Sur l'utilisation de la logique floue dans la synthèse de circuits analogiques

F. Tissafi-Drissi, I. O'Connor, F. Mieyeville, F. Gaffiot
Ecole Centrale de Lyon / CIMIRLY, F-69134 Ecully
http ://leom.ec-lyon.fr
{faress.tissafi-drissi, ian.oconnor, fabien.mieyeville, frederic.gaffiot}@ec-lyon.fr

Résumé

La logique floue est une technique de raisonnement qualitatif provenant du domaine de l'intelligence artificielle. Dans la conception analogique et multi-domaine, des décisions basées sur de l'expérience sont courantes, notamment dans la sélection d'une topologie pour réaliser une fonction et respecter un cahier des charges d'une part, et dans la validation de la faisabilité d'une architecture d'autre part. Nous démontrons dans ce travail l'application de la logique floue à ces deux tâches dans une plate-forme de synthèse analogique et multi-domaine.

1. Introduction

L'augmentation de la fréquence de fonctionnement des circuits intégrés et l'intégration nécessaire de blocs numériques et mixtes avec des fonctions RF (radio-fréquence) exigent le développement d'outils et de méthodes de conception adaptées aux hautes fréquences (quelques GHz) et aux systèmes multi-domaine. De tels outils permettraient de conserver l'expertise acquise lors de la conception d'un circuit (*IP*[1] *reuse*), de pouvoir aisément prendre en compte les évolutions de la technologie, d'augmenter le rendement et d'accélérer le cycle de conception.

La conception d'un système passe nécessairement par la décomposition des fonctions en sous-fonctions jusqu'au plus bas niveau (niveau transistor en général). La gestion de cette décomposition progressive doit se faire de façon intelligente, en répartissant les spécifications de l'ensemble en plusieurs sous-spécifications, en propageant des contraintes de dimensionnement et en faisant remonter les performances réelles des sous-blocs afin de prendre en compte les implications pour les performances du système.

*A. **Contexte pédagagogique***

La troisième année de cycle ingénieur à l'Ecole Centrale de Lyon est caractérisée par l'intégration des élève-ingénieurs généralistes dans une option spécialisée. Les étudiants de l'option "Electronique et Systèmes de Communication" ont, au cours des six premiers mois de l'année universitaire, à traiter un projet personnel d'environ 150 heures, en parallèle avec les cours spécialisés et souvent en connexion avec les axes de recherche du laboratoire d'accueil. Le premier trimestre est consacré à une étude bibliographique du sujet (50 heures) puis, de janvier à mars, ils passent à la phase "active" du projet (100 heures).

L'exemple qui suit est consacré à l'utilisation des techniques provenant du domaine de l'intelligence artificielle dans le cadre de la synthèse de circuits analogiques et de systèmes multi-domaine en général, et la gestion de la hiérarchie en particulier. Il a été démontré que la logique floue peut apporter une solution intéressante des points de vue de la sélection de topologie à partir d'une bibliothèque d'une part, et de la validation de faisabilité d'une solution avant dimensionnement effectif d'autre part.

C'est une étude qui a permis à un élève-ingénieur d'effectuer un travail très pointu d'initiation à la recherche dans un contexte alliant les disciplines de l'électronique, l'informatique et les mathématiques.

*B. **Problématique***

Une plate-forme générique de synthèse de systèmes multi-domaine a été développée au sein du laboratoire pour assister en la prédiction de l'évolution des performances des blocs analogiques, mixtes et multi-domaine dans les systèmes intégrés. Très flexible, cette plate-forme est capable d'associer différentes méthodes de dimensionnement et d'évaluation, cette modularité permettant une application à divers systèmes. Cependant, la plate-forme ne peut fonctionner qu'à un seul niveau hiérarchique, ce qui limite fondamentalement son utilité. L'objectif du projet a été de rajouter cette fonctionnalité à la plate-forme (fig. 1).

Plusieurs méthodes existent pour choisir des architectures faisables de circuit (et donc éliminer les architectures inutiles), dont "branch and bound", heuristiques, programmation non-linéaire [1, 2, 3]. Notre choix s'est porté sur l'utilisation de la logique floue [4], qui présente l'avantage de découpler l'aspect quantitatif des capacités d'une topologie à atteindre le cahier des charges, de celui de règles qualitatives. En outre, elle permet l'estimation de la faisabilité du choix de paramètres des sous-topologies, simulées au niveau

[1] intellectual property

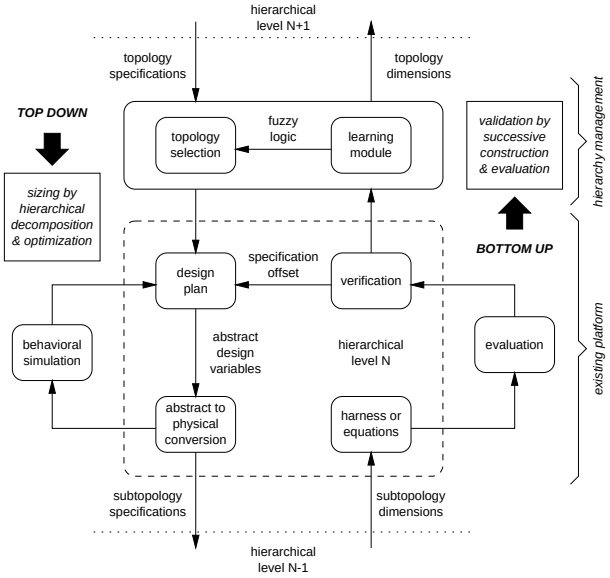


Fig. 1. Place de la logique floue dans la plate-forme existante

système par un simulateur comportemental (tel que AdvanceMS ou SpectreHDL) qui est par définition incapable de fournir des informations concernant les conséquences physiques du choix de paramètres.

Le nombre de configurations possibles d'un circuit analogique ou d'un système multi-domaine pour réaliser une même fonction est extrêmement élevé. Par conséquent, la recherche d'une solution et la décomposition en sous-blocs ne peuvent être systématiques. Le choix d'une solution proposée ne peut se faire qu'en ayant *a priori* une idée du résultat que l'on obtiendra.

Le processus de recherche est tel que l'on peut pour chacun des blocs, avoir en base de données des configurations différentes. Une fois que le processus est arrivé au plus bas niveau (transistors), il doit effectuer une optimisation des paramètres par un bouclage optimiseur - simulateur. Le logiciel doit donc effectuer un choix "intuitif" entre ces différentes configurations, car il ne peut bien entendu pas tester toutes les configurations. En effet, le plus bas niveau demande un très grand nombre de calculs, vu le nombre quasi-infini de configurations possibles.

Dans section 2, nous expliquons brièvement ce que représente la logique floue, pour que la sélection de topologie se fasse sur la base d'un jeu de règles simples et qualitatives. Section 3 décrit la réalisation d'un moteur flou pour la sélection de topologie d'une part, et permettant un auto-apprentissage des correspondances quantitatives des règles en affinant les choix au fur et à mesure que les essais s'accumulent. Il y a en effet nécessité d'avoir, dans le sens opposé à la fonctionnalité de conception, un retour des résultats sur les hypothèses qui ont été faites *a priori*. L'intégration de ce moteur flou dans la plate-forme existante est détaillée en section 4, et les résultats du travail en section 5.

2. La logique floue

A. Théorie de la logique floue

Elle consiste en la discrétisation d'un intervalle de valeurs plausibles, avec dans le cas d'une valeur "frontière", un choix pondérée pour la valeur. L'intérêt réside donc dans la possibilité de passer de variables quantitatives en variables qualitatives ("verysmall", "small" ...) et donc d'utiliser des règles qualitatives pour le choix des configurations.

La phase de choix sera à priori aidée par les règles entrées lors de la conception du logiciel, mais nous pouvons imaginer que celui-ci puisse modifier ou créer des règles de manière autonome grâce à un module de test automatique.

La phase d'apprentissage, quant à elle, se fera ressentir sur le passage aux variables floues, notamment en décalant les différents intervalles de conversion en fonction des résultats obtenus.

Un des avantages de la logique floue est la simplicité des algorithmes permettant le calcul d'une décision et le langage intuitif des règles guidant le choix du logiciel vers telle ou telle configuration.

B. Modélisation et utilisation des règles

B.1 Modélisation des termes flous

Chaque terme flou est modélisé par un intervalle dans lequel la valeur numérique d'une spécification donne une probabilité $P(c)$ de choix de la topologie, et rendant compte de l'ensemble des valeurs qui peuvent être regroupées sous ce terme qualitatif. Cependant les formes d'intervalles flous peuvent prendre des formes différentes. Si en ce qui concerne les termes de frontière ("verysmall" et "verylarge"), les rampes sont adaptées (afin que les valeurs limites soient prisent en compte dans les règles floues), nous avons besoin d'intervalles à support compact (triangle ou trapèze) pour modéliser les termes "intermédiaires" (fig. 2). Dans ce travail nous pouvons ne prendre en compte que les triangles, mais il est évident qu'un processus d'apprentissage devra prendre une plage de valeurs optimales, donc en forme de trapèzes.

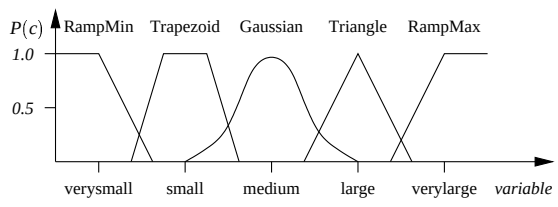
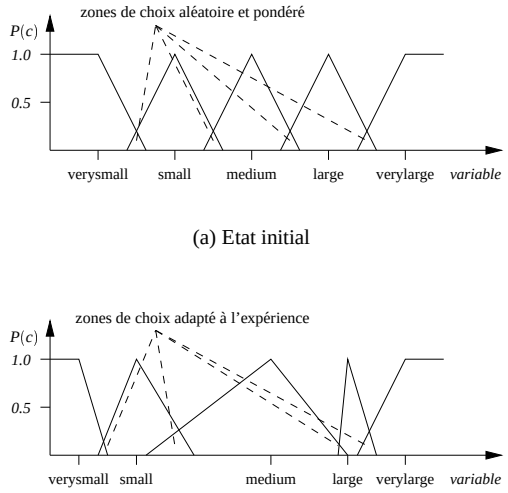


Fig. 2. Modélisation de termes flous par des intervalles de formes différentes

L'avantage de la logique floue est de permettre deux niveaux d'apprentissage. Un premier niveau est représenté par la modification des intervalles en

fonction des valeurs optimisées issues du niveau de décomposition inférieur (fig. 3).



(a) Etat initial

(b) Etat post-apprentissage

Fig. 3. Apprentissage par modification des intervalles floues

Cependant cet apprentissage se base sur une confiance totale en les règles de conception entrées par l'utilisateur. Un deuxième niveau d'apprentissage, beaucoup plus puissant puisque pouvant évaluer les possibilités de n'importe quel modèle, simplement en se basant sur les résultats de simulations obtenus par les logiciels bas-niveau, est la génération automatique de règles.

B.2 Modélisation des règles floues

Une règle en logique floue a une structure "IF-THEN", acceptant une condition et une conclusion (fig. 4).

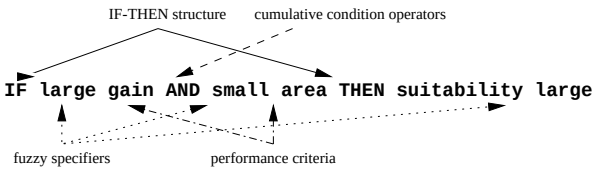


Fig. 4. Exemple d'une règle floue

Elle permet donc de rendre compte de manière qualitative de règles telles qu'elles pourraient être formulés par un designer. Une base de règles est nécessaire pour chaque décomposition d'un bloc en sous-blocs pour permettre à chaque règle de s'exprimer, et de permettre également la prise en compte de toutes ces règles sous forme d'une note de l'adaptabilité de la topologie aux dimensions.

Le mécanisme algorithmique de calcul permet justement de déterminer la correspondance d'une règle aux spécifications. La correspondance des spécifications avec un intervalle concerné par la règle se fait par le calcul de plus bas niveau. Nous faisons ensuite la synthèse de toutes les conditions (en prenant par exemple la plus faible valeur de correspondance) pour connaître l'adaptabilité de la règle aux spécifications (fig. 5).

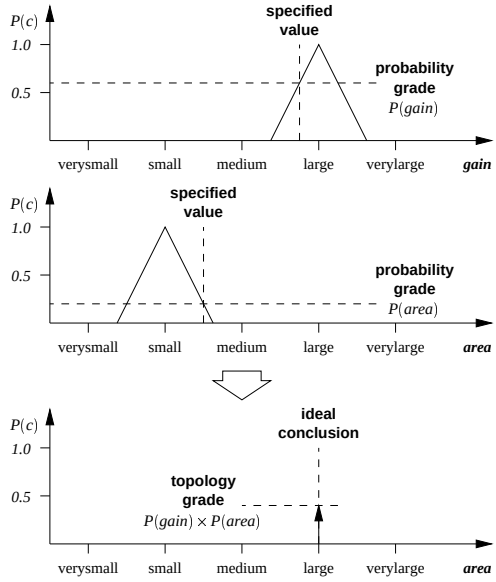


Fig. 5. Modélisation mathématique d'une règle floue

La synthèse pour la correspondance d'une topologie avec les spécifications se fait par la synthèse de toutes les règles. Le calcul consiste alors à faire la synthèse de toutes les règles par un calcul barycentrique de conclusions des règles pondérées par la correspondance des spécifications avec les conditions de la règle. Il est à noter qu'une règle peut être favorable à une topologie (si sa conclusion, représentant sa "suitability" est "large" ou "verylarge"), ou défavorable (terme de conclusion "small" ou "verysmall").

B.3 Utilisation de logiques stricte et large

De part la forme plus ou moins large de ces intervalles, la logique floue permet de moduler la prise de décision. La prise en compte d'une règle pour une topologie implique que la valeur de spécification appartienne à l'intervalle traduisant le terme flou pour la condition liée à la spécification.

Dans un premier temps, il paraît évident de prendre en compte un processus qui permette de prendre en compte le plus largement possible les règles. Nous parlons alors de logique "large". Puisqu'à l'initialisation, les intervalles sont définis de manière automatique, ils n'ont aucune raison de correspondre à la représentation qualitative liés au terme flou, tel que le "comprend" l'utilisateur. Il est donc nécessaire de voir le maximum de règles prises en compte. Ceci est réalisé par le recouvrement des intervalles, de manière à ce qu'à chaque valeur de spécification corresponde deux intervalles flous.

Une fois la phase d'apprentissage passée, nous pouvons envisager de faire appel à une autre forme de logique, la logique stricte. Elle est déterminée de telle manière qu'à chaque valeur de spécification ne corresponde qu'un terme flou. Ceci est réalisé concrètement par le non-recouvrement des intervalles.

3. Réalisation d'un moteur flou

*A. **Fonctionnalité***

Le moteur flou (fig. 6) permet de noter une sous-topologie proposées par l'optimiseur en fonction de règles correspondant à cette configuration et des dimensions de l'optimiseur. Les données en entrée doivent subir au préalable une normalisation, car la construction d'intervalles ne peut se faire qu'avec des valeurs comprises entre 0 et 1.

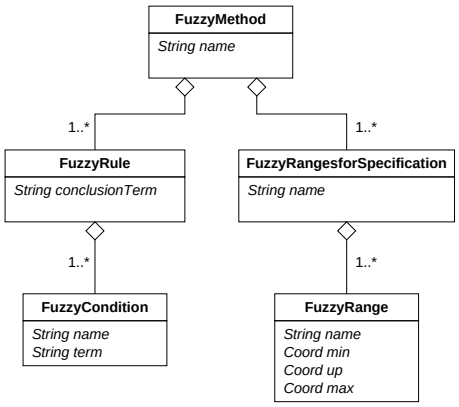


Fig. 6. Architecture générale des classes du moteur flou

L'instanciation des FuzzyRange se fait en entrant dans le constructeur le nom des spécifications souhaitées. L'instanciation des différents paramètres se fait de manière automatique par un découpage identique des intervalles pour chaque spécification (RampMin, Triangle, Triangle, Triangle, RampMax) .

L'instanciation des FuzzyRule se fait par lecture d'un fichier texte, lié à une topologie, et portant le nom de la méthode. La modification des règles de calcul peut se faire en modifiant le fichier texte de règles, et par rechargement de la classe.

*B. **Décision et apprentissage***

B.1 Structure des classes

Le module de décision contient 2 classes et une exception :

- DecisionModule permet le lancement du calcul pour une catégorie, dont nous souhaitons déterminer laquelle des sous-topologies est la plus adaptée.
- FuzzyDecisionLauncher permet le lancement de toutes les catégories à tester, et les stockages des topologies qui conviennent et des catégories qui ne conviennent pas.
- L'exception NoSuitabilityException utilisée par la méthode sendNote() permet quant à elle d'avertir le FuzzyDecisionLauncher qu'aucune des topologies liées à la catégorie n'est adaptée aux spécifications.

B.2 Apprentissage par modifications des intervalles

Le choix de la logique floue a été effectué pour permettre au logiciel de connaître *a priori* la configuration de circuit qui conviendra le mieux aux dimensions proposées par l'optimiseur. Nous avons vu cependant que l'initialisation des intervalles floues se faisait de manière constante, en répartissant les intervalles floues de manière algorithmique et constantes. Or cette construction n'est bien entendu pas adaptée à chaque dimension.

La réduction du temps de calcul par un choix fiable nécessite forcément en contre partie la possibilité d'apprendre, et de "comprendre" les règles qualitatives. Une des innovations du logiciel pourrait être de s'adapter à n'importe quelle configuration de circuits.

Pour cela, nous nous basons sur les règles définies par l'utilisateur, et adaptons chacun des intervalles floues représentant les différents termes d'une spécification. Cette adaptation peut se faire manuellement par une mise en place direct des valeurs correspondantes, mais cette adaptation serait forcement aléatoire, et ferait perdre une grande partie de l'intérêt d'écrire des règles qualitatives en logique floue.

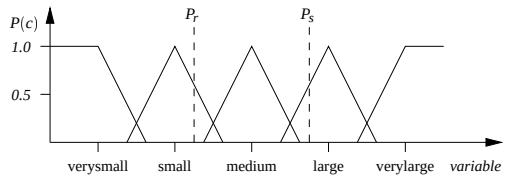
L'adaptation peut se faire également de manière algorithmique, si nous considèrons que le moteur flou peut avoir accès à l'évaluation des paramètres qui sont apparus du fait de sa décision. Cela veut par exemple dire que les données, issues de la décomposition permise par le moteur flou, doivent revenir et permettre la modification des intervalles flous.

B.3 Principe de l'apprentissage par modification d'intervalle

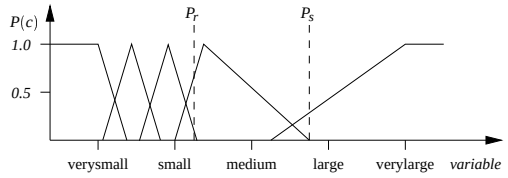
Prenons le cas où la valeur de la performance concernée de la topologie après optimisation P_r est inférieure à la valeur initialement spécifiée P_s. Nous supposons dans le cas le plus général qu'ils ne correspondent pas au même terme flou (la plus grande valeur d'interpolation avec un intervalle n'est pas réalisé par le même intervalle flou).

La décision erronée que l'on souhaite modifier a été prise en utilisant les règles qui font appel à l'intervalle "large", terme flou correspondant le plus à la spécification. Or il s'avère que la valeur optimisée pour la topologie est nettement moindre que celle spécifiée. Nous devons donc modifier les intervalles flous de manière à faire correspondre "large" à la valeur optimisée, et faire en sorte que la valeur de spécification

ne fasse plus intervenir les règles où “large” intervient. Cela nécessite un recadrage des intervalles flous (fig. 7).



(a) Intervalles flous avant apprentissage



(b) Intervalles flous après apprentissage

Fig. 7. Modification des intervalles floues

L’algorithme d’apprentissage définit de nouveaux barycentres des intervalles flous U_p, déterminés par la distance entre l’intervalle directement à gauche de P_r (ici “verysmall”) et P_s, divisé par le nombre d’intervalles à modifier.

Une autre contrainte est l’interdiction et la gestion de modification d’intervalles où de coordonnées dont nous sommes “sûr” (déjà modifiés par un apprentissage précédent, donc provenant de la simulation). Pour cela, des marqueurs de verrouillage ont été ajoutés. Ce sont des valeurs booléennes, qui ont la valeur “vraie” si la coordonnée ne peut être déplacée, “fausse” si elle est libre de ce déplacement.

4. Intégration du moteur dans la plate-forme de synthèse

Le module entier (FuzzyDecisionLauncher, fig. 8) prend en entrée les dimensions des différentes catégories issues de l’optimiseur et permet de retourner la meilleur topologie pour chaque catégorie, ou si aucune des topologies ne réalise les conditions nécessaires, classe dans les unSuitablesTopologies le nom de la catégorie qui ne peut être décomposée, ainsi que la meilleure note des configurations testées. Envoyées au programme, ces vecteurs permettent d’effectuer la décomposition des éléments, ou d’assouplir les contraintes sur les topologies non-décomposables. Nous pouvons par exemple modifier une contrainte en coût pour la catégorie ayant la plus basse note, indiquant qu’elle sera la plus difficile à décomposer avec les dimensions précédentes.

Il faut donc envisager la prise en compte de la méthode de décomposition hiérarchique une fois que

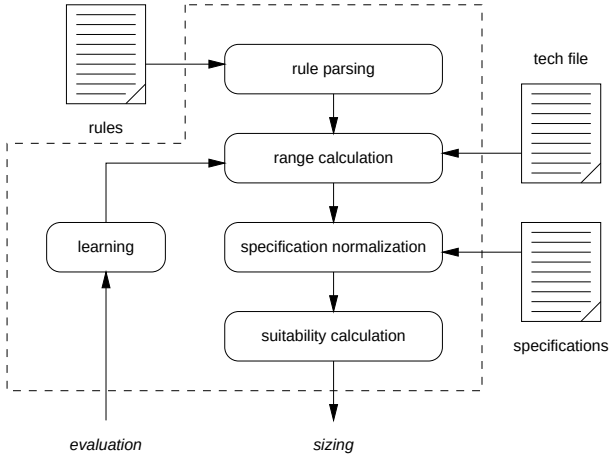


Fig. 8. Déroulement général du moteur flou

toutes les catégories optimisées ont leurs dimensions, et passer ces dimensions au FuzzyDecisionLauncher.

Pour la possibilité de prendre en compte l’apprentissage, l’ensemble des objets FuzzyRange doit être sérialisé, et sauvé après chaque manipulation. La prise en compte de l’apprentissage est réalisable par la réalisation d’un historique de chaque catégorie, qui doit donc permettre de mettre en relation des dimensions optimisées, et les anciennes dimensions proposées au moteur de décision.

5. Résultats

Le moteur de logique floue a été développé et testé (fig. 9), et permet de déterminer le choix d’une configuration sans avoir à l’évaluer. Le module de décision permet de plus le stockage et le passage d’information sur la décomposition, et d’accepter effectivement ou non la décomposition proposée par l’optimiseur.

En ce qui concerne la méthode d’apprentissage, les coordonnées qui représentent les intervalles peuvent être “verrouillées”, c’est à dire mis à jour uniquement sous certaines conditions. Par exemple la valeur “y” ne peut être modifiée car la construction d’un objet de la classe Coord met le marqueur de verrouillage à la valeur “vraie”.

Cependant la modification des intervalles implique un stockage de la base d’intervalles, ce qui n’est pas encore réalisé. De plus la gestion de certaines exceptions (par exemple si la valeur de la spécification est à l’intersection de deux intervalles) ne sont pas prises en compte. De meme la gestion d’exceptions “dures” - refonte complète des intervalles, en prenant en compte les différents apprentissages passés si nous avons une incompatibilité entre règles - n’est pas prise en compte.

Le logiciel permet un apprentissage relatif, qui est suffisant si les règles sont justes et forment un set complet. Cependant le fonctionnement du logiciel est pour l’instant limité au nombre de fichiers de règles, fonction du nombre de topologies représentées. La

```
* Beginning fuzzy decision for all categories
* Beginning fuzzy decision for category ampli
* Sub directories of ampli are:
* (0) inv
* (1) jh
* (2) vcc_load
** current topology: inv
Path: ~/rune/examples/categories/ampli/inv
File: fuzzyRulesFile.txt
Beginning Rules loading in class FuzzyRulesBase
   New Condition Built: medium gain
   New Condition Built: small bp
   New Conclusion Built: large
   End Rule building
   New Condition Built: medium gain
   New Condition Built: medium bp
   New Conclusion Built: verylarge
   End Rule building
Rules loaded in class FuzzyRulesBase
Beginning building fuzzy ranges
   Specification encoded: gain
   (position: 0) Fuzzy term verysmall:
    (0.0,1.0)  (0.1,1.0)  (0.3,0.0)
   (position: 1) Fuzzy term small:
    (0.1,0.0)  (0.3,1.0)  (0.5,0.0)
   (position: 2) Fuzzy term medium:
    (0.3,0.0)  (0.5,1.0)  (0.7,0.0)
   (position: 3) Fuzzy term large:
    (0.5,0.0)  (0.7,1.0)  (0.9,0.0)
   (position: 4) Fuzzy term verylarge:
    (0.7,0.0)  (0.9,1.0)  (1.0,1.0)
   Large fuzzy ranges used
   Building fuzzy ranges (gain) finished
   Specification encoded: bp
   (position: 0) Fuzzy term verysmall:
    (0.0,1.0)  (0.1,1.0)  (0.3,0.0)
   (position: 1) Fuzzy term small:
    (0.1,0.0)  (0.3,1.0)  (0.5,0.0)
   (position: 2) Fuzzy term medium:
    (0.3,0.0)  (0.5,1.0)  (0.7,0.0)
   (position: 3) Fuzzy term large:
    (0.5,0.0)  (0.7,1.0)  (0.9,0.0)
   (position: 4) Fuzzy term verylarge:
    (0.7,0.0)  (0.9,1.0)  (1.0,1.0)
   Large fuzzy ranges used
   Building fuzzy ranges (bp) finished
End building fuzzy ranges
Beginning note for current topology
   Compatibility of dimensions with rule 0: 0.45
   Suitability of rule 0 with current topology: 0.7
   Compatibility of dimensions with rule 1: 0.5
   Suitability of rule 1 with current topology: 0.9
Final suitability of topology with dimen-
sions: 0.8052631578947368
End note for current topology
```

Fig. 9. Déroulement de tests d'une méthode floue

réalisation du fichier de règles par l'utilisateur requiert des connaissances en conception analogique relativement avancées, même si l'écriture des règles est intuitive.

Un système d'apprentissage au sens propre, et non seulement une adaptation à des règles, pourrait être envisagé. Il est possible d'améliorer encore le logiciel en lui permettant de réaliser un système de génération de règles automatiques, dans une phase d'apprentissage, pour qu'il teste différents circuits, et génère directement les règles adéquates. La conception ferait suite à cette phase préliminaire.

6. Conclusion

Dans ce travail, l'utilité potentiel de la logique floue dans la synthèse analogique et multi-domaine a été démontré. En effet, cette technique peut aider en la sélection d'une topologie depuis une bibliothèque, et en la validation de la faisabilité d'une solution générée avant dimensionnement. Nous avons réalisé un moteur de logique flou pour intégration dans une plate-forme de synthèse générique déjà existante.

Références

[1] F. El-Turky and E. E. Perry, "BLADES : An artificial intelligence approach to analog circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 8, no. 6, June 1989.

[2] H. Y. Koh, C. H. Séquin, and P. R. Gray, "OPASYN : A compiler for CMOS operational amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 9, no. 2, Feb. 1990.

[3] P. C. Maulik, L. R. Carley, and R. A. Rutenbar, "Integer programming based topology selection of cell-level analog circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 4, Apr. 1995.

[4] A. Torralba, J. Chávez, and L. G. Franquelo, "FASY : A fuzzy-logic based tool for analog synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 7, July 1996.