

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF1-
UFAS (ALGERIE)

MEMOIRE

Présenté à la faculté de Technologie
Département d'Electronique

Pour l'obtention du Diplôme de

MAGISTER

Option : Contrôle

Par

Mr: Ghedjati Fouaz

Thème

Contrôleur flou reconfigurable sur FPGA

Soutenu le : -- / -- /201 3 devant la commission d'examen :

Mr : Bouloufa Abdeslam	Prof à l'université Ferhat Abbas – Sétif	<i>Président</i>
Mr : Khouni Larbi	MCA à l'université Ferhat Abbas – Sétif	Examineur
Mr : Ziet Lahcene	MCA à l'université Ferhat Abbas – Sétif	Examineur
Mr : Chemali Hamimi	MCA à l'université Ferhat Abbas – Sétif	Rapporteur

:

Remerciements

A l'issue de ce travail je tiens à adresser ma reconnaissance et mes remerciements à toutes les personnes qui ont contribué, chacune à leur manière, à l'accomplissement de ce mémoire.

Tout d'abord, je tiens à remercier très chaleureusement mon directeur de mémoire Monsieur H. Chemali, Maitre de Conférences à l'Université de Sétif, pour son encadrement, son suivi permanent et ses précieux conseils.

Je tiens à remercier vivement Monsieur A. Bouloufa Professeur au département d'Electronique de l'université de Sétif pour avoir accepté de présider le jury de ce mémoire de magister.

Mes remerciements vont aussi à Messieurs L. Khouni et L. Ziet Maitres de conférences au département d'Electronique de Sétif d'avoir accepté d'évaluer ce modeste travail.

J'exprime toute ma gratitude à mon père, ma mère, mes frères et mes sœurs qui m'ont apporté soutien et encouragements tout au long de mes études. Je leur réserve une pensée toute particulière.

Enfin, merci à mes amis qui ont partagé au quotidien mes espoirs et mes inquiétudes, qui m'ont réconforté dans les moments difficiles et avec qui j'ai partagé d'inoubliables instants de détente.

Table des matières

Introduction générale.....	1
1 Les architectures reconfigurables	5
1.1 Introduction	5
1.2 Types de reconfiguration	5
1.2.1 Reconfiguration logique	5
1.2.2 Reconfiguration par ‘jeu d’instructions’	6
1.2.3 Reconfiguration statique et dynamique	6
1.3 Classification des architectures reconfigurables.....	8
1.3.1 Classification selon la granularité.....	8
1.3.2 Classification selon le schéma de reconfiguration.....	8
1.4 Classification selon le mode traitement.....	10
1.4.1 Architecture orientée chemin de données.....	11
1.4.2 Architecture orientée traitement logique	11
1.4.3 Architecture orientée mémoire	11
1.5 Conception des socs reconfigurables.....	12
1.5.1 Les exigences au niveau système pour les SOCs reconfigurables :	13
1.5.2 Flot de conception de SoCs reconfigurables	15
1.6 Exemple de conception d’un système reconfigurable :	18
1.6.1 Création d’un flux de bits de reconfiguration en fonction de différence.....	19
1.6.2 Utilisation de flux binaires et programmation du FPGA.....	19
1.7 Introduction à la reconfiguration d’un contrôleur flou :.....	20
1.7.1 Etude de l’apport de la reconfiguration dans la fuzzification et l’inférence :..	24
1.7.2 Etude de l’apport de la reconfiguration dans la défuzzification :	25
1.8 Conclusion.....	26
2 Logique floue	27
2.1 Introduction	27
2.2 La logique floue.....	28
2.3 Théorie des ensembles flous.....	28
2.3.1 Notion d’appartenance partielle.....	28
2.3.2 Variable linguistique	29
2.3.3 Fonctions d’appartenance	29
2.3.4 Operateurs de la logique floue.....	30

2.3.5	Règles floues	30
2.3.6	La logique floue type-1 et type-2	31
2.3.7	Généralités sur la logique floue type-1.....	31
2.3.8	Généralités sur la logique floue type-2.....	33
2.4	Contrôleur flou	36
2.5	Implémentation de contrôleurs flous [16]	37
2.6	Conclusion.....	38
3	Langage VHDL et Circuits programmables	39
3.1	Introduction	39
3.2	Structure d'une description VHDL	40
3.2.1	L'entité	40
3.2.2	L'architecture	41
3.3	Les modes de travail en VHDL	43
3.3.1	Le mode combinatoire	43
3.3.2	Le mode séquentiel.....	43
3.4	Synthèse et simulation d'une description VHDL	44
3.4.1	Synthèse d'une description VHDL.....	44
3.4.2	Simulation d'une description VHDL.....	44
3.5	Outil de développement Xilinx 13.3.....	45
3.5.1	L'environnement de développement (XILINX ISE 13.3).....	45
3.6	Les circuits programmables.....	47
3.6.1	L'architecture des circuits FPGA	49
3.7	Développement d'un projet VHDL sur FPGA	53
3.8	Conclusion.....	54
4	Conception du contrôleur flou.....	55
4.1	Introduction	55
4.2	Système général.....	61
4.2.1	Schéma bloc de système adopté :	61
4.2.2	Description du système global de ventilation :.....	61
4.2.3	Le gestionnaire :	62
4.2.4	Contrôleur Global.....	63
4.2.5	Aiguilleur :	64
4.2.6	Temporisateur et mode de ventilation :	64
4.3	Gestionnaire embarqué.....	65
4.3.1	Description générale du Gestionnaire Embarqué:	65
4.3.2	L'interface :	67

4.3.3	L'aiguilleur :	67
4.3.4	Le Temporisateur :	67
4.3.5	Le Microprocesseur :	68
4.3.6	Bloc de commande standard :	68
4.3.7	Organigramme de fonctionnement	70
4.3.8	Module d'inférence	75
	Résultats de synthèse, de simulation et d'implantation du FLC_OP :	80
4.4	Le contrôleur flou de CO (FLC_CO) :	82
4.5	Le contrôleur flou mixte (FLC_MIXTE) :	84
4.6	Contrôleur flou de Trafic (circulation des véhicules).....	86
4.7	Contrôleur final :	88
4.7.1	La méthode de ventilation pour chaque zone :	90
4.8	Le contrôleur global :	92
4.9	Impacts et effets de la reconfiguration :	95
4.9.1	L'apport de la reconfiguration du bloc de fuzzification sur le FLC_OP :	96
4.9.2	L'apport de la reconfiguration du bloc de fuzzification sur le FLC_CO :	97
4.9.3	L'apport de la reconfiguration du bloc de fuzzification sur le FLC_trafic.....	97
4.9.4	L'apport de la reconfiguration du bloc de fuzzification sur le FLC global : ...	98
4.9.5	L'apport de la reconfiguration du bloc d'inférence sur le FLC_OP :	98
4.9.6	L'apport de la reconfiguration du bloc d'inférence sur le FLC_CO :	99
4.9.7	L'apport de la reconfiguration du bloc d'inférence sur le FLC du trafic :	99
4.9.8	L'apport de la reconfiguration du bloc d'inférence sur le FLC de global : ...	100
4.9.9	L'apport de la reconfiguration du bloc de défuzzification sur le FLC_OP : .	101
4.9.10	L'apport de la reconfiguration du bloc de défuzzification sur le FLC_CO : .	101
4.9.11	L'apport de la reconfiguration du bloc de défuzzification sur le FLC de trafic : 102	
4.9.12	L'apport de la reconfiguration du bloc de défuzzification sur le FLC global : 102	
4.10	Conclusion.....	103
5	Bibliographie	105

Liste des figures

Figure 1-1 Niveaux de couplage de base de reconfiguration logique.....	6
Figure 1-2 principe de la reconfiguration statique	7
Figure 1-3 principe de la reconfiguration dynamique	7
Figure 1-4 Les différents modèles de reconfiguration dynamique.....	9
Figure 1-5 une vue globale des SOC's reconfigurables.....	12
Figure 1-6 flot de conception	16
Figure 1-7 simulation d'un additionneur complet	19
Figure 1-8 simulation du compteur	20
Figure 1-9 implantation sur FPGA 2S30	20
Figure 1-10 Bloc diagramme du contrôleur flou	21
Figure 1-11 Principe de la reconfiguration partielle.....	23
Figure 1-12 Fonctionnement de LUT.....	24
Figure 1-13 Schéma bloc du contrôleur globale de ventilation.....	26
Figure 2-1 Exemples de fonctions d'appartenance	30
Figure 2-2 Système flou type-1.....	32
Figure 2-3 Structure d'un système flou type-2	34
Figure 2-4 Fonction d'appartenance d'un ensemble flou type-2.....	35
Figure 2-5 Structure générale du contrôleur flou	37
Figure 3-1 Demi_ADD.....	42
Figure 3-2 XILINX ISE 13.3.....	47
Figure 3-3 Classification des circuits logiques.....	48
Figure 3-4 Organisation des circuits FPGA de Xilinx.....	50
Figure 3-5 Schéma d'une cellule CLB (XC4000 de Xilinx)	51
Figure 3-6 Schéma des blocs d'entrée /sortie(IOB).....	52
Figure 3-7 Réseau mémoire SRAM	52
Figure 4-1 Solution proposée.....	59
Figure 4-2 Solution adopté.....	59
Figure 4-3 L'aiguilleur	64
Figure 4-4 Schéma général de gestionnaire embarqué.....	65
Figure 4-5 Organigramme de fonctionnement.....	70
Figure 4-6 Schéma fonctionnel des opérations dans un contrôleur flou.....	71
Figure 4-7 Les fonctions d'appartenance du taux d'opacité.....	72
Figure 4-8 Les fonctions d'appartenances de variation du taux d'opacité.....	73
Figure 4-9 Les fonctions d'appartenances de l'état d'opacité.....	73

Figure 4-10 Méthode de programmation de la fuzzification.....	74
Figure 4-11 Les résultats de la simulation de fuzzification.....	75
Figure 4-12 L'implantation du module de fuzzification sur FPGA.....	75
Figure 4-13 Sélection des règles Actives.....	76
Figure 4-14 Calcul du Minimum.....	77
Figure 4-15 Simulation d'engin d'inférence	77
Figure 4-16 L'implantation d'engin d'inférence sur FPGA	78
Figure 4-17 La conversion des règles actives résultantes.....	79
Figure 4-18 La défuzzification par centre de gravité.....	79
Figure 4-19 Simulation de la défuzzification du FLC_OP.....	80
Figure 4-20 Exécution du programme VHDL du contrôleur OP.\$.....	80
Figure 4-21 La structurent du contrôleur d'opacité (FLC_OP).	81
Figure 4-22 Résultat de la simulation du FLC_OP	82
Figure 4-23 Résultats de la simulation du FLC_OP.....	82
Figure 4-24 Le schéma RTL du contrôleur d'OP (FLC_CO)	83
Figure 4-25 Exécution du programme VHDL du contrôleur op	84
Figure 4-26 .L'implantation du contrôleur de CO sur FPGA.	84
Figure 4-27 Schéma RTL du contrôleur mixte (FLC_MIXTE)	85
Figure 4-28 Résultats de la simulation du contrôleur mixte.	86
Figure 4-29 L'implantation du contrôleur mixte sur FPGA	86
Figure 4-30 Schéma RTL du contrôleur trafic (FLC-Trafic).....	87
Figure 4-31 Résultats de la simulation du contrôleur trafic.	88
Figure 4-32 L'implantation du contrôleur trafic sur FPGA	88
Figure 4-33 Le schéma RTL du contrôleur final (FLC-Final).....	89
Figure 4-34 Résultats de la simulation du contrôleur final.	90
Figure 4-35 L'implantation du contrôleur final sur FPGA.....	90
Figure 4-36 Le schéma RTL du module "répartition-vent" de la zone 1.....	91
Figure 4-37 Implantation du module de répartition sur FPGA de type Spartan 3E.....	92
Figure 4-38 Le schéma RTL du contrôleur global.	93
Figure 4-39 Résultats de la simulation du contrôleur global.....	94
Figure 4-40 L'implémentation du contrôleur global sur un FPGA de type Virtex5.	94

Liste des tableaux

Tableau 4-1 Comparaison entre les différentes approches.....	57
Tableau 4-2 Look up table (pour un mode de ventilation : temporisation.....	68
Tableau 4-3 Codage des fonctions d'appartenances	73
Tableau 4-4 Résultats de la synthèse du module de fuzzification du FLC_OP.	74
Tableau 4-5 Base de règles de contrôleur flou.....	76
Tableau 4-6 Codage des règles.	76
Tableau 4-7 Résultat de synthèse du module d'inférence	77
Tableau 4-8 Résultats de la synthèse du module de la défuzzification du FLC_OP.	79
Tableau 4-9 Résultats de la synthèse du FLC_OP.	81
Tableau 4-10 Résultats de synthèse du contrôleur FLC_CO.	83
Tableau 4-11 Résultats de synthèse du contrôleur mixte.	85
Tableau 4-12 Résultats de synthèse du contrôleur Trafic.	87
Tableau 4-13 Résultats de la synthèse d'un contrôleur final.	89
Tableau 4-14 Méthode de ventilation.....	91
Tableau 4-15 Résultats de la synthèse du module "répartition-vent "de la zone 1	91
Tableau 4-16 Résultats de la synthèse d'un contrôleur global.	93
Tableau 4-17 L'apport de la reconfiguration sur le bloc de fuzzification du FLC_CO.....	96
Tableau 4-18 L'apport de la reconfiguration sur le bloc de fuzzification du FLC_CO.....	97
Tableau 4-19 L'apport de la reconfiguration du bloc de fuzzification sur le FLC de trafic. .	97
Tableau 4-20 L'apport de la reconfiguration du bloc de fuzzification sur le FLC global.	98
Tableau 4-21 L'apport de la reconfiguration du bloc d'inférence du FLC_OP.	98
Tableau 4-22 L'apport de la reconfiguration du bloc d'inférence du FLC_CO.....	99
Tableau 4-23 L'apport de la reconfiguration du bloc d'inférence sur le FLC du Trafic.....	99
Tableau 4-24 L'apport de la reconfiguration du bloc d'inférence sur le FLC globale.	100
Tableau 4-25 L'apport de la reconfiguration du bloc d'inférence du FLC_OP.	101
Tableau 4-26 L'apport de la reconfiguration sur le bloc de défuzzification du FLC_CO	101
Tableau 4-27 L'apport de la reconfiguration du bloc de défuzzification sur le FLC trafic. .	102
Tableau 4-28 L'apport de la reconfiguration du bloc de défuzzification sur le FLC global.	102

Introduction générale

Les avancées technologiques de ces dernières années ont permis la fabrication de nouvelles générations de circuits programmables dotés de structures reconfigurables telles que les FPGA (**F**ield **P**rogrammable **G**ate **A**rray). Ces circuits très denses, constitués de blocs logiques, permettent désormais l'implantation d'une variété de fonctions électroniques complexes. L'action flexible de reconfiguration matérielle a aidé d'une manière efficace à la mise au point de circuits et systèmes de contrôle très performants en mode codesign (logiciel/matériel).

Cette nouvelle approche où un système est subdivisé en soft et hard, à des proportions variables, est considérée comme solution incontournable aux appels et exigences incessants du marché des produits électroniques d'une part, et pour préserver, avec des coûts supportables, la dynamique de l'évolution des besoins de l'homme en termes de sécurité et de bien être d'autre part.

Dans nos jours, on peut rencontrer une multitude de systèmes complexes à large domaine de fonctionnement (robotique, aviation, procédés économiques ou écologiques,...etc.). La recherche de solutions optimales qui permettent de contrôler ces systèmes complexes est toujours posée. Malgré l'évolution prodigieuse de l'électronique et de l'informatique, le recours à la modélisation demeure prépondérant. Comme certains procédés sont toujours qualifiés de non modélisables, une solution à ce problème peut être envisagée à partir d'une représentation du comportement, un modèle flou par exemple, où les relations entre les entrées, les sorties et les états sont exprimées par des règles floues.

Aujourd'hui la logique floue est une technique de contrôle largement étendue à de multiples applications. L'utilisation de contrôleurs flous (**F**uzzy **L**ogic **C**ontrollers : **FLC**) pour capturer la connaissance des experts humains et sa traduction en stratégies de contrôle robustes, sans la nécessité d'un modèle mathématique, a conduit à une augmentation significative du nombre d'applications de contrôle utilisant des techniques d'inférences floues durant ces dernières décennies. Ces systèmes de commande floue peuvent être implémentés selon différentes approches, allant des solutions, soit totalement matérielles ou logicielles à des stratégies hybrides qui permettent d'atteindre des compromis adéquats entre la flexibilité de

programmation et la rapidité du matériel. Les réalisations hybrides nécessitent des processeurs pour les tâches d'exécution de logiciels et du matériel dédié pour la mise en œuvre des activités complexes indissociables, telles que le processus d'inférence floue.

Ce travail ajoute un nouveau point de vue aux efforts continuels dans la recherche concernant la conception de contrôleurs flous reconfigurables. Notre approche œuvrerait à rompre avec la logique floue classique en trois étapes distinctes du procédé de mise en œuvre, fuzzification, règles inférence et noyaux de défuzzification à travers la mise en jeu d'actions de reconfigurations jugées opportunes.

Notre but primordial est de développer des contrôleurs capables de s'adapter avec l'environnement où ils sont implantés grâce à des actions dites de reconfiguration sans l'appui et le recours à des routines de programmation exigeant généralement des espaces de travail étendus en mémoire et temps CPU d'une part et exigeant parfois des re-conceptions quasi totales pour prendre en charge certaines évolutions dictées le progrès et les besoins incessants des humains. Pour y contribuer à atteindre l'objectif visé ci-dessus, une stratégie de mise en œuvre d'un système flou reconfigurable où les différents blocs constitutifs reliés par des interfaces de communication permettent désormais, l'intégration du système flou sous forme d'IP (**I**ntelectual **p**roperty). L'implantation d'un système reconfigurable consiste à trouver un circuit programmable qui a des caractéristiques très avancées, capacité, vitesse, flexibilité et surtout possibilités de modification des aptitudes des ressources à se conformer à certaines exigences pendant le fonctionnement. La technologie moderne nous propose plusieurs produits en ASIC et FPGA pouvant répondre aux opérations de reconfiguration. Mais en pratique il est difficile d'implanter des contrôleurs flous hardware ayant la caractéristique de reconfiguration comme fonction dominante sans une exploitation de blocs de logique eux mêmes configurables dans un environnement d'interconnexions configurables à volonté aussi ; les SRAM-FPGA introduites avec multiples configurations ont été les précurseurs de la reconfiguration matérielle en général. Les avantages par rapport aux ASIC sont : coût de développement plus bas, dépenses en NRE (**N**on **R**ecurring **E**ngineering) moins importantes, corrections et modifications plus simples à réaliser, TTM '**T**ime-**T**o-**M**arket' plus court.

Les FPGA reconfigurables partiellement et dynamiquement offrent un nouvelle espace de conception avec une variété d'avantages : réduction du temps de configuration et sauvegarde de la mémoire en tant que fichiers partiels de reconfiguration (de flux binaires) qui sont plus petits que le contenu de la mémoire.

Le trait le plus saillant des FPGA est la reconfiguration matérielle c'est-à-dire que nous pouvons concevoir et modifier les circuits matériellement en tissu FPGA pour répondre aux besoins d'application spécifiques.

Depuis les années 80, le marché des FPGA a une croissance rapide avec différentes applications dans plusieurs domaines. Leurs avantages sont dus à la flexibilité qui résulte de leur nature programmable par rapport aux systèmes utilisant des circuits intégrés spécifiques (ASIC). Il y a une nouvelle évolution concernant les FPGA dite la reconfiguration dynamique partielle, qui peut être exploitée dans de nombreux domaines d'applications tels que la satisfaction des exigences de l'espace dans les petits systèmes portables.

Les progrès des FPGA ont effectivement étendu les capacités de la logique programmable au royaume des grands systèmes programmables.

Les circuits FPGA avancés tels que les VERTEX pro de XILINX intègrent des cœurs de processeurs de haute performance, ce qui permet de combiner les avantages de la reconfiguration du matériel de logique des FPGA avec la reprogrammation du logiciel des processeurs embarqués.

La méthodologie de conception du matériel et logiciel permet de partager les fonctions c'est à dire implémenter des fonctions dans le matériel pour le traitement à grande vitesse, tandis que d'autres sont mises en œuvre dans le logiciel pour une considération de haute flexibilité. La combinaison du matériel reconfigurable et logiciel programmable dans une puce FPGA, offre désormais une solution de SOPC (System On Chip) pour les systèmes à ressources et tailles limitées.

Tirant partie de la nature reconfigurable et de l'énorme capacité des FPGA avancés, nous développons un contrôleur à base de FPGA personnalisé de haute performance pour servir comme cerveau d'un système de ventilation d'un tunnel routier d'une part et pour améliorer la reconfiguration et répondre aux besoins de l'informatique embarquée d'autre part.

Rappelant que les dispositifs FPGA récents permettent des reconfigurations partielles simultanément avec le fonctionnement normal, appelées reconfigurations dynamiques partielles. Cette fonctionnalité permet d'utiliser une seule unité pour le traitement d'un très grand nombre d'opérations. Cette dernière permet d'utiliser des dispositifs plus petits, moins coûteux et avec une consommation de puissance plus faible, et évitant ainsi l'implantation d'unités de calcul à structures complexes.

Les efforts intellectuels investis dans la reconfiguration par la communauté scientifique et repris par l'industrie concourent tous à améliorer le bien être de l'humain mais certaines incidences liées à la sécurité doivent être surveillées pour éviter tout méfait ou entorse.

Ce travail est structuré en quatre chapitres :

Le premier chapitre est consacré à la description des principaux concepts des architectures reconfigurables et types de reconfiguration. Une classification détaillée de ces systèmes en termes de granularité, de structure et plans de reconfigurations au niveau système est détaillée. La reconfiguration dynamique partielle (**RDP**) utilisée dans notre application est aussi détaillée.

Dans le deuxième chapitre, la logique floue et la structure générale d'un contrôleur flou sont explicitées.

Dans le troisième chapitre, nous décrivons le langage VHDL et l'environnement d'exploitation Xilinx 13.3.

Le quatrième chapitre est dédié à la description de l'application développée. Un tunnel routier de 4.2 Km de longueur et 9 mètres de largeur est étudié. L'apport de la reconfiguration et les résultats obtenus sont analysés.

Une conclusion termine ce mémoire, situe la valeur du travail effectué et consigne les projections et perspectives à suivre pour améliorer cette contribution.

1

Les architectures reconfigurables

1.1 Introduction

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

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

1.2 Types de reconfiguration

1.2.1 Reconfiguration logique

Les architectures reconfigurables au niveau porte les plus répandues sont des architectures à base de FPGA (Figure 1-1). Il existe deux types de technologie de reconfiguration pour ces composants selon que la configuration est effectuée en utilisant des éléments Anti-Fuse ou des mémoires SRAM.

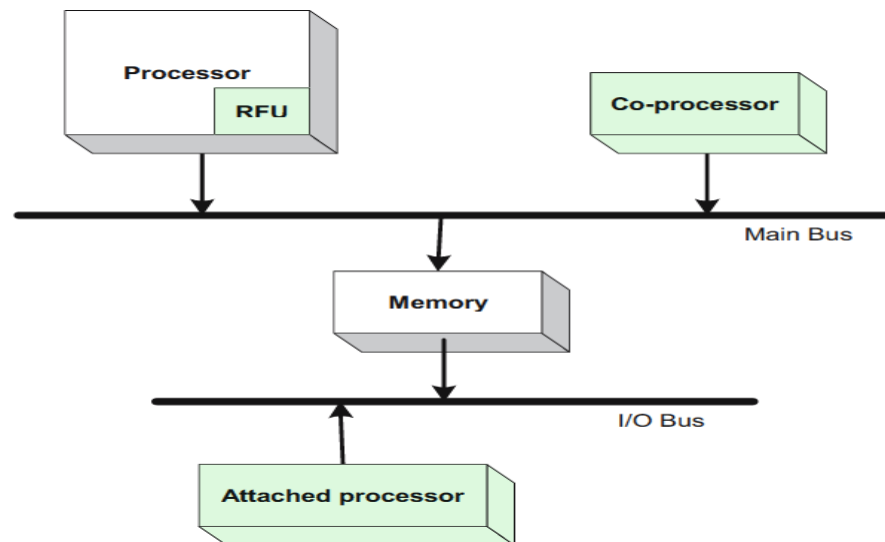


Figure 1-1 Niveaux de couplage de base de reconfiguration logique

La technologie Anti-Fuse utilise un fort courant électrique pour créer la connexion entre deux fils et elle est généralement non-reprogrammable. La technologie basée sur les SRAMs, permet de reconfigurer le composant FPGA en modifiant la fonctionnalité des blocs logique (CLBs) et le réseau d'interconnexion par le chargement d'un flot binaire dans les cellules mémoire SRAM [1] [2] [3].

1.2.2 Reconfiguration par 'jeu d'instructions'

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

1.2.3 Reconfiguration statique et dynamique

Il existe deux importantes approches de reconfiguration : la reconfiguration dynamique et la reconfiguration statique [4].

1.2.3.1 Reconfiguration statique

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

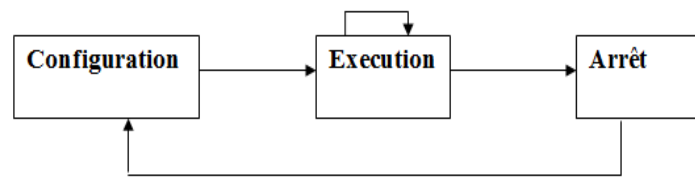


Figure 1-2 principe de la reconfiguration statique

1.2.3.2 Reconfiguration dynamique

La reconfiguration dynamique utilise un plan de l'affectation dynamique qui réattribue le hardware au cours d'exécution. C'est une technique avancée et elle est considérée comme une réalisation flexible du compromis espace/temps. Elle peut augmenter les performances du système en utilisant des circuits hautement optimisés qui sont adaptés et réadaptés, « chargés et déchargés » dynamiquement pendant le fonctionnement du système (Figure 1-3 principe de la reconfiguration dynamique). Dans cette reconfiguration la flexibilité du système est maintenue et la densité fonctionnelle du hardware est augmentée.

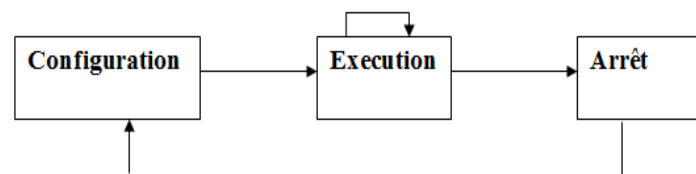


Figure 1-3 principe de la reconfiguration dynamique

1.3 Classification des architectures reconfigurables

Les architectures reconfigurables peuvent être classées selon différents paramètres.

Dans cette section nous distinguons les classifications architecturales suivantes [5] :

1.3.1 Classification selon la granularité

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

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

Une faible granularité procure une plus grande flexibilité d'adaptation du matériel à la structure de calcul souhaitée, mais elle entraîne un temps de configuration important. De plus, le grand nombre d'interconnexions entre les unités rajoute un délai conséquent à l'exécution du module implanté (malgré l'existence de structures permettant de réduire ce délai, comme les chaînes de propagation de retenue que l'on peut trouver maintenant sur quelques FPGA) [6][7].

1.3.2 Classification selon le schéma de reconfiguration

1.3.2.1 Architecture statiquement reconfigurable

Les architectures reconfigurables traditionnelles sont statiquement reconfigurables (**Compile-Time Reconfiguration** : CTR), ou les ressources de reconfiguration sont lancées au démarrage de l'exécution et restent inchangées durant l'application. Afin de reconfigurer ce type d'architecture, le système doit être arrêté pendant la reconfiguration, ensuite relancé avec la nouvelle configuration.

Les architectures FPGA traditionnelles ont été programmées à l'origine de façon sérielle avec un seul contexte, ou seulement une configuration peut être chargée à la fois. Ce type de FPGAs est programmé en utilisant un flux sériel de données, il nécessite la reconfiguration totale pour tout changement.

1.3.2.2 Architecture dynamiquement reconfigurable

Les architectures dynamiquement reconfigurables (**Run-Time Reconfiguration : RTR**) permettent de procéder à la reconfiguration et à l'exécution en même temps. Les différentes reconfigurations dynamiques sont représentées sur la Figure 1-4 et elles sont discutées dans les prochains paragraphes.

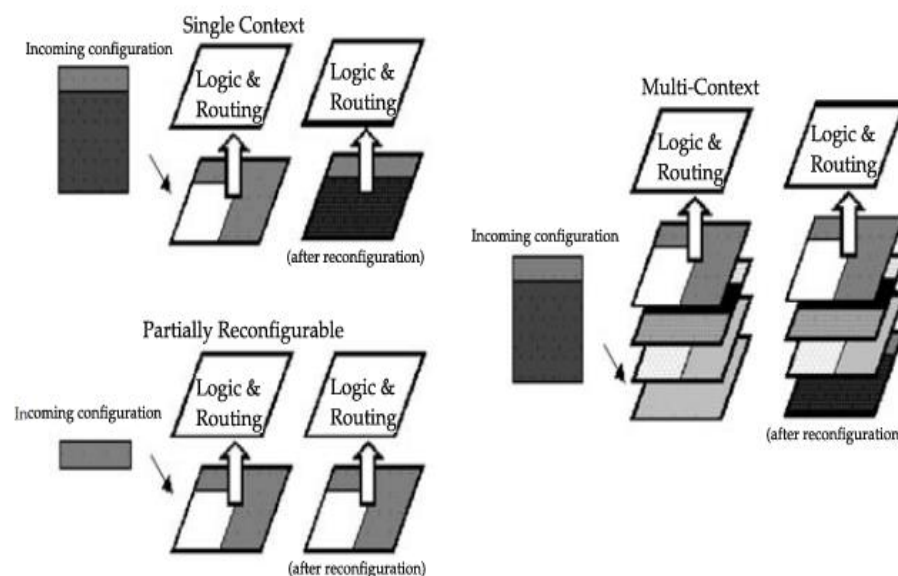


Figure 1-4 Les différents modèles de reconfiguration dynamique

1.3.2.2.1 La reconfiguration dynamique à contexte unique

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

1.3.2.2.2 La reconfiguration dynamique multi-contexte

L'architecture reconfigurable à contextes multiples possède plusieurs plans de configuration. Un plan de configuration unique peut être activé à un certain moment et le passage à un autre plan de configuration unique peut être très rapide (de l'ordre de quelques nanosecondes). L'architecture consiste à stocker un nombre de configurations dans des buffers (généralement en mémoire locale du composant) et de changer de contexte pendant l'exécution en un temps assez faible par rapport à un chargement de contexte à partir d'une mémoire externe. L'architecture 'Chamelon' par exemple, possède un cache permettant de stocker une configuration et de reconfigurer le composant à partir de ce cache en un seul cycle [8].

1.3.2.2.3 Les architectures reconfigurables partiellement [6]

Dans les architectures à reconfiguration partielle, la couche de programmation sous-jacente opère comme une RAM en utilisant des adresses pour spécifier l'emplacement visé reconfiguré continu à suivre ou à bloquer l'exécution courante :

- Configuration partielle fixe.
- Configuration partielle active.

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

1.4 Classification selon le mode traitement

Le type de traitement peut être considéré comme une caractéristique fondamentale du système, puisque il influence les autres caractéristiques des noyaux de ce système. D'après ce raisonnement, une classification générale des architectures reconfigurables basées sur le type de traitement comme principal critère produit [8].

1.4.1 Architecture orientée chemin de données

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

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

1.4.2 Architecture orientée traitement logique

Ces architectures sont destinées à l'origine pour 'mapping' de gros blocs de logique combinatoire moyennant le traitement logique et fusionnant des chemins de données. Les éléments de traitement logique sont utilisés pour implémenter des éléments arithmétiques, les architectures orientées traitement de données devrait valider le 'mapping' de fonctions arithmétiques avec la propagation de retenues.

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

1.4.3 Architecture orientée mémoire

Les architectures orientées mémoire visent les noyaux de traitement qui nécessitent des ressources de stockage. Ces ressources sont, en général, nécessaires pour deux raisons :

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

- Les besoins de l'implémentation : Les éléments mémoires dépendent de méthode d'implémentation.

Habituellement, l'intégration et la mise en place des noyaux orientés mémoires exigent un flot de données et de logique câblées. Par conséquent, pour garantir l'efficacité d'ordonnancement de tels noyaux, ces architectures doivent être orientées des fonctions multiples ou à usage général.

1.5 Conception des socs reconfigurables

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

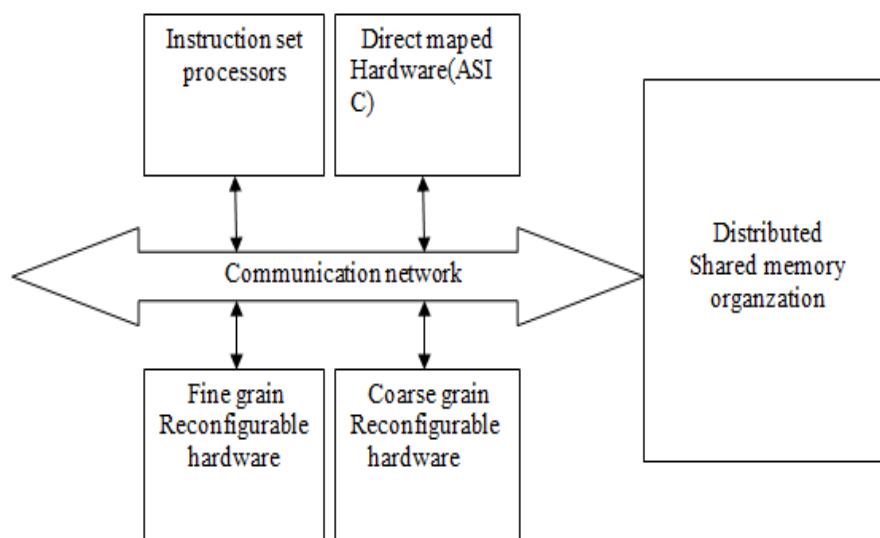


Figure 1-5une vue globale des SOC reconfigurables

En particulier, la présence d'unité reconfigurables, ou ARD (accélérateur reconfigurable dynamiquement), permet d'adapter l'architecture à différents

traitements. Un accélérateur reconfigurable au sein de ce type de plateforme permet d'apporter un degré important de flexibilité à l'architecture. [1]

1.5.1 Les exigences au niveau système pour les SOC's reconfigurables :

L'introduction des ressources reconfigurables dans les systèmes sur puce a créé les besoins des modifications et de prolongement des flots de conception classiques, principalement dans les niveaux d'abstraction plus élevés où les décisions les plus importantes sont prises. Habituellement, il est convenu que le flot de conception générique des systèmes intégrés nécessite les phases clés suivantes :

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

La manière avec laquelle la présence de ressources reconfigurables affecte les principales étapes du flot de conception et exige des besoins supplémentaires fortement corrélés sont discutés ci-dessous.

1.5.1.1 Spécification du système

Dans cette phase, les besoins, les restrictions et les spécifications sont rassemblés sans les ressources reconfigurables, mais un effort supplémentaire doit être consacré pour l'identification des parties qui sont candidates à être implantées avec du matériel reconfigurable. [1]

Les effets des blocs reconfigurables sur la surface, la vitesse et la puissance absorbée devraient être complètement appréhendés avant toute utilisation. [2]

1.5.1.2 Partitionnement et mapping hardware/software

Durant cette phase, une nouvelle dimension va être ajoutée aux systèmes. La partie du système visé qui sera réalisée sur un matériel reconfigurable devrait être identifiée, et les règles nécessaires pour y parvenir sont listées ci-dessous :

- Si l'application a besoin d'accélérateurs matériels, qui ne sont pas utilisés en même temps ; alors un bloc reconfigurable dynamiquement fournira une solution optimale.
- Si éventuellement, les spécifications d'une ou plusieurs parties de l'application changent, ces parties seront implantées par un hardware reconfigurable.
- Si de nouvelles générations de l'application sont envisageables pour mettre à jour le système, alors la partie concernée sera reconfigurable.

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

1.5.1.3 Conception architecturale

Beaucoup de questions sont liées au choix de l'architecture, qui concerne la technologie de reconfiguration utilisée et la partie du système qui doit être reconfigurable. La méthode la plus utilisée est la simulation. Les opérations de partitionnement, de mapping et de l'ordonnancement sont effectuées manuellement par le concepteur ; alors que les résultats et le rendement sont produits par simulation. Ainsi, cette approche est peut être qualifiée d'efficace car à ce stade, il n'existe toujours pas d'outils capables d'intégrer une partie reconfigurable sur une plateforme hardware [9].

Avant de choisir la technologie reconfigurable adéquate pour la modélisation du système, il est nécessaire d'identifier certains paramètres : (a) la taille et volume des blocs reconfigurables, (b) la taille des contextes de reconfiguration, (c) le temps de

reconfiguration et la possibilité de reconfiguration partielle, sans oublier la caractéristique liée à la puissance consommée. Ces paramètres donneront au concepteur une idée sur le comportement des différentes technologies et leurs impacts et affectations sur les performances globales du système.

Les résultats obtenus qui vérifient si modèle arrive ou non à satisfaire les performances du système sont :

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

1.5.2 Flot de conception de SoCs reconfigurables

Dans cette section, nous fournirons la structure générale d'un des flots de conception utilisé dans la modélisation des SoCs reconfigurables. Ce flot a pour objectif d'améliorer le processus de conception d'un SoCs afin d'utiliser les outils disponibles d'une manière optimale [1].

L'idée principale de ce flot de conception est d'identifier les parties du système où l'ajout de technologies reconfigurables a le plus grand effet. C'est très important, car il n'y a toujours pas d'outils commerciaux ou de méthodologies pour supporter la reconfiguration. Le flot de conception est divisé en trois parties comme montré sur la Figure 1-6.

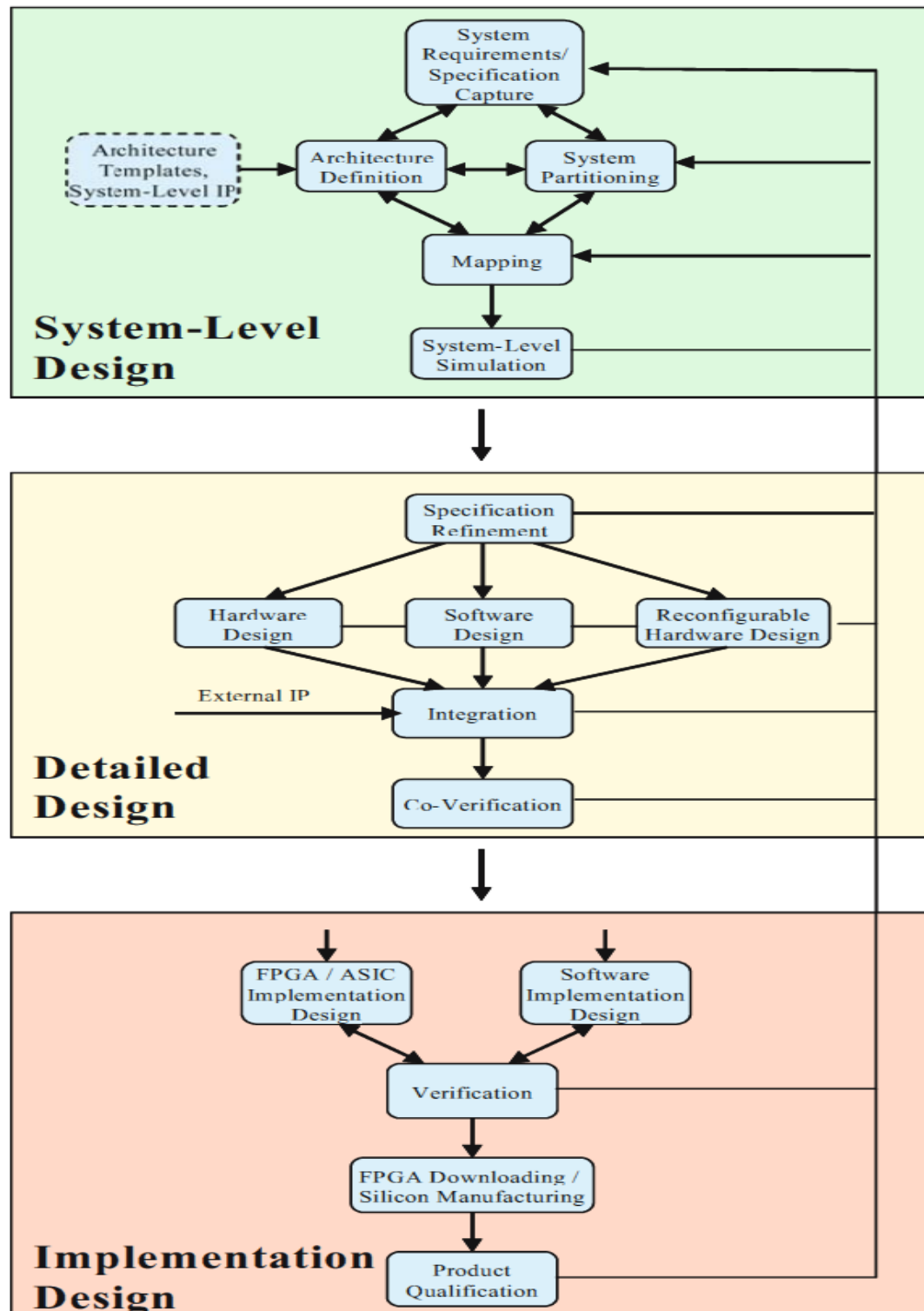


Figure 1-6 flot de conception

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

1.5.2.1 Conception au niveau système (System Level Design-SLD)

Les objectifs principaux de cette phase sont :

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

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

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

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

1.5.2.2 Conception détaillée (Detailed Design-'DD')

Dans cette phase, les spécifications sont affinées et la vérification est planifiée en fonction de la technologie ciblée. Après ceci, le partitionnement individuel des blocs hardware, software et reconfigurables est conçu et ensuite suivi par l'étape vérification.

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

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

1.5.2.3 Implémentation (Implémentation Design -'ID') [1]

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

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

1.6 Exemple de conception d'un système reconfigurable :

Bien qu'il existe une multitude de différents types de modifications qui peuvent être apportées à la conception de l'FPGA, dans cette application qui est un additionneur nous devons d'abord changer la programmation LUT à l'aide de l'Éditeur de l'FPGA. Bien qu'il soit possible de changer les informations de routage, ce n'est pas recommandé en raison de la possibilité de discordance interne lors de la reconfiguration. Nous devons changer l'équation LUT afin de faire un nouveau fichier `adder_test.ncd` pour le circuit reconfigurable. L'élément logique plus petit pouvant être sélectionné est la cellule.

Les attributs sont modifiés en utilisant les équations de LUT. Une fois ces modifications ont été apportées, le nouveau `adder_test.ncd` est utilisé avec l'application `bitgen` pour générer un flux de bits en fonction de différence (difference-based bitstream)

1.6.1 Création d'un flux de bits de reconfiguration en fonction de différence

Un flux de bits de reconfiguration partielle en fonction de différence peut être créé avec l'utilitaire `bitgen` utilisant l'option `commutateur`. Ce commutateur produit un flux binaire qui contient uniquement les différences entre le fichier d'entrée. `ncd` et le fichier original.

1.6.2 Utilisation de flux binaires et programmation du FPGA

Les supports de programmation JTAG série prend en charge les options de la reconfiguration partielle. Parce que les flux binaires partiels sont indiscernables de flux binaires complets, les utilisateurs finaux doit être prudent pour séquencer correctement l'application de ces flux binaires partiels aux dispositifs cibles. [5]

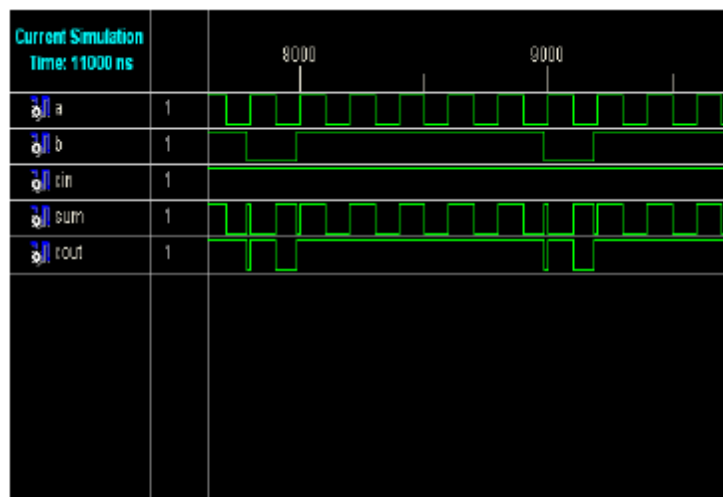


Figure 1-7 simulation d'un additionneur complet

Initialement, le code de l'additionneur complet a été écrit en VHDL et simulé comme dans la Figure 1-7 de même le code du compteur a été écrite et simulé et la Figure 1-8 montre la simulation du compteur. Le flot de bits est généré et lors de l'exécution du programme de l'additionneur le train de bits soustracteur est chargé dans le FPGA et maintenant la sortie de soustracteur est obtenue.

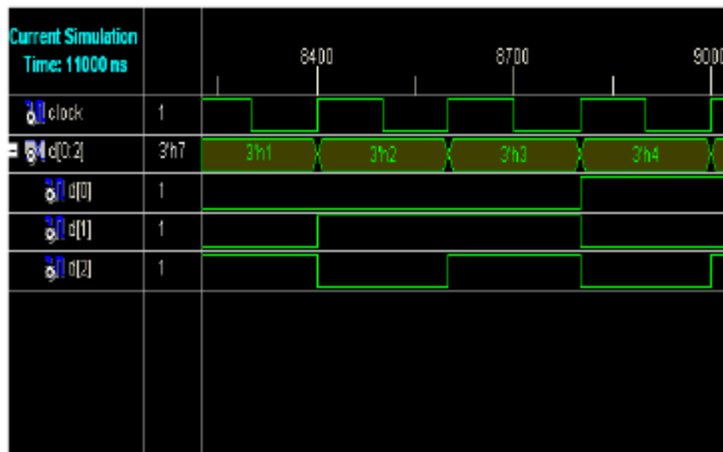


Figure 1-8 simulation du compteur

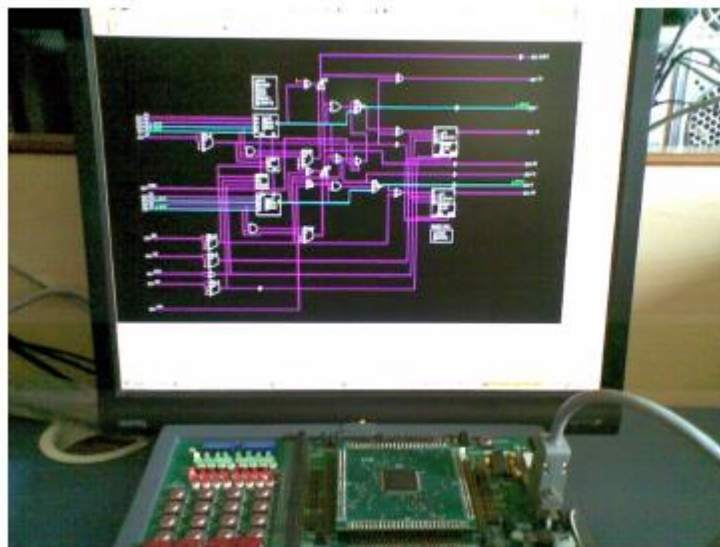


Figure 1-9 implantation sur FPGA 2S30

1.7 Introduction à la reconfiguration d'un contrôleur flou :

Maintenant on examine les impacts des techniques de reconfiguration sur la structure d'un contrôleur flou implanté sur FPGA.

La reconfiguration partielle sur un contrôleur flou développé en VHDL sur une plateforme de Xilinx serait extensivement étudiée :

- Indépendamment de toute application exploitant le contrôleur

- Reconfiguration en liaison avec l'application à contrôler

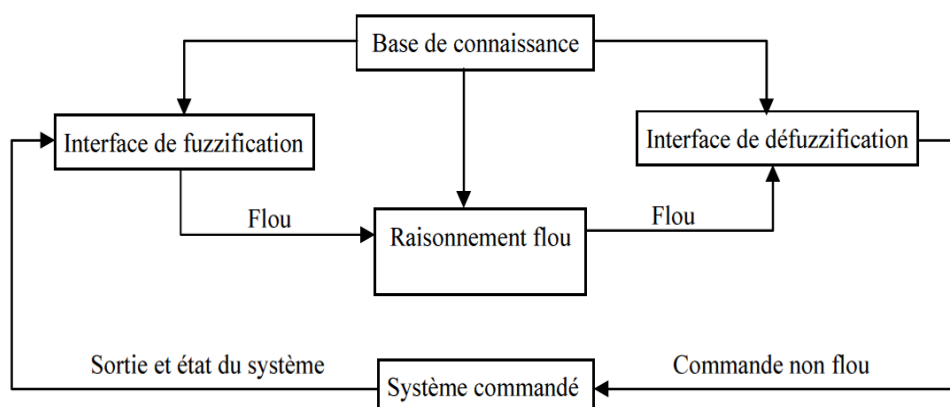


Figure 1-10 Bloc diagramme du contrôleur flou

Comme nous disposons au laboratoire de plusieurs versions de FLC, notre étude est concentrée à trouver la méthode appropriée qui fait appelle à la configuration sans remettre en cause les acquis obtenus des différentes applications, c'est-à-dire l'exploitation de FLC dans le test, la ventilation de tunnel, stabilité et contrôle d'échauffement dans les Socs etc.

Vu que notre objectif est de montrer que grâce à la configuration :

- On annule le recours à des re-conceptions complètes (re-design),
- On évite le recours à de circuits FPGA de plus en plus performants,
- On évite aussi le recours à des partitionnements multiples induisant généralement des optimisations difficiles et des coûts élevés,

Les atouts dont nous disposons sont largement favorisés par la flexibilité grandissante des conceptions sur SRAM-FPGA.

Nous rappelons que les proportions de surface sur FPGA sont majoritairement en faveur des blocs de defuzzification. En effet, la surface du bloc FLC defuzzification occupe jusqu'à 90% du circuit FPGA implantant le FLC global. [31]

En prenant les résultats de synthèse du module de la défuzzification du FLC_OP réalisé au laboratoire de recherche et illustrés sur le tableau ci-

dessous. On remarque que le bloc de défuzzification occupe une grande surface par rapport aux deux autres blocs (fuzzification et inférence), visible à travers le nombre d'IOB (48%).

Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slices	159	4656		3%
Number of Slice Flip Flops	56	9312		0%
Number of 4 input LUTs	281	9312		3%
Number of bonded IOBs	32	66		48%
Number of GCLKs	1	24		4%

Table 1-1 synthèse du module de la défuzzification du FLC_OP

Alors que la fuzzification et le mécanisme d'inférence occupent uniquement une petite surface, comme le montre le tableau ci-dessous. [31]

Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slices	7	4656		0%
Number of Slice Flip Flops	12	9312		0%
Number of 4 input LUTs	2	9312		0%
Number of bonded IOBs	21	66		31%
Number of BRAMs	1	20		5%
Number of GCLKs	1	24		4%

Table 1-2 synthèse du module de la fuzzification du FLC_OP

Connaissant que la reconfiguration partielle doit s'articuler autour d'implantation et changement à volonté que d'une fine partie ou bloc bien spécifique du système sur FPGA, évitant les ressources et signaux globaux. [10]

Cette dernière mesure et contrainte de reconfiguration nous pousse, dans un premier temps, à ne considérer que la reconfiguration agissant sur les blocs de fuzzification et d'inférence (individuellement et simultanément).

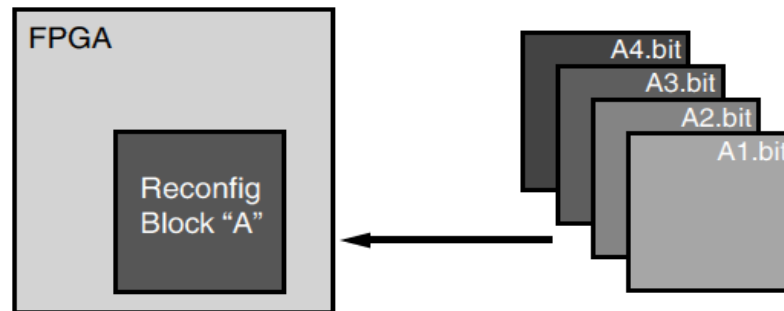


Figure 1-11 Principe de la reconfiguration partielle

A titre indicatif, comme on le voit sur la Figure 1-11, la fonction mise en œuvre dans le bloc Reconfig A est modifiée en téléchargeant un de plusieurs fichiers partiels de bits, A1.bit, A2.bit, A3.bit ou A4.bit. La logique de la conception de FPGA est divisée en deux types différents : la partie reconfigurable et la partie statique. La zone grise du bloc FPGA représente la partie statique et la partie du bloc marqué Bloc Reconfig «A» représente la partie reconfigurable. La logique de fonctionnement statique est toujours préservée et non affectée par le chargement d'un fichier de bit partiel. Alors que le contenu de la partie reconfigurable est remplacé durant l'opération de reconfiguration par le fichier de bit partiel approprié [2].

Il ya beaucoup de raisons pour lesquelles la capacité de matériel de multiplexage dans le temps de manière dynamique sur un dispositif FPGA unique est avantageuse.

Il s'agit notamment de :

- Réduire la taille du dispositif FPGA nécessaires pour mettre en œuvre une fonction donnée, avec des réductions conséquentes des coûts et de la consommation d'énergie.
- Fournir la flexibilité dans le choix des algorithmes et des protocoles disponibles pour une application.
- Permettre aux nouvelles techniques en matière de sécurité de la conception. Amélioration de la tolérance aux pannes FPGA.
- Accélérer la reconfiguration informatique (reconfigurable computing).

On estime comme plus de la réduction de la taille, du poids, des puissance mises en jeu et des coûts induits, la reconfiguration partielle permet aussi fournir de nouvelles versions de conceptions plus performantes sur le même FPGA, initialement impossibles à mettre en œuvre sans cette reconfiguration [2].

1.7.1 Etude de l'apport de la reconfiguration dans la fuzzification et l'inférence :

L'étendue des valeurs d'entrées sont modifiables et peut être remplacées par sélection de LUT (Look Up Table).

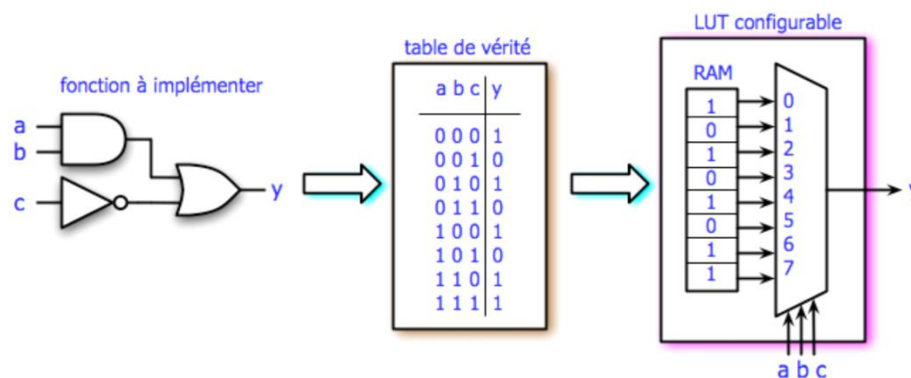


Figure 1-12 Fonctionnement de LUT

La partie de sélection des règles actives peut être aussi reconfigurée moyennant la sélection du nombre de règles actives à la fois.

- 1 règle : pas de flou, propre au système linéaire la réponse peut être prédite nous sommes devant un système linéaire.
- 2 règles : évolution lente sans perturbation majeur (évolution entre deux états).
- 4 règles : les travaux FLC sont basés sur cette considération et l'expertise est considérée imposante et correcte.
- >> 4 règles : une sur charge et un manque d'expertise sont atteints alors apport très négligeable par rapport au raisonnement multiple et additif.

Ces résultats devraient être confirmés par l'implantation pratique du système.

NB : ensembles flous types 2 et autres ne sont pas considérés.

NB : Nous avons étudié aussi la possibilité de produire des configurations des configurations par génération indépendantes du contexte c'est-à-dire, en faisant évoluer une machine d'état et selon l'état présent, le contexte se chargera de produire la configuration par une action sur celle déjà préprogrammée. Mais Elle est très difficile à mettre au point car elle nécessite la connaissance de toutes les évolutions possibles.

1.7.2 Etude de l'apport de la reconfiguration dans la défuzzification :

Plusieurs méthodes existent pour la défuzzification, les plus couramment utilisées sont :

- Méthode du centre de gravité.
- Méthode des hauteurs pondérées.
- Méthode de moyenne des maximums.

Dans notre étude, nous avons constaté que toutes les approches de defuzzification se rejoignent autour de caractéristiques similaires : surface, temps, stabilité, alors notre contribution s'est penché sur la configuration au sein d'une même technique, celle du centre de gravité dans notre cas.

Comme cette méthode repose sur le calcul du centre de gravité la reconfiguration sera très délicate. [30]

Pour mieux cerner le problème et vérifier notre contribution nous avons considéré le système de ventilation développé au laboratoire ou nous avons introduit la reconfiguration.

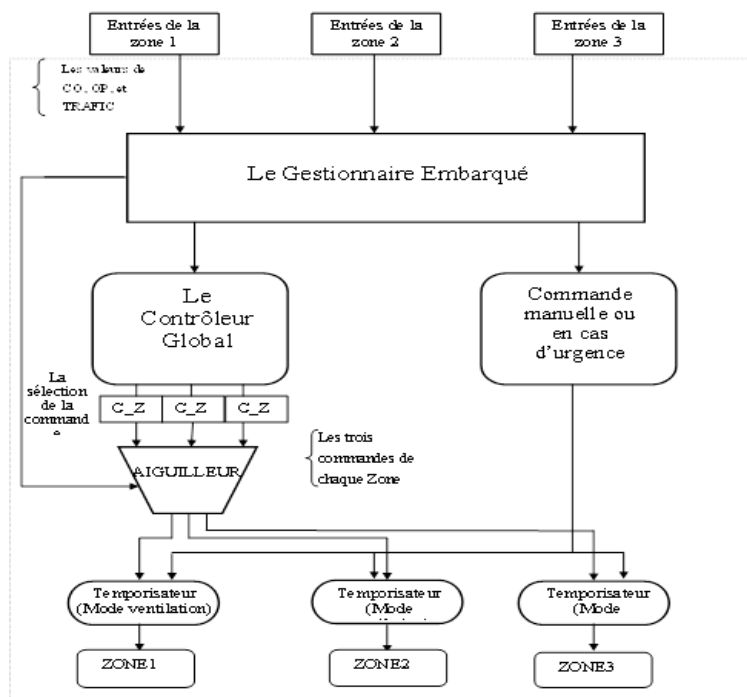


Figure 1-13 Schéma bloc du contrôleur global de ventilation

1.8 Conclusion

Nous avons présenté dans ce chapitre une synthèse générale sur les architectures reconfigurables, en particulier les architectures reconfigurables dynamiquement, et leurs types communément utilisés dans la littérature, ainsi que leurs caractéristiques de classification. Nous avons montré les modifications apportées au flot de conception des SoCs pour leur intégrer des modules reconfigurables. La réadaptation de l'architecture globale du SoC est nécessaire afin de permettre un réajustement des modules ajoutés avec les IPs déjà présents sur le système.

L'utilisation des architectures reconfigurables se développe rapidement dans les domaines des télécommunications, et de contrôle et particulièrement dans les systèmes de formes évolutives tels que les applications industrielles et de grand public à base de contrôleurs flous.

Dans les prochains chapitres, une étude sur l'application des techniques reconfigurable aux systèmes flous sera présentée.

Logique floue

2.1 Introduction

La logique floue, ou plus généralement le traitement des incertitudes, est l'une des classes de l'intelligence artificielle. Elle a pour objet l'étude la représentation des connaissances imprécises et le raisonnement approché. Elle a été connue pour la première fois comme une branche mathématique complémentaire à la théorie de la logique classique, puis elle a trouvé sa place parmi les techniques de commande basées sur l'intelligence artificielle. La logique floue ne remplace pas nécessairement les systèmes classiques de régulation. Elle est complémentaire et utilisée particulièrement lorsqu'on ne dispose pas de modèle mathématique précis du processus à commander, ou lorsque ce dernier présente des non linéarités ou imprécisions.

De plus, l'intérêt de la logique floue réside dans sa capacité de traiter l'imprécis, l'incertain et le vague. Ainsi, le succès de la commande par la logique floue trouve en grande partie son origine dans sa capacité à traduire une stratégie de contrôle d'un opérateur qualifié en ensemble de règles linguistiques facilement interprétables.

Actuellement la logique floue a trouvé son application dans plusieurs domaines, telles que, la gestion, la médecine et la commande des systèmes. Dans ce dernier domaine d'application, on peut citer la commande des bras robotiques, des

machines outils, des réacteurs chimiques, des véhicules, des appareils électroménagers ...etc.

2.2 La logique floue

La logique floue est née de la constatation que la plupart des phénomènes ne peuvent être représentés à l'aide de variables booléennes qui ne peuvent prendre que deux valeurs (1 ou 0).

Exemple : pour considérer une eau à 18 °C comme étant chaude ou froide, n'est-elle pas ni vraiment chaude, ni vraiment tiède ?

Pour répondre à ce type de question, la logique floue considère la notion d'appartenance d'un objet à un ensemble, non plus comme une fonction booléenne mais comme une fonction qui peut prendre toutes les valeurs entre 0 et 1.

Le terme d'ensemble flou apparaît pour la première fois en 1965 lorsque le professeur LOTFI A.ZADEH [1965] de l'université de Berkeley aux USA, publie un article intitulé « Ensembles flous ». Il a réalisé depuis, de nombreuses avancées théoriques majeures dans le domaine et a été rapidement accompagné par de nombreux chercheurs développant des travaux théoriques.

En 1974 M. Mamdani expérimentait la théorie énoncée par Zadeh sur une chaudière à vapeur, matériel dont on connaît la complexité, introduisant ainsi la commande floue dans la régulation d'un processus industriel, plusieurs applications ont alors vu le jour en Europe (1980: développement des études sur les ensembles floue), pour des systèmes parfois très complexes. [12]

A partir de 1985, un grand développement des applications des régulateurs flous a été fait par les chercheurs Japonais.

2.3 Théorie des ensembles flous

2.3.1 Notion d'appartenance partielle

Dans la théorie des ensembles, un élément appartient ou n'appartient pas à un ensemble.

La notion d'ensemble est à l'origine de nombreuses théories mathématiques. Cette notion essentielle ne permet cependant pas de rendre compte des situations pourtant simples et rencontrées fréquemment. Par exemple : parmi des fruits, il est

facile de définir l'ensemble des pommes ; par contre, il sera plus difficile de définir l'ensemble des pommes mûres.

On conçoit bien que la pomme mûrit progressivement, la notion de pomme mûre est donc graduelle. C'est pour prendre en compte de telles situations qu'a été créée la notion d'ensemble flou. La théorie des ensembles flous repose sur la notion d'appartenance

Partielle : chaque élément appartient partiellement ou graduellement aux ensembles flous qui ont été définis. [13]

2.3.2 Variable linguistique

La logique floue est aussi appelée « logique linguistique » car ses valeurs de vérité sont des mots du langage courant « plutôt vrais, presque faux, ... ». cette valeur de vérité dépend du prédicat, en clair la logique floue n'est surtout pas une logique imprécise mais, une logique qui s'adapte à l'être humain en laissant une place entre la certitude du vrai et la certitude du faux. [13]

Chaque variable linguistique est caractérisée par un ensemble tel que :

$[x, \mu_A(x), U, G]$

- x est le mon de la variable.
- $\mu_A(x)$ est l'ensemble des valeurs linguistiques que peut prendre x (la fonction d'appartenance).
- U l'univers de discours associé avec la valeur de base.
- G est la règle syntaxique pour générer les valeurs linguistique de x .

2.3.3 Fonctions d'appartenance

Un ensemble flou est défini par sa « fonction d'appartenance », qui correspond à la notion de « fonction caractéristique » en logique classique. Elle permet de mesurer le degré d'appartenance d'un élément à l'ensemble flou, Les fonctions d'appartenance peuvent avoir différentes formes : triangulaires, trapézoïdales, Monotones (croissantes ou décroissantes), ou en forme de cloche (Gaussiennes). [14]

Généralement, les plus souvent utilisées sont les fonctions de forme trapézoïdale ou triangulaire.

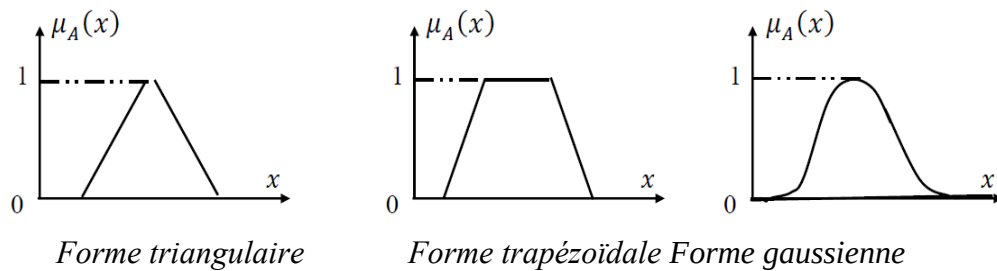


Figure 2-1 Exemples de fonctions d'appartenance

2.3.4 Opérateurs de la logique floue

Une fois les ensembles flous définis, des opérations mathématiques concernant ce type d'ensembles ont été développées. Les opérateurs mathématiques élaborés ressemblent beaucoup à ceux reliés à la théorie des ensembles conventionnels. Les opérateurs de l'intersection, l'union et la complémentation sont traduits respectivement, par les opérateurs " *ET*, *OU* et *NON* ".

Soit A et B deux ensembles flous, dont les fonctions d'appartenance sont $\mu_A(x)$, $\mu_B(y)$ respectivement. Cependant, les plus répandus sont ceux dits « de Zadeh ».

2.3.4.1 Intersection

L'opérateur logique correspondant à l'intersection d'ensembles est le ET. Le degré de vérité de la proposition « A ET B » est le minimum des degrés de vérité de A et de B . $\mu(A \text{ ET } B) = \min(\mu(A), \mu(B))$

2.3.4.2 Union

L'opérateur logique correspondant à l'union d'ensembles est le OU. Le degré de vérité de la proposition « A OU B » est le maximum des degrés de vérité de A et de B . $\mu(A \text{ OU } B) = \max(\mu(A), \mu(B))$

2.3.4.3 Complément

L'opérateur logique correspondant au complément d'un ensemble est la négation.

$$\mu(\text{Non } A) = 1 - \mu(A)$$

2.3.5 Règles floues

La logique floue a pour objectif de formaliser et de mettre en œuvre la façon de raisonner d'un être humain. Pour cela, elle peut être classée dans le domaine de l'intelligence artificielle. L'outil le plus utilisé dans les applications de la logique

floue est la base de règles floues. Une base de règles floues est composée de règles qui sont généralement utilisées en parallèle, mais peuvent également être enchaînées dans certaines applications. Une règle est du type :

SI « prédicat » **ALORS** « conclusion ».

Par exemple : « Si température élevée et pression forte ALORS ventilation forte et soupape grande ouverte ».

Les bases de règles floues, tout comme les systèmes experts classiques, fonctionnent en s'appuyant sur une base de connaissance issue de l'expertise humaine.

Un prédicat (encore appelé prémisses ou condition) est une combinaison de propositions par des opérateurs ET, OU, NON.

Les propositions « température élevée » et « pression forte » de l'exemple précédent sont combinées par l'opérateur ET pour former le prédicat de la règle. [12]

2.3.6 La logique floue type-1 et type-2

La logique floue permet la représentation et le traitement de connaissances imprécises ou approximatives. Le nombre d'applications basées sur la théorie de la logique floue est augmenté considérablement ces dernières années car cette logique est exprimée usuellement par des règles linguistiques de la forme Si –Alors. Elle est utilisée pour résoudre les problèmes de décision en contrôle ou pour décrire le comportement dynamique d'un système inconnu ou mal défini. La première génération de la logique floue est appelée la logique floue type-1. Une nouvelle logique floue appelée logique floue type-2 est introduite pour mieux représenter l'imprécision et l'incertitude. [15]

2.3.7 Généralités sur la logique floue type-1

Les systèmes flous permettent d'exploiter et de manipuler efficacement les informations linguistiques émanant de l'expert humain grâce à un fondement théorique important. En plus, le système mis en œuvre peut être intégré facilement dans une boucle de commande ou d'identification. La structure de base d'un système flou se divise en trois parties principales comme le montre la Figure 2-2. [12]

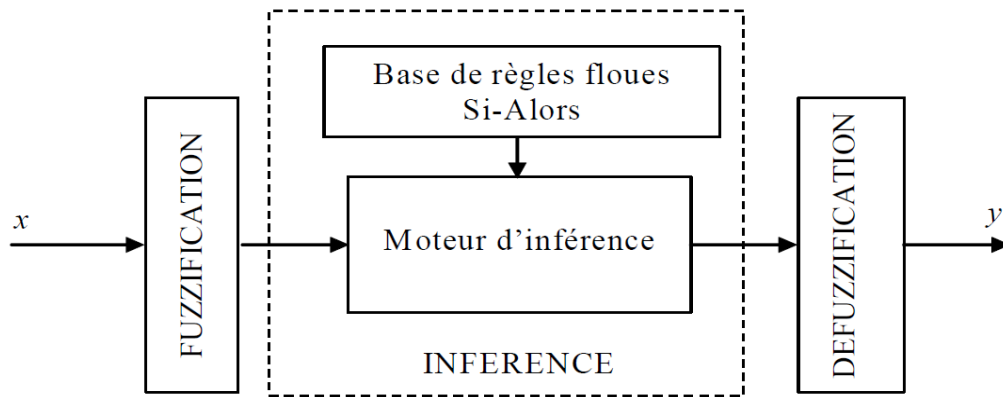


Figure 2-2 Système flou type-1

2.3.7.1 Fuzzification

La fuzzification proprement dite consiste à définir des fonctions d'appartenances pour les différentes variables linguistiques. Le but est la conversion d'une grandeur physique en une grandeur linguistique. Il s'agit d'une projection de la variable physique sur les ensembles flous caractérisant cette variable. Cette opération permet d'avoir une mesure précise sur le degré d'appartenance de la variable d'entrée à chaque ensemble flou, afin de garantir la couverture uniforme de l'univers de discours et d'éviter les indécisions ou les confusions entre les règles.

[16]

2.3.7.2 Inférence

Les connaissances de l'opérateur humain sur un processus donné sont transformées en un ensemble de règles floues de la forme suivante :

Si « prémisses » **Alors** « conclusion ».

Où la prémisses est un ensemble de conditions liées entre elles par des opérateurs flous.

La partie conclusion peut être une description d'évolution dans le cas d'identification ou une action dans le cas de commande. Les opérateurs flous utilisés dans la partie prémisses sont les conjonctions : "ET", "OU".

L'interprétation de ces conjonctions dépend directement du type du moteur d'inférence adopté. La relation entre la prémisses et la conclusion "Alors" peut être traduite par le produit ou le minimum. [15]

Le Table 2-1 Exemple de matrice d'inférence représente un exemple de matrice d'inférence pour les deux variables linguistique d'entrée **teta** et **Dteta**.

La case en bleu du Table 2-1 Exemple de matrice d'inférence représente la règle :

Si **teta** = N et **Dteta** = P alors **U** = Z.

Lorsque toutes les cases de la matrice d'inférence sont remplies, il s'agit de règles d'inférence complètes. Cependant, cette représentation graphique devient plus complexe, s'il y a plus de trois variables linguistiques d'entrée.

F		teta		
		N	Z	P
Dteta	N	PG	P	Z
	Z	P	Z	N
	P	Z	N	NG

Table 2-1 Exemple de matrice d'inférence

2.3.7.3 Défuzzification

La commande nécessite un signal précis, il faudra donc transformer la fonction d'appartenance résultante obtenue à la sortie du moteur d'inférence en une valeur précise. Cette opération est appelée défuzzification. Parmi les méthodes utilisées dans la littérature on peut citer : [15]

- Le centre de gravité.
- La méthode de la hauteur.
- La méthode de la hauteur modifiée.
- La méthode de la valeur maximum.
- La méthode de la moyenne des centres.

2.3.8 Généralités sur la logique floue type-2

Lorsque nous ne pouvons pas déterminer le degré d'appartenance d'un élément à un ensemble par 0 ou 1, on utilise les ensembles flous type-1. De même, Lorsque nous ne pouvons pas déterminer les fonctions d'appartenance floues par des nombres réels dans $[0, 1]$, on utilise alors les ensembles flous type-2. Les ensembles flous type-1 peuvent être considérés comme une approximation du premier ordre de l'incertitude, alors que les ensembles flous type-2 seront considérés comme

approximation du deuxième ordre.

Le concept des ensembles flous type-2 a été introduit par Zadeh [Zad, 75], comme extension du concept de l'ensemble flou ordinaire appelé ensemble flou type-1. Un ensemble flou type-2 est caractérisé par une fonction d'appartenance floue, c'est à dire, la valeur d'appartenance (degré d'appartenance) de chaque élément de l'ensemble est un ensemble flou dans $[0, 1]$. De tels ensembles peuvent être utilisés dans les situations où nous avons de l'incertitude sur les valeurs d'appartenance elles mêmes. L'incertitude peut être soit dans la forme de la fonction d'appartenance ou dans l'un de ses paramètres.

La structure d'un système flou type-2 est représentée dans la Figure 2-3 Nous allons supposer dans cette section que les fonctions d'appartenance des prémisses et des conséquences sont de type-2. [10]

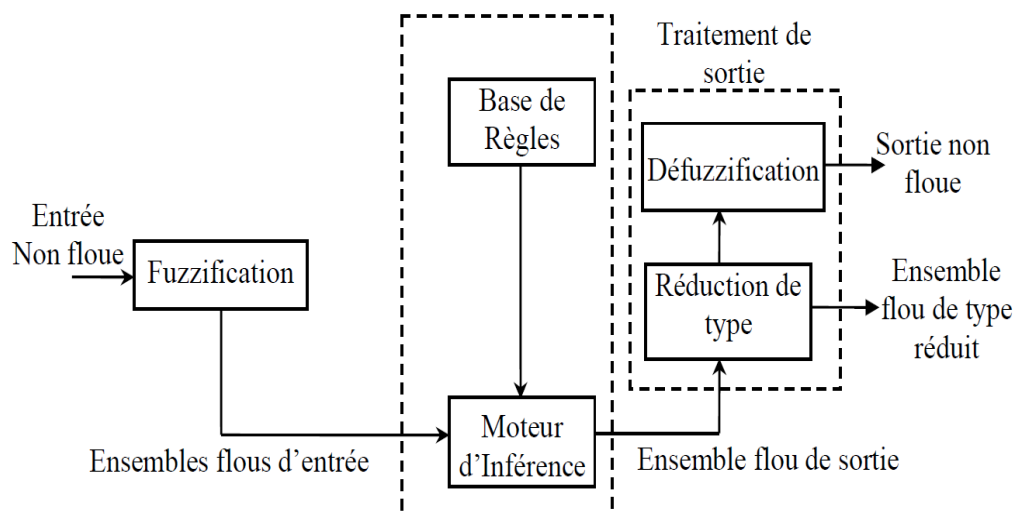


Figure 2-3 Structure d'un système flou type-2

2.3.8.1 Fuzzification

Contrairement à la fonction d'appartenance type-1, La fonction d'appartenance type-2 donne plusieurs degrés d'appartenance (ou dimensions) pour chaque entrée. Par conséquent, l'incertitude sera mieux représentée. Cette représentation va nous permettre de tenir compte de ce qui a été négligé par le type-1. Cet aspect est illustré dans la Figure 2-4: construction d'un ensemble flou type-2 à partir d'un ensemble flou type-1. [12]

2.3.8.2 Inférence

La différence entre le type-1 et le type-2 réside seulement dans la nature des fonctions d'appartenance, donc, la structure des règles dans le cas du type-2 va rester exactement la même. La seule différence étant que quelques (ou toutes) les fonctions d'appartenance seront de type-2.

Il n'est pas nécessaire que toutes les fonctions d'appartenance des prémisses et des conséquences soient de type-2. Il suffit qu'une seule fonction d'appartenance dans une prémisse ou dans une conséquence soit de type-2 pour que tout le système le soit aussi.

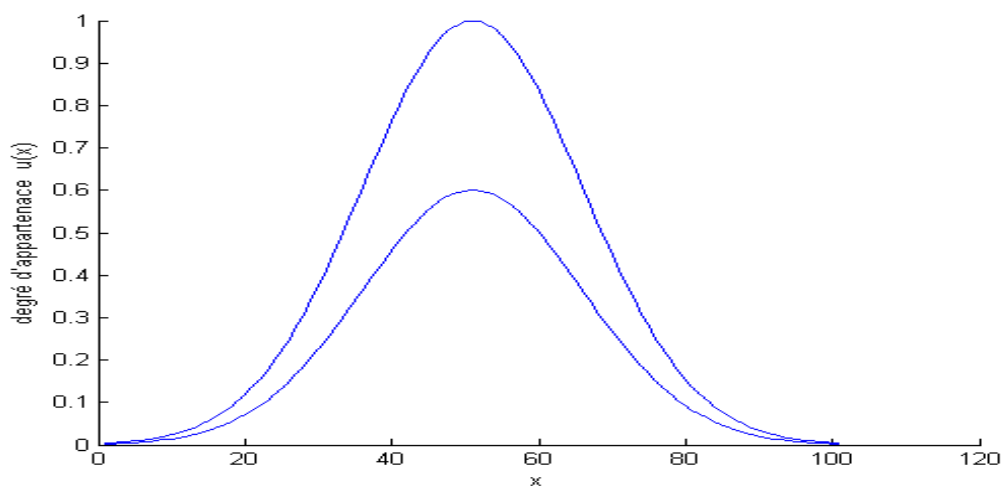


Figure 2-4 Fonction d'appartenance d'un ensemble flou type-2

2.3.8.3 Réduction de type

Dans un système flou type-1, où les ensembles de sortie sont des ensembles flous type-1, nous effectuons la défuzzification dans le but d'obtenir une valeur numérique (ensemble type-0) représentant la combinaison des ensembles de sortie.

Dans le cas du type-2, les ensembles de sortie sont des ensembles de type-2; donc nous devons utiliser des versions étendues des méthodes de défuzzification de type-1 appelées la réduction de type [Kar-98]. Cette opération va transformer l'ensemble flou type-2 résultant en un ensemble flou type-1 appelé ensemble type réduit, qui sera défuzzifié par la suite. Cet ensemble type réduit résultant prend en considération plus d'informations au sujet des incertitudes des règles que la valeur défuzzifiée (un nombre).

Parmi les méthodes de réduction de type utilisées on cite : [11]

- La réduction de type par le centre de gravité.

- La réduction de type par la hauteur.
- La réduction de type par le centre des ensembles.

2.3.8.4 Défuzzification

A la fin de l'étape de réduction de type, on obtient un ensemble type réduit qui est un ensemble flou type-1, alors il est nécessaire de le transformer en une valeur numérique bien déterminée. [15]

2.4 Contrôleur flou

Le contrôle flou, qui est probablement le domaine d'application le plus important de la logique floue, n'a été développé qu'à partir de 1973 par Mamdani. Depuis, il a attiré l'attention de beaucoup de chercheurs partout dans le monde.

Dans les années 80, ce sont les japonais qui utilisèrent la logique floue dans des produits industriels pour résoudre des problèmes de réglage et de commande. Les domaines où la logique floue a été appliquée avec succès sont variés et nombreux. Ainsi, la logique floue n'est pas limitée au domaine du contrôle, mais elle est aussi utilisée dans d'autres domaines qui nécessitent des décisions basées sur l'expérience humaine comme les diagnostics médicaux, le traitement d'image, la vision ...etc.

Les contrôleurs flous sont des programmes exécutés sur des microcontrôleurs classiques. Cela limite parfois l'application à des procédés qui demandent une fréquence élevée de commande. Pour surmonter ce problème, l'implémentation matérielle (hardware) du contrôleur semble être la seule solution.

La structure d'un tel contrôleur est généralement, composée des blocs représentés sur la Figure 2-5.

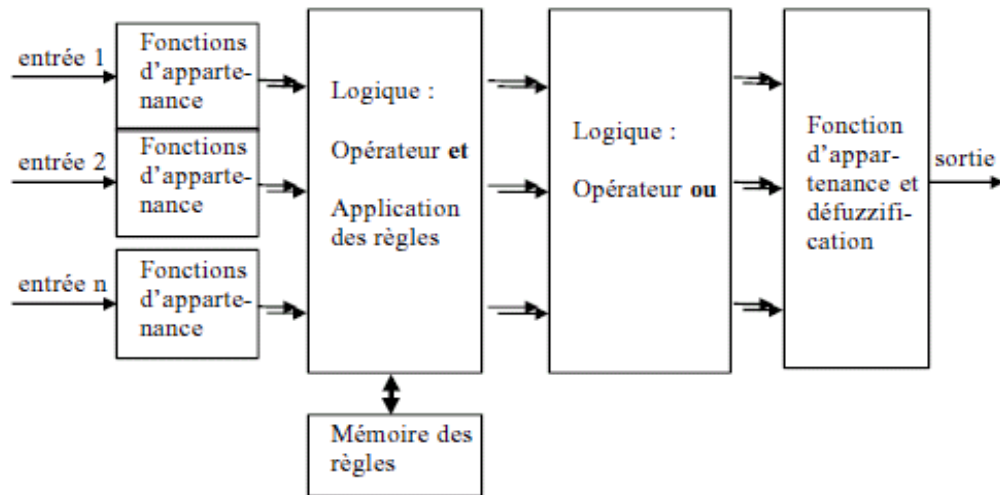


Figure 2-5 Structure générale du contrôleur flou

2.5 Implémentation de contrôleurs flous [16]

Généralement, un projet flou est décrit par les caractéristiques suivantes :

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

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

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

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 l'application).

2.6 Conclusion

Les fonctions d'appartenance floues type-1 sont bidimensionnelles, par contre, les fonctions d'appartenance floues type-2 sont tridimensionnelles. La nouvelle (troisième) dimension des ensembles flous type-2 fournit un degré de liberté supplémentaire permettant de prendre en charge la modélisation des incertitudes. De ce fait, les ensembles flous type-2 ont la capacité de modéliser les incertitudes parce que leurs fonctions d'appartenance sont eux-mêmes floues.

Les ensembles flous type-2 sont très efficaces dans les circonstances où il nous est difficile de déterminer exactement les fonctions d'appartenance pour les ensembles flous ; par conséquent, ils sont très efficaces pour l'incorporation des incertitudes. [17]

L'implantation matérielle « hardware » d'un système à base de logique floue.

Peut s'effectuer de plusieurs façons, surtout les processeurs flous numériques et analogiques. L'approche numérique est relativement facile à développer comparée aux approches analogiques. Les processeurs flous numériques offrent une plus grande flexibilité de configuration. Leurs performances par contre restent dépendantes des méthodes d'inférence utilisées par les méthodes de défuzzification. [21]

Langage VHDL et Circuits programmables

3.1 Introduction

Le choix d'un langage spécifique est généralement lié à plusieurs conditions et 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 en passant par différentes étapes de conception.

Le VHDL est un langage de description matériel qui permet de faciliter le développement d'un circuit numérique en fournissant une méthode rigoureuse de description du fonctionnement et de l'architecture du circuit désiré.

Le VHDL est développé dans les années 80 aux États-Unis, il est ensuite devenu une norme IEEE numéro 1076 en 1987. Révisé en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité du langage, cette norme est vite devenue un standard en matière d'outils de description de fonctions logiques. A ce jour, on utilise le langage VHDL pour concevoir des ASIC, programmer des composants type PLD, CPLD et FPGA, et concevoir des modèles de simulation numériques ou des bancs de tests.

L'abréviation VHDL signifie VHSIC **H**ardware **D**escription **L**anguage (VHSIC: **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit). Ce langage a été écrit pour réaliser la simulation de circuits électroniques. On l'a ensuite étendu en lui rajoutant des extensions pour permettre la conception (synthèse) de circuits logiques programmables (PLD).

Il permet au code écrit d'être portable, c'est à dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit. Il faut avoir à l'esprit que ce langage dit de haut niveau permet de matérialiser les structures électroniques d'un circuit.

3.2 Structure d'une description VHDL

La structure d'une description VHDL (Module VHDL) est un ensemble d'éléments VHDL avec lesquels nous allons décrire un système numérique, celui-ci peut être constitué d'une simple porte logique jusqu'à un système complexe. La structure de description VHDL est constituée d'une entité qui définit l'interface et d'une ou plusieurs architectures qui définissent le fonctionnement. [25]

3.2.1 L'entité

L'entité est une vue externe du module VHDL (structure de description), elle définit toutes les entrées et les sorties.

Syntaxe générale de l'entité :

```
EntityNom_du_moduleis  
  Port (Nom_de_l'entree1 : intype_du_signal ;  
        Nom_de_l'entree2 : intype_du_signal ;  
        Nom_de_sortie1 ;outtype_du_signal  
  ) ;  
EndNom_du_module ;
```

Les entrées et les sorties de l'entité sont toujours représentées par des ports :

- **In**: port d'entrée.

- **Out:** port de sortie.
- **Input:** port d'entrée /sortie.
- **Buffer** : une sortie rebouclée en entrée.

Les types utilisés pour les signaux d'entrée et de sortie sont :

- Le `std_logic` pour un signal.
- Le `std_logic_vector` pour un bus composé de plusieurs signaux.

3.2.2 L'architecture

L'entité décrit l'interface au modèle VHDL. L'architecture décrit la fonctionnalité fondamentale de l'entité. Une architecture est toujours liée à une entité et décrit le comportement de celle-ci. [27]

La syntaxe de l'architecture est la suivante :

```
ARCHITECTURE architecture_name OF entity_name IS
    [Declarations]
BEGIN
    (Code)
END architecture_name ;
```

Il existe trois manières différentes pour décrire une architecture :

3.2.2.1 Description comportementale

La description comportementale décrit le traitement de données à l'aide de process dont le corps est constitué d'instructions séquentielles.

3.2.2.2 Description flot de données

Pour décrire des éléments simples, des tables de vérité ou autres multiplexeurs, il est très pratique d'utiliser la description dite 'flot de données'. Ce type de description se déroule d'une manière concurrente. Tous les signaux sont assignés en même temps, il suffit de placer à la droite d'une équation ce que l'on veut assigner au signal. [29]

3.2.2.3 Description structurelle

Une description structurelle décrit un système sous forme d'une liste de primitives interconnectées entre elles. Dans ce type de description on relie entre eux, des composants préalablement décrits, en VHDL. L'exemple ci-dessous nous montre les trois types de description de l'architecture d'un demi-additionneur.

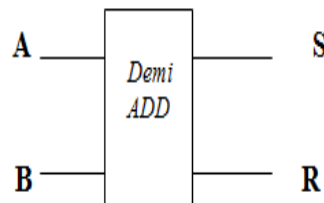


Figure 3-1 Demi_ADD

<pre> library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL; Entity Demi_ADD is port(A,B: in bit;R,S:out bit); EndDemi_ADD; </pre>	<pre> --Description flot de données -description Flot de données ArchitectureFlot_ADD of Demi_ADDis Begin S <= A xor B ; R <= a and B ; Endflot _ADD; </pre>
<pre> --description Structurelle Architecturestructurelle of Demi_ADD is ComponentXou Port (A, B : in bit ; S, R : out bit) ; End component ; Component ET Port(E1,E2 :in bit ;S : out bit); End component; For U1: Xou use entity work. Xou (A_xou) ; For U2: ET use entity work. ET (A_ET); Begin U1:Xou port map (A, B, S); U2:ET port map (A, B, R) ; End structurelle ; </pre>	<pre> --Description comportementale Architecture comport1 of Demi_ADD is Begin Process If ((A = '0') and (B = '0')) then S <= 0 ;R <= 0 ; If ((A = '1') and (B = '0')) then S <= 1 ; R <= 0 ; If ((sel0 = '0') and (B = '1')) then S <= 1 ; R <= 0 ; If ((A = '1') and (B= '1')) then S <= 0 ; R <= 1 ; Endprocess ; End comport_Demi_ADD; </pre>

3.3 Les modes de travail en VHDL

Le VHDL utilise deux modes de fonctionnement : le mode combinatoire (ou concurrent) et le mode séquentiel. Chacun de ces modes est utilisé dans des cas bien précis.

3.3.1 Le mode combinatoire

En mode combinatoire (ou concurrent), toutes les instructions d'une description VHDL sont évaluées et affectent les signaux de sortie en même temps, l'ordre dans lequel les instructions sont écrites n'a donc aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.

Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres. Avec VHDL il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, cela n'est pas toujours évident.

3.3.2 Le mode séquentiel

Le mode séquentiel utilise les process dans lesquels le temps est une variable essentielle. Un process est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement, c'est-à-dire les unes à la suite des autres. Il permet d'effectuer des opérations sur les signaux en utilisant l'instruction standard de la programmation structurée comme dans les systèmes à microprocesseurs. Dans le mode séquentiel, on distingue :

- **Le mode séquentiel asynchrone** : dans lequel les changements d'état des sorties peuvent se produire à des instants difficilement prédictibles (retards formés par le basculement d'un nombre souvent inconnu de fonctions),
- **Le mode séquentiel synchrone** : dans lequel les changements d'état des sorties se produisent tous à des instants quasiment connus (un temps de retard après un front actif de l'horloge).

3.4 Synthèse et simulation d'une description VHDL

Le développement d'un projet VHDL nécessite l'utilisation de deux outils : le simulateur et le synthétiseur. Le premier va nous permettre de simuler notre description VHDL avec un fichier de simulation appelé « test-bench » ; cet outil interprète directement le langage VHDL et il comprend l'ensemble du langage. L'objectif du synthétiseur est très différent : il doit traduire le comportement décrit en VHDL en fonctions logiques de base, celles-ci dépendent de la technologie choisie ; cette étape est nommée « synthèse ». L'intégration finale dans le circuit cible est réalisée par l'outil de placement et de routage. Celui-ci est fourni par le fabricant de la technologie choisie. [24]

3.4.1 Synthèse d'une description VHDL

La synthèse d'une description VHDL consiste à concevoir le schéma logique décrit par ce programme. Elle est très intéressante pour la conception d'un circuit intégré spécifique à une application, qu'il s'agisse d'un ASIC (Application Specific Integrated Circuit) ou d'un ASIC programmable par l'utilisateur appelé FPGA (Field Programmable Gate Array). Dans ce cas, la synthèse se fait généralement en deux étapes :

- la traduction du programme en un schéma logique fonctionnel basé sur des bascules et des portes logiques.
- la traduction de ce schéma fonctionnel en un schéma technologique adapté aux bibliothèques et à la structure de l'ASIC ou du FPGA.

3.4.2 Simulation d'une description VHDL

La possibilité de simuler avec des programmes VHDL devrait considérablement faciliter l'écriture de test avant la programmation des circuits et évite ainsi de nombreux essais sur prototype qui sont coûteux et dont les erreurs sont plus difficiles à trouver. Bien que la simulation offre de grande facilité de test, il est toujours nécessaire de concevoir les circuits en vu des tests de fabrication, c'est-à-dire en permettant l'accès à certains signaux internes. [27]

On distingue deux types de simulations :

- **Simulation logique**

Le modèle VHDL peut être validé par simulation logique en utilisant des Test-benchs écrits en VHDL. Un test-bench comprend une entité vide, une déclaration d'une instance de composants testés, et un ensemble de stimuli, ou vecteur de test. La simulation après synthèse logique ne prend cependant en compte que les délais des portes logiques. [27]

- **Simulation temporelle**

Après placement et routage, il est possible d'extraire les valeurs des éléments parasites (résistance, capacité) associés aux interconnexions et de calculer les délais correspondants. Ces délais sont stockés dans un fichier SDF (Standard Delay Format), puis réinjectés dans la description VHDL obtenue après synthèse logique. Cette fois-ci la simulation (après placement et routage) prend en compte les délais des portes logiques et des interconnexions.

3.5 Outil de développement Xilinx 13.3

Le projet de développement du contrôleur flou englobe des parties logicielles et matérielles. La première étape est la spécification de l'architecture du contrôleur flou. La conception est composée de plusieurs étapes et fait appel à un outil de développement garantissant : simplicité, modularité et efficacité d'accomplissement.

La conception du modèle **VHDL** du contrôleur et de sa synthèse fait appel à l'environnement de développement **XILINX ISE 13.3**. La validation et la vérification fonctionnelle fait appel au simulateur « VHDL » **I Sim simulator**.

3.5.1 L'environnement de développement (XILINX ISE 13.3)

XILINX FOUNDATION est un des logiciels de CAO « Conception Assistée par ordinateur » les plus répandus, sur les plates formes de type pc, pour la mise en œuvre des circuits logiques programmables (FPGA, CPLD, PLD...etc.) ISE Fondation 13.3 est un environnement intégré de développement des systèmes numériques ayant pour but une implémentation matérielle sur CPLDS, FPGA de la compagnie XILINX. Les designs peuvent être décrits sous trois formes principales :

- Schémas.
- langage de description matérielle (HDL) comme VHDL ou Verilog.
- diagrammes d'états.

ISE intègre donc différents outils permettant de passer à travers tout le flot de conception d'un système numérique :

- un éditeur de textes, de schémas et de diagramme d'état.
- un compilateur **VHDL**/Verilog.
- un outil de simulation.
- des outils pour des gestions des contraintes temporelles.
- des outils pour la synthèse.
- des outils pour la vérification.
- des outils pour l'implantation sur circuits programmables

Toutes les fonctions de logiciel sont accessibles à partir du navigateur ISE. L'écran principal fait alors apparaître Plusieurs fenêtres comme le montre la Figure 3-2.

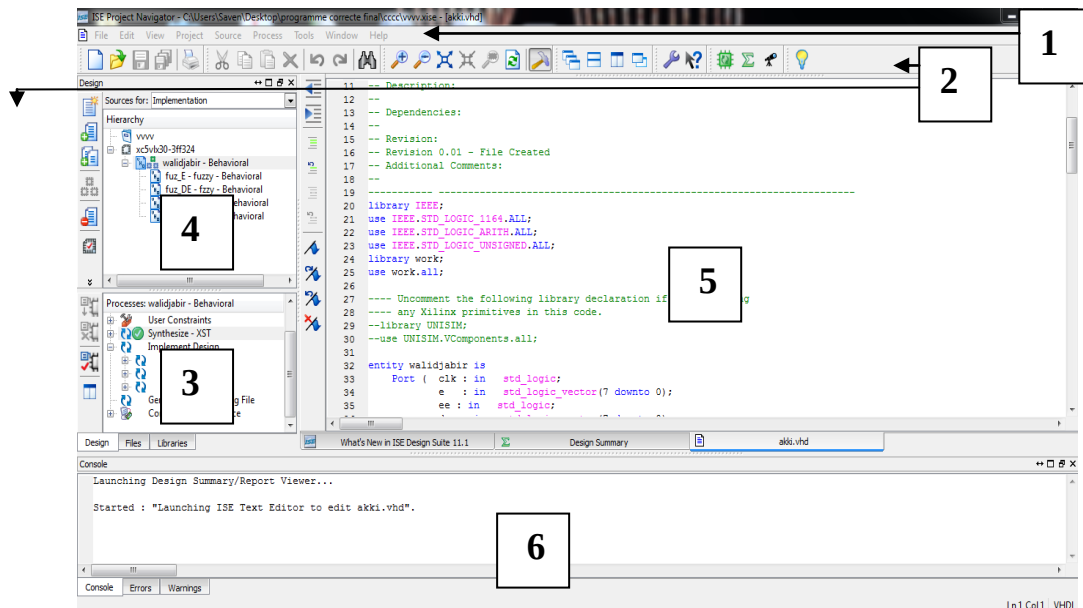


Figure 3-2XILINX ISE 13.3

1. Menus déroulants
2. Icones de raccourci
3. Processus de traitement (Fenêtre des processus) : permet d'ordonner des processus de traitement ou même de création.

Ces fonctions sont les mêmes que celles des menus déroulants et des raccourcis.

4. L'affichage hiérarchique des sources (fenêtres de sources) : permet l'affichage des sources actives du projet, dans le cas d'une simulation (Behavioral Simulation) ou d'une saisie / implantation (Synthèses / Implémentation).
5. fenêtre d'édition des sources : permet la saisie de sources (par exemple un schéma logique) ainsi que l'affichage des résultats (par exemple le résultat d'une simulation).
6. fenêtre d'édition des messages : permet l'affichage de messages (par exemple les erreurs précisant pourquoi un processus n'a pas été mené à son terme).

3.6 Les circuits programmables

De nombreuses familles de circuits programmables et reprogrammables sont apparues depuis les années 70 avec des noms très divers suivant les constructeurs.

La Figure 3-3 donne une classification possible des circuits numériques en

précisant où se situe chaque type de ces circuits dans cette classification. [18]

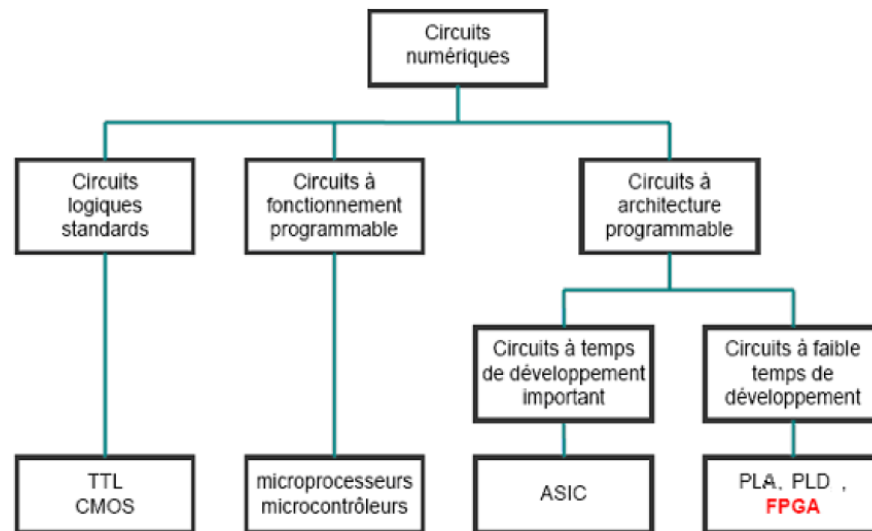


Figure 3-3 Classification des circuits logiques

Comme nous allons implanter notre contrôleur flou sur FPGA, nous allons se limiter aux définitions des circuits à architecture programmable et nous approfondir un peu plus sur les circuits FPGA.

PLA (Programmable Logic Array) : circuits dont les fonctions peuvent être codées en une somme de produit, un produit de somme ou un mélange des deux.

PLD (Programmable Logic Device) : nom générique donné à l'ensemble des circuits monolithiques formés des cellules logiques (comportant des fusibles, des anti fusibles ou des mémoires) qui peuvent être programmés et parfois reprogrammés par l'utilisateur.

SPLD (Simple Programmable Logic Device) : circuit programmable élémentaire composé d'un bloc d'entrée, d'une matrice ET, d'une matrice OU et d'un bloc de sortie.

CPLD (Complexe Programmable Logic Device) : circuit programmable hiérarchique regroupant un ensemble de circuits programmables élémentaires (**SPLD**) et un réseau d'interconnexion programmable.

ASICs (Application Specific Integrated Circuit) : circuits produits pour un seul

client selon les spécifications de ce dernier.[18]

FPGA (Field Programmable LogicArray) : circuits présentés par XILINX depuis 1980, ils diffèrent des CPLD par leur architecture, l'espace mémoire, le degré d'intégration des composants et le cout, les FPGA nécessitent un outil de placement et de routage. [26]

3.6.1 L'architecture des circuits FPGA

Dès l'origine, les FPGA, tels que Xilinx les a inventés, avaient la réputation de mettre à disposition de l'utilisateur une conception rapide, fiable et simple. Les progrès technologiques ont permis d'accéder à des matrices logiques programmables de plusieurs millions de portes.

Cette complexité actuelle reste absolument gérable et permet la réalisation d'applications très performantes moyennant une bonne connaissance des ressources offertes et le respect d'une méthodologie de conception. **La Figure.II.4** montre l'architecture simplifiée d'un FPGA de Xilinx (pour les familles Spartan-IIE, Virtex-E et Virtex-II). Les principales caractéristiques de ces trois familles sont :

- Complexités allant de 1500 à plus de 8 millions de portes.
- Faible consommation.
- Grande souplesse d'utilisation des entrées/sorties avec adaptation d'impédance (Virtex-II) et configuration en mode différentiel (Spartan-IIE, Virtex-E et Virtex II).
- Fonctions mémoire (distribuée et blocs de RAM).
- Dispositifs de gestion des horloges (DLL et DCM).
- Multiplieurs câblés (Virtex-II)

Et bien d'autres possibilités permettant d'optimiser à la fois la performance et la densité des fonctions logiques et/ou arithmétiques.

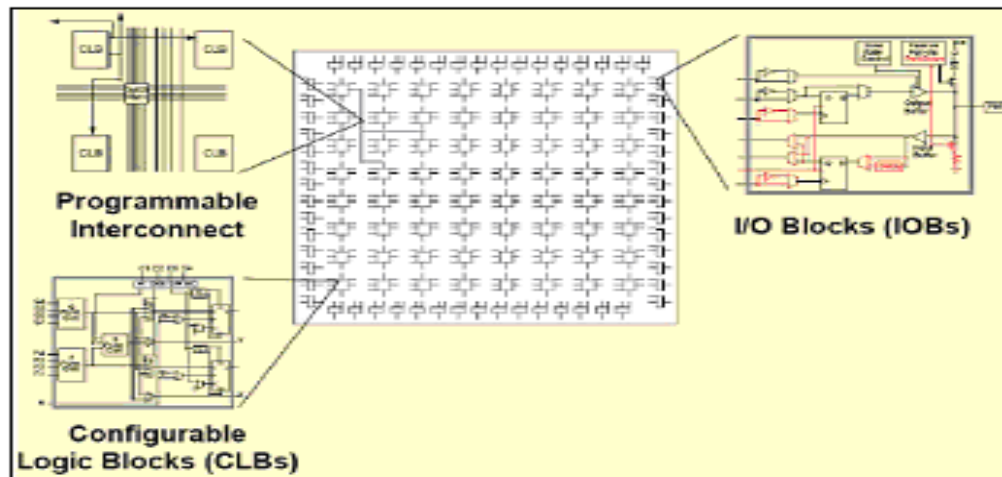


Figure 3-4 Organisation des circuits FPGA de Xilinx

Dans ce qui suit, on va faire une description de l'architecture utilisée par Xilinx, Celle-ci se présente sous forme de deux couches :

- Une couche appelée **Circuit Configurable**.
- Une couche réseau mémoire **SRAM** (Static Read Access Memory). [26]

3.6.1.1 Les circuits configurables :

3.6.1.1.1 Les CLB (Configurable Logic blocs) :

C'est l'unité fondamentale du bloc logique qui fournit des éléments utilitaires pour la logique combinatoire et la logique séquentielle,

Les blocs logiques configurables sont les éléments déterminants des performances du circuit FPGA. Chaque CLB est un bloc de logique combinatoire composé de générateurs de fonctions à quatre entrées (LUT) et d'un bloc de mémorisation/synchronisation composé de bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB. La LUT (Look Up Table) est un élément qui dispose de quatre entrées, il existe donc $2^4 = 16$ combinaisons différentes de ces entrées. L'idée consiste à mémoriser la sortie correspondant, à chaque combinaison d'entrée dans une petite table de 16 bits, la LUT devient ainsi un petit bloc générateur de fonctions. La Figure 3-5 montre le schéma simplifié d'un CLB de la famille XC4000 de XILINX.

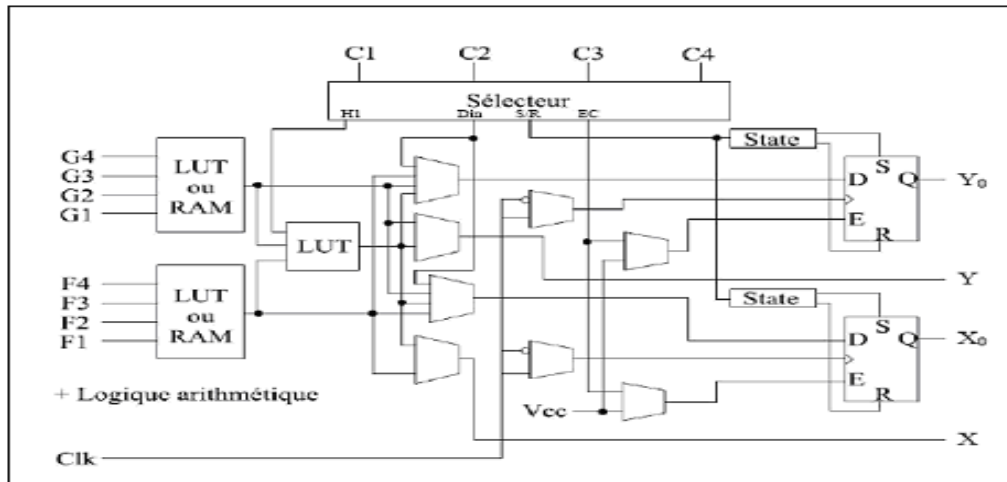


Figure 3-5 Schéma d'une cellule CLB (XC4000 de Xilinx)

3.6.1.1.2 Les IOB (Input Output Bloc)

Ils constituent l'interface entre les bornes du circuit et les CLB. Le dispositif de routage Versa Ring offre les ressources nécessaires à l'interconnexion des CLB aux IOB. Nous pouvons ainsi modifier le système implanté sur le FPGA sans interférer avec l'attribution des bornes. Cette caractéristique s'avère importante si nous souhaitons développer des nouvelles versions d'un produit tout en conservant la compatibilité au niveau du boîtier. [27]

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 signal bidirectionnel ou être inutilisé (état haute impédance). La Figure 3-6 présente la structure de ces blocs.

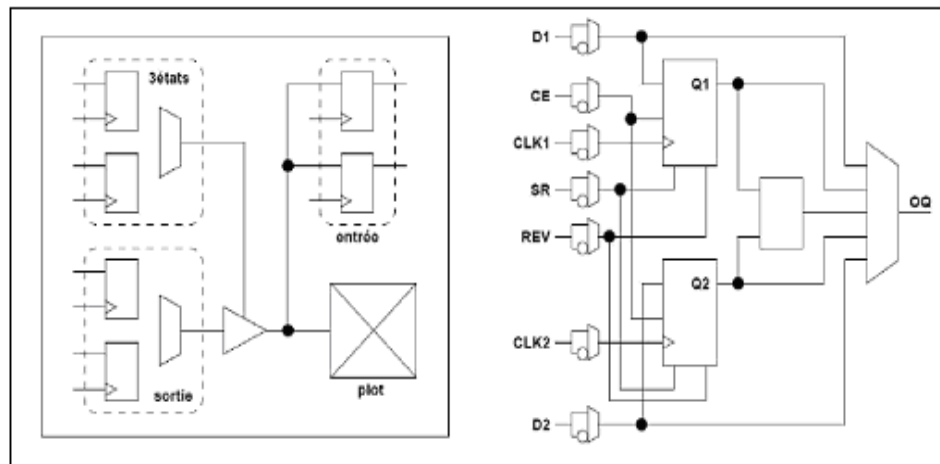


Figure 3-6 Schéma des blocs d'entrée /sortie (IOB)

3.6.1.2 Réseau mémoire SRAM

La programmation du circuit FPGA, appelé aussi LCA (LogicCellsArray), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM. [26]

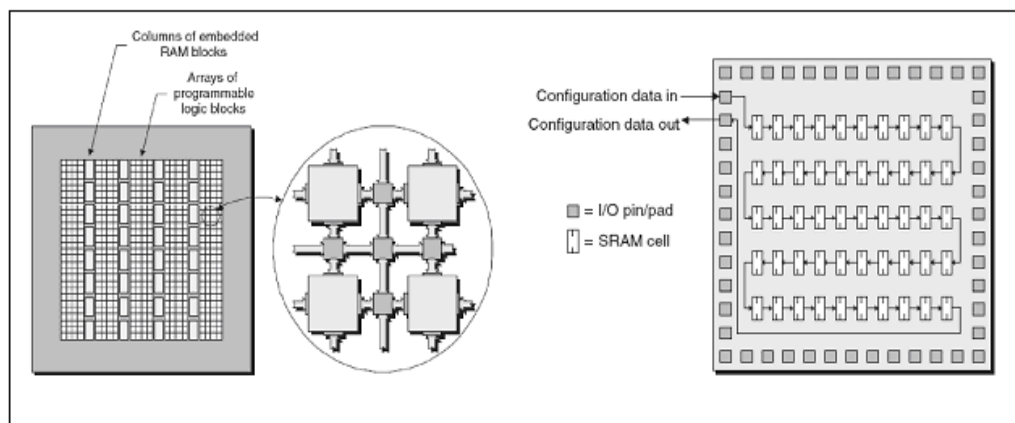


Figure 3-7 Réseau mémoire SRAM

3.7 Développement d'un projet VHDL sur FPGA

Pour implanter un projet VHDL sur un circuit programmable FPGA, on doit suivre les étapes données par la figure ci-dessous :

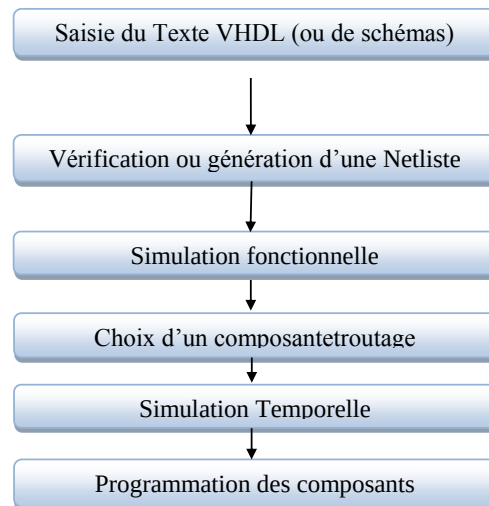


Figure 3-8 Développement d'un projet sur FPGA

- L'étape de vérification permet de valider la syntaxe du programme.
La (Netlist) est un fichier contenant la description de l'application sous forme d'équations logiques.
- Lors de l'étape de simulation fonctionnelle, on valide l'application indépendante de l'architecture et des temps de propagation du futur circuit cible.
- L'étape de routage génère les informations permettant d'intégrer l'application dans le circuit choisi. une (retro annotation) est effectuée dans le (Netlist), les temps de propagation du composant cible y sont pris en compte.
- Enfin l'étape de l'implantation permet de charger le modèle dans un circuit FPGA utilisant un port série de l'ordinateur, puis de tester le circuit.

3.8 Conclusion

La conception de circuits programmables complexes est devenue une tâche possible grâce aux facilités offertes par les nouvelles plateformes de développement couplées à des langages tels que VHDL et à l'implantation sur circuits type FPGA.

La reconfiguration est devenue la clé sur laquelle repose l'évolution continue à moindre frais.

Conception du contrôleur flou

4.1 Introduction

Les contrôleurs flous les plus développés et exploités dans l'industrie, sont en fait dédiés à exécuter des fonctions où les paramètres d'entrée sont certes très liés au point de fonctionnement, et variant conformément à l'état du système difficile à modaliser dans une certaine plage. Mais vu les difficultés à implanter des capteurs très diverses et travaillant sur des situations étendues et donc imposant des routines à développer sur les contrôleurs (les délais, les temps de réponse, la stabilité, l'optimisation des ressources n'œuvrent pas toutes sur la même direction et c'est pourquoi les contrôles est asservissements y sont adaptés, c'est ce que nous proposons ici dans ce travail, c'est-à-dire au lieu d'exécuter des routines sur les contrôleurs et faire en sorte à ce que les sorties seraient orientées aux points souhaités par la supervision d'un organe d'optimisation et de gestion externe, nous proposons d'effectuer des reconfigurations adéquates aux conditions, ainsi les contrôleurs demeureront simples et d'une architecture simple flexible et en conséquence les caractéristiques des FPGA exploitées ne seront pas très pointues et avancées et donc un gain substantiel sur les ressources d'implantation des systèmes à FLC seront désormais simples et réalisables.

Le système à multiple FLC serait constitué d'un nombre de FLC reconfigurables partiellement sous la supervision d'un FLC (une méthode de suivie de reconfiguration subdivise la partie de configuration en deux : une fixe et l'autre

modifiable à volonté). Ce FLC de reconfiguration gère les états de configuration des autres et donc contrôlant et aiguillant selon un mécanisme n'altérant pas les méthodes floues et additivement profiter de la souplesse de programmation des FPGA pour y définir les contextes et les situations sur lesquelles reposera le système flou global.

Notre contribution s'est manifestée à travers les orientations suivantes :

- Subdiviser le système FLC global en multiple FLC caractérisant les situations qui s'imposent.
- Organiser les FLC et sans modifiant de contraintes en termes d'exploitation ou de stratégies.
- Définir l'approche de gestion et issues à suivre ; Développer l'organe de supervision et de contrôle et ainsi que les contextes d'action
 - types d'action directe
 - actions de reconfiguration
 - stabilité et robustesse

Comme la reconfiguration est très dépendante du matériel utilisé, nous nous limitons aux changements de contextes, de cycles d'états et réorientation d'activités vers les solutions prioritaires par exemple :

- Les solutions les plus rapides.
- Les solutions les moins coûteuses (Industrie, matière première et pertes)
- Obtention d'un système globale à base de technologie (caractéristiques de base) bon marché sans recourir aux circuits très avancés onéreux et imposant des exigences et restrictions aux signaux d'E/S.

Objectifs visés	Approches appliquées	résultats	Nouvelles approches
Modèle monobloc du tunnel	Contrôleurs flous en cascade CO,OP	Fonctionnalité des conditions correctes : traduction En temps de ventilation complexe	-amélioration de la gestion par introduction de la variation du trafic -indice de performance : énergie/confort
Modèle multi-bloc du tunnel	Chaque zone à un modèle -asservissement. -économie d'énergie Gestion	Gestion complexe +équipements réparties+problème aux interfaces des zones	-contrôle flou+gestionnaire pour garantir un indice de performance (énergie/confort) -acquisition précise des paramètres -reconfiguration des modes

Tableau 4-1 Comparaison entre les différentes approches.

Nous pouvons d'avance affirmer que gérer un tunnel routier, de longueur supérieur à 1800m est une tâche complexe et délicate. Les approches basées sur l'acquisition des paramètres tels que CO ,OP, Trafic, et la détermination d'un programme informatique développé autour d'un système à microprocesseur et le recours aux techniques de commande et d'asservissement classiques ont fait l'objet de plusieurs travaux et publications [8] ,[1],[2] mais ne répondent pas toujours aux exigences de sécurité, de maintenance et de gestion aisée. En effet des corrections et des interventions sur terrain demeurent les seules actions pour une exploitation qualifiée bonne et acceptable. Nous affirmons aussi que les difficultés de modélisation d'une part et les solutions économiquement adaptées aux approches suivies d'autre part sont les points essentiels qui ont réduit la pertinence des solutions exploitées. Plusieurs approches de modélisation de notre tunnel sont considérées.

- Comme première approche, le tunnel est considéré comme un seul bloc : Les difficultés liées à la détermination de la position des capteurs, des ventilateurs, de leurs puissances ainsi que du système de commande ne favorisent pas cette solution. En pratique, seul le paramètre influant Co est pris en charge, les commandes développées, en conséquence, optimisent uniquement l'énergie.
- Considération de plusieurs paramètres CO, OP, Trafic, mais tout en modélisant le tunnel comme un seul bloc : Etant donné que l'influence de ces paramètres est difficile à quantifier (à n'importe quelle position du tunnel) même si un emplacement uniforme de capteurs et de batteries de ventilateurs est adapté

(néanmoins, le recours à des méthodes d'estimation ou de simplification suite à certaines actions permet d'atteindre certains objectifs). Une conclusion directe à tirer est qu'une consommation excessive d'énergie n'offre pas toujours le confort souhaité.

- Configuration du tunnel en zones : La subdivision en zones apparaît comme solution et bonne approche afin de mieux cerner les difficultés inhérentes aux complexités du milieu mais en pratique, elle est difficile à mettre en œuvre. Car il n'est pas simple de trouver les paramètres sur lesquels obéira le partage en zones en plus de la difficulté de sélection du matériel et équipements de ventilation adaptés aux zones.
- Sans la définition d'objectifs claires et quantifiables tels que : performance, confort/énergie et la réussite à produire une stratégie d'exploitation, il sera très difficile de mettre au point une solution qui prendra en charge toutes les doléances et les objectifs à moindre coût.
- Après une étude rigoureuse de ces différentes approches ci-dessus et vu l'arrivée de nouveaux équipements embarqués, une solution reposant sur les techniques floues (pour une meilleure prise en charge des difficultés de modélisation des paramètres et fonctionnalités du système complexe à étudier) est choisie.

La solution proposée est montrée sur la Figure 4-1 où les systèmes à base de contrôleurs flous, traitent les signaux collectés (acquisition des signaux issus des capteurs) et un bloc dénommé gestionnaire embarqué ayant pour tâche la détermination des corrections nécessaires pour atteindre les objectifs ciblés en termes d'indice de performance confort /énergie d'une part et la facilité de gestion du système complet (sécurité, maintenance, évolutions et reconfigurations) d'autre part. Cette solution mènera probablement à mieux raffiner les actions au niveau de chaque zone et palliera aux effets d'interzones.

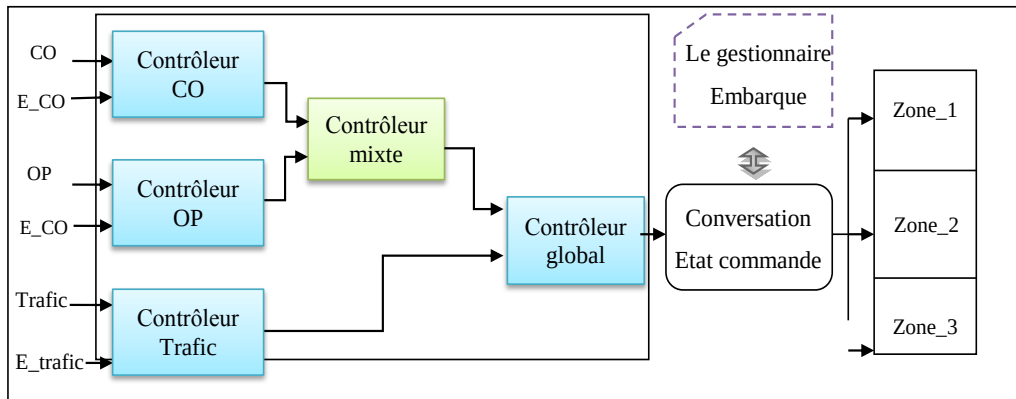


Figure 4-1 Solution proposée

- Afin de renforcer la sécurité et la maintenance et pour une meilleure prise en charge des contraintes de chaque zone nous avons adopté la solution illustrée sur la Figure 4-2.

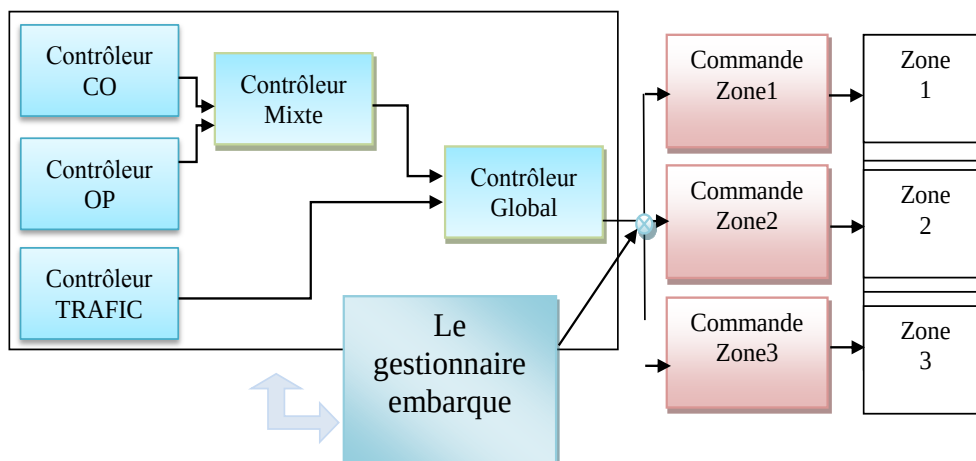


Figure 4-2 Solution adoptée

Les différentes tâches et les différentes activités développées dans ce travail sont :

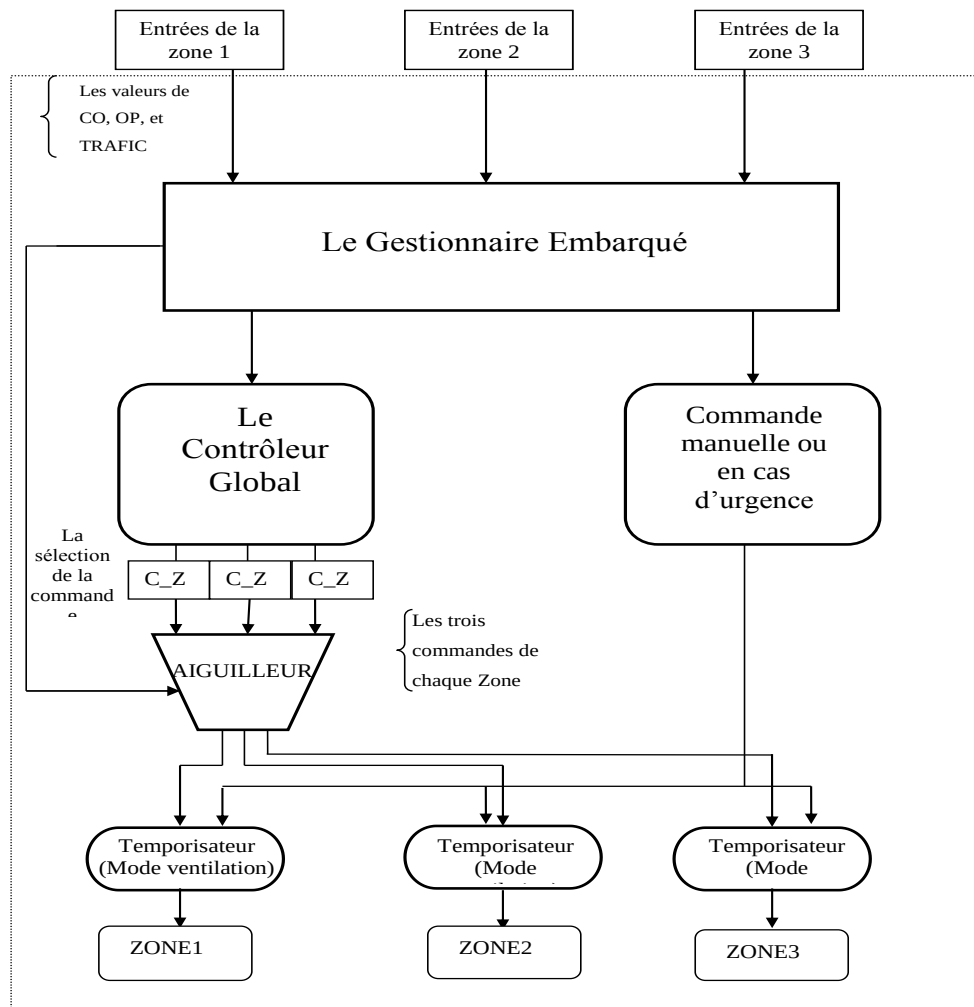
- Conception et simulation en VHDL sous une plateforme de Xilinx 13.3 des contrôleurs flous : contrôleur flou de CO, contrôleur flou d'OP, contrôleur flou mixte CO /OP, contrôleur flou de trafic et contrôleur flou traitant final.
- Modélisation des zones et répartitions des équipements de ventilation.
- Gestion des stratégies de ventilation

La commande de ventilation n'est pas si simple qu'elle apparaît, car il n'est pas correct de croire à un simple mot de commande activant ON/OFF des batteries de ventilateurs. Les équipements sont soumis à différentes restrictions et aux modes opératoires. Le résultat de ventilation est aussi très complexe à quantifier à tout point donné. Dans ce travail, nous avons considéré que :

- Les batteries de ventilation sont les mêmes sur toute la longueur du tunnel (faciliter la maintenance).
- L'activation de ventilation est traduite en une variable de temporisation (pour éviter la génération des commandes temporelles par TIMER ou software).
- Disposition des capteurs : chaque ensemble de capteurs sur une surface donnée est relié à une électronique adaptée pour la production de la valeur moyenne (les valeurs instantanées posent des problèmes vu l'étendue des surfaces considérées)
- Le matériel dans les trois zones définies est quasi-identique.
- Les équipements de production d'énergie et les armoires de commande sont installés près de l'entrée du tunnel afin de faciliter les interventions et l'exploitation.

4.2 Système général

4.2.1 Schéma bloc de système adopté :



4.2.2 Description du système global de ventilation :

Comme le système de contrôle flou global est inadapté à l'exploitation de tout le système, car le tunnel est subdivisé en plusieurs zones ayant plusieurs entrées/sorties dédiées alors on a proposé l'utilisation d'un mécanisme dit 'gestionnaire' pour la collecte des informations issues des capteurs (pour prétraitements) et les commandes aiguillées aux différentes zones traduites en termes de stratégies de modes de ventilation. Le gestionnaire est désormais l'organe qui alimente les contrôleurs à des périodes prédéfinies (estimées) et se charge de toutes les autres activités (exploitation par défaut ou standard, gestion des alarmes et des modes

d'urgence, prise en charge et amélioration de performance par réglage et reconfiguration).



4.2.3 Le gestionnaire :

Le rôle du gestionnaire dans notre système est aussi important que celui des contrôleurs flous du système global de commande, puisque il simplifie la programmation du contrôleur global et effectue les opérations réparties dans les zones, calcule les temps de fonctionnement des ventilateurs ou les paramètres convertibles en mode de ventilation (détermination des vitesses, fréquences, actions sur les cartes de commande de puissance des batteries de ventilateurs, etc.).

Le gestionnaire est encore utilisé pour la supervision et l'observation de fonctionnement du système complet afin de contribuer à l'amélioration effective de fonctionnement, il calcule l'énergie utile dans une période de temps bien définie comme il adapte en conséquence les différentes méthodes de ventilation pour achever et atteindre une méthode de ventilation qui consomme moins d'énergie et qui conservera et maintiendra des seuils de sécurité et des confort prédéfinis, le gestionnaire est aussi utilisé pour appliquer une méthode de 'ventilation standard', cette dernière est recommandée avant activation et désactivation manuelles pour l'apport initial de paramètres de fonctionnement supposés : corrects, par défaut, sécurisants, ou de démarrage. La gestion de commandes prioritaires, d'alarmes, d'états d'urgence fait partie des tâches imposées à ce gestionnaire car il est très complexe de

les intégrer dans les contrôleurs flous vu leurs apparitions inattendues et leurs réponses par gestions non synchrones (interruptions).

4.2.4 Contrôleur Global

Le contrôleur global est considéré comme un 'Top Level Design' qui contient des sous-contrôleurs flous inter- connectés, les entrées de ces contrôleurs flous sont les sorties du gestionnaire. Il y a six tâches et procédures définies à effectuer pour l'obtention de la véritable commande :

- Détermination de l'état de CO : C'est l'étape où le contrôleur fou de CO lit la valeur de Co fournie par le gestionnaire, exécute les trois fonctions (Fuzzification, Inférence, Défuzzification) pour déterminer l'état de Co dans la zone sélectionnée.
- Détermination de l'état d'opacité : C'est l'étape où le contrôleur flou d'OP lit la valeur d'OP à partir du gestionnaire, effectue les trois fonctions de fuzzification, d'inférence, et de défuzzification, pour déterminer l'état d'OP dans la zone à l'étude.
- La troisième étape considère et produit un état de Co et Op simultané. Les sorties des contrôleurs flous de Co et Op sont combinées dans un contrôleur fou (contrôleur mixte Co_Op), moyennant une matrice d'inférences réadaptée.
- Détermination de l'état de trafic : C'est l'étape où le contrôleur TRAFIC lit la valeur de TRAFIC à partir du gestionnaire (se basant sur l'étude liée au calcul de la vitesse de variation de trafic ou de la détermination d'une valeur qui traduit la densité de trafic en valeur efficace de pollution). Cette fonction est liée directement au nombre équivalent de véhicules polluants : c'est pourquoi nous avons introduit des valeurs floues de trafic telles que : trafic très dense, trafic dense, ou en termes de, trafic faible fort, très fort,...etc.
- Détermination de l'état général d'une zone : pour créer une méthode de ventilation simple et efficace on traite la sortie de la troisième étape (état mixte de Co, Op) avec la valeur de trafic dans un contrôleur fou (contrôleur- état). La sortie de ce contrôleur est l'état final de Co, OP, TRAFIC d'une zone.

- Détermination de la commande : la sortie du contrôleur état final est exploitée pour implémenter une commande de ventilation traduite en temps d'activation et sélections des ventilateurs correspondants à la stratégie prédéfinie.

L'ordre des contrôleurs flous est choisi dans notre contrôleur global après une étude extensive et test de tous les ordres possibles. On a noté, en simulation que les différences étaient petites en terme de l'état final.

4.2.5 Aiguilleur :

Après acquisition des informations traitées par le gestionnaire, le contrôleur global envoie les commandes via l'aiguilleur aux équipements de ventilation des différentes zones.

Ce système d'aiguillage permet de réduire la circuiterie de commande (contrôleur rapide action de ventilation lente)

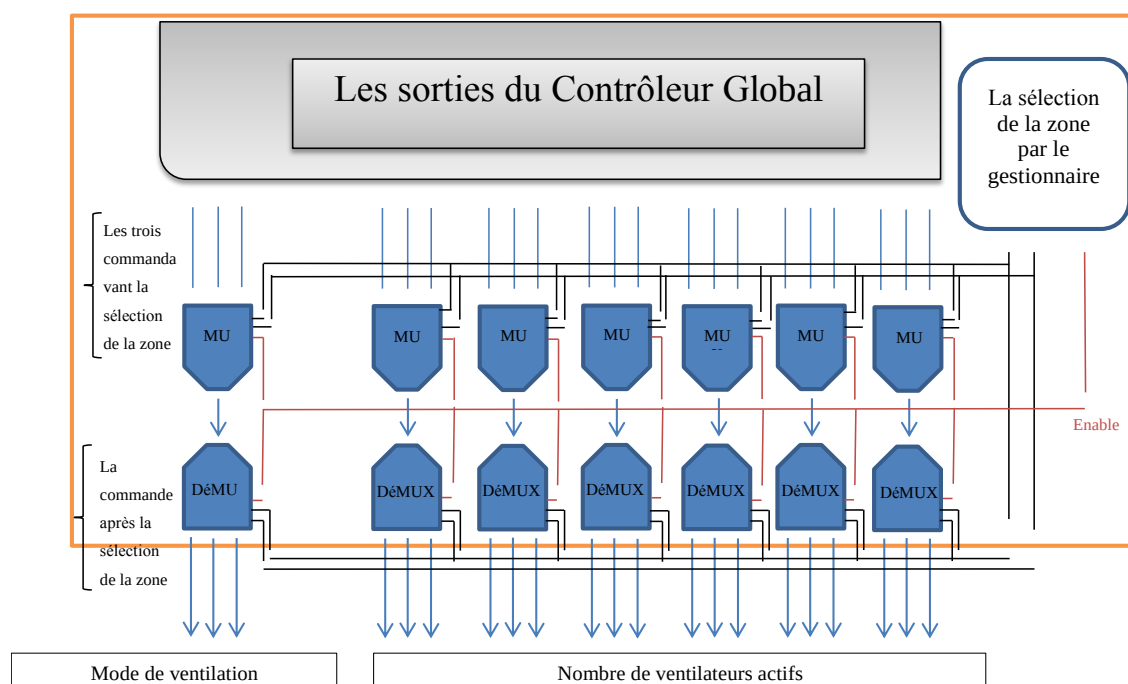


Figure 4-3L'aiguilleur

4.2.6 Temporisateur et mode de ventilation :

Le temporisateur reçoit la commande de contrôleur globale et la traduit en mode de ventilation en se basant sur le temps de ventilation (autre mode peut être :

détermination des vitesses, fréquences, actions sur les cartes de commande de puissance des batteries de ventilateurs, etc.).

4.3 Gestionnaire embarqué

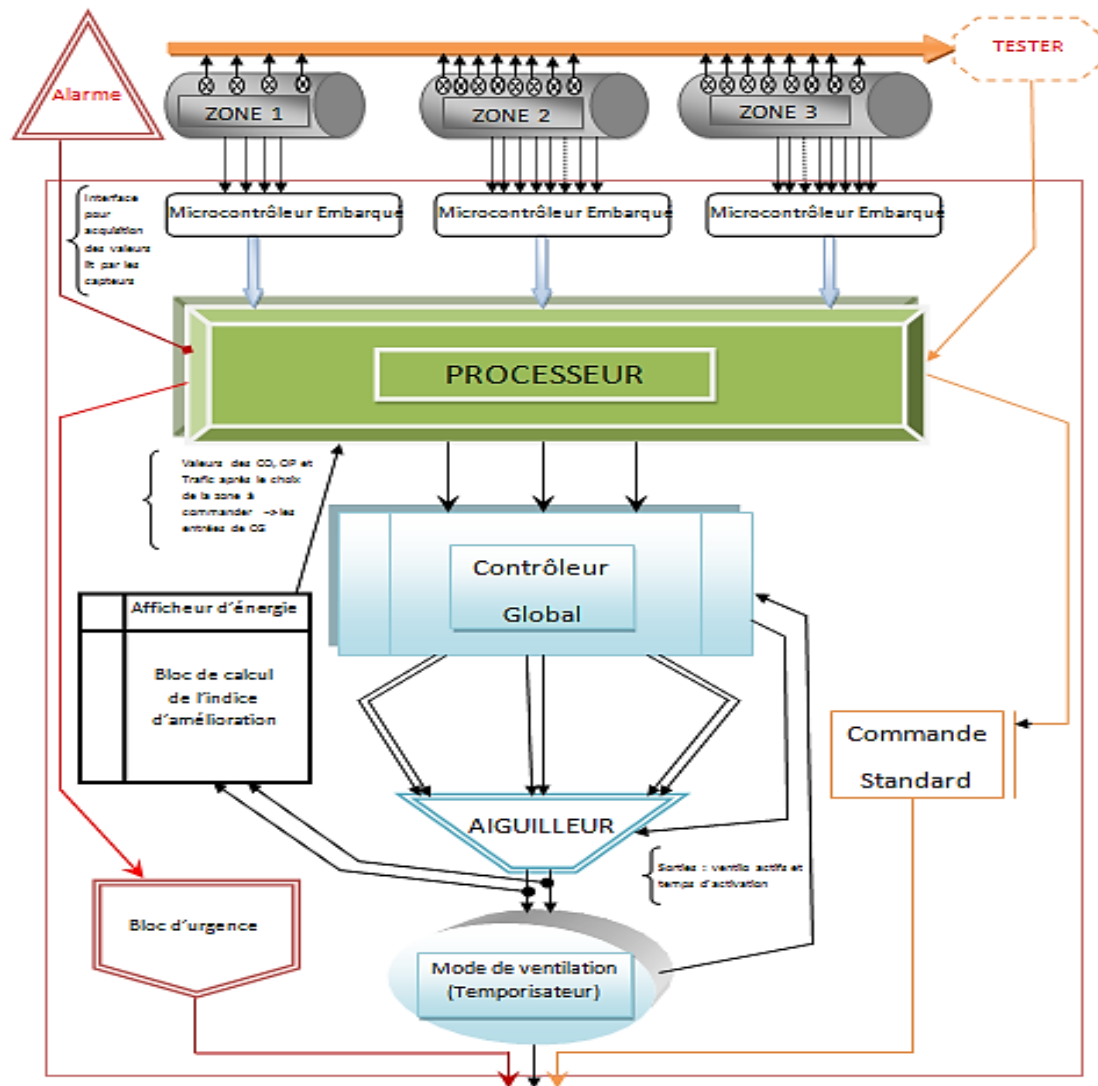


Figure 4-4 Schéma général de gestionnaire embarqué

4.3.1 Description générale du Gestionnaire Embarqué:

Le gestionnaire embarqué comprend plusieurs composants : Le cœur de ce gestionnaire est un microprocesseur qui reçoit les valeurs de Co, Op, TRAFIC acquis à l'aide de capteurs dédiés, et les envoie au contrôleur global ; mais avant que le microprocesseur lise ces valeurs émanant des capteurs, nous proposons que ces valeurs passent par des interfaces qui ont la véritable mission de systèmes embarqués où il y a la technique de collecte de multitudes données, c.-à-d. les signaux de sorties

des capteurs installés sur toute l'étendue définie et conformément à un déploiement constructif et aidant à déterminer les effets des polluants. Pour comprendre pourquoi on utilise l'interface, on doit s'intéresser au nombre de capteurs dans chaque zone, la méthode de lecture de leurs valeurs et en fin à la production de ces fameuses valeurs de Co, Op, TRAFIC d'une zone exploitées comme entrées de contrôle des différents contrôleurs flous.

A titre d'exemple, on a dans la première zone (1600m) quatre capteurs de Co monté sur un support déplaçable (permettre de changer les positions), la distance inter capteurs est uniforme et égale à 360 m. Pour le capteur d'Opacité, on a quatre capteurs, le premier sera positionné à 100m de l'entrée du tunnel, les quatre autres capteurs sont répartis également sur la distance restante, et on a une câble piézo-électrique le long du tunnel pour la mesure de trafic. Pour la deuxième zone (1000m), on place trois capteurs de Co, trois capteurs d'opacité. Pour la troisième zone (1600m), on a quatre capteurs de Co, quatre capteurs d'opacité.

Etant donné ce nombre important de capteurs à lire, mais représentatifs localement, il est recommandé de réaliser un prétraitement par éléments embarqués. Ces circuits embarqués transmettent ensuite leurs données au microprocesseur.

Comme des procédures additionnelles liées au traitement des cas d'urgence, de maintenance et de conditions pour réadaptation et suivi des opérations afin d'y rechercher des conditions optimales et favorisantes à la gestion de tout l'ouvrage qu'est le tunnel, des missions correspondantes y sont affectées au gestionnaire.

En cas d'urgence le gestionnaire désactive le contrôleur global et active le bloc d'urgence. Le bloc de commande standard peut intervenir en cas d'erreur ou mal fonctionnement dans le contrôleur global. Le microprocesseur définit la priorité de la commande (commande d'urgence, commande standard, commande normale).

Il y a d'autres blocs dans notre gestionnaire :

- Temporisateur
- Aiguilleur
- Bloc Alarme
- Bloc de configuration et d'amélioration
- Bloc d'commande standard
- Bloc de suivi, Test et maintenance.

4.3.2 L'interface :

On a choisi comme interface des microcontrôleurs embarqués dans les trois zones. Ces microcontrôleurs reçoivent les données issues de tous les capteurs de la zone (Co, Op, TRAFIC), ensuite réalise un premier traitement : Pour le Co et le OP, par exemple, il calcule la moyenne de chaque paramètre.

La détermination de l'état de trafic est très difficile puisque cet état est lié au nombre de véhicules polluants et il n'est pas lié uniquement au nombre total de véhicules. Quand on mesure un effet de CO, il est le résultat d'un certain nombre de véhicules il y a un certain temps déjà passé, donc non égal au nombre de véhicules présents dans le tunnel à un instant t donné. La variation ou vitesse de pollution est plus représentative que le nombre de véhicule, c'est pourquoi dans cette étude, nous avons modélisé le trafic par une valeur floue de type : trafic dense ou fort, trafic moyen ou moyennement polluant, etc.

Après cette opération, le microcontrôleur embarqué envoie les valeurs calculées au gestionnaire.

4.3.3 L'aiguilleur :

On a déjà vu le rôle de l'aiguilleur, ses activités avec le gestionnaire embarqué sont explicitées ci-dessous : Pendant la phase d'initialisation, le gestionnaire embarqué (microprocesseur) lit les valeurs traitées par le microcontrôleur embarqué de la première zone, ensuite active l'aiguilleur et le positionne à la première zone. Après cela l'aiguilleur laisse passer la commande¹ vers le temporisateur et ainsi de suite la procédure se répète pour la commande des autres zones.

4.3.4 Le Temporisateur :

Un temporisateur est un circuit électronique qui permet de mettre en route un système pendant un certain temps, ou qui permet de le mettre en route au bout d'un certain temps. Les applications d'un temporisateur sont multiples et variées, et on peut aussi bien avoir besoin d'activer un circuit pendant quelques secondes que pendant quelques heures voir plusieurs jours. Un temporisateur peut être construit à partir d'un simple monostable, mais nous verrons qu'il existe d'autres solutions, un peu moins simples mais qui permettent d'obtenir des durées de temporisation très longues.

Dans notre système, on a besoin, d'un temporisateur programmable puisque on a différentes durées d'activation des ventilateurs (voir look up table ci-dessous).

Le Code binaire	La période correspondante
01	T1
10	T2
11	T3

Tableau 4-2 Look up table (pour un mode de ventilation : temporisation)

Les durées d'activation varient selon la commande envoyée par le contrôleur global. Ces périodes d'activation seront très utiles au fonctionnement du bloc dénommé 'amélioration' pour calculer l'indice de performance.

4.3.5 Le Microprocesseur :

C'est le composant qui gère notre système global de ventilation, il est lié à tous les composants et établit les connexions entre les microcontrôleurs embarqués et le contrôleur global.

Pendant la phase d'initialisation, le microprocesseur vérifie les signaux d'urgence et du test, si non présents, il enchaîne le fonctionnement normal. Une attente d'un temps T_s nécessaire au microcontrôleur embriqué de la première zone de produire les valeurs des capteurs, et leur prétraitement.

Un organigramme de fonctionnement global doit être établi pour une meilleure gestion des ressources du système développé.

4.3.6 Bloc de commande standard :

Comme les équipements de puissance et de commandes associées peuvent passer par des situations parfois non désirables (erreurs dans le contrôleur global, panne de capteurs, nouvelle situation ou inconnue etc.), le système intervient selon plusieurs

variantes pour corriger ou imposer des modes d'actions : ex mode manuel, ou mode par défaut.

4.3.6.1 Le bloc d'urgence :

Dans le cas d'urgence, un bloc dédié au traitement des différents cas, incendie, accident...etc. sera activé selon des procédures définies par des experts.

4.3.6.2 Bloc de supervision et amélioration de performances :

Ce bloc est introduit pour apporter les correctifs nécessaires, suites aux lacunes et faiblesses introduites dans les différents blocs constitutifs ainsi de l'approche globale suivie. Son rôle et son impact sont déterminants car il s'agit d'améliorer les performances de notre système. En effet, on s'appuie sur ce bloc pour changer la méthode de ventilation relative à un indice de performance calculé à partir de l'énergie consommée et du degré de confort souhaité dans le tunnel pendant un temps prédéfini.

4.3.6.3 L'indice d'amélioration

Selon une méthode mathématique de pondération, on calcule l'indice réparti en deux paramètres : le premier concerne l'énergie évaluée par un certain pourcentage de cet indice, le deuxième paramètre caractérise le confort représenté par le reste du pourcentage de l'indice. Dans notre travail, nous donnons à l'énergie 60% de la valeur de l'indice d'amélioration et 40% pour le confort.

4.3.6.4 Fonctionnement du Bloc de supervision et amélioration de performances

Le bloc d'amélioration calcule l'énergie consommée par les ventilateurs et le confort dans le tunnel pendant des périodes définies par un expert.

Calcul d'énergie : le nombre de ventilateurs actifs et leurs périodes d'activation à partir de l'aiguilleur sont utilisées pour calculer et pondérer cette énergie.

Calcul de confort : la valeur de confort est déterminée à partir de deux paramètres, le premier c'est le changement brusque d'opacité (passage du niveau d'opacité bas à un niveau très grand), le deuxième paramètre est le niveau de monoxyde de carbone (calcul approché).

Ensuite ce bloc envoie cet indice vers le microprocesseur, qui déterminera une méthode de ventilation à partir de l'indice calculé (préserve la méthode si la valeur de l'indice est acceptable). Pour changer la méthode de ventilations, le microprocesseur redéfinit les connexions entre le contrôleur-état-général et les blocs qui contiennent les commandes de la zone (com.-zone) par un Switch.

4.3.7 Organigramme de fonctionnement

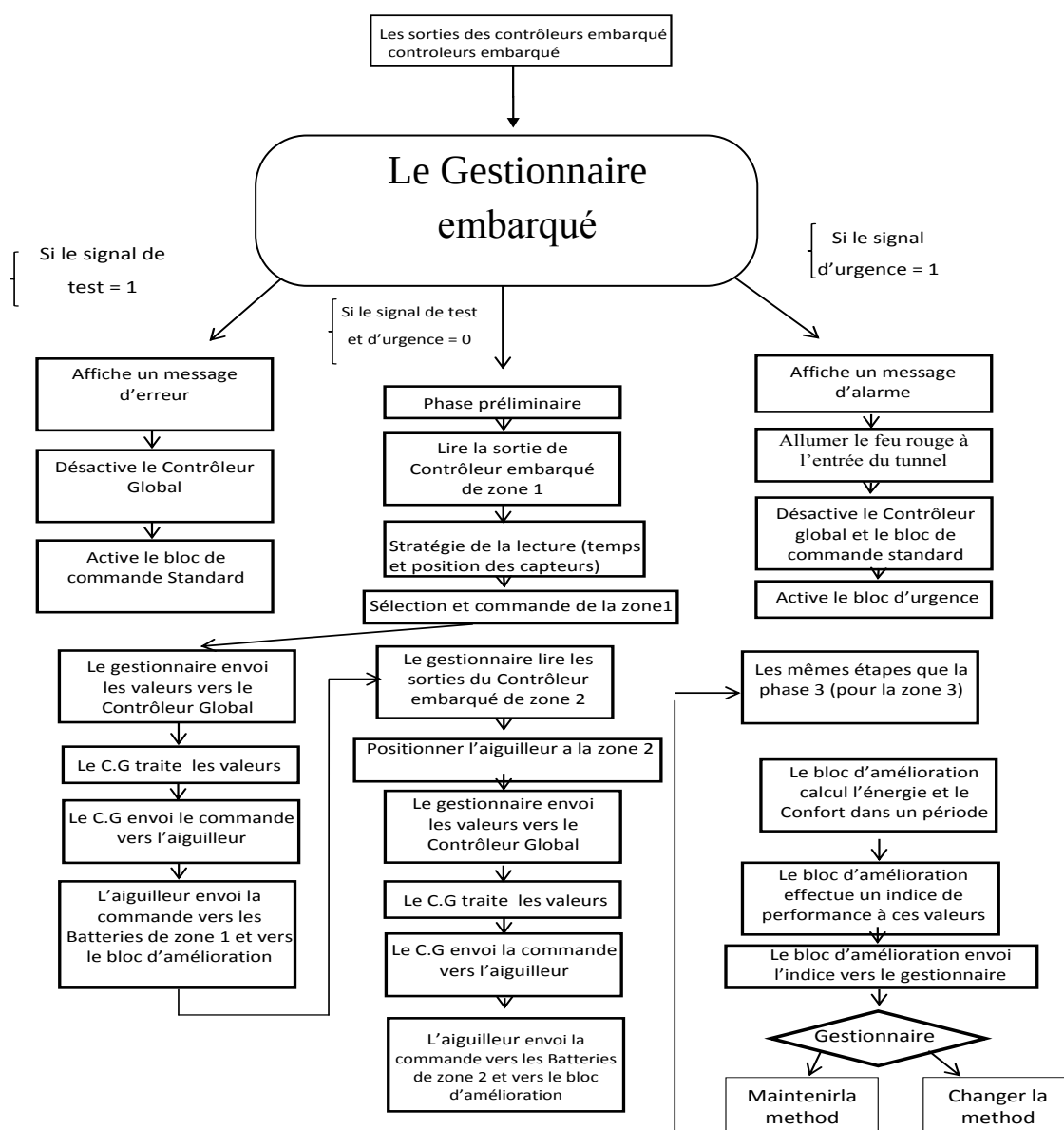


Figure 4-5 Organigramme de fonctionnement

Dans ce chapitre, l'implémentation structurelle de notre contrôleur flou est développée. Cette implémentation s'appuie directement sur les données consignées dans les chapitres précédents. Pour simplifier l'explication, la méthode de subdivision du travail en deux blocs simples est adoptée :

- Bloc pour le Contrôleur flou (FLC).
- Bloc pour la gestion de ventilation.

Afin d'éviter les répétitions d'une phase commune aux différents contrôleurs développés, nous avons considéré dans une première étape un exemple sur lequel nous avons détaillé la procédure suivie pour le contrôleur d'opacité mais présenter uniquement les résultats de simulation et d'implémentation pour les autres.

4.3.7.1 Bloc du Contrôleur flou (FLC) :

Ce bloc lui-même est partagé en quatre entités comme suit :

Deux entités de fuzzification pour l'entrée qui est **OP** et sa variation de l'erreur **E_OP**.

Entité d'inférence.

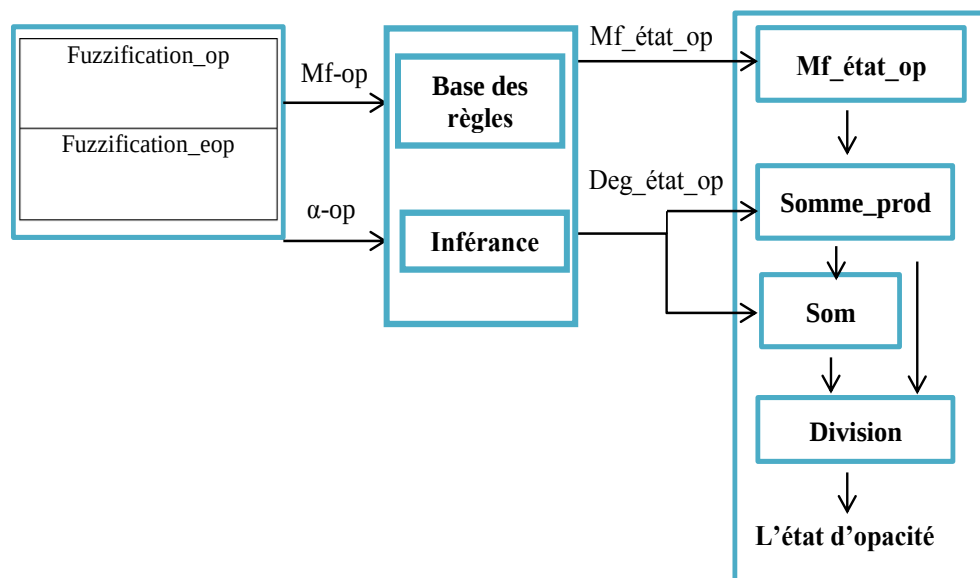


Figure 4-6 Schéma fonctionnel des opérations dans un contrôleur flou.

4.3.7.2 Bloc de fuzzification :

En général, pour fuzzifier, il faut :

- Donner l'univers du discours : plage de variations possibles de l'entrée considérée.
- Une partition en classe floue de cet univers.
- Les fonctions d'appartenances de chacune de ces classes.

4.3.7.3 Fuzzification de l'entrée OP:

4.3.7.4 Les variables de commande : On considère les variables de commande du FLC-OP suivantes :

- L'entrée **op** codé sur 7 bits, elle représente le taux d'opacité.
- L'entrée **e-op** codée sur 7 bits, elle représente la variation du taux d'opacité.
- La sortie **état-op** codée sur 7 bits, elle représente la situation du taux d'opacité à atteindre.

4.3.7.5 La fonction d'appartenance

Il n'y a pas de règle précise pour définir les formes de fonctions d'appartenances alors, pour simplifier les traitements, nous avons choisi des formes triangulaires et/ou trapézoïdales, les fonctions d'appartenances du taux d'opacité et de sa variation sont présentées sur les Figure 4-7 et Figure 4-8 respectivement.

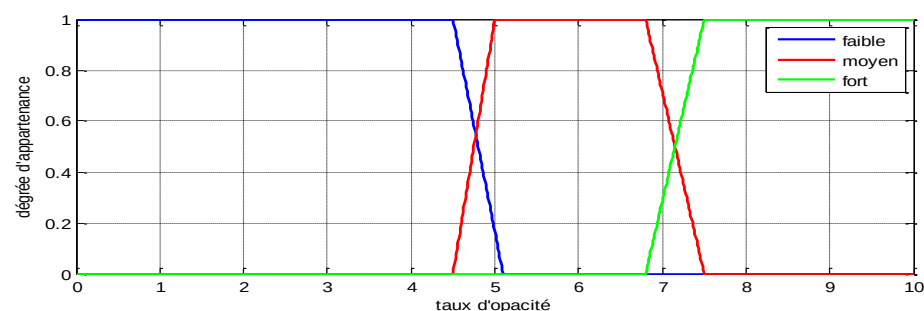


Figure 4-7 Les fonctions d'appartenance du taux d'opacité.

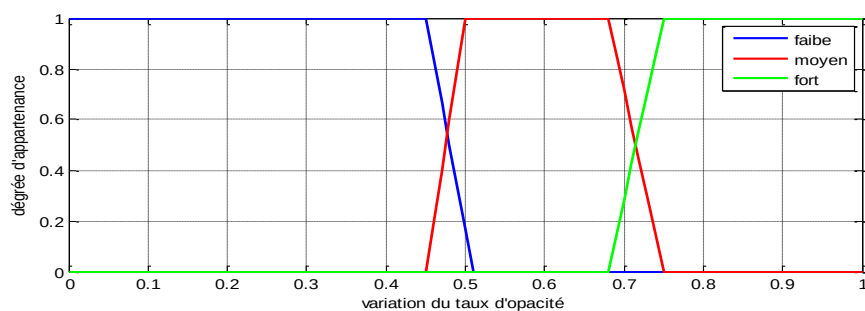


Figure 4-8 Les fonctions d'appartenances de variation du taux d'opacité.

La fonction d'appartenance	Le codage
FB : Faible	001
MOY : Moyen	010
FR : Fort	011

Tableau 4-3 Codage des fonctions d'appartenances

Les fonctions d'appartenance de la sortie (l'état d'opacité) sont présentées sur la Figure 4-9

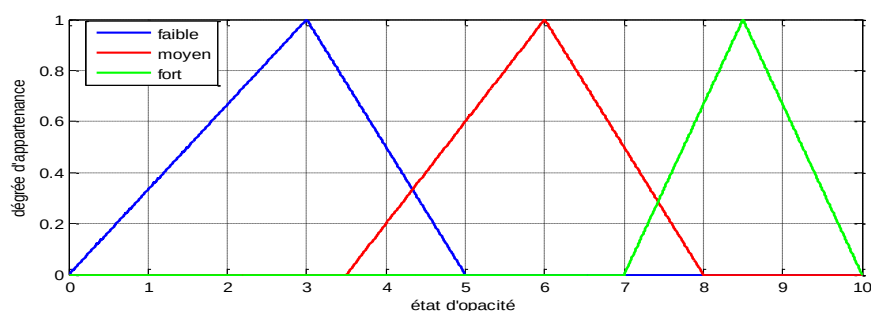


Figure 4-9 Les fonctions d'appartenances de l'état d'opacité.

4.3.7.6 Technique de fuzzification :

Les entrées du système flou sont normalisées et discrétisées suivant la forme des fonctions d'appartenance et la longueur de l'univers de discours du contrôleur flou (triangulaire, trapézoïdale, Gaussienne...) avec deux chevauchements au maximum dans notre cas. Après avoir discrétisé les entrées du système, on définit les tables à deux dimensions qui contiennent les valeurs correspondantes. Les valeurs obtenues sont enregistrées de la manière suivante :

Fx1 :3 bits	Dx1 :3 bits	Fx2 :3 bits	Dx2 :3 bits
-------------	-------------	-------------	-------------

Fx1, Fx2 : sont les fonctions d'appartenance, Dx1, Dx2 : sont leur degré respectif.

Le tableau utilisé dans le programme VHDL du module de fuzzification est décrit de la manière suivante Figure 4-10.

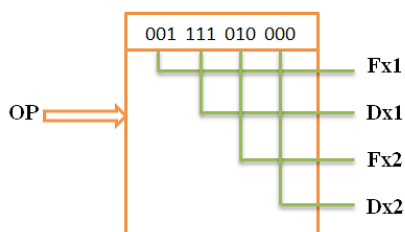


Figure 4-10 Méthode de programmation de la fuzzification

NB : nous avons vu que le circuit FPGA de type spartan 3E suffit pour l'implantation de chaque contrôleur seul mais sa surface devient insuffisante pour l'implantation du contrôleur globale (ce qui nous a mené à choisir le FPGA de type virtex5).

Après élaboration d'un programme en VHDL du module du fuzzification du contrôleur flou d'OP, les résultats de la synthèse (sur FPGA Spartan 3E) sont donnés dans le Tableau 4-4.

Device Utilization Summary (estimated values)				[1]
Logic Utilization	Used	Available	Utilization	
Number of Slices	7	4656		0%
Number of Slice Flip Flops	12	9312		0%
Number of 4 input LUTs	2	9312		0%
Number of bonded IOBs	21	66		31%
Number of BRAMs	1	20		5%
Number of GCLKs	1	24		4%

Tableau 4-4 Résultats de la synthèse du module de fuzzification du FLC_OP.

On remarque que ce module occupe une très petite surface du FPGA La simulation du module sous ISIM simulator est montrée sur la Figure 4-11.

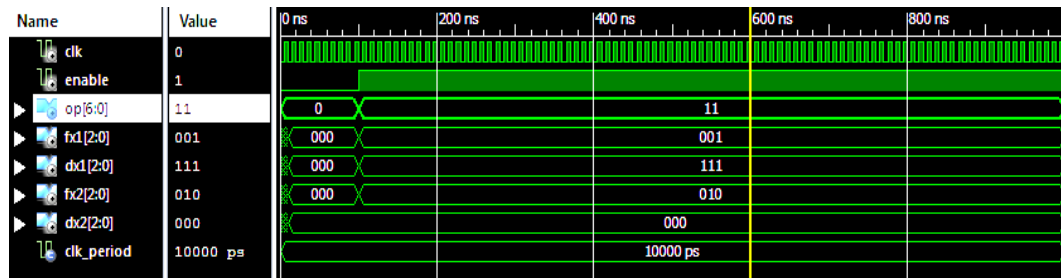


Figure 4-11 Les résultats de la simulation de fuzzification.

On remarque que l'entrée OP est considérée faible (001) avec un degré (111) et moyen (010) avec un degré (000) ; si la valeur d'entrée (mesurées) d'opacité est 11 (0001011).

L'implantation de ce bloc sur FPGA (Spartan 3E) est donnée par la Figure 4-12.

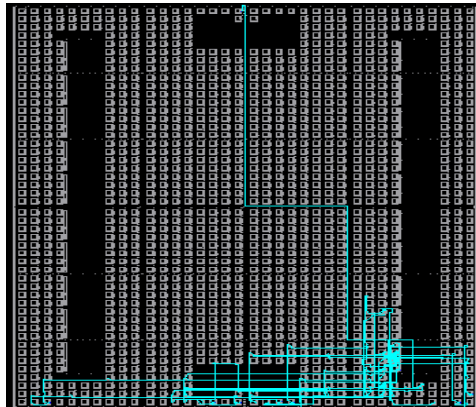


Figure 4-12 L'implantation du module de fuzzification sur FPGA

4.3.8 Module d'inférence

Le bloc d'inférence reçoit les résultats de fuzzification des entrées qui comprend les fonctions d'appartenances et leurs degrés d'appartenances. Il comprend deux systèmes qui fonctionnent en parallèle : le calcul des minimums et la sélection des règles actives.

a) Sélection des règles actives :

Les règles floues représentent le cœur du système flou. La base des règles est celle de Mamdani, les fonctions d'appartenance au nombre de trois pour chaque variable d'entrée au maximum, constitue la matrice des règles (3x3, soient 9 règles au maximum), Tableau (IV.5), Tableau (IV.6).

e-op	op	Faible	Moyen	Fort
Faible		Faible	Moyen	Fort
Moyen		Faible	Moyen	Moyen
Fort		Moyen	Fort	Fort

Tableau 4-5 Base de règles de contrôleur flou

Abréviation	Conclusion	Code binaire
FB	Faible	001
MOY	Moyen	010
FR	Fort	011

Tableau 4-6 Codage des règles.

La configuration ci-dessous des bases de règles a été adoptée où on sélectionne quatre règles actives quasi-simultanément. On adresse ainsi la mémoire des règles par le résultat de la combinaison des fonctions d'appartenance issues du bloc de fuzzification comme le montre la Figure 4-13.

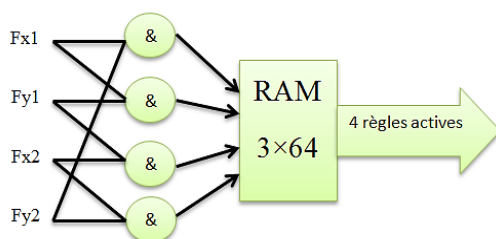


Figure 4-13 Sélection des règles Actives

- b) Engin d'inférence : Pour calculer la valeur correspondante au degré de vérité de chaque règle active, on a utilisé la méthode d'inférence « minimum ». Pour cela, on a utilisé quatre blocs de calcul de « MIN » comme le montre la Figure 4-14.

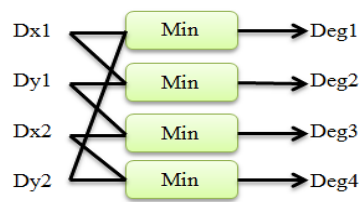


Figure 4-14 Calcul du Minimum

Les avantages de cette méthode sont les suivants :

- Réduction du temps de calcul du degré de vérité en utilisant les quatre blocs de "min".
- Les quatre règles actives et leurs degrés de vérité respectifs sont en sortie au même front d'horloge.

Les résultats de synthèse du module d'inférence sont représentés dans le

Tableau 4-7.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	24	4656	0%
Number of 4 input LUTs	44	9312	0%
Number of bonded IOBs	49	66	74%
Number of GCLKs	1	24	4%

Tableau 4-7 Résultat de synthèse du module d'inférence

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-15.

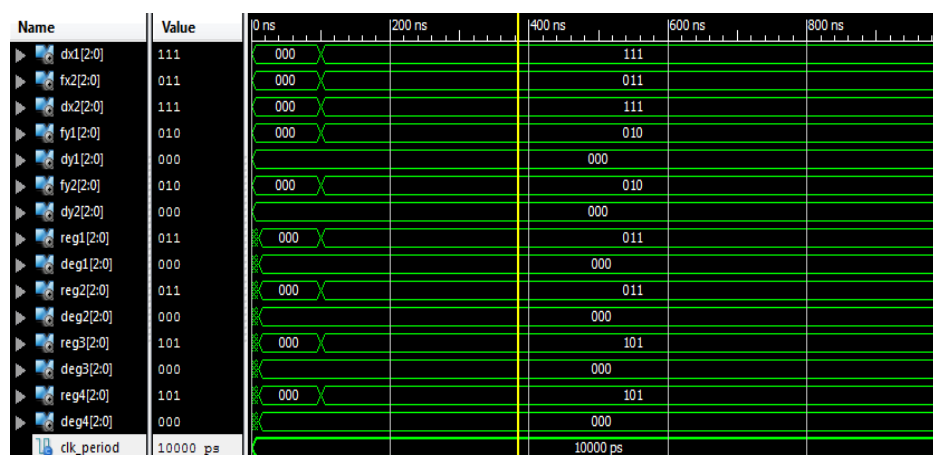


Figure 4-15 Simulation d'engin d'inférence

L'implantation de ce bloc sur FPGA est donnée par la Figure 4-16.

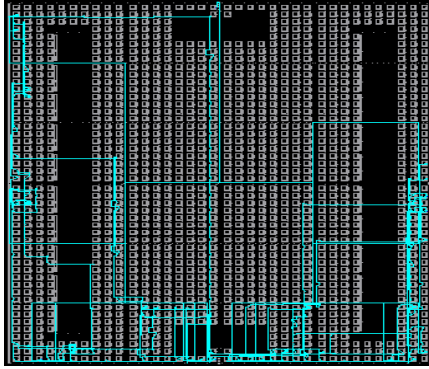


Figure 4-16 L'implantation d'engin d'inférence sur FPGA

Le bloc d'inférence n'occupe qu'une petite proportion du FPGA.

c) Module de défuzzification :

Les calculs exigés dans la défuzzification rendent le temps développement du module de défuzzification très supérieur à celui des autres modules. Dans ce module, on convertit les résultats de l'inférence en valeurs exploitables dans la commande. Il est à rappeler que les performances d'un contrôleur en terme de vitesse et de surface FPGA dépendent de la conception de ce bloc. Il est donc impératif de choisir les algorithmes qui donnent le bon compromis temps/surface. Dans cette étude, on utilise la méthode du centre de gravité (équation 1).

$$u = \frac{\sum_{i=1}^n x_i \mu_i}{\sum_{i=1}^n \mu_i} \quad (1)$$

μ : degré de vérité de la règle active i. x : centre de gravité de la règle active i.

La Figure 4-17 montre la conversion des règles actives résultantes du bloc d'inférence en valeurs réelles.

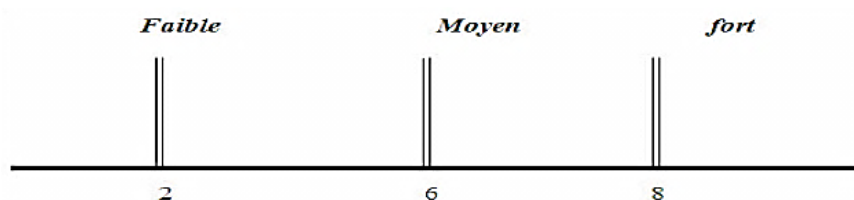


Figure 4-17 La conversion des règles actives résultantes.

Les opérations implémentées dans le module de défuzzification sont : la multiplication, l'addition et la division. (Figure 4-18)

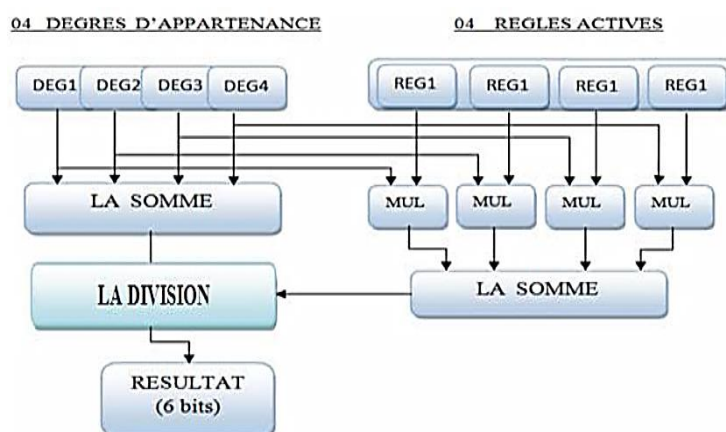


Figure 4-18 La défuzzification par centre de gravité.

Après élaboration d'un programme en VHDL détaillé du module de la défuzzification du contrôleur flou d'opacité, les résultats de la synthèse sont représentés sur le Tableau 4-8:

Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slices	159	4656	3%	
Number of Slice Flip Flops	56	9312	0%	
Number of 4 input LUTs	281	9312	3%	
Number of bonded IOBs	32	66	48%	
Number of GCLKs	1	24	4%	

Tableau 4-8 Résultats de la synthèse du module de la défuzzification du FLC_OP.

On remarque que le bloc de défuzzification occupe une surface grande par rapport aux deux autres blocs (fuzzification et inférence), surtout le nombre d'IOB (48%).

La simulation du contrôleur est montrée sur la Figure 4-19.

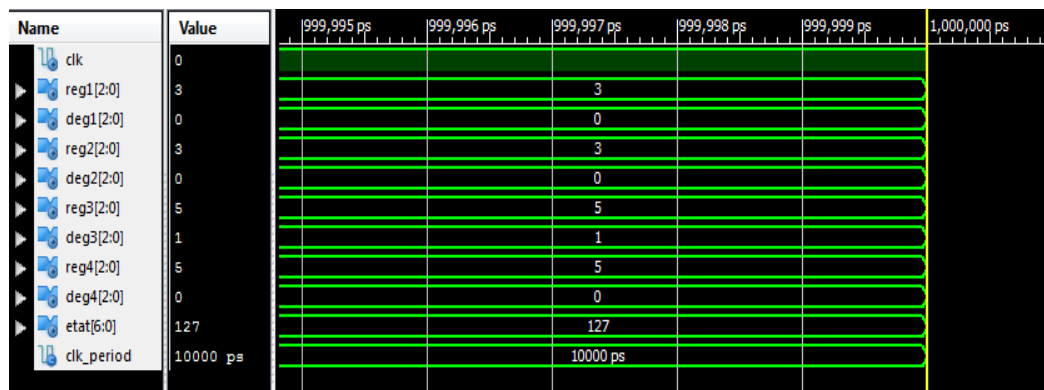


Figure 4-19 Simulation de la défuzzification du FLC_OP.

Résultats de synthèse, de simulation et d'implantation du FLC_OP :

Après élaboration d'un programme en VHDL du contrôleur flou d'opacité, l'exécution de ce programme est montrée dans la Figure 4-20.

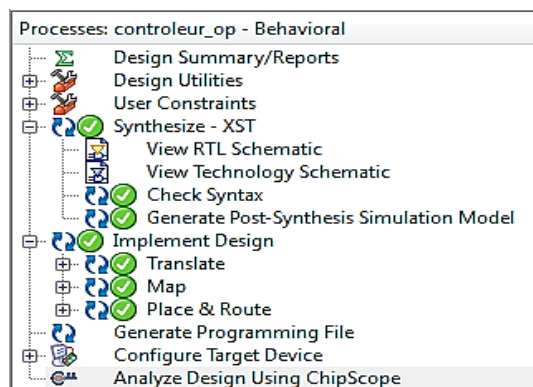


Figure 4-20 Exécution du programme VHDL du contrôleur OP.

Le programme global d'OP (FLC_OP), est constitué de tous les modules précédents (Figure 4-21), la synthèse de ce bloc nous donne les résultats suivants représentés sur le Tableau 4-9.

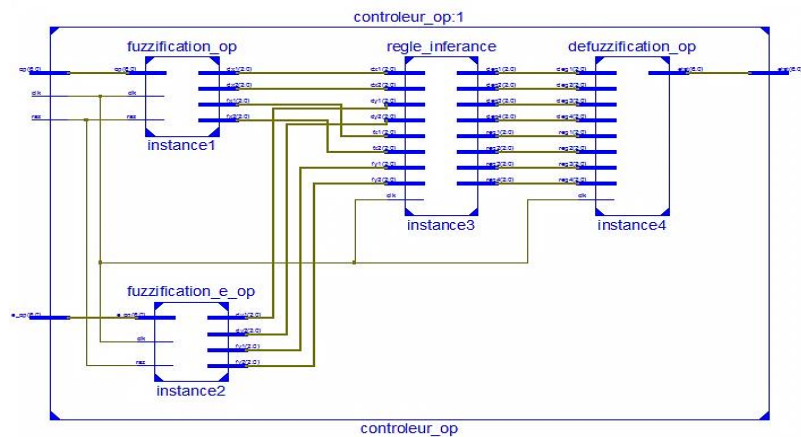


Figure 4-21La structurent du contrôleur d’opacité (FLC_OP).

Famille	Spartan3E (XC3S500E)		Virtex 5 (XC5VLX20T)	
Number of Slices	156/4656	3%	56/12480	0%
Number of Slice Flip Flops	64/9312	0%	205/12480	1%
Number of 4 input LUTs	382/9312	3%	26/247	11%
Number of bondedIOBs	23/66	34%	23/172	13%
Number of BRAMs	2/20	10%	1/26	3%
Number of GCLKs	1/24	4%	1/32	3%
Maximum Frequency	142.074MHz		140.004MHz	

Tableau 4-9Résultats de la synthèse du FLC_OP.

L’analyse des résultats obtenus dans le Tableau 4-9, montre une large étendue de fréquences de travail du contrôleur. Les FPGA de type Spartan 3E consomment plus d’IOB alors que les virtex5 utilisent beaucoup de LUT. La réponse en 142.074MHzpour la Spartan 3E est réellement trouvée sans aucune contrainte de charge, de délai, d’effet d’interconnexion ou de positon de pin d’entrées/sorties sélectionnées à l’avance. Il est connu que les contraintes réduisent presque de moitié les valeurs initiales trouvées.

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-22.

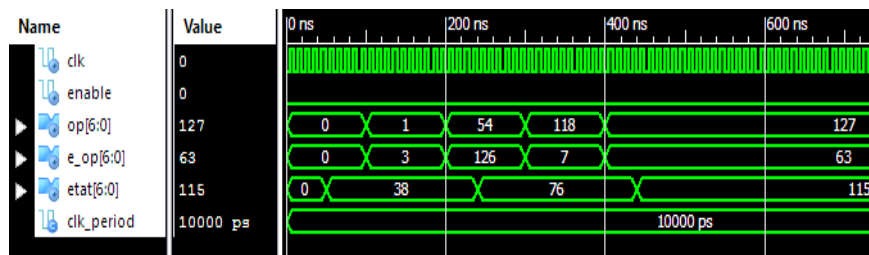


Figure 4-22 Résultat de la simulation du FLC_OP

Dans ce diagramme temporel, on voit que pour un couple opacité-erreur d'opacité(118,7), le contrôleur a produit la sortie(76) vérifiable manuellement.

L'implantation de ce bloc sur FPGA est montrée à la Figure 4-23.

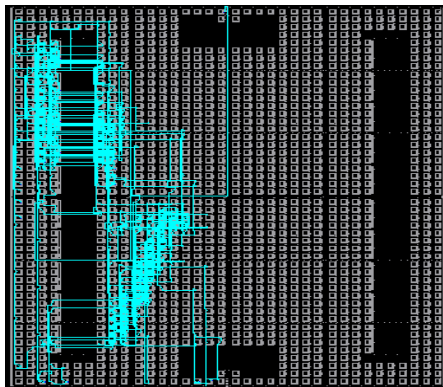


Figure 4-23 Résultats de la simulation du FLC_OP.

4.4 Le contrôleur flou de CO (FLC_CO) :

Le rôle de ce bloc est de contrôler le niveau de CO dans le tunnel. Ses variables de commande sont :

- L'entrée Co qui représente le niveau de CO (de 0 à 90), elle est codée sur 7 bits.
- L'entrée e-co qui représente la variation du niveau de CO (de 0 à 9), elle est codée sur 7 bits.
- La sortie état-Co qui représente la situation du taux de CO à atteindre (de 0 à 90), elle est codée sur 7 bits.

- Les fonctions d'appartenance des variables de la commande sont données en annexe.

Après synthèse, le schéma RTL du contrôleur de CO (FLC_CO) est montré sur la Figure 4-24.

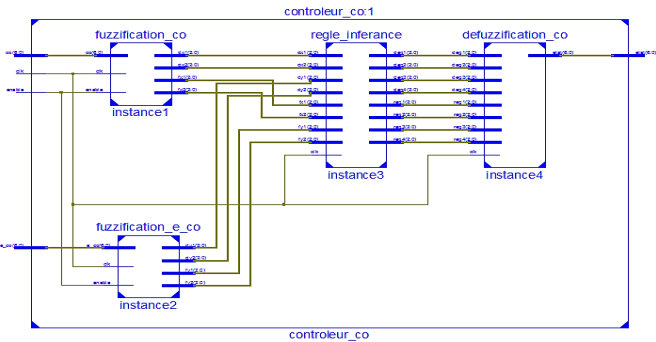


Figure 4-24Le schéma RTL du contrôleur d’OP (FLC_CO)

Les résultats de synthèse du contrôleur CO sur différents FPGA sont donnés dans le tableau suivant.

Famille	Spartan3E	(XC3S500E)	Vertex 5	(XC5VLX20T)
Number of Slices	56/12480	0%	56/12480	0%
Number of Slice Flip Flops	218/12480	1%	218/12480	1%
Number of 4 input LUTs	26/247	10%	26/247	10%
Number of bondedIOBs	23/172	13%	23/172	13%
Number of BRAMs	1/26	3%	1/26	3%
Number of GCLKs	1/32	3%	1/32	3%
Maximum Frequency	139.821MHz		139.821MHz	

Tableau 4-10Résultats de synthèse du contrôleur FLC_CO.

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-25.

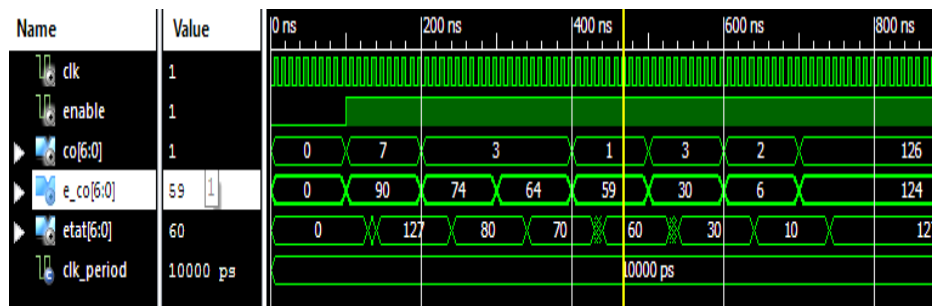


Figure 4-25 Exécution du programme VHDL du contrôleur op

Dans ce diagramme temporel, on voit que pour un couple Co-erreur d'entrée(2,6), le contrôleur a produit la sortie(10) vérifiable manuellement.

L'implantation de ce bloc sur FPGA est montrée à la Figure 4-26.

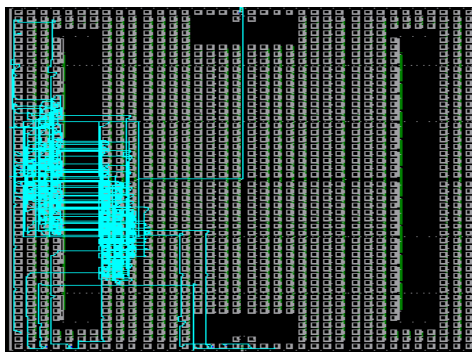


Figure 4-26. L'implantation du contrôleur de CO sur FPGA.

4.5 Le contrôleur flou mixte (FLC_MIXTE) :

Le rôle de ce bloc est de connaître l'état mixte selon l'état du CO et l'état de l'opacité (OP), donc les entrées/sorties du système sont :

- Etat –Co : (codée sur 7 bits) représente la valeur de l'état de monoxyde de carbone. Cette entrée varie dans l'intervalle [0, 90].
- Etat-op : (codée sur 7 bits) représente la valeur de l'état d'opacité. Cette entrée varie dans l'intervalle [0, 10].
- Etat-mixte : (codée sur 7 bits) représente la valeur du régime de ventilation à atteindre. Cette sortie varie dans l'intervalle [0,80].

Les fonctions d'appartenance des entrées et de la sortie sont données en annexe .Après synthèse, le schéma RTL du contrôleur mixte est montré sur la Figure 4-27.

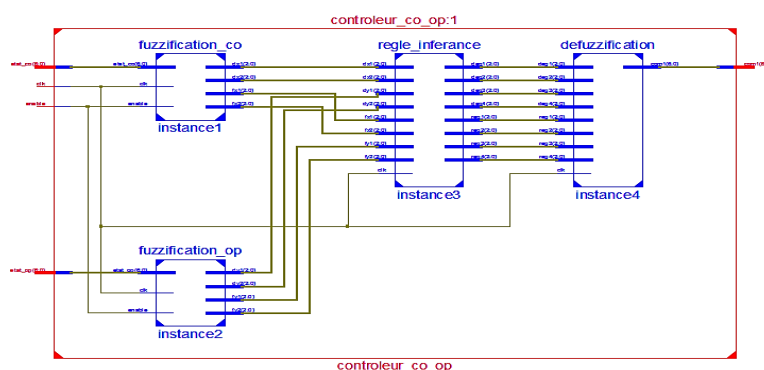


Figure 4-27 Schéma RTL du contrôleur mixte (FLC_MIXTE)

Les résultats de synthèse du contrôleur mixte sur différents FPGA sont donnés dans le tableau suivant.

Famille	Spartan3E (XC3S500E)	Vertex 5 (XC5VLX20T)
Number of Slices	198/4656 4%	56/12480 0%
Number of Slice Flip Flops	116/9312 1%	217/12480 1%
Number of 4 input LUTs	354/9312 3%	26/247 10%
Number of bondedIOBs	23/66 34%	23/172 13%
Number of BRAMs	2/20 10%	1/26 3%
Number of GCLKs	1/24 4%	1/32 3%
Maximum Frequency	142.366MHz	139.821MHz

Tableau 4-11 Résultats de synthèse du contrôleur mixte.

Les FPGA de type spartan 3E ont une réponse avec une fréquence de 142.366 MHz, et virtex5 avec 139.821 MHz, donc il y a une large étendue de fréquence de travail du contrôleur.

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-28.

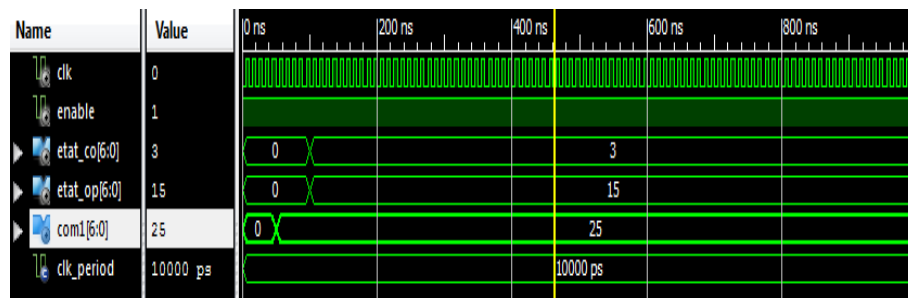


Figure 4-28 Résultats de la simulation du contrôleur mixte.

Dans ce diagramme temporel, on voit que pour un couple état-Co-état-op(3,15), le contrôleur a produit la sortie(25) vérifiable manuellement.

L'implantation de ce bloc sur FPGA est montrée à la Figure 4-29.

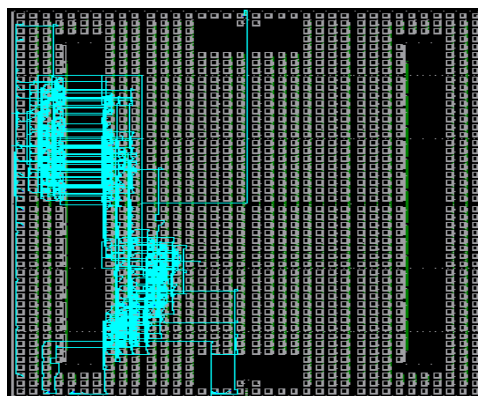


Figure 4-29 L'implantation du contrôleur mixte sur FPGA

4.6 Contrôleur flou de Trafic (circulation des véhicules)

Le contrôleur de trafic donne en sortie l'état de trafic à atteindre selon les seuils préétablis et la variation de la valeur mesurée dans le tunnel.

- **Trafic** : (codée sur 7 bits) représente la valeur du trafic (Circulation des Véhicules).

Cette entrée varie dans l'intervalle [0, 400].

- **E-Trafic** : (codée sur 5 bits) représente la valeur de l'erreur de ce trafic.

Cette entrée varie dans l'intervalle [0, 40].

- **L'état de Trafic :** (codée sur 7 bits) représente la valeur de l'état du trafic à atteindre.

Cette sortie varie dans l'intervalle [0,400].

Après synthèse, le schéma RTL du contrôleur de la ventilation est montré sur la Figure 4-30.

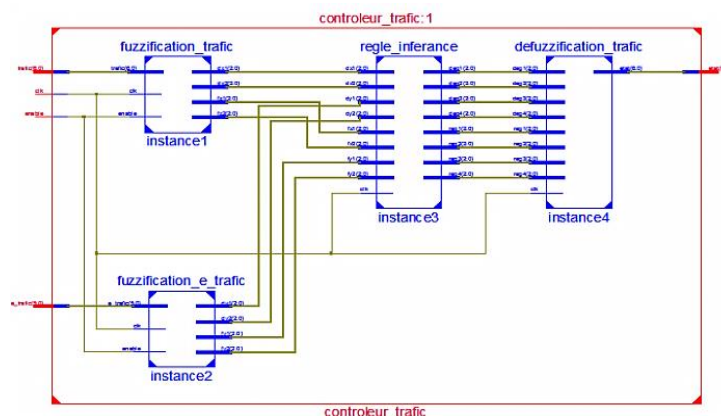


Figure 4-30 Schéma RTL du contrôleur trafic (FLC-Trafic)

Les résultats de synthèse du contrôleur trafic sur différents FPGA sont donnés dans le suivant.

Famille	Spartan3E (XC3S500E)		Vertex 5 (XC5VLX20T)	
Number of Slices	177/4656	3%	68/12480	0%
Number of Slice Flip Flops	64/9312	0%	230/12480	1%
Number of 4 input LUTs	306/9312	3%	40/258	15%
Number of bonded IOBs	22/66	33%	22/172	12%
Number of BRAMs	2/20	10%	1/26	3%
Number of GCLKs	1/24	4%	1/32	3%
Maximum Frequency	142.366MHz		139.821MHz	

Tableau 4-12 Résultats de synthèse du contrôleur Trafic.

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-31.

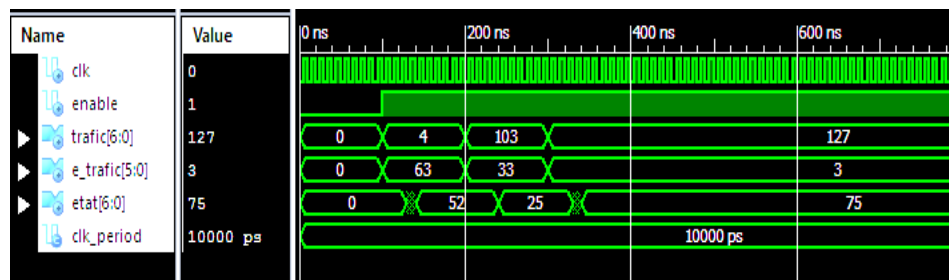


Figure 4-31 Résultats de la simulation du contrôleur trafic.

Dans ce diagramme temporel, on voit que pour un couple d'entrée trafic-erreur trafic (127,3), le contrôleur a produit la sortie(75) vérifiable manuellement.

L'implantation de ce bloc sur FPGA est montrée à la Figure 4-32.

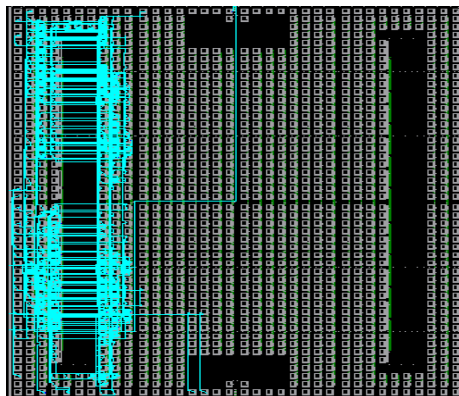


Figure 4-32 L'implantation du contrôleur trafic sur FPGA

4.7 Contrôleur final :

Ce contrôleur nous permet d'avoir l'état final à partir de l'état de **Co**, **op** et de l'état du trafic.

- **Etat- mixte** : (codée sur 7 bits) représente la valeur du régime de ventilation trouvé précédemment .Cette entrée varie dans l'intervalle [0, 90].
- **Etat-Trafic** : (codée sur 7 bits) représente la valeur de l'état du trafic. Cette entrée varie dans l'intervalle [0, 400].
- **Commande 2** : (codée sur 6 bits) représente la valeur du régime de ventilation final à atteindre. Cette sortie varie dans l'intervalle [0,63].

Le schéma RTL du contrôleur final (FLC-final) est montré sur la Figure 4-33.

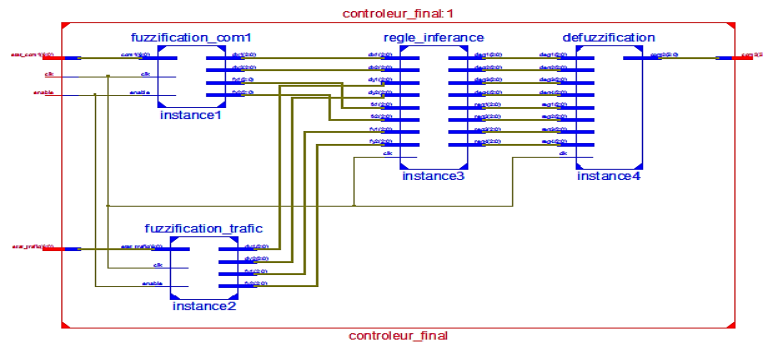


Figure 4-33 Le schéma RTL du contrôleur final (FLC-Final)

Les principaux résultats du rapport de la synthèse sont présentés dans le tableau suivant.

Famille	Spartan3E (XC3S500E)		Vertex 5 (XC5VLX20T)	
Number of Slices	157/4656	3%	56/12480	0%
Number of Slice Flip Flops	63/9312	0%	190/12480	1%
Number of 4 input LUTs	280/9312	3%	26/220	11%
Number of bonded IOBs	22/66	33%	22/172	12%
Number of BRAMs	2/20	10%	1/26	3%
Number of GCLKs	1/24	4%	1/32	3%
Maximum Frequency	145.699MHz		140.368MHz	

Tableau 4-13 Résultats de la synthèse du contrôleur final.

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-33.

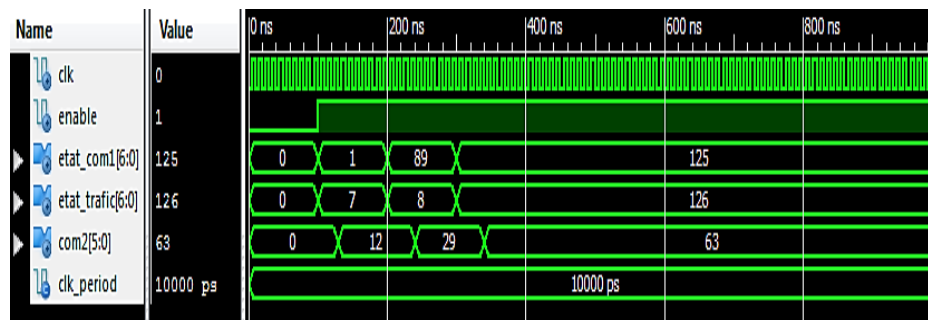


Figure 4-34 Résultats de la simulation du contrôleur final.

Dans ce diagramme temporel, on voit que pour un couple d'entrée `etat_com1` et `etat_trafic` (125,126), le contrôleur a produit la sortie(63) vérifiable manuellement.

L'implantation de ce bloc sur FPGA est montrée à la Figure 4-35.

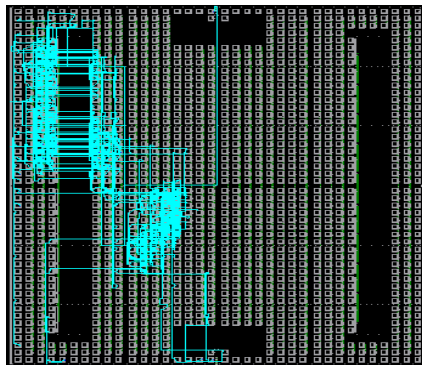


Figure 4-35 L'implantation du contrôleur final sur FPGA

4.7.1 La méthode de ventilation pour chaque zone :

C'est le module qui nous donne la répartition des ventilateurs actifs le long de tunnel.

La répartition des ventilateurs nécessite une étude de la dynamique de l'aire dans le tunnel afin de garantir le confort et d'optimiser l'énergie ; et comme il est difficile de garantir cette étude, alors nous propose une répartition qui suit le changement de taux du CO le long du tunnel.

Il y a plusieurs méthodes de ventilation, parmi ces méthodes (M1) pour la zone 1.

Régime	L'état	Répartition proposée	Temps
Régime_0	Très faible et faible	(000, 000, 000, 000)	0
Régime_1	Moyen	(010, 010, 010 ,010)	T1
Régime_2	Fort	(010, 010, 101, 101)	T2
Régime_3	Très fort	(101, 111, 111, 111)	T3
Régime_4	Critique	(111, 111,111, 111)	T3

Tableau 4-14Méthode de ventilation

La synthèse de ce module, nous génère le schéma RTL présenté dans la Figure 4-36.

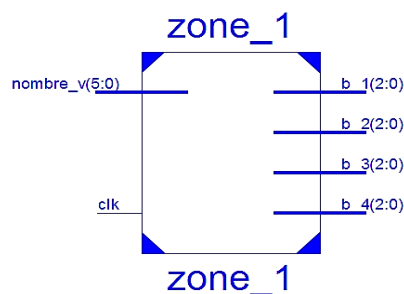


Figure 4-36Le schéma RTL du module "répartition-vent" de la zone 1.

les résultats de la synthèse sont classés dans le Tableau 4-15Résultats de la synthèse du module " répartition-vent "de la zone 1:

Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slices	12	4656	0%	
Number of 4 input LUTs	22	9312	0%	
Number of bonded IOBs	18	66	27%	

Tableau 4-15Résultats de la synthèse du module " répartition-vent "de la zone 1

Ce module occupe une très petite surface du FPGA.

L'implantation de ce module sur FPGA est montrée sur la Figure 4-37 Implantation du module de répartition sur FPGA de type Spartan 3E..

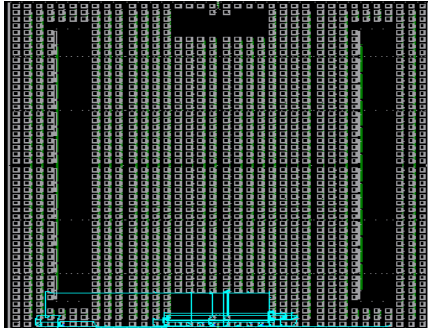


Figure 4-37 Implantation du module de répartition sur FPGA de type Spartan 3E.

4.8 Le contrôleur global :

Après le développement des cinq contrôleurs précédents, nous allons les interconnectés (instanciation) pour construire un contrôleur globale qui a les entrées sorties suivantes :

- **Co** et **e-co** : (codées sur 7 bits) représente la valeur du taux du **Co** et de sa variation **e-co** : respectivement, elles sont les entrées du première contrôleur FLC_CO.
- **Op** et **e-op** : (codée sur 7 bits) représente la valeur du niveau de l'opacité OP et de sa variation **e-co**, elles sont les entrées du deuxième contrôleur FLC_OP.
- **Trafic** et **é-traffic** : (codée sur 7 bits et 6 bits respectivement) représente la valeur du couple de trafic et erreur de trafic.
- **Stratégie-ventilation** : représente le nombre de ventilateur dans chaque batterie, elle représente la sortie du contrôleur global.

Après La synthèse du contrôleur globale, nous génère le schéma RTL présenté dans la Figure 4-38Le schéma RTL du contrôleur global..

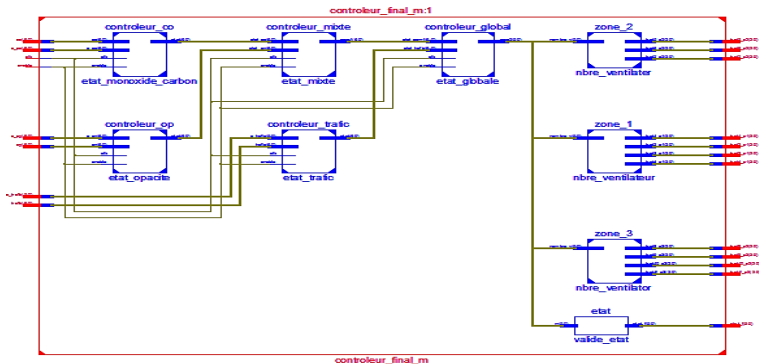


Figure 4-38Le schéma RTL du contrôleur global.

Après l’instanciation de tous les modules de notre contrôleur, la synthèse nous génère le rapport illustré sur le tableau suivant.

Famille	Vertex 5 (XC5VLX20T)	
Number of Slices	56/12480	2%
Number of Slice Flip Flops	190/12480	11%
Number of 4 input LUTs	26/220	8%
Number of bonded IOBs	22/172	44%
Number of BRAMs	1/26	19%
Number of GCLKs	1/32	3%
Number of DSP-48Es	15/24	62%
Maximum Frequency	78.368MHz	

Tableau 4-16Résultats de la synthèse d’un contrôleur global.

La simulation du contrôleur sous ISIM est montrée sur la Figure 4-39. Résultats de la simulation du contrôleur global..

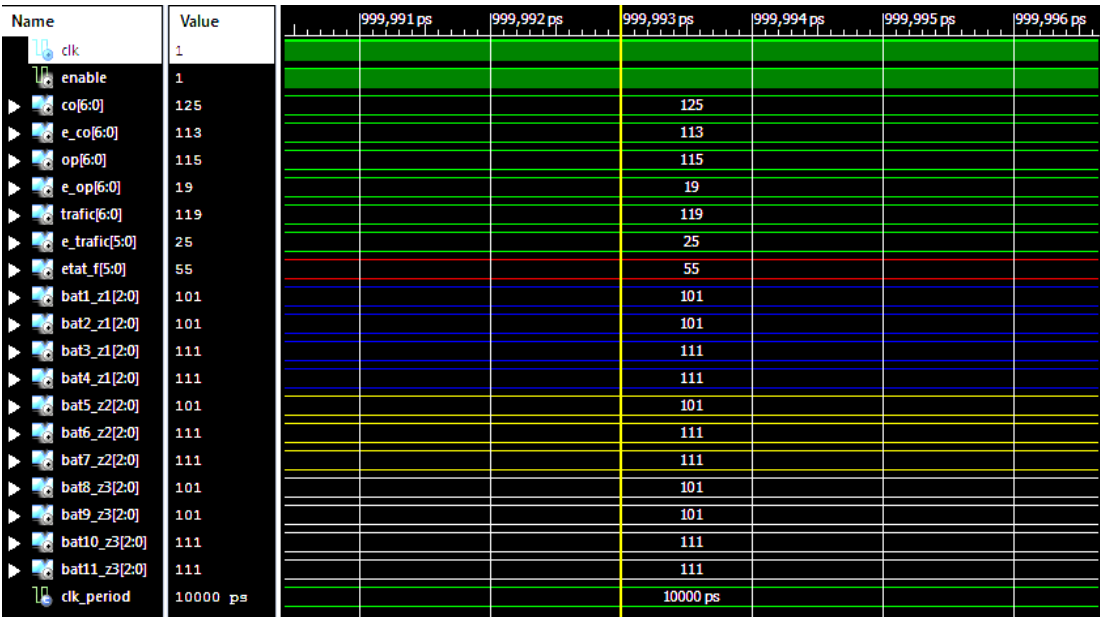


Figure 4-39 Résultats de la simulation du contrôleur global.

L'implémentation du contrôleur sur un FPGA de type Virtex5 est donnée sur la Figure 4-40.

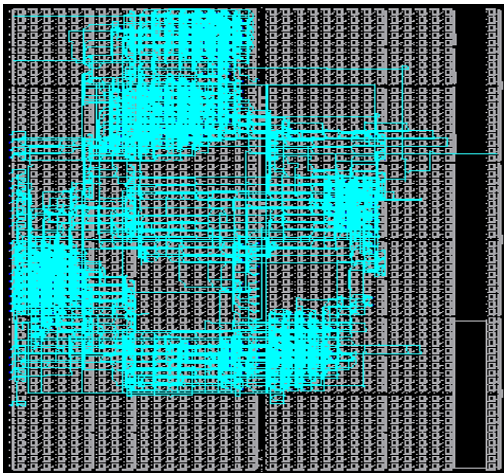


Figure 4-40 L'implémentation du contrôleur global sur un FPGA de type Virtex5.

4.9 Impacts et effets de la reconfiguration :

L'étude du comportement du contrôleur développé est effectuée à différents niveaux d'expérimentation, moyennant l'utilisation extensive du FLC mis au point par l'équipe de recherche de notre laboratoire [31]. Pour montrer les effets et impacts sur la structure d'une part, et recenser les avantages et gains récoltés d'autre part, de l'action de reconfiguration, nous avons estimé que la meilleure évaluation reposerait sur les facteurs : énergie consommée et confort produit et. Donc évaluer la configuration à travers une application et non la considération d'un cas général car une étude détaillée du système de ventilation en question a montré que tous les changements pouvant être opérés sur les structures des différents contrôleurs assurant la commande de ventilation couvre toutes les actions sujettes au renforcement du contrôle qui préserve efficacement les ressources exploitables.

Comme nous avons voulu soumettre le contrôleur original (sans reconfiguration) à toutes les épreuves de reconfiguration afin d'établir les procédures dont l'impact est positif c'est-à-dire vérifier et quantifier l'apport de notre approche.

Les étapes suivies pour réaliser cette activité sont définies ci-dessous :

1. Notre approche sera appliquée seulement sur quatre contrôleurs : FLC_OP, FLC_CO, FLC_trafic et FLC_global.
2. La reconfiguration modifie les blocs de fuzzification, inférence et défuzzification de chaque FLC.
3. On présente d'abord les résultats de la synthèse de chaque contrôleur sans reconfiguration dans un tableau (mode1).
4. On représente ensuite les résultats de la synthèse de chaque contrôleur dans un tableau équivalent pour différents contextes considérés où chaque contexte correspond à une reconfiguration spécifique donnée.
5. Nous discutons l'effet de la reconfiguration sur le fonctionnement (ex : fréquence maximale de fonctionnement) de chaque FLC.

NB : Dans notre étude, nous avons constaté que toutes les approches de defuzzification se rejoignent autour de caractéristiques similaires : surface, temps, stabilité, alors notre contribution s'est penché sur la configuration au sein d'une même technique, celle du centre de gravité dans notre cas.

4.9.1 L'apport de la reconfiguration du bloc de fuzzification sur le FLC_OP :

	Sans reconfiguration Valeur sur 7 bits	Avec reconfiguration (Valeur sur les 4 bits du poids faible)
Nbr of Slices	56/12480	40/12480
Nbr of F Flops	205/12480	152/12480
Nbr of 4 LUTs	26/247	26/247
Nbr of IOBs	23/172	23/172
Nbr of BRAMs	1/26	1/26
Nbr of GCLKs	1/32	1/32
Maximum Frequency	140.004MHz	163.654MHz

Tableau 4-17L'apport de la reconfiguration sur le bloc de fuzzification du FLC_CO

4.9.2 L'apport de la reconfiguration du bloc de fuzzification sur le FLC_CO :

	Sans reconfiguration Valeur sur les 7 bits	Avec reconfiguration (Valeur sur les 4 bits du poids faible).
Nbr of Slices	56/12480	40/12480
Nbr of F Flops	218/12480	180/12480
Nbr of 4 LUTs	26/247	20/247
Nbr of IOBs	23/172	23/172
Nbr of BRAMs	1/26	1/26
Nbr of GCLKs	1/32	1/32
Maximum Frequency	139.821MHz	160.544MHz

Tableau 4-18L 'apport de la reconfiguration sur le bloc de fuzzification du FLC_CO

4.9.3 L'apport de la reconfiguration du bloc de fuzzification sur le FLC_trafic

	Sans reconfiguration Valeur sur les 7 bits	Avec reconfiguration (Valeur sur les 4 bits du poids faible).
Nbr of Slices	68/12480	32/12480
Nbr of F Flops	230/12480	143/12480
Nbr of 4 LUTs	40/258	36/247
Nbr of IOBs	22/172	22/172
Nbr of BRAMs	1/26	1/26
Nbr of GCLKs	1/32	1/32
Maximum Frequency	139.821MHz	162.832 MHz

Tableau 4-19L 'apport de la reconfiguration du bloc de fuzzification sur le FLC de trafic.

4.9.4 L'apport de la reconfiguration du bloc de fuzzification sur le FLC global :

	Sans reconfiguration Valeur sur les 7 bits	Avec reconfiguration (Valeur sur les 4 bits du poids faible).
Nbr of Slices	56/12480	42/12480
Nbr of F Flops	205/12480	164/12480
Nbr of 4 LUTs	26/247	22/247
Nbr of IOBs	23/172	23/172
Nbr of BRAMs	1/26	1/26
Nbr of GCLKs	1/32	1/32
Maximum Frequency	78,368MHz	90.664MHz

Tableau 4-20 L'apport de la reconfiguration du bloc de fuzzification sur le FLC global.

Il est clair selon les tableaux que les résultats obtenus pour une reconfiguration du bloc de fuzzification des contrôleurs FLC_OP, FLC_CO, FLC_trafic et FLC_global montrent que le module de fuzzification du contrôleur occupe une surface plus petite avec une augmentation significative de la fréquence maximale de fonctionnement.

4.9.5 L'apport de la reconfiguration du bloc d'inférence sur le FLC_OP :

	Sans reconfiguration	2 règles	8 règles	16 règles
Nbr of Slices	56/12480	28/12480	114/12480	230/12480
Nbr of F Flops	205/12480	112/12480	385/12480	788/12480
Nbr of 4 LUTs	26/247	24/247	62/247	128/247
Nbr of IOBs	23/172	23/172	23/172	23/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	1/32
Maximum Frequency	140.004MHz	162.104MHz	122.731MHz	99.214MHz

Tableau 4-21 L'apport de la reconfiguration du bloc d'inférence du FLC_OP.

4.9.6 L'apport de la reconfiguration du bloc d'inférence sur le FLC_CO :

	Sans reconfiguration	2 règles	8 règles	16 règles
Nbr of Slices	56/12480	28/12480	112/12480	228/12480
Nbr of F Flops	218/12480	108/12480	400/12480	812/12480
Nbr of 4 LUTs	26/247	14/247	60/247	123/247
Nbr of IOBs	23/172	23/172	23/172	23/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	2/32
Maximum Frequency	139.821MHz	159.234MHz	124.054MHz	124.054MHz

Tableau 4-22L'apport de la reconfiguration du bloc d'inférence du FLC_CO.

4.9.7 L'apport de la reconfiguration du bloc d'inférence sur le FLC du trafic :

	Sans reconfiguration	2 règles	8 règles	16 règles
Nbr of Slices	68/12480	34/12480	142/12480	262/12480
Nbr of F Flops	230/12480	122/12480	462/12480	798/12480
Nbr of 4 LUTs	40/258	30/247	75/247	146/258
Nbr of IOBs	22/172	22/172	22/172	22/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	2/32
Maximum Frequency	139.821MHz	156.604MHz	128.842MHz	101.331MHz

Tableau 4-23L'apport de la reconfiguration du bloc d'inférence sur le FLC du Trafic.

4.9.8 L'apport de la reconfiguration du bloc d'inférence sur le FLC de global :

	Sans reconfiguration	2 règles	8 règles	16 règles
Nbr of Slices	56/12480	28/12480	118/12480	236/12480
Nbr of F Flops	205/12480	104/12480	418/12480	840/12480
Nbr of 4 LUTs	26/247	20/247	38/247	75/247
Nbr of IOBs	23/172	23/172	23/172	23/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	1/32
Maximum Frequency	78,368MHz	88.112MHz	72.556MHz	69,001MHz

Tableau 4-24L 'apport de la reconfiguration du bloc d'inférence sur le FLC globale.

Les résultats obtenus pour une reconfiguration du bloc d'inférence des contrôleurs FLC_OP, FLC_CO, FLC_trafic et FLC_global montrent que : le module d'inférence occupe une surface plus petite et l'exécution se fait avec une fréquence plus élevée, si le nombre de règles est petit. Par contre, il est clair qu'on va assurer plus de précision de commande lors de l'augmentation de nombres de règles, mais le module d'inférence nécessite une surface plus grande et la fréquence de travail diminue de manière très prépondérante.

4.9.9 L'apport de la reconfiguration du bloc de défuzzification sur le FLC_OP :

	Sans reconfiguration	2 règles actives	8 règles Actives	16 règles actives
Nbr of Slices	56/12480	28/12480	114/12480	230/12480
Nbr of F Flops	205/12480	112/12480	385/12480	788/12480
Nbr of 4 LUTs	26/247	24/247	62/247	128/247
Nbr of IOBs	23/172	23/172	23/172	23/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	1/32
Maximum Frequency	140.004MHz	163.128MHz	124.003MHz	100.564MHz

Tableau 4-25L 'apport de la reconfiguration du bloc d'inférence du FLC_OP.

4.9.10 L'apport de la reconfiguration du bloc de défuzzification sur le FLC_CO :

	Sans reconfiguration	2 règles actives	8 règles Actives	16 règles actives
Nbr of Slices	56/12480	28/12480	112/12480	228/12480
Nbr of F Flops	218/12480	108/12480	400/12480	812/12480
Nbr of 4 LUTs	26/247	14/247	60/247	123/247
Nbr of IOBs	23/172	23/172	23/172	23/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	2/32
Maximum Frequency	139.821MHz	160.222MHz	125.434MHz	125.124MHz

Tableau 4-26L 'apport de la reconfiguration sur le bloc de défuzzification du FLC_CO

4.9.11 L'apport de la reconfiguration du bloc de défuzzification sur le FLC de trafic :

	Sans reconfiguration	2 règles actives	8 règles Actives	16 règles actives
Nbr of Slices	68/12480	34/12480	142/12480	262/12480
Nbr of F Flops	230/12480	122/12480	462/12480	798/12480
Nbr of 4 LUTs	40/258	30/247	75/247	146/258
Nbr of IOBs	22/172	22/172	22/172	22/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	2/32
Maximum Frequency	139.821MHz	158.103MHz	131.132MHz	102.911MHz

Tableau 4-27L'apport de la reconfiguration du bloc de défuzzification sur le FLC de trafic.

4.9.12 L'apport de la reconfiguration du bloc de défuzzification sur le FLC global :

	Sans reconfiguration	2 règles actives	8 règles Actives	16 règles actives
Nbr of Slices	56/12480	28/12480	118/12480	236/12480
Nbr of F Flops	205/12480	104/12480	418/12480	840/12480
Nbr of 4 LUTs	26/247	20/247	38/247	75/247
Nbr of IOBs	23/172	23/172	23/172	23/172
Nbr of BRAMs	1/26	1/26	1/26	1/26
Nbr of GCLKs	1/32	1/32	1/32	1/32
Maximum Frequency	78,368MHz	92.883MHz	70.501MHz	69,401MHz

Tableau 4-28L'apport de la reconfiguration du bloc de défuzzification sur le FLC global.

Les résultats obtenus pour une reconfiguration du bloc défuzzification des contrôleurs FLC_OP, FLC_CO, FLC_trafic et FLC_global montrent que : le bloc de défuzzification occupe une surface plus petite et la fréquence de fonctionnement devienne plus élevée, si le nombre de règles est petit. Par contre, on va assurer plus de précision de commande lors de l'augmentation de nombres de règles, mais le module d'inférence nécessite une surface plus grande et la fréquence de travail diminue significativement.

4.10 Conclusion

Le système de la ventilation développée est implanté sur une carte FPGA de type Virtex5 (XCS3S200). Il fonctionne avec une fréquence de 78.755 Mhz et occupe 11% de la surface du FPGA.

Selon les résultats obtenus pour une reconfiguration du bloc de fuzzification on peut démontrer que la reconfiguration a un apport positif sur le fonctionnement du module de fuzzification car il devient plus rapide (augmentation de la fréquence de fonctionnement), et occupe moins de surface sur le FPGA.

La reconfiguration du module d'inférence et de défuzzification a permis de prouver l'effet avantageux agissant sur les performances de la commande et l'augmentation significative de la précision lors de l'augmentation de nombre de règles et le module d'inférence nécessite une surface plus grande, alors que la fréquence de travail diminue de manière très prépondérante. Par contre le module d'inférence occupe une surface plus petite et l'exécution se fait avec une fréquence plus élevée, si le nombre de règles est petit.

Conclusion générale et Perspectives

Cette étude basée sur les techniques de reconfiguration et exploitant VHDL comme outil de programmation à l'échelle comportementale et structurelle pour concevoir un système flou, a montré qu'une nouvelle version et stratégie s'introduite pour la gestion des ressources par l'inclusion de la reconfiguration au niveau software et hardware et prenant en charge très tôt dans le concept de développement les partitionnements et les modules à restructurer. Une solution au problème délicat posé par la gestion du fonctionnement d'un contrôleur flou composé de plusieurs IP soit dans une même puce ou dans des puces différentes. Elle peut être qualifiée de solution simple mais porteuse, car elle agit uniquement sur la fonctionnalité et évite tout recours à la restructuration des IPs, et son impact est immédiat indépendamment du nombre d'IPs et de leurs activités spécifique.

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

Conception d'un contrôleur flou reconfigurable au niveau du chemin de données .On note aussi le caractère générique utilisé dans le différent contrôleur constituant le système pour assurer la souplesse de la programmation, portabilité du produit et permettre des extensions à volonté.

La mise au point de la reconfiguration est réussie grâce au bon partitionnement en blocs fonctionnels .On tire une conclusion importante en qualifiant que le partitionnement est la clé de la reconfiguration mais l'optimisation globale des ressources se complique.

La reconfiguration d'un système complet et de contrôleurs flous est possible par un gestionnaire interne ou externe.

Cet ensemble de règles a permis l'implantation d'un système flou qui a une architecture reconfigurable au niveau des éléments d'interconnexion.

Bibliographie

[1] Nikolaos S. Voros Intracom S.A., Patra, Greece and Kontantinos Masselos " System Level Design of Reconfigurable Systems-on-Chip," Imperial College of Science Technology and Medicine, London, U.K,2005.

[2] Yale Patt, Jim Smith and Mateo Valero " Fine- and Coarse-Grain Reconfigurable Computing," Technical University Delft, The Netherlands, 2007.

[3] Scott Hauck and Andre ´ Dehon " Reconfigurable Computing the theory and Practice of FPGA based Computation," Elsevier,2008.

[4] Christophe Bobda" Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications ," University of Kaiserslautern, Germany, c 2007 Springer.

[5] Yoonjin Kim Rabi N. Mahapatra " Design of Low-Power Coarse-Grained Reconfigurable Architectures ," © 2011 by Taylor and Francis Group, LLC.

[6] Shaon Yousuf, Adam Jacobs, Ann Gordon-Ross " Partially Reconfigurable System-on-Chips for Adaptive Fault Tolerance ," NSF Center for High-Performance Reconfigurable Computing (CHREC) Department of Electrical and Computing Engineering, University of Florida, Gainesville, FL 32611, ©2011 IEEE.

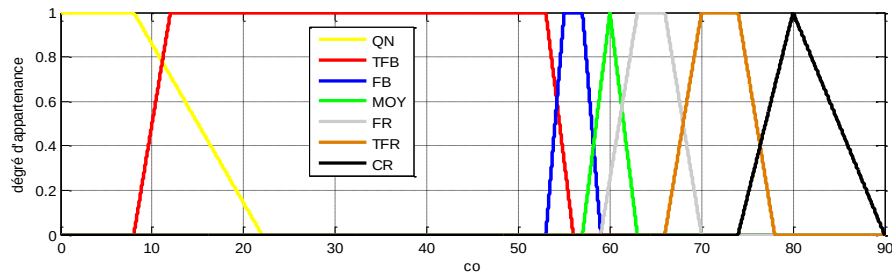
- [7]** Partial Reconfiguration User Guide,Xilinx,Inc.
http://www.xilinx.com/support/documentation/user_guides/ug702.pdf, UG702 (v12.3) October 5, 2010.
- [8]** Michaël Grand " Conception d'un crypto-système reconfigurable pour la radio logicielle sécurisée," Thèse de Doctorat Présentée à L'université de Bordeaux1, Ecole Doctorale Des Sciences Physiques de L'ingénieur, Soutenue le : 02 Décembre 2011.
- [9]** Pauls. Graham Maya Gohale " Reconfigurable Computing Accelerating Computation with Field-Programmable Gate Arrays," Los Alamos National Laboratory, NM, U.S.A.© 2005 Springer.
- [10]** Feijun Song & Samuel M. Smith. "Takagi-Sugeno Type Fuzzy logic Controller with only 3 RULEs for a 4 Dimensional Inverted Pendulum System", IEEE Int. Conf. System,Man & Cybernetics, pp.3800-3805, 2000.
- [11]** E.J.McDonald, Runtime FPGA Partial Reconfiguration. Aerospace Conference, 2008 IEEE,Los Angeles,2008,pp1-7.
- [12]** Robert Fuller, "Fuzzy Reasoning and Fuzzy Optimization," TUCS General Publications, No. 9, Turku Centre for Computer Science, Abo, 1998.
- [13]** M. Ibrahim, Fuzzy Logic for Embedded Systems Applications, Elsevier Science, MA,USA, 2004.
- [14]** C. W. Silva, Intelligent Control-Fuzzy Logic Application, CRC, Boca Raton, FL, 1995.

- [15]** Lotfi A.Zadah, "Fuzzy Logic," University of California, Berkeley, 1989.
- [16]** Prasad, Ram., "Fuzzy Logic Control," class handout, New Mexico State University, Electrical & computer Engineering, 1996.
- [17]** Peri, Vamsi Mohan, "Fuzzy Logic Controller for an Autonomous Mobile Robot," Jawaharlal Nehru Technological University, India May, 2002.
- [18]** Maxfield, Clive, "The Design Warrior's Guide to FPGAs," Newnes, 2004.
- [18]** Balasaheb S. Darade, Abhishek Singh Chauhan and Tarun A.Parmar, "Paper Presentation On Programming FPGA's Using Handel-C," Jawaharlal Nehru Engineering College Aurangabad.
- [19]** Xilinx Inc. "Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs," XAPP462 (V1.1), Jan 2006.
- [20]** K. Mabasa, Mohamed Akil, Thierry Grandpierre, B.J. Van Wyk and M.A. Van Wyk "Automatic VHDL Code Generation for Fuzzy Logic Systems," African Journal of Science and Technology, 2008.
- [21]** Gersnoviez and M. Brox, "Using Xfuzzy Environment for the Whole Design of Fuzzy Systems," Proc. IEEE International Conference on Fuzzy Systems, London, July 23-26, 2007.
- [22]** D. R. Lopez, S. Sanchez-Solano, and A. Barriga, "Xfuzzy: A Design Environment for Fuzzy Systems," Proc. 7th IEEE International Conference on Fuzzy Systems, pp.1060-1065, Anchorage, May 1998.

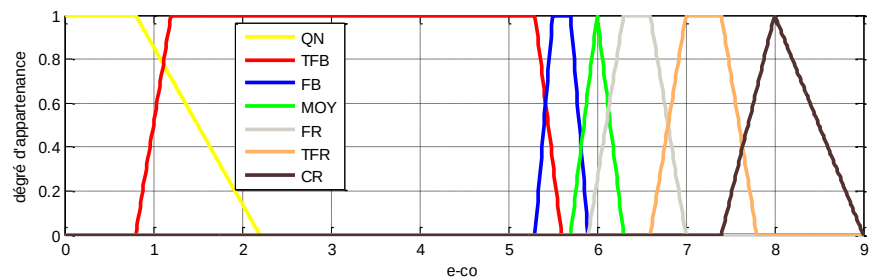
- [23]** F.J.M. Velo, L. Baturone, S.S. Solano and A. Barriga, "Rapid design of fuzzy systems with Xfuzzy," Proceedings of the IEEE 12th International Conference on Fuzzy Systems, May 25-28, Sevilla, Spain, pp: 342-347.2003.
- [24]** Volnei A. Pedroni "Digital Electronics And Design With VHDL," © 2008 by Elsevier Inc.
- [25]** Weng Fook Lee " VHDL Coding and Logic Synthesis," A Harcourt Science and Technology Company, 2000 by Academic Press.
- [26]** Enoch O. Hwang" Microprocessor Design Principles and Practices," © Brooks / Cole 2004.
- [27]** Douglas L. Perry " VHDL: Programmingby Example," Fourth Edition,Copyright © 2002 by The McGraw.
- [28]** International IEEE conference, "Simulation analysis of longitudinal ventilation system with jet fan speed control for MPC strategy in a road tunnel", Alaska, USA, 2012.
- [29]** Oucherfi Myriam,"Contribution à l'étude de la ventilation transversale lors d'un incendie en tunnel routier", thèse de doctorat, Université Claude Bernard – Lyon I, 6 Mars 2009.
- [30]** Mezzah Ibrahim, "Etude et conception d'un microcontrôleur IP sur FPGA", mémoire, présentée à la Faculté des Science de l'ingénieur, Département d'électronique, université Ferhat Abbas, Sétif, 2008.
- [31]** Khellaf Djamel, "Amélioration des performances de ventilation d'un tunnel par contrôleurs embarqués", Mémoire de Master, Université de Sétif, Institut d'électronique, Présenté le 25 juin 2013.

ANNEXE

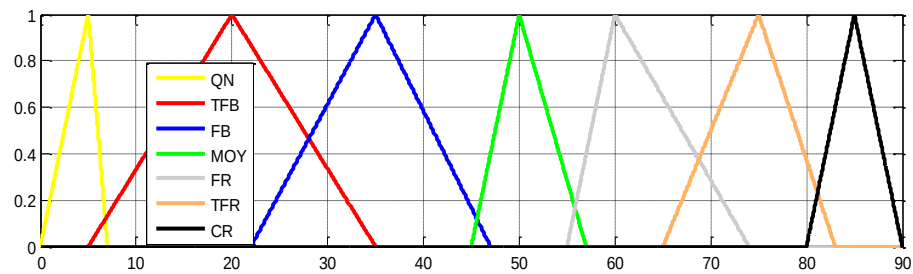
Contrôleur flou de monoxyde de carbone:



Fonction d'appartenance de CO

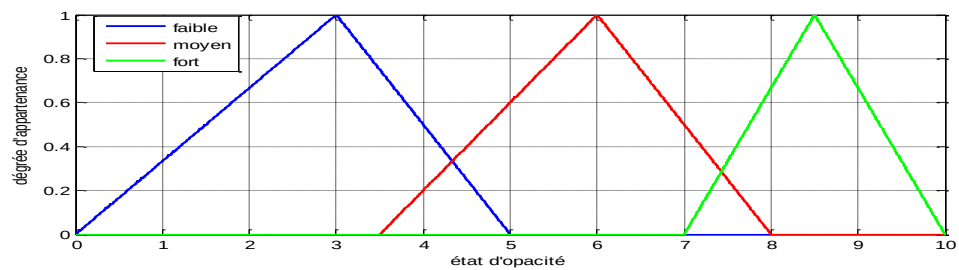


Fonction d'appartenance d'E_CO

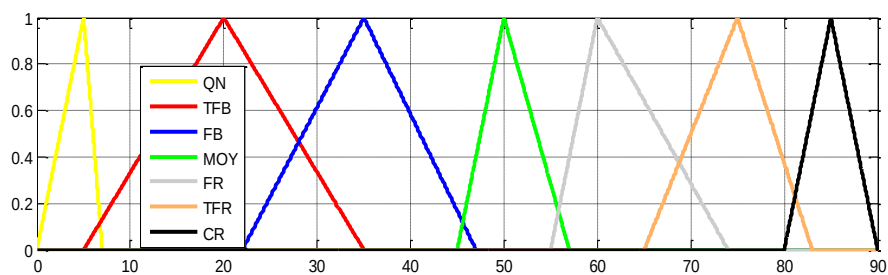


Fonction d'appartenance de L'état CO

Contrôleur mixte :

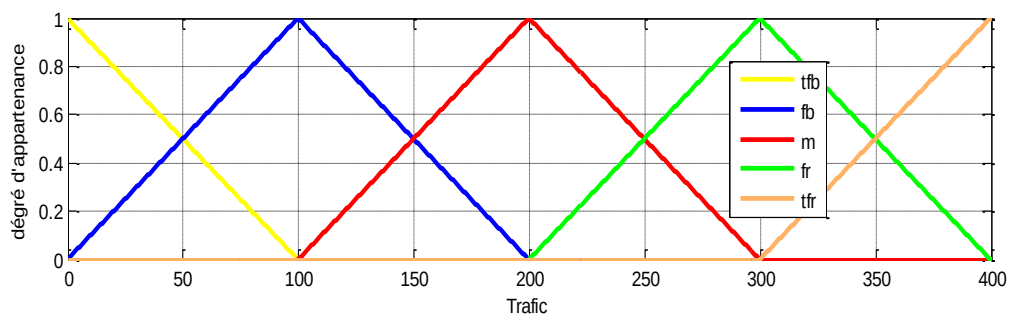


Fonction d'appartenance de l'état d'opacité

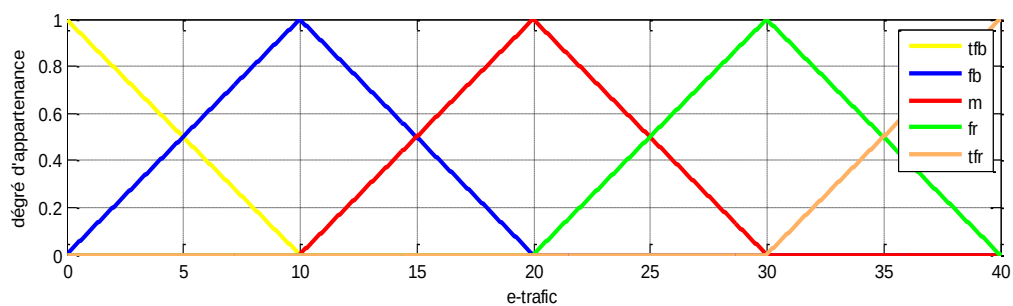


Fonction d'appartenance de l'état de CO.
Fonction d'appartenance de l'état mixte.

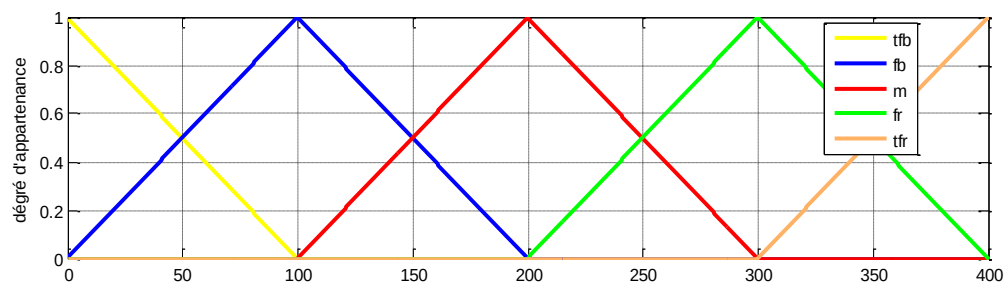
Contrôleur trafic :



Fonction d'appartenance du trafic.



Fonction d'appartenance d'e_traffic.



Fonction d'appartenance de l'état de trafic.

Programme principale

Library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity controleur_global is

Port (clk : in STD_LOGIC;

enable : in STD_LOGIC;

co : in STD_LOGIC_VECTOR (6 downto 0);

e_co : in STD_LOGIC_VECTOR (6 downto 0);

op : in STD_LOGIC_VECTOR (6 downto 0);

e_op : in STD_LOGIC_VECTOR (6 downto 0);

trafic : in STD_LOGIC_VECTOR (6 downto 0);

e_trafic : in STD_LOGIC_VECTOR (5 downto 0);

etat_f : out STD_LOGIC_VECTOR (5 downto 0);

bat1_z1 : out STD_LOGIC_VECTOR (2 downto 0);

bat2_z1 : out STD_LOGIC_VECTOR (2 downto 0);

bat3_z1 : out STD_LOGIC_VECTOR (2 downto 0);

bat4_z1 : out STD_LOGIC_VECTOR (2 downto 0);

bat5_z2 : out STD_LOGIC_VECTOR (2 downto 0);

bat6_z2 : out STD_LOGIC_VECTOR (2 downto 0);

bat7_z2 : out STD_LOGIC_VECTOR (2 downto 0);

bat8_z3 : out STD_LOGIC_VECTOR (2 downto 0);

bat9_z3 : out STD_LOGIC_VECTOR (2 downto 0);

bat10_z3 : out STD_LOGIC_VECTOR (2 downto 0);

```
        bat11_z3 : out STD_LOGIC_VECTOR (2 downto 0) );

endcontroleur_global;

architecture Behavioral of controleur_final_m is

    component controleur_co is port(

        clk      : in  STD_LOGIC;

        enable    : in  STD_LOGIC;

        co       : in  STD_LOGIC_VECTOR (6 downto 0);

        e_co      : in  STD_LOGIC_VECTOR (6 downto 0);

        etat      : out STD_LOGIC_VECTOR (6 downto 0));

    end component;

    componentcontroleur_op is port(

        clk : in  STD_LOGIC;

        enable : in  STD_LOGIC;

        op : in  STD_LOGIC_VECTOR (6 downto 0);

        e_op : in  STD_LOGIC_VECTOR (6 downto 0);

        etat : out STD_LOGIC_VECTOR (6 downto 0));

    end component;

    componentcontroleur_mixte is port(

        clk : in  STD_LOGIC;

        enable : in  STD_LOGIC;

        etat_co : in  STD_LOGIC_VECTOR (6 downto 0);

        etat_op : in  STD_LOGIC_VECTOR (6 downto 0);

        com1 : out STD_LOGIC_VECTOR (6 downto 0));

    end component;
```

```
component controleur_trafic is port (  
    clk : in STD_LOGIC;  
    enable : in STD_LOGIC;  
    trafic : in STD_LOGIC_VECTOR (6 downto 0);  
    e_trafic : in STD_LOGIC_VECTOR (5 downto 0);  
    etat : out STD_LOGIC_VECTOR (6 downto 0));  
end component;  
  
component controleur_global is port (  
    clk : in STD_LOGIC;  
    enable : in STD_LOGIC;  
    etat_com1 : in STD_LOGIC_VECTOR (6 downto 0);  
    etat_trafic : in STD_LOGIC_VECTOR (6 downto 0);  
    com2 : out STD_LOGIC_VECTOR (5 downto 0));  
end component;  
  
component zone_1 is port (  
    nombre_v : in std_logic_vector(5 downto 0);  
    bat1_z1 : out std_logic_vector(2 downto 0);  
    bat2_z1 : out std_logic_vector(2 downto 0);  
    bat3_z1 : out std_logic_vector(2 downto 0);  
    bat4_z1 : out std_logic_vector(2 downto 0));  
end component;  
  
component etat is port (  
    m : in std_logic_vector(5 downto 0);  
    etat_f : out std_logic_vector(5 downto 0));  
end component;
```

```
component zone_2 is port (  
    nombre_v:instd_logic_vector(5 downto 0);  
    bat5_z2:outstd_logic_vector(2 downto 0);  
    bat6_z2:outstd_logic_vector(2 downto 0);  
    bat7_z2:outstd_logic_vector(2 downto 0));  
end component;  
  
component zone_3 is port (  
    nombre_v:instd_logic_vector(5 downto 0);  
    bat8_z3:outstd_logic_vector(2 downto 0);  
    bat9_z3:outstd_logic_vector(2 downto 0);  
    bat10_z3:outstd_logic_vector(2 downto 0);  
    bat11_z3:outstd_logic_vector(2 downto 0));  
end component;  
  
    signal s1,s3,s4 : std_logic_vector(6 downto 0);  
    signal s5 : std_logic_vector(6 downto 0);  
    signal commande : std_logic_vector(5 downto 0);  
  
for all :controleur_co      use entity work.controleur_co (Behavioral);  
for all :controleur_op      use entity work.controleur_op (Behavioral);  
for all :controleur_mixte use entity work.controleur_mixte (Behavioral);  
for all :controleur_trafic use entity work.controleur_trafic(Behavioral);  
for all :controleur_global use entity work.controleur_global(Behavioral);  
for all :zone_1      use entity work.zone_1(Behavioral);--modifie  
for all :zone_2      use entity work.zone_2(Behavioral);  
for all :zone_3      use entity work.zone_3(Behavioral);  
for all :etat use entity work.etat(Behavioral);
```

begin

etat_monoxide_carbon: controleur_co port map (clk,enable,co,e_co,s1);

etat_opacite: controleur_op port map (clk,enable,op,e_op,s5);

etat_mixte: controleur_mixte port map (clk,enable,s1,s5,s3);

etat_trafic: controleur_trafic port map (clk,enable,trafic,e_trafic,s4);

etat_globale: controleur_global port map (clk,enable,s3,s4,commande);

nbre_ventilateur:zone_1 port map (commande,bat1_z1,bat2_z1,bat3_z1,bat4_z1);

nbre_ventilater:zone_2 port map (commande,bat5_z2,bat6_z2,bat7_z2);

nbre_ventilator:zone_3 port map (commande,bat8_z3,bat9_z3,bat10_z3,bat11_z3);

valide_etat: etat port map (commande,etat_f);

endBehavioral;

