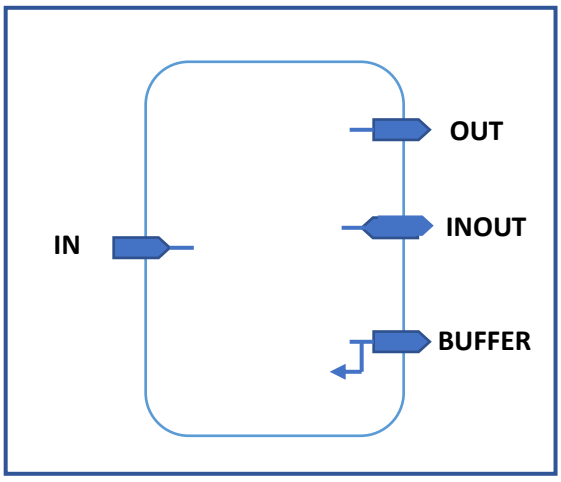

Figure II. 9: Modes de ports d'une entité VHDL

- **IN** L'entité lit un signal d'entrée fourni par l'extérieur
- **OUT** L'entité fournit un signal de sortie, mais ne peut pas relire ce signal.
- **INOUT** Le signal est bidirectionnel
- **BUFFER** L'architecture de l'entité fabrique un signal utilisable en sortie, qui peut aussi être relu par l'entité comme un signal interne.

Des valeurs génériques peuvent être définies dans l'entité (Generic) qui servira à déclarer les propriétés et les constantes du circuit, quelle que soit l'architecture

II.5.3. Déclaration d'architecture (Architecture)

Le corps d'une architecture définit le corps d'une entité de conception. Il spécifie les relations entre les entrées et les sorties. Cette spécification peut être exprimée sous forme comportementale, structurelle, flot de données, les trois formes peuvent coexister à l'intérieur d'un même corps d'architecture. La partie déclarative déclare des objets(Signaux internes, fonctions, procédures, constantes...) qui seront visibles et utilisables par l'entité de conception. Toutes les instructions qui décrivent le corps de l'architecture sont des instructions concurrentes qui s'exécutent de façon asynchrone les unes par rapport aux autres [28].

```
architecture  identifiant of nom_entité is
 -- déclaration de l'architecture:
 -- types
 -- signaux
 -- components
Begin
 -- code de la description
 -- instructions concurrentes
 -- équations booléennes
 -- components
       -- process (liste de sensibilité)
       Begin
        -- code de la description
        end process;
 end identifiant;
```

Figure II. 10:Déclaration d'architecture

II.6. Fonctionnement concurrent et séquentiel

II.6.1. Fonctionnement concurrent

Dans un circuit logique, toutes les portes fonctionnent simultanément. On dit alors que les portes fonctionnent de manière concurrente, c'est à dire que l'ensemble des opérations se déroulent en parallèle[30]. Prenant comme Example la fonction logique de la figure ci-dessous :

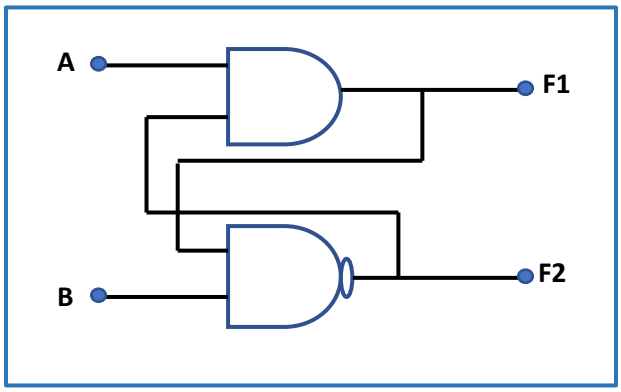

Figure II. 11:Fonction logique simple

Les portes logique de cette fonction s'exécute en parallèle voici l'architecture qui décrit cela

```
architecture description of fonction_simple is
 begin
 F1 <= (A AND F2);
F1 <= (B NAND F1);
end description;
```

Figure II. 12:Description d'une fonction logique

Dans un circuit, toutes les portes fonctionnent simultanément. Le langage VHDL disposent d'instructions concurrentes qui vont nous permettre de décrire le comportement d'un circuit. Une description VHDL est constitué d'un ensemble de processus concurrents. L'ordre dans lequel ces processus apparaissent dans la description n'est pas important.[28]

Le langage VHDL disposent de 5 instructions concurrentes, soit:

• l'affectation simple ... <= ....

La syntaxe générique d'une affectation simple est la suivante:

```
Signal_1 <= Signal_2 fonction_logique Signal_3;
```

• l'affectation conditionnelle when ... else

La syntaxe générique d'une affectation conditionnelle est la suivante:

```
Signal_S <= Signal_1 when Condition_1
else Signal_2 when Condition_2
else .... .... Signal_X when Condition_X
else Signal_Y;
```

• l'affectation sélective with ... select

La syntaxe générique d'une affectation d'un signal sélectionné est la suivante:

```
with Vecteur_De_Commande select
Signal_Sortie <=Valeur_1 when Etat_Logique_1,
                Valeur_2 when Etat_Logique_2,
       .... ....Valeur_N when others;
```

• l'instanciation de composant

La syntaxe générique d'une instanciation de composant est la suivante:

```
[Label:] Nom_Composant port map(Nom_port => Nom_Signal, .... );
```

Lors de l'utilisation d'un composant, il est nécessaire de déclarer celui-ci et d'indiquer, la paire entité-architecture qui doit être utilisée. Cela s'appelle la configuration. La syntaxe générique de la déclaration d'un composant avec la configuration est la suivante: [28]

```
--Déclaration du composant component
 Nom_Composant is
port (Nom_Entree : in std_logic; ....
         Nom_Sortie : out std_logic);
 end component;
```

- **Génération conditionnelle**

Les syntaxes génériques d'une génération conditionnelle sont les suivantes:

```
-- structure itérative
[label] : for variable in valeur_debut to valeur_fin generate
instructions_concurrentes ;
end generate [label] ;
```

```
-- structure conditionnelle
 [label] : if condition generate
instructions_concurrentes ;
end generate [label] ;
```

II.6.2. Fonctionnement séquentiel

Comme le fonctionnement concurrent n'est pas toujours le plus confortable pour la description des systèmes , en particulier les systèmes très complexes, le VHDL permet également un fonctionnement algorithmique qui combine des instructions exécutées en série avec d'autres en fonctionnement concurrent.[29]

- **Le process**

est une structure particulière du VHDL qui est principalement réservée pour contenir des instructions qui ne doivent pas nécessairement avoir leur valeur définie pour toutes les entrées. Cela oblige la structure du processus à stocker ses valeurs de signal et peut (pas toujours) conduire à des sous-circuits séquentiels. De plus, en simulation seules les instructions internes à cette structure sont exécutées lorsque l'un des signaux de sa liste de sensibilité change de valeur.[29]

La syntaxe générique d'un process est la suivante:

```
[Label:] process (Liste_De_Sensibilité)
 --zone de Déclarations
 Begin
 --Zone pour instructions Séquentielles
 end process [Label];
```

- **Instruction case…is…when**

La syntaxe générique de l'**Instruction case…is…when** est la suivante:

```
--Instruction case….is….when
Case  Etat_Logique is
When condition_1 => instruction_1 ;
When condition_2 => instruction_2 ;
When others => instruction_N ;
End case ;
```

- **Instructions des boucles(Loop)**

Les syntaxes génériques des instructions des boucles sont les suivantes:

```
--boucle for
[label] : for variable in valeur_debut to valeur_fin
loop
instructions_repetitives ;
end loop [label] ;
```

```
--boucle while
[label] : while conditions loop
instructions_repetitives ;
end loop [label] ;
End case ;
```

```
--boucle générale
[label] : loop  --boucle infinite
instructions_repetitives ;
end loop [label] ;
End case ;
```

II.8. Conclusion :

Au début de ce chapitre on a discuté sur les PLDs et les circuits FPGAs ,on a illustrer l'architecture de cette dernière, par la suite on a présenté le langage le VHDL qui est l'un des langages utilisés pour programmer les circuits FPGAs, on a expliqué la structure d'un code écrit en ce langage puis on a déterminer les deux modes de fonctionnement concurrent et séquentielle ainsi que les instructions utilisées.

Chapitre 3

Conception d'un contrôleur flou en VHDL

III.1. Introduction

Les premières implémentations des systèmes d'inférences flous ont été réalisées par des logiciels fonctionnant sur des processeurs à implémentation logicielle. Cependant, en raison des progrès continus des technologies de fabrication de circuits intégrés, de nombreuses alternatives ont été proposées ces dernières années pour la mise en œuvre matérielle des systèmes flous en utilisant des FPGA ou des ASIC [31].

Au long de ce chapitre un contrôleur flou de type Mamdani sera décrit en langage VHDL. Les paramètres d'un contrôleur flou d'une application d'un système de freinage automatique seront pris comme exemple, où les détails ont été donnés dans l'article [32]. La figure III.1 illustre un schéma synoptique d'un contrôleur flou avec deux entrées et une sortie pour un système de freinage.

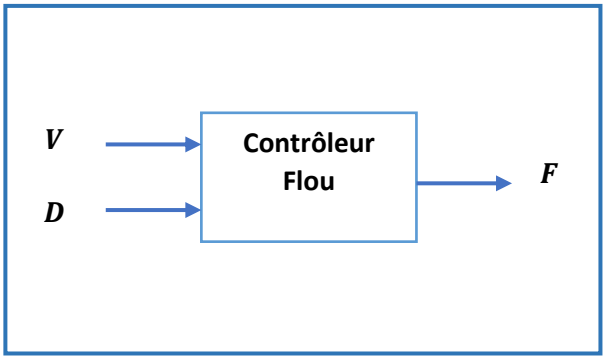

Figure III. 1 : contrôleur flou pour un système de freinage

Où, *V* est la vitesse, *D* est la distance, et *F* est l'angle de freinage.

Dans ce système a logique floue, il existe deux facteurs qui influencent la valeur de freinage, le premier est la vitesse, si la voiture roule à grande vitesse, il faudra alors un freinage important et lorsque la voiture roule à basse vitesse, le freinage généré sur le système sera faible. Le deuxième facteur est la distance de la voiture à l'obstacle. Plus la voiture est proche de l'obstacle, plus il faut un gros freinage pour éviter la survenue d'une collision, en gros deux variables d'entrée du contrôleur flou seront utilisées, la distance D et la vitesse V , la variable de sortie du système est l'angle de freinage F

III.2. La fuzzification

La description VHDL de la partie fuzzification est divisée en deux étapes principales :

1. Partitionnement des fonctions d'appartenance sur l'univers de discours, et
2. Calcul des degrés d'appartenance

Les valeurs des fonctions d'appartenance ont été mises à l'échelle sous la forme d'une valeur hexadécimale représentée avec le signe X.

L'axe des valeurs du degré d'appartenance des deux entrées et la sortie est donné dans l'intervalle allant de X00 à XFF.

III.2.1 Partitionnement des fonctions d'appartenance

III.2.1.1 Les entrées

Les fonctions d'appartenance établies sont représentées par des fonctions trapézoïdales et triangulaires. Ces fonctions conviennent à ce projet puisqu'elles simplifient la conception en code VHDL, en plus d'obtenir un résultat efficace adapté à nos variables de contrôle.

- **La vitesse V**

Les sous-ensembles flous de la variable linguistique V peuvent être définies à l'aide des termes linguistiques (lente, modérée, rapide). La représentation graphique des fonctions d'appartenance de V est illustrée à la figure III.2. Les valeurs de partitionnement des fonctions d'appartenance sont représentées dans le tableau 1 en format décimal et en format hexadécimal.

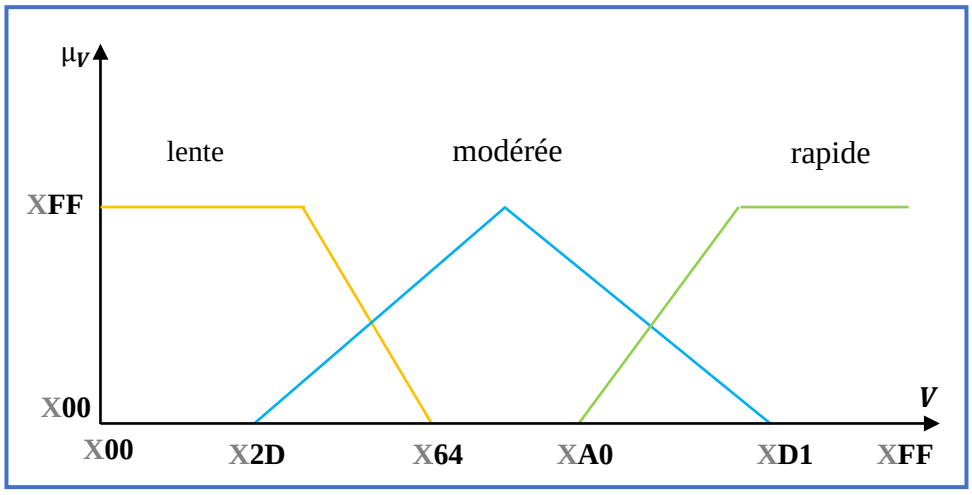

Figure III. 2 : Fonctions d'appartenance de l'entrée V

Tableau III. 1 : Intervalles des sous-ensembles flous de la variable V.

Sous-ensembles flous	Partitionnement en decimal(cm/s)	Partitionnement en hexadécimal
lente	De 0 à 100	De X00 à X64
moyenne	De 45 à 209	De X2D à XD1
rapide	De 160 à 255	De XA0 à XFF

- **La distance *D***

La deuxième variable d'entrée du contrôleur flou du système de freinage est la distance D. Le taux de changement de distance prend trois termes linguistiques (proche, moyenne et loin). La représentation graphique de leurs fonctions d'appartenance est illustrée à la figure III. 3.

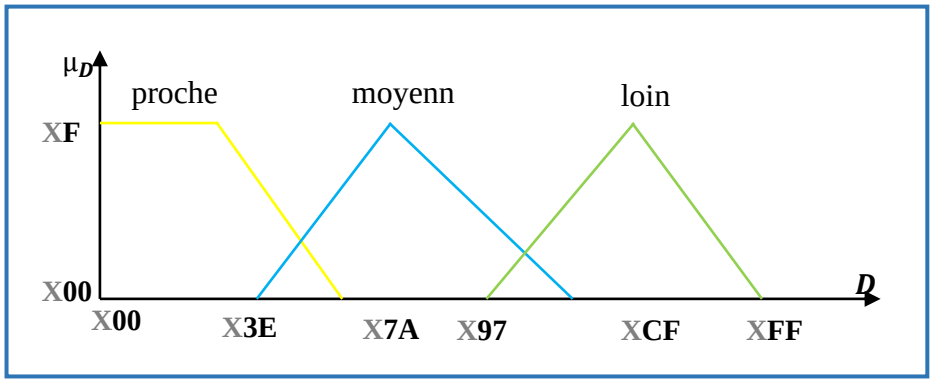

Figure III. 3 : Fonctions d'appartenance de l'entrée D

III.2.1.2. La sortie

Les fonctions d'appartenance de sortie du contrôleur sont des singletons

- **Angle de freinage *F***

Cinq termes linguistiques différents (TF: très faible, F :faible, M :moyen, E :élevé et TE :très élevé) sont utilisés pour décrire la variable de freinage. La figure III. 4 montre la représentation graphique des fonctions singletons.

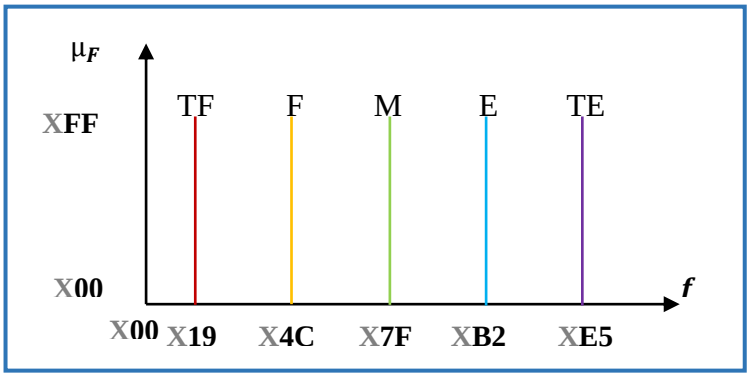

Figure III. 4 : Fonctions d'appartenance de la sortie F

III.2.1.3 Description en VHDL des variables du contrôleur floues

La Figure III.5 montre les paramètres qui ont été prises en considération en VHDL. La fonction d'appartenance trapézoïdale est définie par quatre points et deux pentes (Figure III.5(a)), et la fonction d'appartenance triangulaire est définie par trois points et deux pentes (Figure III.5(b)).

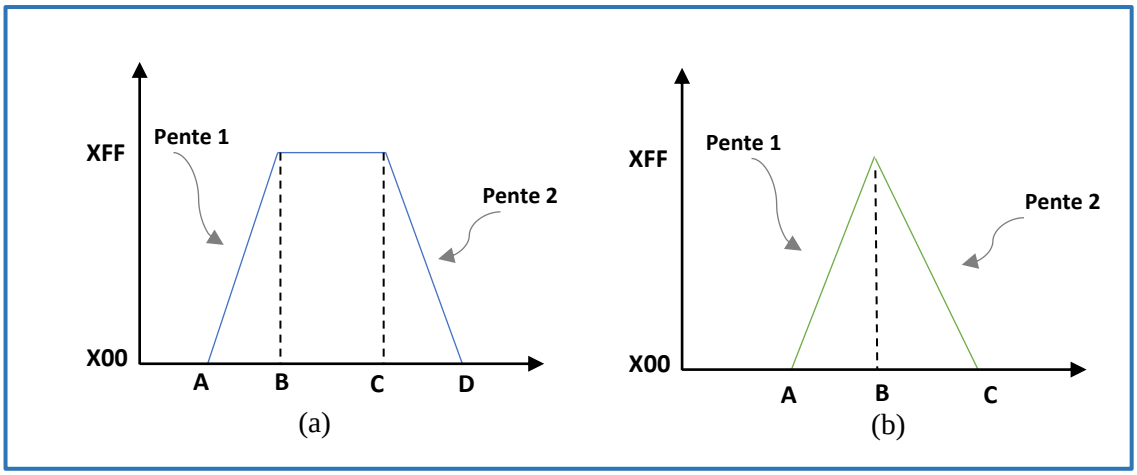

Figure III. 5 : (a) : Fonction d'appartenance trapézoïdale, et (b) : Fonction d'appartenance triangulaire.

Les fonctions d'appartenances des deux entrées et de la sortie sont toutes décrites dans un seul paquet(Package) nommée «membership_functions » comme suit :

- Description de l'entrée *V*
  - Description des fonctions trapézoïdales

```
type V_type is (lente,moderée, rapide, none);
---Fonctions trapezoidales
type v_mf_trap is record term: v_type;
point1:std_logic_vector(7 downto 0); pente1:std_logic_vector(7 downto 0);
point2:std_logic_vector(7 downto 0); pente2:std_logic_vector(7 downto 0);
point3:std_logic_vector(7 downto 0); point4:std_logic_vector(7 downto 0);
 end record;
type v_mfs is array(natural range <>)of v_mf_trap;
constant v_trapezoidal: v_mfs:=(
(term => lente, point1 => x"00", point2 => x"00", point3 => x"32",point4 => x"64",
pente1 => x"00", pente2 => x"05"),
(term => moderée, point1 => x"00", pente1=> x"00", point2 => x"00", pente2 => x"00",
point3 => x"00", point4 => x"00"),
(term => rapide, point1 => x"A0", point2 => x"BF", point3 => x"FF", point4=> x"FF",
pente1 => x"08", pente2 => x"00"),
(term => none , point1 => x"FF", point2 => x"FF", point3 => x"FF", point4=> x"FF",
pente1=> x"FF", pente2 => x"FF"));
```

- Description des fonctions triangulaires

```
--fonctions triangulaires
type v_mf_tri is record term: v_type;
point1: std_logic_vector(7 downto 0); pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0); pente2: std_logic_vector(7 downto 0);
point3: std_logic_vector(7 downto 0);
end record;
type v_mfs_1 is array(natural range <>)of v_mf_tri;
constant v_triangular: v_mfs_1 :=(
(term => lente, point1 => x"00", point2 => x"00", point3 => x"00", pente1 => x"00",
pente2 => x"00"),
(term => moderée, point1 => x"2D", point2 => x"7F", point3 => x"D1", pente1 => x"03",
pente2 => x"03"),
(term => rapide, point1 => x"00", point2 => x"00", point3 => x"00", pente1 => x"00",
pente2 => x"00"),
(term => none, point1 => x"FF", point2 => x"FF", point3 => x"FF", pente1 => x"FF", pente2
=> x"FF"));
```

- Description de l'entrée *D*

  - Description des fonctions trapézoïdales

```
type D_type is (proche, moyenne,loin,none);
--fonctions trapezoidales
type D_mf_trap is record
term: D_type;
point1: std_logic_vector(7 downto 0); pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0); pente2: std_logic_vector(7 downto 0);
point3: std_logic_vector(7 downto 0); point4: std_logic_vector(7 downto 0);
 end record;
type D_mfs is array(natural range <>)of D_mf_trap;
constant D_trapezoidal: D_mfs:=
((term => proche, point1 => x"00",point2 => x"00",point3 => x"2F",point4=> x"4C", pente1
=> x"00", pente2 => x"08"),
(term => moyenne, point1 => x"00", pente1 => x"00", point2 => x"00", pente2 =>
x"00",point3 => x"00",point4=> x"00"),
(term => loin, point1 => x"00",point2 => x"00",point3 => x"00",point4=> x"00", pente1 =>
x"00", pente2 => x"00"),
(term => none, point1 => x"FF",point2 => x"FF",point3 => x"FF",point4=> x"FF", pente1 =>
x"FF", pente2 => x"FF"));
```

- Description des fonctions trapézoïdales

```
----fonctions triangulaires
type D_mf_tri is record
term: D_type;
point1: std_logic_vector(7 downto 0); pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0); pente2: std_logic_vector(7 downto 0);
point3: std_logic_vector(7 downto 0);
end record;
type D_mfs_1 is array(natural range <>)of D_mf_tri;
constant D_triangular: D_mfs_1 :=
((term => proche, point1 => x"00",point2 => x"00",point3 => x"00", pente1 => x"00", pente2
=> x"00"),
(term => moyenne, point1 => x"3E",point2 => x"7A",point3 => x"B7", pente1 => x"04", pente2
=> x"04"),
(term => loin, point1 => x"97",point2 => x"CF",point3 => x"FF", pente1 => x"04", pente2 =>
x"05"),
(term => none, point1 => x"FF",point2 => x"FF",point3 => x"FF", pente1 => x"FF", pente2 =>
x"FF"));
```

- **Description de la sortie *F***

  - Description des fonctions singletons

```
constant très_faible : std_logic_vector := x"19";
constant faible : std_logic_vector := x"4C";
constant moyen : std_logic_vector := x"7F";
constant  élevé : std_logic_vector := x"B2";
constant très_élevé : std_logic_vector := x"00E5";
type singleton is array (0 to 4) of std_logic_vector(7 downto 0);
signal f: singleton :=(très_faible,faible,moyen,élevé,très_élevé);
```

III.2.2 Calcul des degrés d'appartenance

Après la déclaration des fonctions d'appartenance, nous devons calculer leur degré d'appartenance ou chaque valeur d'entrée est appliquée à sa fonction d'appartenance pour trouver la valeur d'entrée floue.

Les sous-ensembles représentée par des fonctions d'appartenance trapézoïdales sont divisées en 4 parties :

- Avant le point A et après le point D

$$\mu = 0$$

- Entre le point A et le point B
  - La pente est ascendante : $Pente1 = XFF/(B - A)$

$$\mu = (valeur\ d'entrée - A)\ (pente1)$$

- Entre le point B et le point C

$$\mu = 1$$

- Entre le point C et le point D
  - La pente est descendante : $Pente2 = XFF/(D - C)$

$$\mu = (B - valeur\ d'entrée\ )\ (pente2)$$

Les sous-ensembles représentée par des fonctions d'appartenance triangulaires sont divisées en 3 parties :

- Avant le point A et après le point C

$$\mu = 0$$

- Entre le point A et le point B
  - La pente est ascendante : $Pente1 = XFF/(B - A)$

$$\mu = (valeur\ d'entrée - A)\ (pente1)$$

- Entre le point B et le point C
  - La pente est descendante : $Pente2 = XFF/(D - C)$

$$\mu = (B - valeur\ d'entrée\ )\ (pente2)$$

Cet exemple a deux entrées, ayant trois fonctions d'appartenance chacune, il y a au total six degrés d'appartenance à calculer. Puisqu'une valeur d'entrée spécifique ne croise qu'a deux fonctions d'appartenance, la plupart des dégréés seront donc nuls. Le pseudo-code suivant illustre le déroulement de ce processus :

- **Degré d'appartenance de l'entrée *V***

```
For i in 0 to 2 loop
----------------------trapezoidal function-----------------
if(v_input<v_trapezoidal(i).point1) then
   uv(i)<=(others=>'0');
elsif (v_input<v_trapezoidal(i).point2) then
   uv(i)<= (v_input - v_trapezoidal(i).point1)*(v_trapezoidal(i).pente1);
elsif (v_input<v_trapezoidal(i).point3) then
   uv(i)<=x"00FF";
elsif (v_input<v_trapezoidal(i).point4) then
   uv(i) <=(v_trapezoidal(i).point4-v_input)*(v_trapezoidal(i).pente2);
----------------------triangular function----------------
elsif (v_input<v_triangular(i).point1) then
   uv(i)<=(others=>'0');
elsif (v_input<v_triangular(i).point2) then
   uv(i)<= (v_input - v_triangular(i).point1)*(v_triangular(i).pente1);
elsif (v_input<v_triangular(i).point3) then
   uv(i) <=(v_triangular(i).point3-v_input)*(v_triangular(i).pente2);
 else
  uv(i)<=(others=>'0');
end if;
end loop;
```

- **Degré d'appartenance de l'entrée *D***

```
For i in 0 to 2 loop
-------------------------trapezoidal function-----------------------
if(d_input<d_trapezoidal(i).point1) then
   ud(i)<=(others=>'0');
elsif (d_input<d_trapezoidal(i).point2) then
   ud(i)<= (d_input - d_trapezoidal(i).point1)*(d_trapezoidal(i).pente1);
elsif (d_input<d_trapezoidal(i).point3) then
   ud(i)<=x"00FF";
elsif (d_input<d_trapezoidal(i).point4) then
   ud(i) <=(d_trapezoidal(i).point4-d_input)*(d_trapezoidal(i).pente2);
-------------------------triangular function----------------------
elsif (d_input<d_triangular(i).point1) then
   ud(i)<=(others=>'0');
elsif (d_input<d_triangular(i).point2) then
   ud(i)<= (d_input - d_triangular(i).point1)*(d_triangular(i).pente1);
elsif (d_input<d_triangular(i).point3) then
   ud(i) <=(d_triangular(i).point3-d_input)*(d_triangular(i).pente2);
 else
   ud(i)<=(others=>'0');
end if;
end loop;
```

III.3 Règles d'inférence

Une fois que les degrés d'appartenance ont été déterminés dans l'étape de fuzzification, l'étape suivante consiste à utiliser des règles linguistique pour déterminer quel décision doit être prise.

Notre exemple contient deux entrée avec trois termes linguistique chacune ce qui fait au totale neuf règles. Le tableau suivant résume ces règles

Tableau III. 2 : Table de règles

Output-F		V		
		Lente	Modérée	Rapide
D	Proche	Moyen	Elevé	T.elevé
	Moyenne	Faible	Moyen	T. élevé
	Loin	T.faible	Faible	Elevé

Maintenant que la table de règles est formée la partie suivante consiste à inférer sur les règles tout en utilisant la méthode de Mamdani

Les opérateurs utilisés pour définir les règles sont AND et OR lorsqu'il existe plusieurs antécédents.

L'opérateur flou AND est utilisé pour évaluer la conjonction des antécédents des règles et l'opérateur OR pour évaluer la disjonction, puisque AND est le minimum et OR est le maximum, une technique d'inférence min-max est utilisée pour la mise en œuvre des règles linguistiques , tel que défini ci-dessous :

1. OR : C = maximum (A, B)
2. AND : C = minimum (A, B)
3. Règles avec la même sortie : C = maximum (minimum (A1, B1), minimum (A2, B2))

La description VHDL pour l'évaluation des règles à plusieurs antécédents se fait avec les fonctions maximum et minimum, qu'on a créé dans un paquet nommé « min_max_functions » comme indiqué ci-dessous :

- Fonction minimum :

```
function minimum ( a, b : std_logic_vector) return
std_logic_vector is
variable min : std_logic_vector (15 downto 0) := (others=> '0');
 begin
 if (a < b) then min := a;
 else min := b;
 end if;
 return min;
end minimum;
```

- Fonction maximum :

```
function maximum ( a, b : std_logic_vector) return
std_logic_vector is
variable max : std_logic_vector (15 downto 0) := (others=> '0');
 begin
 if (a > b) then max := a;
 else max := b;
 end if;
 return max;
end maximum;
```

Une fois les fonctions maximum et minimum définies, on obtient la valeur de chaque règle, évaluée comme suit :

```
Rule1:Inf(0) <= minimum(uv(0),ud(2));

Rule2_et_Rule3:Inf(1)<=maximum(minimum(uv(0),ud(1)), minimum(uv(1), ud(2)));

Rule4_et_Rule5:Inf(2)<=maximum(minimum(uv(0),ud(0)), minimum(uv(1), ud(1)));

Rule6_et_Rule7:Inf(3)<=maximum(minimum(uv(1),ud(0)), minimum(uv(2), ud(2)));

Rule8_et_Rule9:Inf(4)<=maximum(minimum(uv(2),ud(0)), minimum(uv(2), ud(1)));
```

III.4 La défuzzification

Pour le processus de défuzzification, la méthodes du centre de gravité est choisie. Le principe de cette méthode est formulé par l'équation mathématique suivante

$$z* = \int \mu A(z) z dz \,/ \int \mu A(z) z dz \qquad \textbf{(III.1)}$$

Chaque sortie floue est multipliée par sa position de singleton correspondante, La somme de ce produit est divisée par la somme de toutes les sorties floues pour obtenir le résultat de la sortie finale suivant l'algorithme ci-dessous

```
sum = 0;
For i = 1 to n faire
product = (s(i) × f(i)) + product;
sum = f[i] + sum;
end  ;
output = product /sum
```

Où

n : le nombre des singleton de sortie

s : le tableau des singletons de sortie

f : le tableau des résultat des règles évaluées

Afin de décrire cette méthode en VHDL des fonctions « sum », « prod »et « div » ont été créés dans un paquet nommé « math_functions » , Les codes VHDL correspondants à ces fonctions sont décrits respectivement comme suites:

```
-----------------somme--------------------
function sum ( a : singleton) return std_logic_vector is
variable somme : std_logic_vector (15 downto 0) := (others=> '0');
 begin
 for i in 0 to 4 loop
somme:=a(i)+somme;
 end loop;
 return somme;
end sum;
```

```
------------------produit-------------
function prod ( a,b : singleton) return std_logic_vector is
variable produit : std_logic_vector (31 downto 0):= (others=> '0');
 begin
 for i in 0 to 4 loop
produit:= produit+(a(i)*b(i));
 end loop;
```

```
---------------------division--------------------------
function div ( num,den : std_logic_vector) return std_logic_vector is
variable Quns,numuns: unsigned(31 downto 0);
variable denuns: unsigned(15 downto 0);
variable Q: std_logic_vector(31 downto 0);
begin
  numuns := unsigned(num);
  denuns := unsigned(den);
  Quns := numuns/denuns;
  Q := std_logic_vector(Quns);
  return q;
  end div;
```

La description de cette méthode est donc la suivante :

```
product<=prod(f,Inf);
somme <= sum(Inf);
freinage<= div(product,somme);
```

III.5 Simulation et résultats

Le VHDL offre la possibilité de créer une entité de conception pouvant interagir avec une conception matérielle. Cette entité est connue sous le nom de 'Test Bench' et elle fournit bien plus qu'un simple fichier de simulation et la sortie qui en résulte. Cela vient du fait que le Test Bench peut modifier les valeurs de simulation qu'il fournit au modèle en fonction de sa sortie . Par exemple, si le modèle testé est une unité logique arithmétique, le Test Bench peut vérifier

itérativement la correction par des tests successifs. S'il découvre une erreur dans le processus, il peut fournir une entrée supplémentaire au modèle pour identifier la source du problème. De plus, le banc de test peut être exécuté tout au long du cycle de conception. Si une erreur est rencontrée lors de l'ajout d'une nouvelle révision, elle peut être signalée immédiatement, ce qui réduit considérablement le temps nécessaire au débogage du système.[29]

Pour notre exemple, de différentes valeurs d'entrée afin de tester le déroulement du code VHDL sont choisies. Le test est donné pour des signaux périodiques de 25 ns à des valeurs en format hexadécimales :

- les valeurs hexadécimales x"1B", x"BA", x"AE" et x"BD" sont prises pour déterminer l'entrée V , et
- les valeurs x"7F", x"48", x"DE", x"5A" pour la deuxième entrée D.

Le code VHDL de l'architecture du Test Bench correspondent:

```
architecture archi of tb_FLC is
 component FLC is
port (V_input,D_input: in std_logic_vector(7 downto 0);
       f_output: out std_logic_vector(31 downto 0));
end component ;
signal tb_V_input,tb_D_input :std_logic_vector(7 downto 0);
signal tb_f_output: std_logic_vector(31 downto 0);
begin
c:FLC port map(tb_v_input,tb_d_input,tb_f_output);
process
begin
tb_v_input<=x"1B" ; wait for 25 ns;
tb_v_input<=x"BA" ; wait for 25 ns;
tb_v_input<=x"AE" ; wait for 25 ns;
tb_v_input<=x"DB" ; wait for 25 ns;
end process;
process
begin
tb_d_input<=x"7F" ; wait for 25 ns;
tb_d_input<=x"48" ; wait for 25 ns;
tb_d_input<=x"DE" ; wait for 25 ns;
tb_d_input<=x"5A" ; wait for 25 ns;
end process;
end archi;
```

Le Test Bench a été compilé et simulé à l'aide du logiciel de simulation Model Sim la figure ci-dessous montre les résultats obtenues pour les différentes variables d'entrée

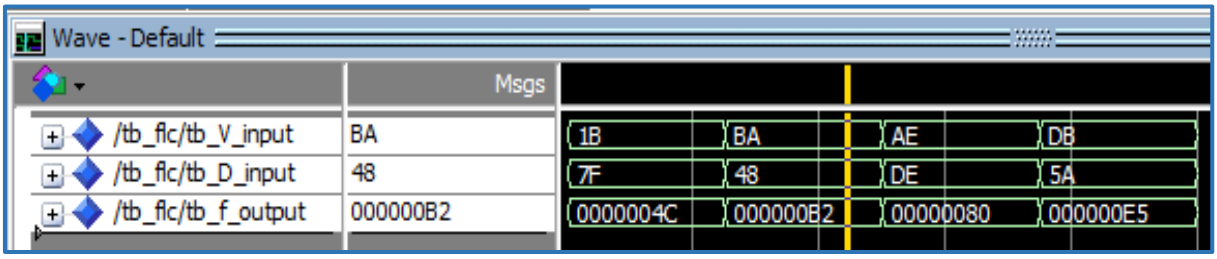

Figure III. 6: Rapport de simulation

Le tableau suivant montre les résultats obtenue après les avoir convertis en valeurs réelles

Tableau III. 3 : Résultat de simulation en format réelle et en format hexadécimal.

Les valeurs réelles de V	Les valeurs de V en hexadécimale	Les valeurs réelles de D	Les valeurs de D en hexadécimale	Les valeurs réelles de F	Les valeurs de F en hexadécimale
27	1B	80	7F	30	4C
174	AE	139	DE	51	80
186	BA	45	48	70	B2
189	DB	57	5A	90	E5

III.5.1 Résultats théoriques

Pour l'essai théorique la valeur hexadécimale x"BA" est prise pour déterminer l'entrée V, et la valeur hexadécimale x"4F" est prise pour déterminer l'entrée D. Les dégrées d'appartenance de chaque entrée sont représentées dans les deux figures suivantes :

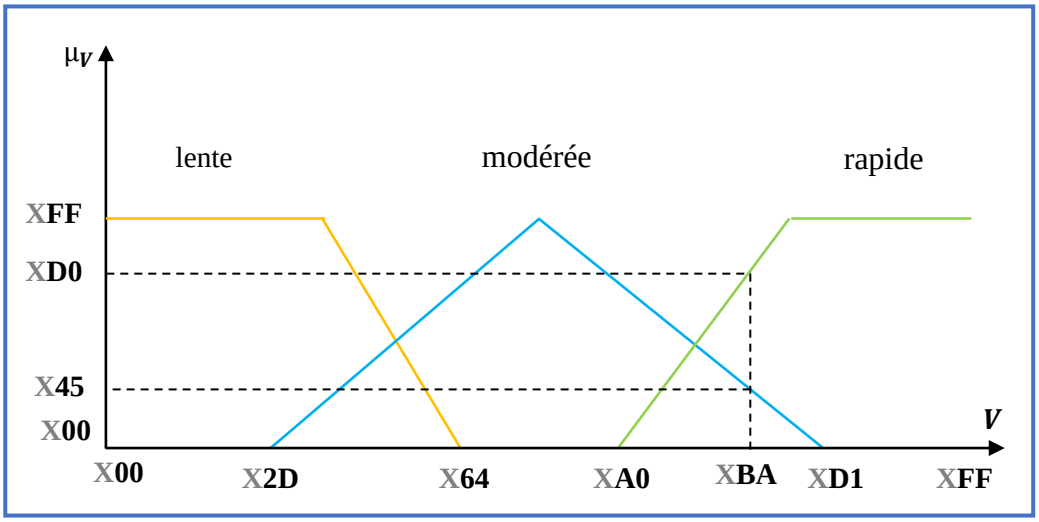

Figure III. 7: Degré d'appartenance de V=x"BA"

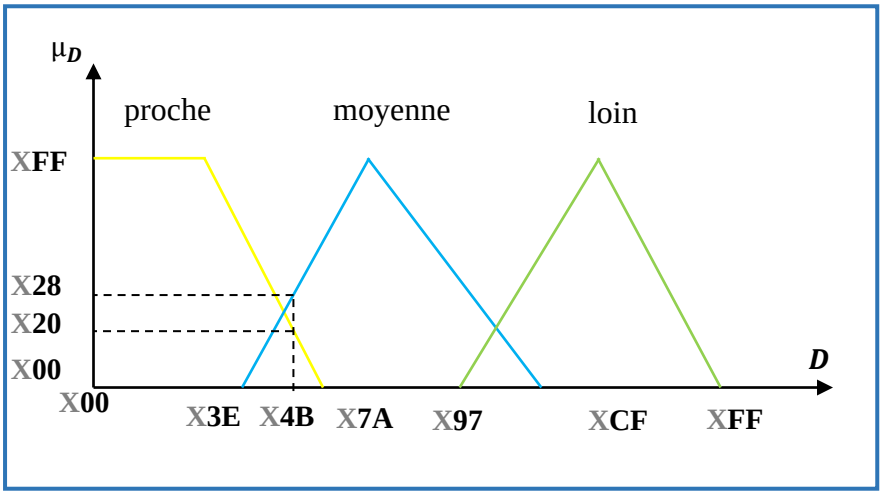

Figure III. 8 : Degré d'appartenance de D=x"4F"

Les dégrées d'apparence de la sortie *F* seront déterminés en inférant sur les dégrée d'appartenances obtenus par projection des quatre règles activées suivantes et selon la méthode Madani

- **IF** *V* is moderee **and** *D* is proche **then** *F* is E (min=x"20")
  x''45'' x''20''
- **IF** *V* is moderee **and** *D* is moyenne **then** *F* is M (min=x"28")
  x''45'' x''28''
- **IF** *V* is rapide **and** *D* is proche **then** *F* is TE (min=x"20")
  x''D0'' x''20''
- **IF** *V* is rapide **and** *D* is moyenne **then** *F* is TE (min= x"28")
  x''D0'' x''28''

(max= x"28")

Les résultats obtenus sont présentés dans la figure suivante :

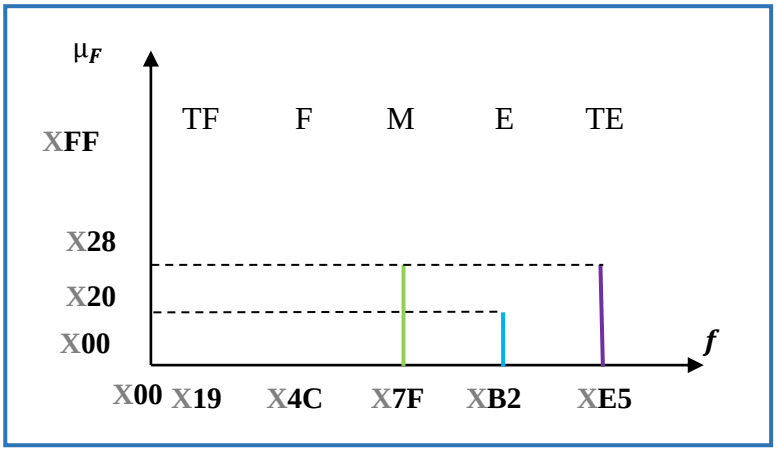

Figure III. 9: Résultats d'inférence Mamdani

L'étape de défuzzification sera effectuée en exploitant les résultats d'inférence obtenus et la méthode du centre de gravitée

$$\boldsymbol{F} = \frac{x"28" * x"7F" + "x"20" * x"B2" + x"28" * x"E5"}{x28 + x"20" + x"28"} = x"B2"$$

III.5.2 Discussion des résultats

D'après la comparaison entre les résultats obtenue à partir de la simulation du logiciel Modélisme et les calculs faits théoriquement on constate que les résultats de la simulation sont les résultats attendus

III.5 Conclusion

Dans ce chapitre, une approche a été présenté pour décrire un contrôleur à logique floue de type Mamdani en utilisant la description VHDL. Un exemple d'un contrôleur flou d'un système de freinage avec deux entrées et une sortie a été choisi.

Conclusion générale

Conclusion

N'étant pas nécessairement vraies ou fausses, les informations vagues, imprécises ou approximatives sont mieux décrites avec la logique floue qu'avec la logique classique. La logique floue autorise un changement graduel ou des intermédiaires dans le passage d'un état a un autre. Elle permet aussi une appartenance partielle d'un élément dans une classe donnée ainsi qu'une infinité de niveaux comprise entre 0 et 1. La logique floue représente une généralisation de la logique booléenne [33]. L'incorporation de la logique floue dans les systèmes de contrôle donne naissance à ce que nous appellerons des systèmes de contrôle flous. Ces derniers peuvent être implémentés sur de nombreux processeurs tel que les FPGAs.

Les FPGAs sont devenus les processeurs numériques de choix dans de nombreuses situations, elle sont une architecture appropriée pour la mise en œuvre des contrôleurs tels que le PID, Logic floue. Une conception FPGA peut être décrite en utilisant des langages de description, parmi ces langages le VHDL. Le VHDL est un langage de description très utile à connaître et à utiliser ce dernier est l'un des langages les plus utilisé dans la description des circuits électroniques.

L'application du control flou sur le système de freinage automatique et sa conception en langage VHDL montre des résultats fiables et précis. Les résultats de variation de vitesse et de distance de l'architecture proposée obtenus par le logiciel de simulation ModelSim sont en adéquation avec les résultats attendus. Les variations de vitesse et de distance ont tendance à ralentir la voiture, plus la vitesse est élevée ainsi que la distance entre la voiture et l'obstacle est proche, plus le ralentissement résultant est important, et vice versa.

En perceptives, il serait bien de poursuivre ce travail en :

- Implémentant l'application en temps réel.
- Utilisant d'autres méthodes de défuzzification pour faire une comparaison.
- Implémentant d'autres niveaux de types de contrôleurs flous
- Utilisant Fuzzy Toolbox -Fuzzy inference System(FIS) de MATLAB pour la simulation et la comparaison des résultats entre le contrôleur implémenté en VHDL et le contrôleur simulé en MATLAB.

Annexe A

Code VHDL du paquet « membership_functions »

Annex A

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

package membership_functions is
--V-----
type V_type is (lente, moderée, rapide, none);
type v_mf_trap is record term: v_type;
point1: std_logic_vector(7 downto 0);
 pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0);
 pente2: std_logic_vector(7 downto 0);
 point3: std_logic_vector(7 downto 0);
 point4: std_logic_vector(7 downto 0);
 end record;
type v_mf_tri is record term: v_type;
point1: std_logic_vector(7 downto 0);
 pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0);
 pente2: std_logic_vector(7 downto 0);
 point3: std_logic_vector(7 downto 0);
 end record;
type v_mfs is array(natural range <>)of v_mf_trap;
constant v_trapezoidal: v_mfs:=
((term => lente, point1 => x"00",point2 => x"00",point3 => x"32",point4=> x"64",
pente1 => x"00", pente2 => x"05"),
(term => moderée, point1 => x"00", pente1 => x"00", point2 => x"00", pente2 =>
x"00",point3 => x"00",point4=> x"00"),
(term => rapide, point1 => x"A0",point2 => x"BF",point3 => x"FF",point4=> x"FF",
pente1 => x"08", pente2 => x"00"),
(term => none, point1 => x"FF",point2 => x"FF",point3 => x"FF",point4=> x"FF", pente1
=> x"FF", pente2 => x"FF"));
type v_mfs_1 is array(natural range <>)of v_mf_tri;
constant v_triangular: v_mfs_1 :=
((term => lente, point1 => x"00",point2 => x"00",point3 => x"00", pente1 => x"00",
pente2 => x"00"),
(term => moderée, point1 => x"2D",point2 => x"7F",point3 => x"D1", pente1 => x"03",
pente2 => x"03"),
(term => rapide, point1 => x"00",point2 => x"00",point3 => x"00", pente1 => x"00",
pente2 => x"00"),
(term => none, point1 => x"FF",point2 => x"FF",point3 => x"FF", pente1 => x"FF",
pente2 => x"FF"));
type v_degree is array (0 to 4) of std_logic_vector( 15 downto 0);
-----d----------
type D_type is (proche, moyenne,loin,none);
type D_mf_trap is record
```

```
term: D_type;
point1: std_logic_vector(7 downto 0);
 pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0);
 pente2: std_logic_vector(7 downto 0);
 point3: std_logic_vector(7 downto 0);
 point4: std_logic_vector(7 downto 0);
 end record;
 type D_mf_tri is record
term: D_type;
point1: std_logic_vector(7 downto 0);
 pente1: std_logic_vector(7 downto 0);
point2: std_logic_vector(7 downto 0);
 pente2: std_logic_vector(7 downto 0);
 point3: std_logic_vector(7 downto 0);
 end record;

type D_mfs is array(natural range <>)of D_mf_trap;
constant D_trapezoidal: D_mfs:=
((term => proche, point1 => x"00",point2 => x"00",point3 => x"2F",point4=> x"4C",
pente1 => x"00", pente2 => x"08"),
(term => moyenne, point1 => x"00", pente1 => x"00", point2 => x"00", pente2 =>
x"00",point3 => x"00",point4=> x"00"),
(term => loin, point1 => x"00",point2 => x"00",point3 => x"00",point4=> x"00", pente1
=> x"00", pente2 => x"00"),
(term => none, point1 => x"FF",point2 => x"FF",point3 => x"FF",point4=> x"FF", pente1
=> x"FF", pente2 => x"FF"));
type D_mfs_1 is array(natural range <>)of D_mf_tri;
constant D_triangular: D_mfs_1 :=
((term => proche, point1 => x"00",point2 => x"00",point3 => x"00", pente1 => x"00",
pente2 => x"00"),
(term => moyenne, point1 => x"3E",point2 => x"7A",point3 => x"B7", pente1 => x"04",
pente2 => x"04"),
(term => loin, point1 => x"97",point2 => x"CF",point3 => x"FF", pente1 => x"04",
pente2 => x"05"),
(term => none, point1 => x"FF",point2 => x"FF",point3 => x"FF", pente1 => x"FF",
pente2 => x"FF"));
type D_degree is array (0 to 2) of std_logic_vector( 15 downto 0);
------F--------------

constant très_faible : std_logic_vector := x"0019";
constant faible : std_logic_vector := x"004C";
constant moyen : std_logic_vector := x"007F";
constant  élevé : std_logic_vector := x"00B2";
constant très_élevé : std_logic_vector := x"00E5";
type singleton is array (0 to 4) of std_logic_vector(15 downto 0);

end package;
```

Annexe B

Code VHDL du paquet « min_max_functions »

Annex B

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

package min_max_functions is
function minimum ( a, b : std_logic_vector) return std_logic_vector;
function maximum ( a, b : std_logic_vector) return std_logic_vector;
function max ( a : singleton) return integer;
end min_max_functions;

package body min_max_functions is
-----min function---------
function minimum ( a, b : std_logic_vector) return
std_logic_vector is
variable min : std_logic_vector (15 downto 0) := (others=> '0');
 begin
 if (a < b) then min := a;
 else min := b;
 end if;
 return min;
end minimum;
--- maximum function----------
function maximum ( a, b : std_logic_vector) return
std_logic_vector is
variable max : std_logic_vector (15 downto 0) := (others=> '0');
 begin
 if (a > b) then max := a;
 else max := b;
 end if;
 return max;
end maximum;

end min_max_functions;
```

Annexe C

Code VHDL du paquet « math_function »

Annex C

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use work.membership_functions.all;

package math_functions is
function sum ( a : singleton) return std_logic_vector;
function prod ( a,b : singleton) return std_logic_vector;
function div ( num,den : std_logic_vector) return std_logic_vector;
end math_functions;

package body math_functions is
-----------------somme-------------------------------------------------
function sum ( a : singleton) return std_logic_vector is
variable somme : std_logic_vector (15 downto 0) := (others=> '0');
 begin
 for i in 0 to 4 loop
somme:=a(i)+somme;
 end loop;
 return somme;
end sum;
------------------produit-----------------------------------------
function prod ( a,b : singleton) return std_logic_vector is
variable produit : std_logic_vector (31 downto 0):= (others=> '0');
 begin
 for i in 0 to 4 loop
produit:= produit+(a(i)*b(i));
 end loop;
 return produit;
end prod;
---------------------division-----------------------------------------
function div ( num,den : std_logic_vector) return std_logic_vector is
variable Quns,numuns: unsigned(31 downto 0);
variable denuns: unsigned(15 downto 0);
variable Q: std_logic_vector(31 downto 0);
begin
  numuns := unsigned(num);
  denuns := unsigned(den);
  Quns := numuns/denuns;
  Q := std_logic_vector(Quns);
  return q;
  end div;

end math_functions;
```

Annexe D

Code VHDL principal « FLC »

Annex D

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.membership_functions.all;
use work.min_max_functions.all;
use work.math_functions.all;


entity FLC is
port (v_input,d_input: in std_logic_vector(7 downto 0);
       f_output: out std_logic_vector(31 downto 0));
end FLC;

architecture description of FLC is
signal uv:v_degree;
signal ud:d_degree;
signal Inf: singleton;
signal f: singleton :=(très_faible,faible,moyen,élevé,très_élevé);
signal product:  std_logic_vector(31 downto 0);
signal somme: std_logic_vector (15 downto 0):=x"0000";
begin
---------------fuzzification part-------------------------------------------------
process(v_input,d_input)
begin
--------------------------te-----------------------------------------
For i in 0 to 2 loop
----------------------trapezoidal function-----------------
if(v_input<v_trapezoidal(i).point1) then
   uv(i)<=(others=>'0');
elsif (v_input<v_trapezoidal(i).point2) then
   uv(i)<= (v_input - v_trapezoidal(i).point1)*(v_trapezoidal(i).pente1);
elsif (v_input<v_trapezoidal(i).point3) then
   uv(i)<=x"00FF";
elsif (v_input<v_trapezoidal(i).point4) then
   uv(i) <=(v_trapezoidal(i).point4-v_input)*(v_trapezoidal(i).pente2);
----------------------triangular function----------------
elsif (v_input<v_triangular(i).point1) then
   uv(i)<=(others=>'0');
elsif (v_input<v_triangular(i).point2) then
   uv(i)<= (v_input - v_triangular(i).point1)*(v_triangular(i).pente1);
elsif (v_input<v_triangular(i).point3) then
   uv(i) <=(v_triangular(i).point3-v_input)*(v_triangular(i).pente2);
 else
  uv(i)<=(others=>'0');
end if;
```

```
end loop;
----------------------------tr-------------------------------------------------
For i in 0 to 2 loop
-------------------------trapezoidal function----------------------
if(d_input<d_trapezoidal(i).point1) then
   ud(i)<=(others=>'0');
elsif (d_input<d_trapezoidal(i).point2) then
   ud(i)<= (d_input - d_trapezoidal(i).point1)*(d_trapezoidal(i).pente1);
elsif (d_input<d_trapezoidal(i).point3) then
   ud(i)<=x"00FF";
elsif (d_input<d_trapezoidal(i).point4) then
   ud(i) <=(d_trapezoidal(i).point4-d_input)*(d_trapezoidal(i).pente2);
-------------------------triangular function----------------------
elsif (d_input<d_triangular(i).point1) then
   ud(i)<=(others=>'0');
elsif (d_input<d_triangular(i).point2) then
   ud(i)<= (d_input - d_triangular(i).point1)*(d_triangular(i).pente1);
elsif (d_input<d_triangular(i).point3) then
   ud(i) <=(d_triangular(i).point3-d_input)*(d_triangular(i).pente2);
 else
   ud(i)<=(others=>'0');
end if;
end loop;
end process;
-----------------------Inf part-----------------------------------
Rule1:Inf(0) <= minimum(uv(0),ud(2));
Rule2_et_Rule3: Inf(1) <= maximum( minimum(uv(0), ud(1)), minimum(uv(1), ud(2)));
Rule4_et_Rule5: Inf(2) <= maximum( minimum(uv(0), ud(0)), minimum(uv(1), ud(1)));
Rule6_et_Rule7: Inf(3) <= maximum( minimum(uv(1), ud(0)), minimum(uv(2), ud(2)));
Rule8_et_Rule9: Inf(4) <= maximum( minimum(uv(2), ud(0)), minimum(uv(2), ud(1)));
-----------------------defuzzification part---------------------
product <=prod(f,Inf);
somme <= sum(Inf);
f_output <= div(product,somme);
end description;
```

Annexe E

Test Bench VHDL « tb_FLC »

Annex E

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

entity tb_FLC is
end tb_FLC;

architecture archi of tb_FLC is
 component FLC is
port (V_input,D_input: in std_logic_vector(7 downto 0);
       f_output: out std_logic_vector(31 downto 0));
end component ;

signal tb_V_input,tb_D_input :std_logic_vector(7 downto 0);
signal tb_f_output: std_logic_vector(31 downto 0);

begin
c:FLC port map(tb_v_input,tb_d_input,tb_f_output);
process
begin
tb_v_input<=x"1B"; wait for 25 ns;

tb_v_input<=x"BA"; wait for 25 ns;

tb_v_input<=x"AE"; wait for 25 ns;

tb_v_input<=x"DB"; wait for 25 ns;
end process;

process
begin
tb_d_input<=x"7F"; wait for 25 ns;

tb_d_input<=x"48"; wait for 25 ns;

tb_d_input<=x"DE"; wait for 25 ns;

tb_d_input<=x"5A"; wait for 25 ns;
end process;

end archi
```

Bibliographies

[1] J. Yen, R. Langari, and L.A. Zadeh, Industrial Applications of Fuzzy Logic and Intelligent Systems. NJ: IEEE Press, 1995.

[2] R.A. Aliev and R.R. Aliev, Soft Computing and its Applications. World Scientific, New Jersey, 2001

[3] M. McKenna, and B.M. Wilamowski, "Implementing a fuzzy system on a field programmable gate array," in IJCNN'01, International Joint Conf. Neural Networks, Washington, DC, 2001, vol.1, pp. 189–194

[4] G. Lizarraga, R. Sepulveda, O. Montiel, and O. Castillo, "Modeling and simulation of the defuzzification stage using Xilinx system generator and Simulink," in Soft Computing for Hybrid Intelligent Systems, Studies in Computational Intelligence, vol. 154, Berlin Heidelberg: Springer, 2008, pp. 333–343.

[5] H. Zavala, I.Z. Batyrshin, I.J. Rudas, L. Villa Vargas, and O. Camacho Nieto, "Parametric operations for digital hardware implementation of fuzzy systems," in MICAI 2009, LNAI, vol. 5845, Berlin Heidelberg: Springer, 2009, pp. 432–443.

[6] I.J. Rudas, I.Z. Batyrshin, A. Hernández Zavala, O. Camacho Nieto, and L. Villa Vargas, "Digital fuzzy parametric conjunctions for hardware implementation of fuzzy systems," in ICCC2009, IEEE 7th Int. Conf. Computational Cybernetics, Palma de Mallorca, Spain, 2009, pp. 157– 166.

[7]Youssef, A., Telbany, M.E. and Zekry, A. (2018). Reconfigurable generic FPGA implementation of fuzzy logic controller for MPPT of PV systems. Renewable and Sustainable Energy Reviews, 82, pp.1313–1319.

[8] L. Bouselham, M. Hajji, B. Hajji, A. El Mehdi, H. Hajji, "Hardware Implementation of Fuzzy Logic MPPT Controller on a FPGA Platform" ,ENSA-UMP, Morocco, BP 669, 60000 oujda

[9] Monmasson, E. and Cirstea, M.N. (2007). FPGA Design Methodology for Industrial Control Systems—A Review. IEEE Transactions on Industrial Electronics, 54(4), pp.1824–1842.

[10] Zadeh, L. A. (1983). Commonsense knowledge representation based on fuzzy logic.

[11] Mamdani, E.H., Assilian, S., "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," International Journal of Human-Computer Studies, Vol. 51, Issue 2, pages 135 – 147, Aug 1999.

[12] S. sumathi and surekha Paneerselvam: Computational Intelligence Paradigms Theory and Applications using MATLAB®, CRC Press, © 2010 by Taylor and Francis Group, LLC

[13] S. N. Sivanandam, S. Sumathi and S. N. Deepa Introduction to Fuzzy Logic using MATLAB, © Springer-Verlag Berlin Heidelberg 2007.

[14] L. Baghli, "Contribution à la commande de la machine asynchrone, utilisation de la logique floue, des réseaux de neurones et des algorithmes génétiques," Université Henri Poincaré-Nancy 1, 1999.

[15] Klir and Bo Yuan Fuzzy Sets & Fuzzy Logic Theory and Applications George j, © 1995.

[16] L. A. Zadeh, "Fuzzy sets", Information and Control, Vol. 8, pp. 338- 351, (1965).

[17] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller-part I", IEEE Trans. Syst., Man. Cybern., Vol. 20, no. 2, pp. 404-418, (1990)

[18] Kevin Skahill, VHDL for programmable logic (Addison-Esley, Massachusetts, 1996).

[19] P. Graham, B. Nelson and B. Hutchings, "Instrumenting Bitstreams for Debugging FPGA Circuits," in Field-Programmable Custom Computing Machines (FCCM'0l), Pravo, Utah, USA, 2001.

[20] D. Galan, C.J. Jimenez, A. Barriga, S. Sanchez-Solano, VHDL Package for description of Fuzzy Logic Controllers, Design Automation Conference.

[21] S. Brown, Fundamentals of digital Logic With VHDL design, McGraw Hill, 2005

[22] Enrique Mandado Pérez, ValdésD. and Jacobo, L. (2003). Dispositivos lógicos programables. Madrid: Thomson, D.L.

[23] LUBA T., MARKOWSKI M.A., ZBIERZCHOWSKI B., Computerised designing digital circuits in PLD structures (in polish), WK i L, Warsaw, 1993

[24] CHMIEL M., DREWNIOK L., HRYNKIEWICZ E., Single Board PLC Based on PLD, in Conference Proceedings, International Conference Programmable Devices And Systems PDS '95, Institute of Electronics, Silesian Technical University of Gliwice, 1995

[25] Fabricante de FPGA's y creador de Xilinx System Generator y Xilinx ISE

[26] Paul, C. (2018). Architecture exploration of fpga based accelerators for bioinformatics applications.

[27] Kevin Skahill, VHDL for programmable logic (Addison-Esley, Massachusetts, 1996)

[28] Sánchez-Élez, M. (n.d.). INTRODUCCIÓN A LA PROGRAMACIÓN EN VHDL.

[29] W. Ailes, J. (1990). Automated digital hardware synthesis using VHDL. thesis.

[30] Messerli, E. (2007). Manuel VHDL pour la synthèse automatique.

[31] Pinillos, F. (2006). Implementación de controladores difusos con microcontroladores

[32]I.Rizianiza and D M Shoodiqin 2021 J. Phys.: Conf. Ser. 1833 012005

[33] ZAOUCHE, Faika, REKIOUA, D., et al. Maximisation de puissance des systèmes photovoltaïques. 2018. Thèse de doctorat. Université Abderrahmane Mira-Bejaia.

Résumé

Ce travaille présente une méthode de conception d'un contrôleur flou de type Mamdani d'un système de freinage automatique. Ce dernier prend la vitesse et la distance comme variables d'entrées et le freinage comme sortie. Les différentes parties du contrôleur, la fuzzification, les règles d'inférence et la défuzzification sont décrite en langage de description matérielle VHDL qui permet d'augmenter et d'améliorer les performances de l'ensemble du système. La conception VHDL est ensuite simuler avec le logiciel Model Sim et les résultats obtenus sont corrects pour de différentes valeurs de simulation.

Mots clés : contrôleur flou ,VHDL, FPGA

Abstract

This work presents a method for designing a Mamdani-type fuzzy controller of an automatic braking system. The latter takes speed and distance as input variables and the braking as output. The different parts of the controller, the fuzzification, the inference rules and the defuzzification are described in the VHDL hardware description language which allows to increase and improve the performance of the whole system. The VHDL design is then simulated using Model Sim software and the results obtained are correct for different simulation values

Keywords : fuzzy logic controller, VHDL, FPGA