Habilitation à Diriger des Recherches Discipline : Informatique Simulation Concurrente de Systèmes à Evénements Discret : Concepts et Applications

Dominique Federici

▶ To cite this version:

Dominique Federici. Habilitation à Diriger des Recherches Discipline : Informatique Simulation Concurrente de Systèmes à Evénements Discret : Concepts et Applications. Informatique [cs]. Université Pascal Paoli, 2006. tel-00603867

HAL Id: tel-00603867
https://theses.hal.science/tel-00603867

Submitted on 27 Jun 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE DE CORSE

UMR CNRS 6134

Simulation Concurrente de Systèmes à Evénements Discrets : Concepts et Applications

Mémoire présenté et soutenu publiquement à Corté, le 30 Novembre 2006 pour l'obtention de l'**Habilitation à Diriger des Recherches de l'Université de Corse**

Par

Dominique FEDERICI

Composition du Jury

MM. G.A. CAPOLINO (Rapporteur), *Professeur à l'Université de Picardie*

J.J. CHABRIER (Rapporteur), *Professeur à l'Université de Bourgogne*

D. FLOUTIER (Rapporteur), *Professeur à l'Université de Savoie*

P. BISGAMBIGLIA (Examinateur), *Professeur à l'Université de Corse*

J-F. SANTUCCI (Examinateur), *Professeur à l'Université de Corse*

B. STRAUBE (Examinateur), *Privatdozen Pr. -Ing. Habil. à l'Institut Fraunhofer de Dresde, Allemagne*

REMERCIEMENTS

J'exprime ma reconnaissance à Messieurs Gérard-André CAPOLINO, Professeur à l'Université de Picardie, Jean-Jacques CHABRIER, Professeur à l'Université de Bourgogne et Denis FLOUTIER, Professeur l'Université de Savoie pour l'intérêt qu'ils ont porté à ce travail en acceptant de le juger en qualité de rapporteurs.

Que Monsieur Bernd STRAUBE, Privatdozen Pr. -Ing. Habil. à l'Institut Fraunhofer de Dresde, qui me fait l'honneur de participer à ce jury trouve ici l'expression de ma profonde gratitude.

Je voudrais également exprimer toute ma reconnaissance à Messieurs Paul BISGAMBIGLIA et Jean-François SANTUCCI, Professeurs à l'Université de Corse pour leur grande disponibilité et pour avoir accepter de participer à ce jury.

Je remercie tous les chercheurs, et personnels de l'Université de Corse pour leur gentillesse et leur contribution dans la préparation de la soutenance.

A ma famille

AVANT PROPOS

Ce document intitulé « Simulation Concurrente de Systèmes à Evénements Discrets : Concepts et Applications » est présenté en vue de l'obtention de l'habilitation à diriger les recherches en informatique. Il est composé de trois parties. La première présente l'évolution de mes travaux de recherche depuis mon doctorat. La seconde partie décrit mon rapport d'activités en matière de recherche, d'enseignement et d'administration. Enfin, la dernière partie est constituée de cinq publications représentatives afin d'éclairer le lecteur sur le cheminement de ma production scientifique.

TABLE DES MATIERES

Remerciements **3**
Avant Propos **5**
Table des matières **6**

PARTIE 1 - SYNTHESE DES TRAVAUX DE RECHERCHE **8**

Chapitre I - Introduction générale **9**
Chapitre II - Activités de Recherche Doctorales **11**
2.1 - Présentation du problème 11
2.2 - Modélisation des descriptions VHDL 12
2.3 - Modèle de fautes 14
2.4 - Technique de simulation de fautes 15
2.5 - Conclusion 17
Chapitre III - Evolution des concepts et développement de méthodes **18**
3.1- Présentation du problème 18
3.2 - La SCC et la SFC 19
3.3 - Le formalisme DEVS 19
3.4 - Le formalisme BFS-DEVS 22
3.5 - Application aux circuits digitaux 24
3.6 - Résultats 25
3.7 - Génération de vecteurs de test 27
3.8- Conclusion 28
Chapitre IV - Activités de recherche recentes **29**
4.1 - Détection de pannes dans les systèmes d'énergie renouvelable 29
4.1.1 – La modélisation électrique des machines électriques asynchrones 29
4.1.2 – La modélisation et la simulation du stator avec PowerDEVS 30
4.1.3 – Conclusion et Perspectives 31
4.2 - Intégration de la hiérarchie d'abstraction de données spatiales dans un SIG 31
4.3 – Analyse/Synthèse de voix pour la valorisation et l'apprentissage des chants polyphoniques corses 33
Chapitre V - Projet de Recherche **35**

PARTIE 2 - NOTICE INDIVIDUELLE ET RAPPORTS D'ACTIVITES **37**

1 - Curriculum Vitae **38**
1.1 - Etat Civil 38
1.2 - Titres et Diplômes 38
1.3 - Cursus Professionnel 39
2 - Rapport d'activité en matière de Recherche **39**
2.1 - Bref historique 39
2.2 - Encadrements de Chercheurs : 40
2.2.1 - Co-encadrements de stages de DEA et de MASTER Recherche 40
2.2.2 - Co-encadrement de doctorat 40
2.3 - Animation et Administration de la Recherche : 41
2.3.1 - Niveau International 41
3 - Rapport d'activité en matière d'Enseignement **44**
3.1 - Bref historique 44
3.2 - Tableau récapitulatif des activités d'enseignement 44
3.3 - Description des cours de l'année 2005/2006 : 47
3.4 - Encadrement de stagiaires 48
3.5 - Tutorat de moniteur 48
4 - Rapport d'activité en matière d'Administration **49**
4.1 - Bref historique 49
4.2 - Le DEUG MIAS 49
4.3 - Le DESS CCI (MASTER 2 CCI) 49
4.4 - Le DESS ISI (MASTER 2 ISI) 49
4.5 - L'IUP NTIC ISI 49
4.6 – Le Département Informatique 50
4.7 - Délocalisation de formations 50
4.8 - Autres Responsabilités 50
5 - Liste des Publications **50**
6 - Références **53**

PARTIE 3 - PUBLICATIONS REPRÉSENTATIVES **56**

1 - Première Publication **57**
2 - Seconde Publication **74**
3 - Troisième Publication **84**
4 - Quatrième Publication **92**
5 - Cinquième Publication **119**

ANNEXE 1 – LETTRES D'APPRECIATION **126**

Partie 1 - Synthèse des Travaux de Recherche

CHAPITRE I - INTRODUCTION GENERALE

Mes activités de recherche ont débuté en 1995 par un stage de fin d'étude au laboratoire SDEM (Systèmes Dynamiques, Energétiques et Mécaniques - URA CNRS 2053) dirigé par le professeur Jacques-Henri Balbi. Ce stage sous la direction du professeur Jean-François Santucci m'a permis de m'initier au travail de chercheur. A la suite de ce stage en laboratoire le professeur Santucci m'a proposé un sujet de doctorat dans le domaine du test de circuits digitaux. L'objectif de cette thèse était la mise au point d'une méthode de simulation de fautes pour les circuits digitaux. Outre le fait qu'un tel sujet permettait la découverte d'un nouveau thème de recherche au sein du laboratoire, l'originalité de ces travaux consistait en plusieurs points :

- La simulation de fautes devait être réalisée le plus tôt possible lors de la conception des circuits.
- La mise au point d'une technique permettant de réduire le nombre de simulation.
- L'implémentation de la méthodologie à l'aide de la programmation orientée objet.

L'obtention du titre de docteur puis d'un poste de maître de conférences en 1999 m'ont permis de poursuivre mes activités de chercheur.

Tout d'abord, sur la généralisation de la technique employée dans le simulateur de fautes. En effet, la co-direction du doctorat de Mr Laurent Capocchi de 2002 à 2005 a permis la mise au point d'un formalisme BFSDEVS (Behavioral Fault Simulation for Discrete EVent system Specification). Ce formalisme a été validé dans le domaine du test de circuits, et nos travaux actuels consistent à l'appliquer au domaine électro-énergétique avec la détection de pannes des machines électriques asynchrones intervenant au sein de systèmes d'éoliennes.

L'implémentation du simulateur de fautes ne permettant pas une autonomie complète pour la réalisation des tests sur les circuits digitaux, nous avons donc proposé à un étudiant de compléter ce simulateur par un générateur de séquence de test en vue d'obtenir un outil complet. Mr François Giamarchi y travaille depuis 2005 en gardant à l'esprit la notion de généricité.

Ces années m'ont aussi permis de découvrir d'autres domaines d'étude par le biais d'encadrement de stagiaires de DEA (Master Recherche) ou de doctorants :

- Co-encadrement du doctorat de Mr El-Hadi Khoumery sur le thème de la multi-représentation de données anthropologiques, astronomiques et archéologiques à l'aide des Systèmes d'Informations Géographiques.
- Encadrement de Mr Jean-Sébastien Gualtieri sur le thème de la réalisation d'un prototype de simulateur de feux en 3 dimensions.

Ce rapport scientifique présente de manière synthétique les travaux de recherches réalisés jusqu'à ce jour. Une organisation chronologique de ce rapport permet de mieux apprécier des différentes avancées réalisées. Je débute donc par une présentation de mes activités de

recherche doctorales. Je poursuis par la description de la recherche réalisée depuis mon doctorat jusqu'à ce jour. Enfin, une dernière partie permet de présenter les perspectives que je voudrais donner à mes travaux de recherche (Projet de Recherche).

CHAPITRE II - ACTIVITES DE RECHERCHE DOCTORALES

Le sujet de mon Doctorat s'intitule : « Simulation de Fautes Comportementales de Systèmes Digitaux Décrits à Haut Niveau d'Abstraction en VHDL » [Fed99]. Ce chapitre est une synthèse du travail réalisé. Dans un premier temps je présente la problématique de départ. La seconde partie présente notre modélisation des descriptions VHDL des circuits digitaux. Dans la troisième partie, la technique de simulation de fautes développée est explicitée. Enfin, la dernière partie traite de l'implémentation et des résultats obtenus.

2.1 - Présentation du problème

Les logiciels de simulation de fautes sont des outils essentiels intervenant dans le test de circuits. Le test de circuits permet de déceler d'éventuelles défaillances physiques. Ces défaillances pouvant résulter d'un défaut de fabrication ou d'une usure. Les effets de ces défaillances physiques sont décrits à l'aide de modèles de fautes définis à partir d'une description du circuit sous test. Ces modèles de fautes sont utilisés pour générer un ensemble de vecteurs de test qui appliqués en entrée du circuit physique permettent de déceler si le comportement d'un circuit est valide ou pas.

Le but d'un simulateur de fautes est de fournir la liste de toutes les fautes détectées par un vecteur de test. Pour un circuit donné, un simulateur de fautes doit disposer des informations suivantes :

- un modèle du système (pouvant être une interconnexion de portes, ou bien une description dans un langage informatique).
- la liste des fautes : définie par rapport au modèle du système (les fautes portent sur les éléments de base du modèle).
- le vecteur d'entrée : correspondant à l'ensemble des données que l'on applique aux ports d'entrée du système.

Les principales raisons pour lesquelles nous avons choisi de développer un simulateur de fautes comportementales à haut niveau d'abstraction sont :

- tout d'abord, le gain de temps par rapport au niveau porte : les outils existants au niveau logique s'avèrent aujourd'hui dépassés par la complexité croissante des circuits.
- et, l'ajout d'un outil de test à un stade plus avancé dans la conception de circuits digitaux.

Notre travail s'insérait dans le cadre d'un projet européen avec des instituts de recherche et des universités allemandes, espagnoles, portugaise et hollandaise (projet européen BELSIGN[1]).

---

[1] Sponsorisé par l'Union Européenne : Projet Human Capability and Mobility BELSIGN - contrat N° CHRX-CT 94-0459

Nous avons plus particulièrement collaboré avec l'institut Fraunoffer de Dresde (Allemagne) qui m’a accueilli pendant 3 mois en 1996.

Plus précisément, l’objectif de notre travail est de proposer un simulateur de fautes comportementales pour des circuits décrits en langage VHDL [Vhd87].
Notre approche est basée sur le principe de sensibilisation d'un chemin critique [Bar86, Nor89, Min82, One90] qui consiste à mettre en évidence une erreur sur un des éléments du modèle comportemental puis à la propager vers une sortie primaire. La définition d'un modèle comportemental support d'application du générateur, d'un modèle de fautes et d'un principe de génération permettent de mettre en œuvre cette démarche.

Nous proposons donc une approche intégrant les quatre étapes suivantes :

- une étude d’une modélisation adéquate permettant de rendre explicite les concepts de base des descriptions comportementales ;
- le choix d’un modèle de fautes comportementales ;
- la définition d’un principe de propagation de fautes à travers les éléments du modèle ;
- l’implémentation de ce principe en utilisant l’approche orientée objet et notre environnement de modélisation et de simulation.

2.2 - Modélisation des descriptions VHDL

La description écrite en un H.D.L. (Hardware Description Language) par le concepteur est un modèle alphanumérique. Ce type de modèle, utilisé par [Kho84, Lev82, Lin84, Min82, One90] pose un problème pratique de manipulation : un fichier texte est moins manipulable qu'une structure de données accessible par le langage dans lequel l'outil de simulation est développé. Pour remédier à ce problème, une transformation de la description textuelle en un modèle interne de type graphe plus facile à manipuler est développée. De plus notre modèle de type graphe met en évidence la séparation données/commandes [Cou91, Pla93].
Ce modèle interne [Fed96, Fed97] est donc composé de trois entités (cf. figure 2.1) :

- Le modèle de Commande : représenté à l’aide d'un graphe inspiré du formalisme de Petri [Pet81] (notions de place, de transitions et de marquage des places par un jeton), décrit de manière hiérarchique le séquencement des opérations se trouvant dans le modèle de donnée.
- Le modèle de Donnée qui est représenté à l'aide de deux types de nœuds : (i) les nœuds *données* correspondant aux variables et signaux manipulés ; (ii) et les nœuds *opérations* correspondant aux opérations effectuées sur les données.
- Les interactions Commandes/Données. Ces interactions sont réalisées par l'intermédiaire de liens d'exécution et de fin d'exécution, qui se situent toujours entre une place du modèle de commande et un nœud du modèle de donnée.

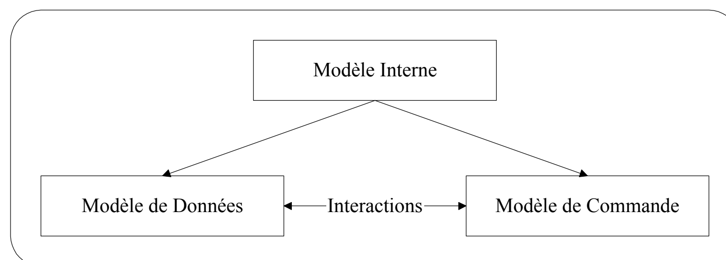


Figure 2.1 : Modélisation interne

La partie commande de notre description VHDL, modélisée à l'aide de Réseaux de Pétri (RdP) [Dav92, Pet81], est composée de trois éléments de base (cf. figure 2.2). L'élément *sélection* permet de modéliser un bloc d'instruction "IF … THEN … ELSE …", l'élément *assignation* représente une instruction d'affection, enfin l'élément *parallélisme* modélise la liaison entre les différents processus d'une description VHDL.

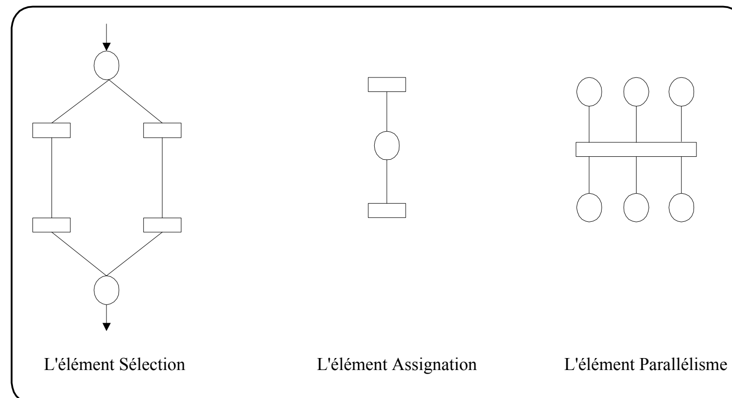


Figure 2.2 : Eléments de base du modèle de commande

La partie donnée de notre modèle interne est représentée à l'aide de deux types de nœuds : les nœuds "donnée" correspondant aux variables et signaux manipulés et les nœuds "opération" correspondant aux opérations effectuées sur les données. Il existe deux classes de nœuds "donnée" l'une représentant les signaux d'entrée/sortie et l'autre représentant les signaux ou variables internes. Chaque nœud "donnée" possède un champ "valeur" permettant de mémoriser sa valeur courante. De plus, un champ additionnel "old" est utilisé pour pouvoir mémoriser la valeur du nœud "donnée" au cycle précédent (pour un nœud donnée noté S, ce champ sera noté S_{old}). Il y a deux types de nœuds opération : les nœuds "décision" représentent un test algébrique ou booléen, le résultat de ce test est pris en compte par une structure sélective ou répétitive du modèle de commande, les nœuds "affectation" représentent l'affectation d'un objet par une opération algébrique ou booléenne (cf. figure 2.3).

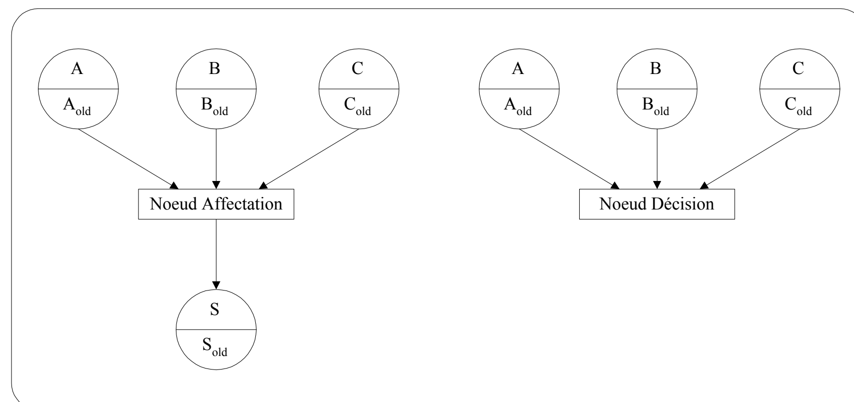


Figure 2.3 : Les nœuds opération

L'interaction commande/données est réalisée par l'intermédiaire de deux liens : un lien d'exécution et un lien de fin d'exécution. Ces liens se situent toujours entre une place du modèle de commande et un nœud opération du modèle de données (cf. figure 2.4).
Lorsqu'une marque est présente sur une place, le lien d'exécution est activé, et une opération dans le modèle de données est exécutée. Une fois l'opération accomplie le lien de fin d'exécution renvoie soit un message de fin d'exécution si le nœud opération est de type affectation, soit un résultat s'il est de type décision.

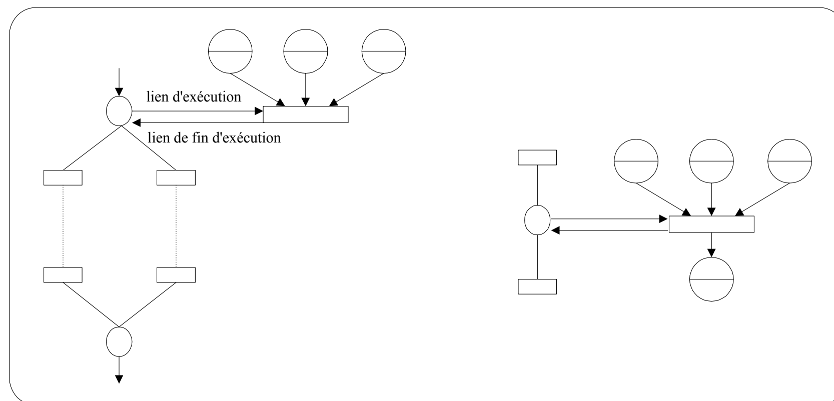


Figure 2.4 : Interaction Commande/Données

2.3 - Modèle de fautes

Il nous a semblé opportun de se reporter à un modèle de fautes déjà existant. La raison pour laquelle le modèle de fautes de S. Ghosh [Gho91] a inspiré notre modèle de fautes est le fait que ses résultats en termes de couverture de fautes sont satisfaisants (90%). Nous avons donc appliqué ce modèle de fautes à notre modélisation des descriptions VHDL, définit trois types de fautes et fait deux hypothèses pour les descriptions que nous souhaitons testées.

Les fautes de type F1 affectent les éléments du modèle de donnée.

- La valeur d'un nœud donnée est collée à V1 ou V2, où V1 et V2 expriment respectivement la borne inférieure et la borne supérieure du domaine de définition du signal ou de la variable représentée par le nœud donnée.
- La sortie d'un nœud opération est collée à V1 ou V2, où V1 et V2 expriment respectivement la valeur inférieure et la valeur supérieure pouvant être calculées par l'opération.

Les fautes de type F2 affectent les éléments du modèle de commande.

- La fonction de sélection d'une transition de commande impliquée dans un sous-graphe de type sélectif est collée à une valeur constante quelque soit le résultat retourné par le nœud décision, le chemin correspondant à la valeur de collage est toujours sélectionné.

Les fautes de type F3 affectent les éléments d'interaction commandes/données.

- Un lien d'interaction entre la transition de type opératif et son nœud opération associé est collé à 0 : cela implique que l'opération ne peut jamais être exécutée.

L'hypothèse de faute unique : on considère qu'une seule faute est présente dans le circuit, la multiplicité de fautes pouvant entraîner des complications (annulation d'une faute par une autre).
L'hypothèse qu'un signal ne peut être affecté dans deux processus différents, afin d'éviter tout conflit de valeur pour un signal.

2.4 - Technique de simulation de fautes

Nous allons à présent décrire les grandes lignes de notre principe de simulation de fautes comportementales (cf. figure 2.5) [San98] :

- La première étape consiste à déterminer la liste globale de fautes.
- Une fois cette liste constituée, il faut sélectionner le vecteur de test, il est constitué des signaux d'entrée du circuit.
- La troisième étape est la Simulation Saine (SS) de notre système, elle consiste à effectuer une simulation du circuit avec l'hypothèse qu'aucune faute n'est présente dans le système.
- Enfin, la dernière étape se nomme "Simulation avec Propagation de Listes de Fautes" (SPLF), elle consiste à propager des listes de fautes pendant une simulation du circuit. Cette propagation est régie par un ensemble de lois décrites ci-dessous.

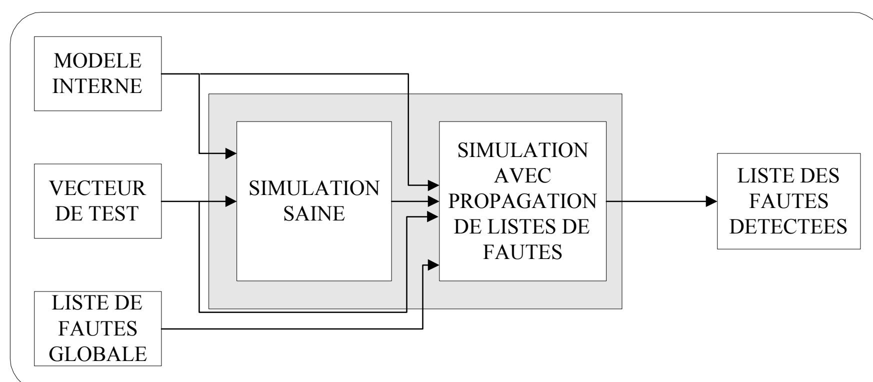


Figure 2.5 : Vue d'ensemble de notre méthodologie

La liste de fautes globales est déterminée à l'aide du modèle de fautes et du modèle interne de la description comportementale VHDL : c'est la liste de toutes les fautes susceptibles de se produire. Le résultat de notre simulation de fautes dépendra directement de la liste globale de fautes; en effet, la liste de fautes détectées est un sous ensemble de la liste globale de fautes.

La détermination du vecteur de test peut se faire de deux façons :

- si le simulateur de fautes est couplé à un générateur de tests, le vecteur de test est une sortie de ce générateur.
- sinon, le vecteur est choisi aléatoirement.

Le but de la SS est d'effectuer une simulation du circuit considéré pour un vecteur d'entrée donné avec pour hypothèse qu'aucune faute n'est présente dans le circuit. Les résultats obtenus après cette phase ont une importance primordiale pour la suite de notre méthode.

C'est la raison pour laquelle toutes les valeurs de tous les signaux et variables à chaque cycle sont mémorisées.

La SPLF consiste à simuler notre description pour un vecteur donné en propageant des fautes à travers notre modèle interne. Cette propagation est régie par un ensemble de principes qui sont décrits ci-dessous.

Principe 1 : Généralités

La propagation des listes de fautes se fait sur le modèle de données, mais dépend directement du modèle de commande.
Le résultat de la SPLF donne la "liste de fautes détectées" : C'est la liste des fautes localement observables sur les sorties primaires du circuit à la fin de la SPLF.

Principe 2 - Cas d'une assignation

Lors de la phase SPLF, lorsqu'une place appartenant à un élément "assignation" du modèle de commande est rencontrée, la règle suivante est appliquée :
Soit l'affectation :

$$\boxed{D1 <= E(Di)}$$

avec $0<i<n$ et E une fonction composée d'opérateurs présents dans les descriptions VHDL. D1,..., Dn étant des nœuds "donnée".

Lors du franchissement d'un élément de base "assignation", on associe à D1 toutes les fautes qui lui affecteraient une valeur différente de la sienne dans la simulation saine au même cycle et lors de la même affectation. La formule qui suit traduit comment nous calculons cette liste :

$$\boxed{L_{D1} = \bigcup_i L_{Di} \cup L_{JUMP} - L}$$

avec :

- L_{D1} étant la liste de fautes associée à D1.
- L_{Di} étant la liste associée à Di des fautes localement observables sur Di ($0<i<n$).
- L_{JUMP} étant soit l'ensemble vide si la valeur de l'attribut old de D_1 est égale à la valeur calculée en utilisant $E(D_i)$, soit le singleton $\{F3_{ASSIGNATION}\}$ (cf. Chapitre III, partie 4.2).
- L étant l'ensemble vide si la valeur de l'attribut old de D_1 est égale à la valeur calculée en utilisant $E(D_i)$. Dans l'autre cas L est égale à la liste de fautes L_{D1} calculée lors du cycle précédent.

Principe 3 - Cas d'une sélection

Lorsque la place rencontrée par le simulateur est la place amont d'un élément "sélection", l'activation du nœud décision va renvoyer au modèle de commande les informations suivantes : la détermination d'une "branche saine" (étant la branche appartenant au chemin sain) et d'une "branche fausse" (appartenant au chemin faux).
Nous stockons dans L_{FAUSSE} toutes les fautes qui impliquent un passage du simulateur sur la branche fausse.

Ces fautes peuvent être :

- de type F1 : elles induisent une réponse au test de la sélection différente de la réponse dans la SS.
- de type F2 : elles impliquent un passage du simulateur dans le chemin faux quelle que soit la réponse au test de la sélection (cf. figure 2.6).

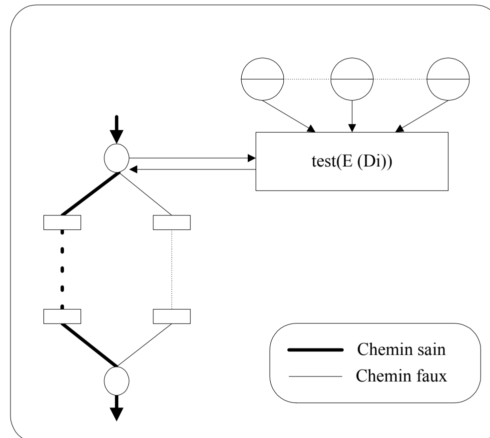


Figure 2.6 : Cas de la sélection

Concernant la simulation de la branche saine, elle s'effectue en utilisant les principes 1 et 2, et le principe 3 si nous rencontrons une autre sélection.
Concernant la branche fausse, pour chaque faute F de la liste L_{FAUSSE}, une simulation de toutes les instructions se trouvant dans la branche fausse est exécutée, avec pour hypothèse qu'aucune autre faute ne peut se produire (hypothèse de faute unique). Une fois arrivé à la jonction (dernière place de l'élément sélection), nous comparons les valeurs des signaux et variables du système avec celle obtenues lors de la simulation saine (sur la même place et au même cycle de simulation). Si pour un signal S, les deux valeurs diffèrent alors nous ajoutons la faute F à la liste de faute associée à S (F est localement observable sur S).

2.5 - Conclusion

Notre outil a été implémenté sur station SUN en langage C++, puis validé sur des descriptions issues des benchmarks d'ITC'99 [Fed00, Fed02]. La définition de ce simulateur de fautes pour des systèmes digitaux décrits à haut niveau d'abstraction a donc permis de présenter nos travaux dans différentes revues et conférences. La suite de ce travail de recherche a consisté à proposer un formalisme de modélisation et de simulation de systèmes à événements discrets intégrant la méthode de simulation concurrente.

Références de la thématique : [1, 3, 4, 5, 13, 14, 15, 16, 17, 18, 19, 20, 21]

CHAPITRE III - EVOLUTION DES CONCEPTS ET DEVELOPPEMENT DE METHODES

Le développement du formalisme BFS-DEVS [Cap05a] est la première évolution concernant mes travaux de recherche postdoctoraux. Ce travail de thèse confié à Mr Laurent Capocchi a permis de définir un socle totalement générique en vue d'accélérer les processus de simulation des systèmes à événements discrets. Ce chapitre décrit en détails le formalisme et sa validation toujours sur le thème du test de circuits digitaux.

3.1- Présentation du problème

Les travaux présentés dans cette section concernent la définition d'un simulateur concurrent pour des systèmes à événements discrets. L'objectif principal de cette recherche est de donner la possibilité de simuler de manière concurrente plusieurs expériences en une seule exécution. La démarche mise en place pour atteindre le but fixé repose sur les étapes suivantes :

1. L'étude des algorithmes de la SCC (Simulation Comparative Concurrente) et de la SFC (Simulation de Fautes Concurrente) [Agr81] avec la technique de propagation de liste de fautes.
2. L'étude du formalisme DEVS (Discrete EVent Specification) [Zei00].
3. L'intégration des algorithmes de la SFC à l'intérieur du formalisme DEVS dans un nouveau formalisme BFS-DEVS (Behavioral Fault Simulation-Discrete EVent Specification).
4. L'utilisation du formalisme BFS-DEVS pour transformer des descriptions comportementales VHDL en des réseaux de modèles BFS-DEVS.

Ces étapes, visibles sur la figure 3.1, permettent de construire le simulateur BFS-DEVS.

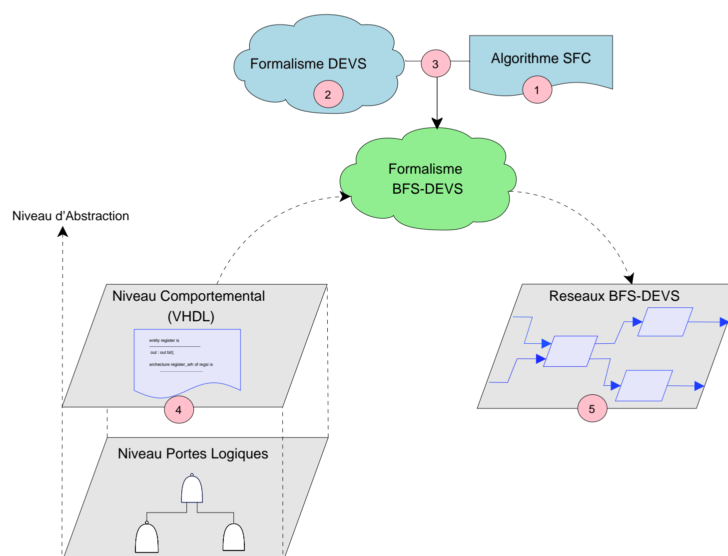


Figure 3.1 : Aperçu de la démarche BFS-DEVS

3.2 - La SCC et la SFC

La SCC [Ulr94] permet la simulation concurrente de plusieurs expériences. Typiquement, les expériences concurrentes sont celles qui sont dues : à la présence de fautes au sein des systèmes digitaux, à l'existence de plusieurs exécutions d'un programme ou à l'exécution de différentes instructions au sein d'un même programme. La puissance essentielle de la SCC réside dans le fait que plusieurs expériences sont simulées de manière *implicite* au travers d'une seule simulation. La SCC (cf figure 3.2) débute avec la simulation de *référence (ou R-expérience)* qui sera à l'origine de toutes les expériences concurrentes (*ou C-expériences).* Dans tous les cas, les simulations manipulent des données et se propagent à travers les éléments ou les instructions de l'expérience. Les C-expériences peuvent diverger de la R-expérience soit parce qu'elles empruntent des *éléments différents de la R-expérience*, soit parce qu'elles affectent des données avec des *valeurs différentes* de la *R-expérience*.

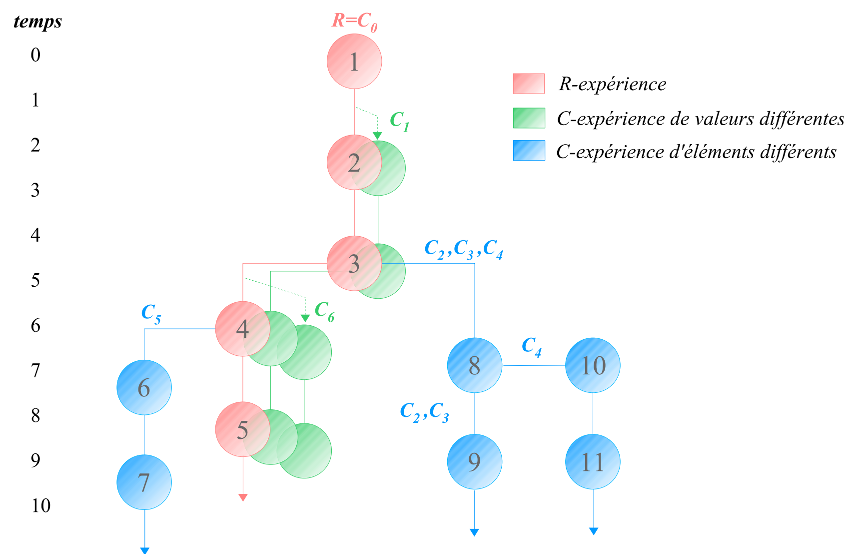


Figure 3.2 : La Simulation Comparative Concurrente (SCC)

La Simulation de Fautes Concurrente (SFC) sur les circuits digitaux a été la première application de la SCC [Bre76]. En effet, le principal domaine d'application dans lequel la SCC a montré la plus importante évolution est le domaine de la simulation de fautes des circuits digitaux décrits au niveau porte logique. L'algorithme concurrent repose sur l'activité des simulations fautives individuelles résultant des expériences fautives. Ces expériences fautives *(ou mauvaises portes)* se détachent de manière indépendante de l'expérience de référence R et peuvent diverger ou converger au cours de la simulation.

3.3 - Le formalisme DEVS

Basé sur la théorie des systèmes, *le formalisme DEVS (Discrete EVent system Specification)* a été introduit par le professeur B.P.Zeigler vers la fin des années 70 [Zei76]. Il permet une modélisation *modulaire* et *hiérarchique* des systèmes à événements discrets. Un système *(ou modèle)* est dit modulaire dans le sens ou il possède des ports d'entrée et de sortie permettant l'interaction avec son environnement extérieur. Dans le formalisme DEVS un modèle est vu comme une *« boîte noire »* qui reçoit et émet des messages sur ses ports d'entrées ou de

sorties. Cette section introduit le formalisme DEVS en distinguant la partie modélisation de la partie simulation.
Le formalisme DEVS définit deux catégories de modèles : les modèles atomiques et les modèles couplés. Les modèles atomiques sont chargés de représenter le(s) comportement(s) du système. Les spécifications classiques d'un MA sont les suivantes :

$$MA =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

où,

- $X = \{(p; v)|p \in Ports_entree; v \in X_p\}$ est la liste des ports et des valeurs d'entrées,
- $Y = \{(p; v)|p \in Ports_sortie; v \in X_p\}$ est la liste des ports et des valeurs de sorties,
- S est l'ensemble des variables d'états,
- $\delta_{int}: S \rightarrow S$ est la fonction de transition interne,
- $\delta_{ext}: Q \times X \rightarrow S$ est la fonction de transition externe où,
  - $Q = \{(s; e) \mid s \in S; 0 \leq e \leq t_a(s)\}$ est l'ensemble des états,
  - e est le temps écoulé depuis la dernière transition.
- $\lambda : S \rightarrow Y$ est la fonction de sortie,
- $t_a : S \rightarrow R^+$ est le temps de vie de l'état S *(réels positifs de 0 à ∞).*

Les modèles réagissent à deux types d'événements : les *événements externes* et les *événements internes*. Les événements externes proviennent d'un autre modèle, déclenchent la fonction de transition externe et mettent à jour le temps de vie du composant. Les événements internes correspondent à des changements d'états du modèle, déclenchent les fonctions de transitions internes et de sorties, le modèle calcul ensuite grâce à la fonction *ta* la date du prochain événement interne. La figure 3.3 illustre un modèle atomique en action.

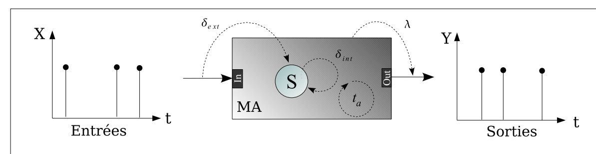


Figure 3.3 : Modèle Atomique DEVS

Le formalisme DEVS utilise la notion de *hiérarchie de description* qui permet la construction de modèles dits .couplés. à partir d'un ensemble de sous-modèles et de trois relations de couplage avec ces sous-modèles. Un modèle couplé est constitué de sous-modèles qui peuvent être *atomiques ou couplés* et il possède les *trois relations de couplage* suivantes :

- *une relation de couplage interne* pour le couplage entre les ports des sous-modèles qui composent le modèle couplé *(en rouge sur la figure 3.4)*,
- *une relation de couplage des entrées externes* pour le couplage entre les ports d'entrées du modèle couplé et les ports d'entrées des sous-modèles *(en bleu sur la figure 3.4)*,
- *une relation de couplage des sorties externes* pour le couplage entre les ports de sorties du modèle couplé et les ports de sorties des sous-modèles *(en vert sur la figure 3.4)*.

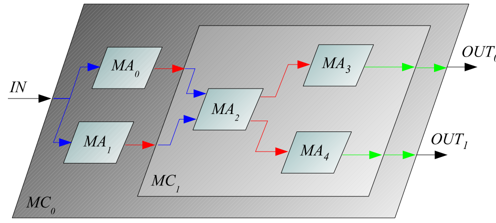


Figure 3.4 : Modèle Couplé DEVS

Dans le cas du formalisme DEVS classique avec port les spécifications d'un modèle couplé sont les suivantes :

$$MC = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$$

où,

- $X = \{(p; v) \mid p \in ports_entree; v \in X_p\}$ est la liste des ports et des valeurs d'entrées.
- $Y = \{(p; v) \mid p \in ports_sortie; v \in Y_p\}$ est la liste des ports et des valeurs de sorties.
- D est la liste des composants constituant le modèle couplé.
- $M_i = < X_i, Y_i, S_i, \delta_{ext,i}, \delta_{int,i}, \lambda_i, t_{a,i} >$ est un modèle atomique,
- Pour chaque modèle $i \in D \cup \{MC\}$, I_i est l'ensemble des modèles qui influencent i,
- $Z_{i,j}$ est la fonction de translation des sorties de i vers j telle que :
  - $Z_{MC,j} : X_{MC} \rightarrow X_j$ est la fonction de couplage des entrées externes,
  - $Z_{i,MC} : Y_i \rightarrow Y_{MC}$ est la fonction de couplage des sorties externes,
  - $Z_{i,j} : Y_i \rightarrow X_j$ est la fonction de couplage interne.

L'une des propriétés importantes du formalisme DEVS est qu'il fournit automatiquement un simulateur pour chacun des modèles. DEVS établit une distinction entre la modélisation et la simulation d'un système tel que n'importe quel modèle DEVS puisse être simulé sans qu'il ne soit nécessaire d'implémenter un simulateur spécifique. Chaque modèle atomique est associé à un *simulateur* chargé de gérer le comportement du composant et chaque modèle couplé est associé à un *coordinateur* chargé de la synchronisation temporelle des composants sous-jacents. L'ensemble de ces modèles est géré par un coordinateur spécifique appelé *« Root »* (cf figure 3.5).

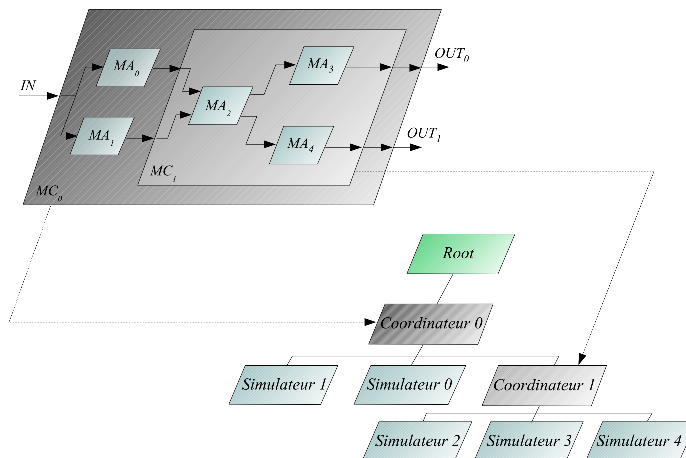


Figure 3.5 : Simulation DEVS

L'ordre d'exécution des fonctions comportementales (δ_{int}, δ_{ext}, λ, ta) d'un modèle atomique DEVS sous-entend la présence d'un mécanisme de simulation. Ce mécanisme est géré par le composant simulateur qui, comme le coordinateur, peut recevoir ou émettre 4 types de messages :

- Le **message d'initialisation (i,t)** permet à tous les acteurs d'effectuer une synchronisation temporelle initiale.
- Le **message de transition interne (*,t)** permet la gestion d'un événement interne avec l'exécution de la fonction δ_{int}.
- Le **message de sortie (y,t)** permet le transport des sorties *(données par* λ*)* aux éléments parents et résulte de la réception d'un message (*,t).
- Le **message de transition externe (x,t)** permet la gestion d'un événement externe avec l'exécution de la fonction δ_{ext}.

Cependant, le formalisme DEVS ne possède aucun algorithme permettant d'accomplir une simulation de comportements fautifs de manière concurrente. Nous proposons donc d'intégrer à DEVS des algorithmes spécifiques afin de définir un nouveau formalisme capable de simuler de manière concurrente des fautes comportementales au sein de systèmes tels que les circuits digitaux.

3.4 - Le formalisme BFS-DEVS

Afin d'appliquer les algorithmes de la SCC dans le domaine de la simulation de fautes comportementales pour des systèmes discrets, nous définissons un noyau de simulation générique appelé "*BFS-DEVS*" *(Behavioral Fault Simulation for DEVS)* [Cap05b].
Cette approche qui sépare le noyau de simulation de la représentation des modèles permet:

- d'intégrer les algorithmes de la SCC indépendamment des domaines d'applications,
- de définir des modèles spécifiques sans se soucier des algorithmes de simulation,
- de spécifier facilement le comportement fautif d'un modèle *(ou faute)*,
- de réutiliser les modèles définis dans les librairies.

Le nouveau formalisme BFS-DEVS permet de spécifier le comportement fautif d'un modèle. Les spécifications d'un modèle atomique BFS-DEVS sont équivalentes à celle d'un modèle atomique DEVS ajoutées des règles suivantes :

- la transition dans un nouvel état fautif résulte de l'arrivée sur un port d'une valeur fautive,
- le comportement fautif est spécifié à l'intérieur d'une nouvelle fonction de transition externe fautive δ_{fault} .

Un modèle atomique DEVS n'admet pas de comportement fautif et est considéré comme *« sain ».* Il est représenté par la structure classique décrite dans la partie 3.1.3 :

$$MA = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

Afin de prendre en compte le comportement fautif d'un tel modèle, nous modifions sa structure de la façon suivante :

$$MA' = < X', Y', S', \delta'_{int}, \delta'_{ext}, \delta_{fault}, \lambda', t'_a >$$

où,

- $X' = X \cup X^f$ est la liste des ports et des valeurs d'entrées avec $X^f = \{(p, v_f)\}$ le nouvel ensemble des ports et des *valeurs fautives* d'entrées.
- $Y' = Y \cup Y^f$ est la liste des ports et des valeurs de sorties avec $Y^f = \{(p, v_f)\}$ le nouvel ensemble des ports et des *valeurs fautives* de sortie.

- $S' = S \cup S^f$ est la liste des états avec S^f le nouvel ensemble des *états fautifs* pris par le modèle lorsqu'il reçoit une valeur fautive.
- $F = \{F_i\} \cup \emptyset$, $i \in R^+$ est l'ensemble des fautes F_i.
- $\delta'_{int} : S' \times F \rightarrow S$ est la fonction de transition interne avec la restriction : $\delta'_{int}(s,\emptyset) = \delta_{int}(s)$.
- $\delta'_{ext} : Q \times X' \times F \rightarrow S$ est la fonction de transition externe qui vérifie : $\delta'_{ext}(s,\emptyset,e,x) = \delta_{ext}(s,e,x)$ si $x \in X$.
- δ_{fault}: $Q \times X^f \times F \rightarrow S^f$ est la *fonction de transition externe fautive* amenant le système dans un état fautif $s_f \in S^f$.
- $\lambda' : X' \times F \rightarrow Y'$ est la fonction de sortie qui vérifie : $\lambda'(s,\emptyset) = \lambda(s)$.
- $t'_a : S' \times F \rightarrow R^+_\infty$ est la fonction d'avancement du temps avec la restriction : $t'_a(s,\emptyset) = t_a(s)$.

Les spécifications BFS-DEVS sont similaires à celles du formalisme DEVS. La différence apparaît lorsque l'on considère l'arrivée d'une valeur fautive x_f ($x_f \in X^f$) sur le modèle. Dans ce cas, le nouvel état fautif est déterminé par la fonction de transition externe fautive δ_{fault} . Si une valeur sa*ine* x ($x \in X$) arrive sur le modèle, le nouvel état du système est toujours déterminé par la fonction de transition externe δ'_{ext} .
Les spécifications sur le couplage d'un modèle BFS-DEVS sont identiques à celle d'un modèle DEVS. Cependant, si le modèle de fautes utilisé contient des fautes pouvant modifier la structure du modèle *(fautes structurelles)*, le couplage peut être modifié au cours de la simulation [Kof00]. Nous considérons uniquement les fautes comportementales et la propriété de *« closure under coupling »* permettant la hiérarchie de composition entre les modèles BFS-DEVS est préservée.

Concernant la partie simulation, c'est grâce à l'introduction d'un nouveau type de message (x_f, t) représentant un événement externe fautif que nous sommes capables d'activer le comportement fautif d'un modèle.
En effet, nous savons que la communication entre composants dans la hiérarchie de simulation se fait par le biais de quatre types de messages. Si nous voulons distinguer le comportement fautif au sein de la simulation il faut définir un nouveau type de message qui, au même titre que le *x-message* (x, t), permettra d'activer le modèle lorsqu'un événement fautif x_f est présent sur ces ports à l'instant t. Le coordinateur, par la connaissance de l'ensemble des valeurs d'entrées X', aura alors la possibilité d'envoyer deux types de messages externes :

- un *x-message* sur ces fils imminents si ceux-ci doivent avoir un comportement sain*,*
- un *f-message* sur ces fils imminents si ceux-ci doivent avoir un comportement fautif.
- Grâce à cette distinction nous sommes capables d'effectuer les simulations saines et fautives en concurrence.

Le mécanisme de la simulation BFS-DEVS (figure 3.6) s'appuie sur les notions de la SCC *(cf. section 3.2)* pour accomplir la simulation concurrente de fautes au sein des modèles BFS-DEVS. Ce mécanisme consiste en une simulation saine *(R-expérience)* concurrencée par plusieurs simulations fautives *(C-expériences)* induites par des fautes comportementales pouvant intervenir à l'intérieur des modèles. L'une des propriétés importantes de ce mécanisme est qu'il utilise une technique de propagation de liste de fautes (LFi sur la figure 3.6) permettant le rassemblement des simulations fautives pour une meilleure observabilité.

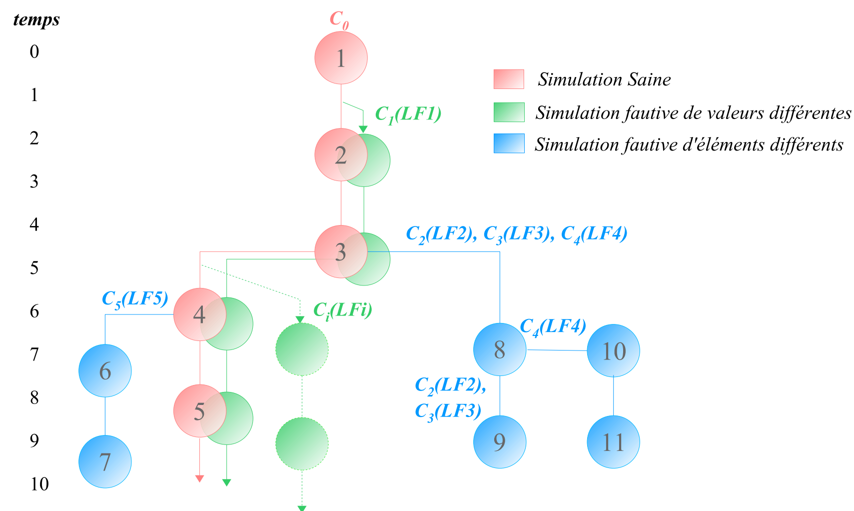


Figure 3.6 : Simulation BFS-DEVS

3.5 - Application aux circuits digitaux

Le domaine d'application choisi pour valider notre approche est celui de la simulation de fautes comportementales au sein de circuits digitaux décrits en VHDL. Le sous-ensemble VHDL étudié ainsi que le modèle de fautes utilisé est le même que celui présenté dans le chapitre 2. La première phase consiste en la modélisation (transformation) des descriptions VHDL textuelles en composants BFS-DEVS.

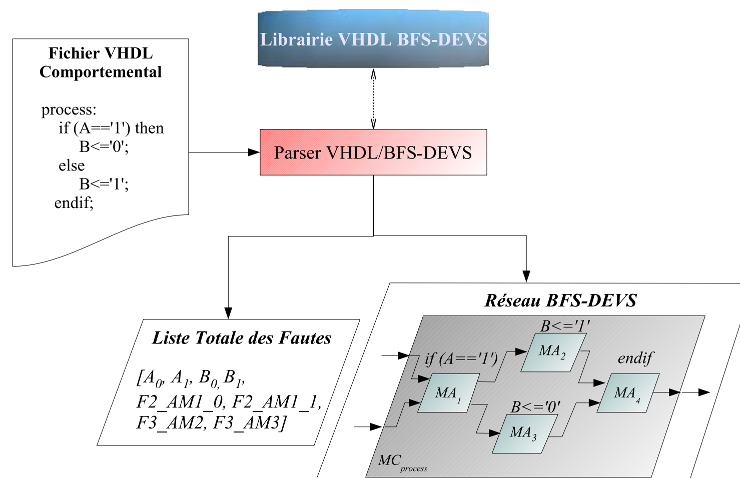


Figure 3.7 : La transformation VHDL/BFS-DEVS

Le réseau BFS-DEVS de la figure 3.7 est obtenu grâce à un parser qui analyse le fichier VHDL en utilisant une librairie de modèles spécifique au domaine des descriptions VHDL comportementales. Cette librairie est construite par l'utilisateur en considérant que n'importe

quelle description VHDL comportementale peut être représentée par l'association des quatre modèles BFS-DEVS de base [Cap03, Cap04] suivants (figure 3.7) :

- **le modèle atomique Assignment** représentant une instruction d'affectation VHDL (MA2 et MA3).
- **le modèle atomique Conditional** représentant une instruction conditionnelle VHDL (MA1).
- **le modèle atomique Junction** associé aux instructions de fin de code comme endif, endcase, etc. (MA4).
- **le modèle couplé Process** associé à l'instruction process VHDL (MCprocess).

Deux modèles atomiques BFS-DEVS supplémentaires sont ajoutés à la librairie pour les besoins de la simulation concurrente :

- **le modèle atomique Generator** : Il permet la génération d'événements destinés à exécuter les composants du réseau BFS-DEVS.
- **le modèle atomique ProcessEngine** : Il permet la gestion de la synchronisation des modèles couplés Process présents dans le réseau BFS-DEVS.

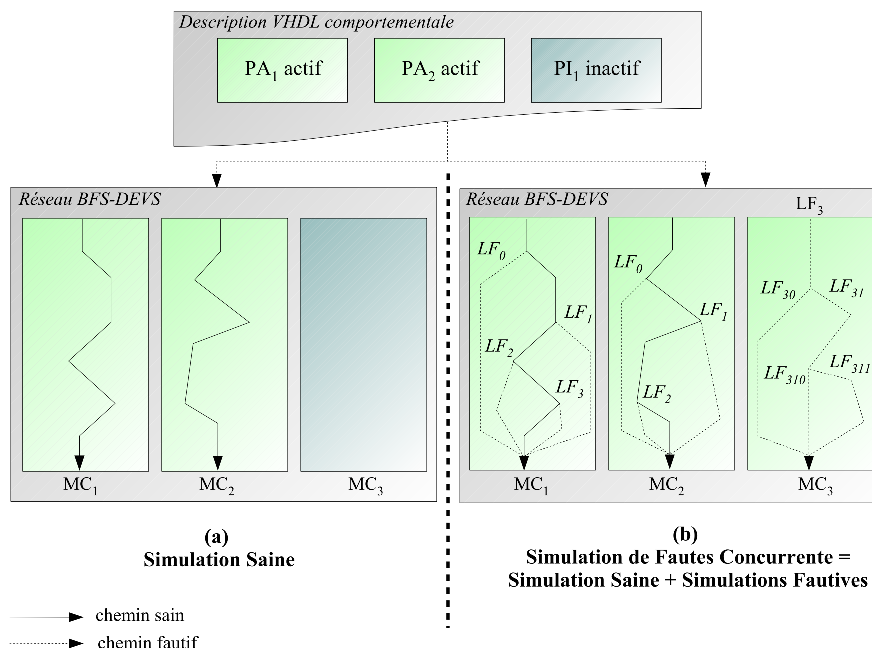


Figure 3.8 : Simulation de fautes concurrente BFS-DEVS

La simulation BFS-DEVS consiste à réaliser une simulation saine concurrencée par des simulations fautives (cf. figure 3.8) [Cap05, Cap06a]. La propagation des listes de fautes se faisant par découpage et réorientation. Les règles de propagation des listes de fautes sont issues des travaux réalisés pendant mon doctorat (cf. section 2.4).

3.6 - Résultats

Nous avons implémentés en langage Python [Bea00] le simulateur de fautes ainsi que les composants VHDL-BFS-DEVS. Cet outil a été validé sur 10 descriptions VHDL (cf Tableau 3.1) issues des benchmarks ITC'99 [Cor97,Cor00].

benchmarks	#lignes	#proc	#affectations	#conditionnelles	#signaux	#variables
B01	110	1	35	12	6	1
B02	70	1	19	7	4	1
B03	141	1	56	14	7	14
B04	102	1	40	12	7	13
B05	310	3	99	46	19	5
B06	128	1	50	11	8	1
B07	92	1	33	9	4	6
B08	89	1	22	8	8	4
B09	103	1	34	8	7	2
B10	167	1	74	19	13	8

Tableau 3.1 : Caractéristiques des 10 premier benchmark ITC'99

L'architecture du prototype est présentée sur la figure 3.9. Le Parser VHDL/BFS-DEVS est issu d'une adaptation de l'outil GENESI (GENErateur de StimulI) [Pao04]. Nous avons utilisés une séquence de test pseudo-aléatoire composée de 10000 vecteurs.

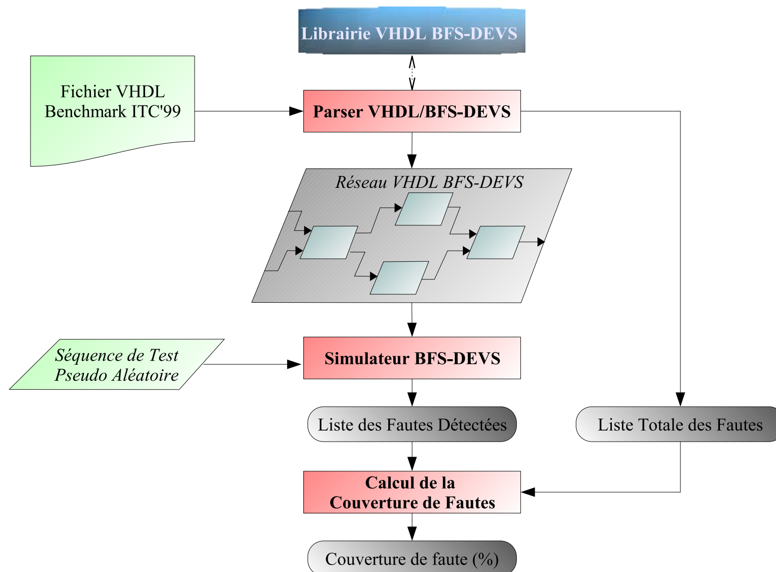


Figure 3.9 : Architecture du prototype

Le tableau 3.2 présente la couverture de fautes obtenue ainsi que la couverture d'instruction représentative de la testabilité d'un programme [McC76].

benchmarks	Fautes	Fautes détectées	Couverture d'instructions (%)	Couverture de fautes (%)
B01	79	73	100	92,94
B02	48	42	100	87,50
B03	130	108	100	83,07
B04	105	89	100	84,76
B05	251	61	69,58	24,30
B06	95	78	100	82,10
B07	76	66	91,83	86,84
B08	64	63	100	98,43
B09	68	57	100	83,82
B10	163	143	100	87,67

Tableau 3.2 : Résultats de la simulation BFS-DEVS

Ces résultats montrent qu'il possible d'obtenir des couvertures de fautes acceptables compte tenu des séquences de test pseudo-aléatoire appliquées.

3.7 - Génération de vecteurs de test

A la suite de la validation de notre simulateur de fautes concurrent sur des descriptions comportementales VHDL, la problématique concernait la génération des vecteurs de test qui était pseudo-aléatoire. C'est la raison pour laquelle, nous avons envisagé le développement d'un générateur de vecteur de test déterministe. En effet, le couplage de BFS-DEVS avec ce générateur permettrait d'obtenir un outil de test complet (cf. figure 3.10).

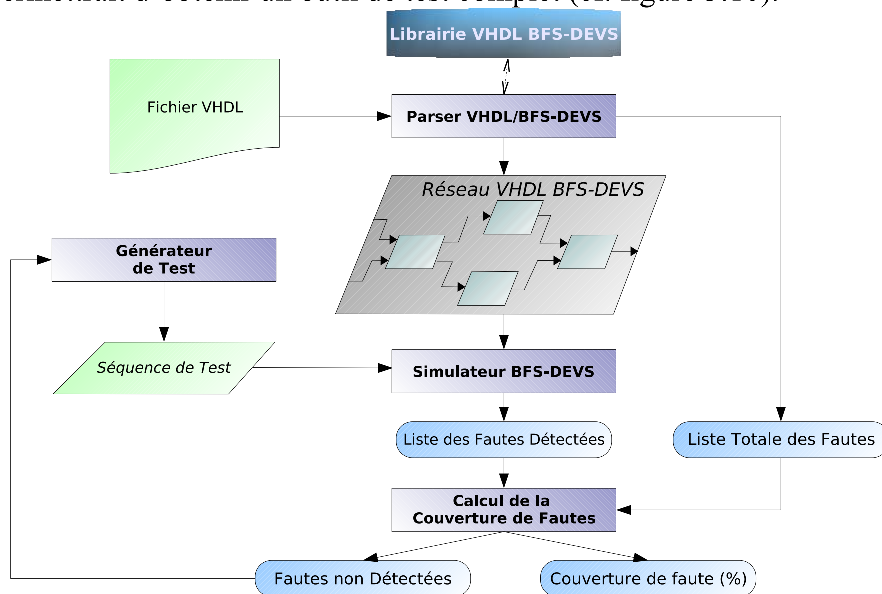


Figure 3.10 : Outil de test complet

Ce travail confié à Mr Giamarchi s'est décomposé en plusieurs étapes décrites ci-après.

La première étape consiste à établir une métrique probabiliste qui permet d'évaluer le degré de détection d'une faute. Cette méthode est basée sur les critères de contrôlabilité et d'observabilité d'un circuit et permet de dire si une faute est facilement testable ou pas.
Cette étude a permis de valider les résultats obtenus avec le simulateur de fautes concurrent. En effet, les fautes difficilement testables sont celles non observées dans nos résultats.

La seconde étape basée sur le graphe de dépendance des descriptions VHDL consiste à définir la séquence de test à appliquer en entrée du circuit pour pouvoir détecter une faute donnée. Le graphe de dépendance est un graphe orienté représentant les dépendances entre les différentes instructions d'un circuit (cf figure 3.11).

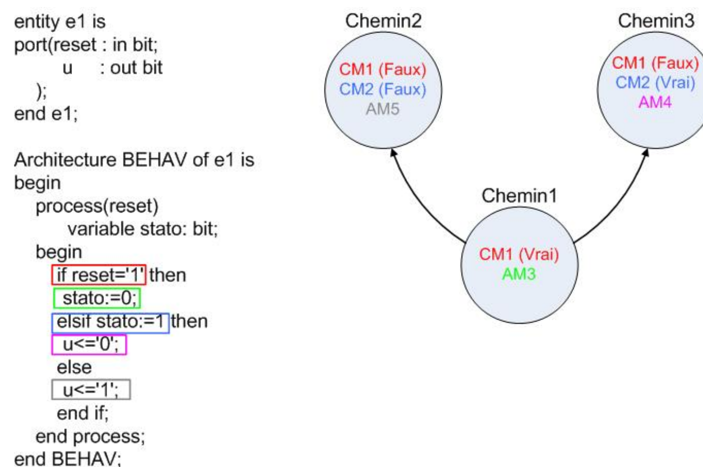


Figure 3.11 : Exemple de graphe de dépendance

La méthodologie en cours de développement s'oriente vers un algorithme de parcours du graphe de dépendance avec propagation de contraintes. Ce parcours se fait dans deux sens :

- depuis le nœud lié à la faute jusqu'aux entrées du circuit (contrôlabilité).
- depuis le nœud lié à la faute jusqu'aux sorties primaires du circuit (observabilité).

3.8- Conclusion

Notre travail réalisé sur le thème des circuits digitaux a permis l'émergence d'un nouveau thème de recherche dans l'équipe informatique du laboratoire SPE UMR 6134 : « la simulation concurrente de systèmes à événements discret ». Notre méthodologie doit à présent être éprouvée dans d'autres champs disciplinaires.

Références de la thématique : [2, 7, 10, 11, 12]

CHAPITRE IV - ACTIVITES DE RECHERCHE RECENTES

Parallèlement à mon axe de recherche principal, j'ai pu découvrir de nouvelles thématiques directement liées aux projets de recherche portés par l'université de Corse.
Dans la première section, nous présentons le travail entrepris avec BFS-DEVS en vue de l'appliquer dans le projet « Energies Renouvelables » : l'application choisie étant la détection de pannes des machines électriques asynchrones intervenant au sein de systèmes d'éoliennes.
Je participe notamment au co-encadrement de la thèse de Mr El-Hadi Khoumery et de Mr Jean-Sébastien Gualtieri dans le cadre du projet de recherche « Inventaire et valorisation des patrimoines » de l'université de Corse. Ces thématiques sont décrites dans la seconde et la troisième partie de ce chapitre.

4.1 - Détection de pannes dans les systèmes d'énergie renouvelable

Le projet « Energies Renouvelables » entre dans le cadre des **6 projets pluridisciplinaires** reconnus par l'Institut de l'Environnement (IE) de l'Université de Corse. Il a pour but de regrouper les compétences des chercheurs de différentes disciplines de l'Université de Corse sur la thématique des énergies renouvelables. Dans ce cadre, nous avons débuté une collaboration avec l'université de Picardie dont la finalité concerne la détection de pannes dans les machines électriques intervenant au sein de systèmes de production d'énergie renouvelable [Cap06b]. La répartition des rôles est la suivante :

- Les membres du laboratoire CREA - UPRES EA3299 travaillent sur la partie modélisation des machines électriques asynchrones.
- Notre équipe adapte la méthode de simulation concurrente à ce domaine bien spécifique en vue d'obtenir un outil de modélisation et de simulation.

Pour parvenir à nos fins, nous avons organisés notre travail de la manière suivante :

- Etude de la méthode de modélisation des machines électriques asynchrones développée par l'équipe du CREA.
- Vérification de l'applicabilité de DEVS à de tels systèmes.
- Elaboration d'une librairie BFS-DEVS pour les machines électriques asynchrones.
- Elaboration d'un modèle de pannes.
- Application de la méthode de simulation concurrente pour la détection de pannes
- Validation avec les résultats obtenus sur bancs d'essai.

La première étape est explicitée dans la première section. La seconde phase, en cours de réalisation, est décrite dans la section 4.1.2. Enfin la dernière parie décrit de manière détaillée la suite à donner à nos travaux.

4.1.1 – La modélisation électrique des machines électriques asynchrones

Notre première étape consiste en l'étude des modèles mathématiques des machines électriques asynchrones [Yaz05]. Dans un premier temps nous nous sommes restreint à une partie de ces machines : le stator.

Le modèle électrique du stator, représenté sur la figure 4.1, fait intervenir à la fois des éléments passifs comme les résistances r_s et des éléments actifs comme les inductances propres L_s et les inductances mutuelles L_{ms}.

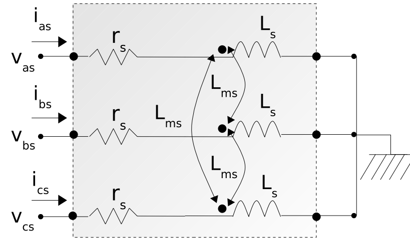


Figure 4.1 : Modèle électrique du stator

La mise en équation de ce circuit, établie grâce à la loi des mailles, donne le système d'équation :

$$\begin{cases} v_{as} = r_s.i_{as}(t) + L_s \frac{d}{dt} i_{as}(t) - \frac{L_{ms}}{2} \left[\frac{d}{dt} i_{bs}(t) + \frac{d}{dt} i_{cs}(t) \right] \\ v_{bs} = r_s.i_{bs}(t) + L_s \frac{d}{dt} i_{bs}(t) - \frac{L_{ms}}{2} \left[\frac{d}{dt} i_{as}(t) + \frac{d}{dt} i_{cs}(t) \right] \\ v_{cs} = r_s.i_{cs}(t) + L_s \frac{d}{dt} i_{cs}(t) - \frac{L_{ms}}{2} \left[\frac{d}{dt} i_{as}(t) + \frac{d}{dt} i_{bs}(t) \right] \end{cases}$$

Ce système montre que chaque phase x ($x \in \{a,b,c\}$) est composée des trois termes suivants:

- $r_s.i_{xs}(t)$: la chute résistive,
- $L_s.\frac{d}{dt}i_{xs}(t)$: l'effet inductif produit par l'enroulement considéré,
- $-\frac{L_{ms}}{2} \left[\frac{d}{dt} i_{x's}(t) + \frac{d}{dt} i_{x''s}(t) \right]$: l'effet inductif produit par les enroulements voisins des autres phases.

Les tensions d'alimentations $v_{as}(t)$,$v_{bs}(t)$et $v_{cs}(t)$ sont des fonctions sinusoïdales du temps de la forme:

$$\begin{cases} v_{as}(t) = V_m \sin(2\pi f t) \\ v_{bs}(t) = V_m \sin(2\pi f t - \frac{2\pi}{3}) \\ v_{cs}(t) = V_m \sin(2\pi f t - \frac{4\pi}{3}) \end{cases}$$

avec $V_m = \frac{Um.\sqrt{2}}{\sqrt{3}}$, U_m étant la tension simple non efficace et f la fréquence du signal.

4.1.2 – La modélisation et la simulation du stator avec PowerDEVS

L'environnement PowerDEVS [Kof03a] propose un ensemble de librairies composées de modèles atomiques et couplés permettant la modélisation des systèmes dans le formalisme DEVS. Nous utilisons les librairies appelées "Continuous" et "Sources" car elles sont composées d'éléments de base comme l'intégrateur ou les sources de tension sinusoïdales. Ces modèles vont nous permettre de modéliser le système d'équations représentant le stator (cf. Figure 4.2).

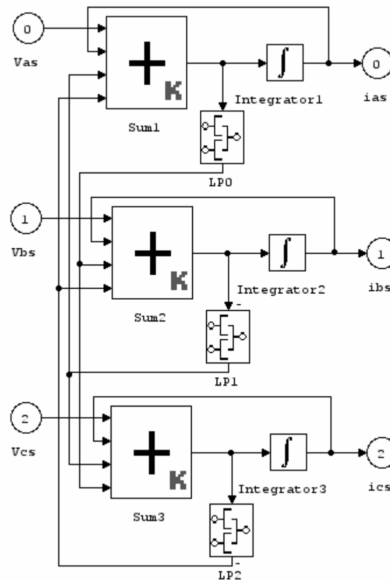


Figure 4.2 : Modèle Couplé du Stator

L'environnement PowerDEVS permet donc la modélisation et la simulation des systèmes de puissance avec couplage. L'application au modèle du stator nous a permis de tester les méthodes de quantification QSS (Quantized State System)[Kof03b, Kof05] et de voir à quel point elles peuvent être efficaces en termes de rapidité de simulation et d'exactitude. Certes cette approche est commune à ce genre de système mais n'a jamais été expérimenté dans l'environnement PowerDEVS, et nous conforte dans le choix du formalisme DEVS pour la suite de nos travaux.

4.1.3 – Conclusion et Perspectives

Les recherches actuelles concernent la modélisation et la simulation d'une machine asynchrone complète. Nous intégrerons ensuite la partie mécanique, présente dans le cas de systèmes d'éoliennes. Une phase de comparaison de nos résultats avec ceux issus du banc d'essai du laboratoire CREA permettra de valider notre approche.
Nous nous attèlerons ensuite à la définition d'un modèle de fautes pour de tels systèmes : cela nécessitera des séries de tests sur banc d'essai. Nous transformerons ensuite les modèles PowerDEVS validés en modèles BFS-DEVS [Cap03] en vue d'effectuer une simulation des défauts grâce notre méthode de simulation concurrente.

Référence de la thématique : [6]

4.2 - Intégration de la hiérarchie d'abstraction de données spatiales dans un SIG

Le projet « Inventaire et valorisation des patrimoines » entre dans le cadre des **6 projets pluridisciplinaires** reconnus par l'Institut de l'Environnement (IE) de l'Université de Corse. Il tente de répondre à une double urgence : la conservation et la transmission notamment des patrimoines culturels locaux menacés.

Le travail de doctorat de Monsieur El-hadi Khoumeri concerne le problème de la définition des notions d'abstraction et de vue de données spatiales soulevé par les anthropologues, les archéologues et les astronomes du projet [Kho05a, Kho05b].
Ces derniers ont entrepris une série de travaux concernant notamment :

- Le recensement et la localisation de sites néolithiques, de toponymes, de légendes.
- Des mesures de l'orientation des sites funéraires néolithiques corses.
- Un travail d'analyse à effectuer à différents niveaux d'interprétation :
    - Analyse des localisations des sites par rapport la topographie et à la toponymie
    - Analyse des liens entre sites et légendes
    - Analyse des orientations d'un point de vue astronomique.

L'intégration de plusieurs niveaux d'abstraction dans un SIG (Système d'Information Géographique) facilite la manipulation de ces données. Notre travail se décompose en plusieurs étapes :

- Formalisation des concepts de domaines de niveaux d'abstraction.
- Conception orientée objet de ces concepts en vue d'automatiser les fonctions de transfert automatiques entre les différents niveaux d'abstraction.
- Implémentation en Visual Basic.

L'utilisation des SIG dans le domaines des Sciences Humaines et Sociales (SHS) reposera de toute façon sur la capacité des chercheurs dans le domaine des SIG à progresser dans les problématiques liées à la multi-représentation.

Les recherches en multi-représentation peuvent être classifiées en deux types d'approches : des approches dites « orientées traitements » utilisant des algorithmes de généralisation [Rua99] ou bien des approches dites « orientées représentations » qui utilisent une base de données où sont stockés les différents niveaux de détail [Van01, Spa00, Bed02]. Au vue des analyses et synthèses de ces deux types d'approches, nous pouvons affirmer que la définition de concepts autorisant une combinaison des deux approches permettrait de répondre efficacement aux problèmes mis en évidence. En effet l'utilisateur se trouve dans certains cas dans l'obligation de chercher des nouvelles représentations non encore stockées dans la base de données. Il ne peut dans ce cas se satisfaire uniquement des approches orientées représentations. Il doit donc faire appel à des approches dites de traitements ce qui implique une utilisation de la généralisation afin de générer de nouvelles représentations.

Après une analyse des capacités des SIG en matière de recherche en SHS et une mise en parallèle avec les problèmes liés notamment à l'archéoastronomie, nous pouvons lister les problèmes à résoudre d'un point de vue informatique :

- Représentation des informations spatiales à différentes échelles
- Représentation des informations spatiales à différents niveaux d'interprétation
- Représentation des informations spatiales correspondant à différents points de vue d'une localisation
- Difficultés à relier de façon géométrique les informations spatiales.
- Difficultés à déduire et à représenter des données issues de l'archéoastronomie

Nous avons défini les concepts de base afin de proposer une solution aux problèmes ci-dessus : notamment les notions de niveaux d'abstraction et de domaines de données spatiales. En effet une définition précise des notions de domaines et de niveaux d'abstraction permettra de proposer une solution aux 3 premières difficultés mises en évidence précédemment. Les

deux dernières difficultés seront résolues plus facilement pour chaque domaine et pour chaque niveau d'abstraction de données spatiales.

Une zone spatiale devra pouvoir donc être visualisée selon différents points de vue et à différents niveaux d'abstraction. A une zone spatiale correspondra donc plusieurs représentations spatiales qui pourront être générées automatiquement par l'utilisateur.
Dans notre étude nous représentons un objet spatial comme étant un élément spatial élémentaire appartenant à une représentation d'un domaine donné et à un niveau d'abstraction donné. Notre approche a consisté à développer des concepts qui permettent à un utilisateur de définir autant de niveaux d'abstraction (niveaux de détails) qu'il désire. Il va de plus pouvoir définir des fonctions de transfert d'information permettant la génération automatique des représentations lorsque l'utilisateur veut changer de niveau d'abstraction. Ces fonctions devront permettre :

- d'une part des transferts d'information d'un niveau d'abstraction moins détaillé vers un niveau plus détaillé ; nous appelons ce type de génération de représentation la ***décomposition***,
- d'autre part des transferts d'information inverses ; nous appelons dans ce cas le processus de génération de représentation l'**agrégation.**

Bien sûr l'utilisateur aura à sa disposition un ensemble de fonctions de transfert prédéfinies : nous pouvons citer comme exemple de fonction de transfert les algorithmes de généralisation [RUA 99]. Ces fonctions de transfert d'information permettent de déduire automatiquement une représentation associée à un domaine D et à un niveau d'abstraction N à partir d'une représentation associée à un même domaine D et à un niveau d'abstraction N'.

La conception orientée objets de ces différents concepts a permis l'implémentation en Visual Basic d'un premier outil mettant en évidence les points fort de notre approche. La prochaine étape consistera à la réalisation d'un couplage avec un outil de SIG du commerce.

Référence de la thématique : [8, 9, 22, 23]

4.3 – Analyse/Synthèse de voix pour la valorisation et l'apprentissage des chants polyphoniques corses

Dans le cadre du programme de recherche de l'Université de Corse «Inventaire et valorisation du patrimoine», la création d'une médiathèque est en cours d'étude. Les objectifs du projet sont de proposer dans l'espace « chant » de la future médiathèque, une description et une caractérisation du chant corse et une méthode d'apprentissage du chant par mimophonie (méthode gestuelle définie à l'auditorium de Pigna) [Gua06]. La mimophonie est une série de gestes qui indiquent au chanteur à quelle hauteur il doit placer sa note par rapport à une tonalité fondamentale

Dans le cadre de ce projet en collaboration avec l'IRCAM (Institut de Recherche et Coordination Acoustique/Musique), Mr Gualtieri a débuté son doctorat en se familiarisant avec les domaines d'études suivants :

- l'analyse et la caractérisation des chants polyphoniques,
- l'étude de la gestuelle musicale et de la synthèse de voix

Un séjour de recherche de 4 mois à l'IRCAM (en 2006) a permis à Mr Gualtieri d'étudier le logiciel Max/MsP[Max], ainsi que les domaines directement liés : la voix, l'analyse et la synthèse de la voix, le contrôle gestuel.
Concernant la partie captation des gestes : nous avons entrepris la création de gants équipés de capteurs. Nous avons donc conçu deux gants ayant chacun six capteurs : cinq capteurs de flexion pour les doigts et un accéléromètre afin de connaître les déplacements de la main sur deux axes.

Les prochaines étapes concernent :

- Etude du langage de mimophonie.
- Définition formelle d'un langage informatique équivalent au langage de mimophonie.
- Définition formelle des tâches de captation des gestes et de synthèses des sons
- Implémentation du logiciel.

CHAPITRE V - PROJET DE RECHERCHE

Dans ce mémoire, les principales activités de recherche auxquelles j'ai participé ces onze dernières années (1995-2006) sont présentées. Ces recherches concernent essentiellement la définition d'une méthode de simulation concurrente pour des systèmes à événements discrets. J'ai donc contribué à la naissance et au développement de cette thématique au sein de l'axe « modélisation et conception de systèmes » de l'UMR CNRS 6134. Le travail réalisé notamment en encadrant des jeunes chercheurs a permis d'obtenir un outil de simulation concurrente et de le valider dans le domaine d'application des circuits digitaux. L'orientation de nos travaux a toujours été guidée par les axes de recherches de notre laboratoire et par les projets structurants de l'Université de Corse. C'est pourquoi nous avons décidé d'appliquer notre méthodologie aux circuits électriques présents dans les systèmes d'éoliennes, et plus particulièrement à la détection de pannes dans ces derniers. Ce travail, issu d'une collaboration avec le laboratoire CREA (Centre de Robotique, d'Electrotechnique et d'Automatique) d'Amiens fait partie du projet structurant « Energies Renouvelables » de l'Université de Corse.
L'encadrement de jeunes chercheurs m'a de plus permis de découvrir de nouvelles thématiques, plus particulièrement issues du projet «Inventaire et valorisation du patrimoine». Ces recherches portent d'une part sur la multi-représentation de données dans un SIG, utile notamment en archéoastronomie, et sur d'autre part l'analyse et la synthèse de voix.

L'orientation future de mes activités de recherches concerne bien entendu la finalisation des travaux initiés dernièrement, mais aussi à plus long terme le développement et l'implication dans de nouvelles actions de recherches.
Mes perspectives de recherche à moyen terme s'orientent donc principalement vers une implication forte dans les deux projets structurants de l'université de Corse:

- Au sein du projet « énergies renouvelables » afin de créer un outil, basé sur notre formalisme, capable de détecter les pannes dans les systèmes d'éoliennes. Cela se traduira notamment par une demande de financement auprès de l'Agence Nationale de la Recherche.
- Au sein du projet «Inventaire et valorisation du patrimoine» dans le domaine d'analyse et synthèse de la voix pour la valorisation et l'apprentissage des chants polyphoniques corses.

Les prochains mois devraient aussi permettre de finaliser notre outil de test complet en proposant le couplage d'un générateur de test avec notre simulateur de fautes.

Concernant les perspectives de recherches à plus long terme, je compte les orienter pour fédérer le plus grand nombre de chercheurs. Tout d'abord, j'envisage d'appliquer notre formalisme à d'autres domaines d'applications : la modélisation et la simulation de centrales nucléaires, la détection d'incidents sur des réseaux électriques qui est un sujet d'actualité,…
Une autre thématique qui me tient à cœur concerne le couplage du formalisme DEVS à un Système d'Information Géographique dans le but de créer un outil de simulation visuelle en temps réel de phénomènes naturels. Le domaine d'application des feux de forêts semble particulièrement adapté à la validation de cet outil, puisque cela permettrait aux sapeurs-

pompiers d'avoir un outil d'aide à la décision qui décrirait en temps réel le développement d'un incendie.

Enfin, l'aspect valorisation de la recherche est aujourd'hui essentiel, et dans ce cadre nous allons mettre en œuvre une demande de création d'entreprise en relation avec le CNRS et l'Université de Corse. La projet de base de cette entreprise concernera le développement d'un environnement de modélisation et de simulation basé sur les travaux de notre équipe de recherche (réseaux de neurones, logique floue, modélisation 3D, couplages SIG, simulation concurrente, etc…).

Partie 2 - Notice Individuelle et Rapports d’Activités

---

1 - CURRICULUM VITAE

1.1 - Etat Civil

Dominique Marie Augustin FEDERICI
Né le 07 Mai 1973 à Bastia (Haute-Corse)
Marié, deux enfants

Adresse professionnelle
Bâtiment PPDB - Quartier Grossetti
Université de Corse
BP 52 - 20250 Corte
Tél. : 04 95 45 02 29
Fax : 04 95 45 01 62
e-mail : federici@univ-corse.fr

Adresse personnelle
Lieu dit u Campu
20215 Venzolasca

1.2 - Titres et Diplômes

1999 : **Doctorat en Informatique**
« Simulation de Fautes Comportementales de Systèmes Digitaux Décrits à Haut Niveau d'Abstraction en VHDL ».
Thèse effectuée sous la direction du Professeur Jean-François Santucci.
Soutenue le 11 Janvier 1999 (mention Très Honorable avec les Félicitations du jury) devant le jury composé de
MM. P. BISGAMBIGLIA, *Maître de Conférences à l'Université de Corse*
S. BOURENNANE, *Professeur à l'Université de Corse*
D. FLOUTIER (Rapporteur), *Professeur à l'Ecole Supérieure d'Ingénieurs d'Annecy*
A. RUCINSKI (Rapporteur), *Professeur à l'Université du New Hampshire, USA*
J.-F SANTUCCI (Directeur de Thèse), *Professeur à l'Université de Corse*
B. STRAUBE, *Privatdozen Dr. -Ing. Habil. à l'Institut Fraunhofer de Dresde, Allemagne*

1995 : **DESS "Ingénierie des systèmes"**
obtenu à l'Université de Corse (mention AB) : Informatique, Modélisation des systèmes/Optimisation, Analyse fonctionnelle et numérique, Commande et Automatique.

1994 : **Maîtrise de Physique et Applications**
obtenue à l'Université de Corse.
Contenu : Mathématiques, Physique, Automatique, Electronique, Informatique.

1993 : **Licence de Physique et Applications**
obtenue à l'Université de Corse (Mention AB).
Contenu : Mathématiques, Physique, Automatique, Electronique, Informatique.

1992 : **D.E.U.G. A "Sciences et Structures de la Matière"**
obtenu à l'Université de Corse.
Contenu : Mathématiques, Physique, Chimie, Thermodynamique, Informatique.

1990 : **Baccalauréat série C**
obtenu en Juin 1990 au Lycée Giocante de Casabianca à Bastia.

1.3 - Cursus Professionnel

2005 : **Directeur du Département Informatique de la faculté des Sciences et Techniques**

2004 - 2005 : **Responsable pédagogique du MASTER Compétence Complémentaire en Informatique et du MASTER Intégration des Systèmes d'Information**

2003 - 2004 : **Responsable pédagogique du DESS Intégration des Systèmes d'Information**

2002 - 2004 : **Responsable pédagogique du DESS Compétence Complémentaire en Informatique**

2000-2002 : **Responsable pédagogique du DEUG Mathématiques, Informatique Application aux Sciences**

1999 : **Maître de Conférences en Informatique**
en poste à l'Université de Corse.

1998-1999 : **Attaché Temporaire d'Enseignement et de Recherche en Informatique**
à l'Université de Corse.

2 - RAPPORT D'ACTIVITE EN MATIERE DE RECHERCHE

2.1 - Bref historique

J'ai débuté mes activités de recherche en Avril 1995 au sein du S.D.E.M. (laboratoire Systèmes Dynamique, Energétique et Mécaniques - URA CNRS 2053) de l'Université de Corse dans le cadre de mon stage de fin d'études. Le professeur Jean-François Santucci m'a permis d'effectuer ce stage dans le cadre du projet CATSAT (conception d'un satellite). L'Université du New Hampshire, partie prenante du projet, m'a accueilli en Mai et Juin 1995.
Au terme de ce stage, les professeurs Santucci et Bisgambiglia m'ont proposé un sujet de Doctorat en Informatique sur le thème du test des circuits digitaux. Durant ces années de Doctorat, j'ai vu évoluer le laboratoire en une Unité Mixte de Recherche du CNRS (N°6134 Systèmes Physiques de l'Environnement). Ce travail s'insérait dans le projet européen

BELSIGN qui regroupait plusieurs universités et instituts de Recherche de la communauté européenne. J'ai ainsi été amené :

- à travailler trois mois au sein de l'institut Fraunhofer de Dresden en 1996.
- à présenter mes travaux dans les différents séminaires du projet.
- à accueillir un chercheur de l'université Complutense de Madrid en 1997.

J'ai présenté mon Doctorat le 11 Janvier 1999, puis j'ai intégré en Septembre 1999 l'Université de Corse en qualité de Maître de Conférence en Informatique. Dans le cadre de mes fonctions de Maître de conférences, j'ai réalisé mes travaux de recherche en tant que membre de l'UMR 6134.

2.2 - Encadrements de Chercheurs :

Depuis ma nomination en tant que maître de conférences, j'ai participé à l'encadrement d'étudiants dans le cadre de leur travail de recherche.

2.2.1 - Co-encadrements de stages de DEA et de MASTER Recherche

2005 : Encadrement du stage de MASTER Recherche de Mr **Jean-Sébastien Gualtieri** : « Développement d'un outil de visualisation en trois dimensions pour la formation et l'aide à la gestion de crises : Application aux feux de forêt **».** Niveau d'encadrement : 100%.

2004 : Encadrement du stage de DEA de Mr **François Giamarchi**: « Implémentation d'un générateur de test à haut niveau ». Niveau d'encadrement : 70%.

2002 : Co-encadrement du stage de DEA de Mr **El-Hadi Khoumery** : « Définition d'une technique visant à modifier des descriptions VHDL pour améliorer leur testabilité ». Niveau d'encadrement : 70%.

2.2.2 - Co-encadrement de doctorat

- Co-Direction du doctorat de Mr **Laurent Capocchi** : « Simulation concurrente de fautes comportementales pour des systèmes à événements discrets : Application aux circuits digitaux ». Niveau d'encadrement : 70%. Thèse soutenue le 25 Novembre 2005 à l'université de Corse. Directeur de Thèse : le professeur Paul Antoine Bisgambiglia.
- Co-Direction du doctorat de Mr **El-Hadi Khoumery :** « Représentation de données spatiales hiérarchisées ; Application à l'archéoastronomie ». Niveau d'encadrement : 30%. Date de soutenance : le 28 Novembre 2006. Directeur de Thèse : le professeur Jean-François Santucci.
- Co-Direction du doctorat de Mr **François Giamarchi.** Le travail de doctorat de Mr Giamarchi porte sur la réalisation d'un générateur de séquences de tests pour des circuits digitaux. Niveau d'encadrement : 50%. Soutenance prévue avant fin 2007. Directeur de Thèse : le professeur Paul Antoine Bisgambiglia.
- Co-Direction du doctorat de Mr **Jean-Sébastien Gualtieri**. Le travail de doctorat de Mr Gualtieri concerne l'analyse et la synthèse de voix pour la valorisation et l'apprentissage des chants polyphoniques corses. Soutenance prévue avant fin 2008. Directeur de Thèse : le professeur Jean-François Santucci.

2.3 - Animation et Administration de la Recherche :

Dans le cadre de ma fonction d'enseignant-chercheur j'ai participé à différents projets de recherche, et j'ai été amené à participer à différentes manifestations regroupant des chercheurs. Je présente dans cette partie ces collaborations et ces implications.

2.3.1 - Niveau International

Projet CATSAT (Cooperative Astrophysics and Technology SATellite)
Responsable Université de Corse.
Notre équipe de recherche entretient une collaboration étroite avec le laboratoire « Electrical and Computer Engineering » du professeur Andrzej Rucinski de l'Université du New Hampshire (UNH) depuis de nombreuses années. Dans le cadre de cette collaboration, j'ai effectué mon stage de fin d'étude (Juin-Juillet 1995) dans ce laboratoire au sein du projet CATSAT.
CATSAT est un projet de conception d'un petit satellite scientifique proposé par l'USRA (Universities Space Research Association), en réponse au programme STEDI (Student Explorer Demonstration Initiative), lancé par USRA et la NASA. CATSAT est une collaboration de trois universités : UNH, WSU (Weber State Universty) en Utah, et Leicester au Royaume Uni.
L'objectif scientifique de ce satellite concerne l'étude l'origine des rayons gamma. Le travail qui m'a été confié portait sur les détecteurs de rayons X de faible énergies qui étaient des photodiodes à avalanche.

Projet HCM (Human Capability and Mobility)
Participation en tant que chercheur.
Contrat n° CHRX-CT 94-0459
Nom du réseau: BELSIGN (BEhavioraL deSIGN methodologies for digital systems)
Durée : 4 ans – (1995-1999)
Financement : Union Européenne.

Le but du projet est de constituer un réseau cohérent de compétences dans le domaine de la conception et du test de systèmes digitaux à partir de spécifications à haut niveau d'abstraction.
Les objectifs principaux sont :

- la synthèse et la validation d'architectures de systèmes à partir de spécifications de hauts niveaux ;
- le développement de nouvelles méthodes pour la conception logicielle/matérielle ;
- l'étude d'une nouvelle approche pour la génération de séquences de test à partir de descriptions comportementales ;
- l'utilisation de HDL (Hardware Description Languages) dans la conception de systèmes. En tant que standard IEEE, VHDL jouera en particulier un rôle important dans les nouveaux outils de synthèse et de validation ;
- la création d'une infrastructure pour la formation de jeunes chercheurs aux techniques de conception de systèmes digitaux.

Ce réseau permet d'une part, de coordonner des activités de recherche dans ces domaines sur des problématique clé et d'autre part, de sensibiliser la communauté scientifique (universitaire et industrielle) au rôle fort que peuvent jouer des méthodologies de conception et de test à haut niveau d'abstraction.

Le réseau est constitué des partenaires suivants :

- Université Complutense de Madrid (Espagne);
- Université de Cantabria (Espagne);
- Université Polytechnique de Madrid (Espagne);
- Institut de Recherche GMD (Allemagne);
- Institut de Recherche FhG-IIS/EAS Dresde (Allemagne);
- Université de Duisburg (Allemagne);
- Université de Twente (Pays-Bas);
- INESC d'Aveiro (Portugal);
- Université de Corse (France);

Projet INTERREG III – Volet A, Thème NTIC
Durée : 4 ans – (2002-2006)
Financement : 183 000 - UE
Participation en tant que chercheur.

Ce projet transdisciplinaire qui regroupe 26 chercheurs de l'UMR 6134, 10 partenaires Français et Italiens devra permettre à l'ensemble des chercheurs impliqués de s'approprier les Nouvelles Technologies de l'Information et de la Communication afin d'assurer la mutualisation, initiée au sein de l'UMR SPE 6134, des connaissances acquises dans les différentes disciplines (Physique, Chimie, Mathématique et Informatique).

Déplacements et Séjours

- En Juin et Juillet 1995, j'ai effectué mon stage de DESS Ingénierie des Systèmes au sein du laboratoire « Electrical and Computer Engineering » de l'Université du New Hampshire (UNH). Le sujet s'intitulait : « Asymétrie de la réponse des photodiodes à avalanche ».
- En Février, Mars et Avril 1996, j'ai effectué un déplacement dans le cadre de mon doctorat à l'institut Fraunhoffer de Dresden en Allemagne. Ce séjour m'a permis de compléter mes connaissances dans le domaine du test de circuit.
- Cours d'été organisés par le réseau Belsign à Almeria (Espagne) en Juillet 97.

Présentations et Interventions

- Présentation d'un article à la conférence HLDVT 2000 à San Francisco le 10 Novembre 2000.
- Présentation d'un article « 5th Advanced Technology Workshop USA » à la base de l'US Air Force d'Hanscom en Juillet 1996.
- Présentation d'un article « 5th Advanced Technology Workshop Europe » à l'INSA de Toulouse en Juin 1996.
- Quatrième séminaire HCM Belsign Workshop à Santander (Espagne) en Octobre 96.
- Sixième séminaire HCM Belsign Workshop à Aveiro (Portugal) en Octobre 97.

Rapporteur

- Reviewer des conférences 39th DAC et 40th DAC.

2.3.2 - Niveau National

Projet Energies Renouvelables EnR :
Participation en tant que chercheur. Début : Juin 2005.
Le projet « EnR » entre dans le cadre des **6 projets pluridisciplinaires** reconnus par l'Institut de l'Environnement (IE) de l'Université de Corse. Il a pour but de regrouper les compétences des chercheurs de différentes disciplines de l'Université de Corse sur la thématique des énergies renouvelables.
Le projet « EnR » s'articule autour de deux axes complémentaires :

- l'étude et la gestion des systèmes à sources renouvelables d'énergie et leur intégration dans le tissu socio-économique ;
- l'étude de matériaux nouveaux pour l'énergie et le photovoltaïque.

GDR SEEDS
Durée : 5 ans – Début : Fin 2006.
Financement : 20 000 Euros /an.
Responsable Université de Corse.
Intitulé : Modélisation des machines électriques en vue de la surveillance et du diagnostic par la méthode des circuits internes équivalents (CIE): prise en compte de phénomènes non-linéaires.
Laboratoires : CREA (Amiens), CEGELY (Lyon), LSEE (Béthune), LEEI (Toulouse), LEG (Grenoble), LAII (Poitiers), SPE (Corte), ……
L'objectif de cette action est la modélisation des machines électriques par une approche simple, afin d'intégrer dans un modèle physique tous les paramètres pouvant caractériser les défauts électromécaniques les plus courants.
Le laboratoire SPE s'intéresse essentiellement aux problèmes de modélisation et simulation de systèmes complexes décrits à haut niveau d'abstraction. Dans le cadre de ce GDR l'équipe informatique du laboratoire SPE s'intéressera à la modélisation et à la simulation des machines électriques sans un premier temps, et à la simulation de fautes dans de tels systèmes dans un second temps.

2.3.3 - Niveau Régional

Projet Microélectronique : Développement d'un outil de test de circuit et amélioration de la testabilité des circuits
Durée : 18 mois – (Janvier 2003-Juin 2004)
Financement : 15 000 Euros - Collectivité Territoriale de Corse.
Coresponsable du Projet

Ce projet propose d'intervenir au début du processus de conception de systèmes complexes (systèmes embarqués, télécommunications, systèmes réactifs...) afin de proposer le développement de méthodologies permettant de tester et d'améliorer la testabilité de tels systèmes.
L'objectif principal de ce projet est la mise au point d'un outil de test complet permettant d'intervenir au plus tôt lors de la conception de circuits digitaux.

3 - RAPPORT D'ACTIVITE EN MATIERE D'ENSEIGNEMENT

3.1 - Bref historique

Depuis 1996, j'exerce de fonctions d'enseignant à l'Université de Corse :

- Septembre 1996 à Septembre 1998 : En qualité de vacataire.
- Septembre 1998 à Septembre 1999 : En qualité d'Attaché Temporaire d'Enseignement et de Recherche à la Faculté de Sciences et Techniques.
- Depuis Septembre 1999 : En qualité de Maître de Conférences à la Faculté de Sciences et Techniques.

3.2 - Tableau récapitulatif des activités d'enseignement

Le tableau ci-après représente l'ensemble des enseignements dispensés par année et par filière.

Année Universitaire	1996-1997	1997-1998	1998-1999	1999-2000	2000-2001	2001-2002	2002-2003	2003-2004	2004-2005	2005-2006
Qualité	**Vacataire**	**Vacataire**	**A.T.E.R.**	**Mcf**	**Mcf**	**Mcf**	**Mcf**	**Mcf**	**Mcf**	**Mcf**
DEUG SV2	Initiation à l'informatique (0/0/42)	Initiation à l'informatique (0/0/21)	Initiation à l'informatique (12/6/18)	Initiation à l'informatique (6/6/12)						
DEUG SM2		Programmation en Pascal (0/0/15)	Programmation en Pascal (0/0/15)							
IUT Réseaux et Communications		Internet&HTML (0/8/0)								
Licence Physique et Applications		Algorithmique (0/9/0)		Algorithmique (0/0/18)						
Maitrise de Chimie				Outils Bureautiques (6/6/24)						
Maitrise Physique et Applications	Internet et HTML (0/0/3)	Algorithmique (0/9/0)	Programmation en langage C (0/3/18)							
DESS Compétences Complémentaire en Informatique		Système d'exploitation UNIX (0/0/12) Algorithmique (0/2/0)	CPOO-C++ (0/0/20) Génie Logiciel (2/0/12) Système d'exploitation UNIX (0/0/16)	Système d'exploitation UNIX (0/0/18) Langage C (0/0/18) CPOO-C++ (0/0/18) Génie Logiciel (2/0/12) Programmation Visual Basic (12/0/15)	Système d'exploitation UNIX (6/6/10) Génie Logiciel (9/0/12) CPOO-C++ (0/0/18) Programmation Visual Basic (12/9/20)	Système d'exploitation UNIX (6/8/0) Génie Logiciel (8/0/6) Algorithmique et Base de la Programmation (12/8/12)	Système d'exploitation UNIX (12/12/18) Génie Logiciel (9/0/6) Programmation Visual Basic (0/18/15) Algorithmique et Base de la Programmation (12/7,5/12)	Système d'exploitation UNIX (9/12/10,5) Génie Logiciel (6/0/6) Programmation Visual Basic (0/15/15) Algorithmique et Base de la Programmation (9/7,5/9)		
DEUG MIAS 1				Introduction aux systèmes d'exploitation (6/6/12) Systèmes d'exploitation Avancés	Introduction aux systèmes d'exploitation (12/12/0) Algorithmique et Structures de données	Introduction aux systèmes d'exploitation (12/12/0) Projet Professionnel	Introduction aux systèmes d'exploitation (12/12/0) Programmation Pascal			

				(9/9/18)	(0/18/0)	(0/0/24)	(0/0/18)			
Année Universitaire	1996-1997	1997-1998	1998-1999	1999-2000	2000-2001	2001-2002	2002-2003	2003-2004	2004-2005	2005-2006
Qualité	Vacataire	Vacataire	A.T.E.R.	Mcf	Mcf	Mcf	Mcf	Mcf	Mcf	Mcf
IUP NTIC ISI 2					Système d'exploitation UNIX (14/14/0) Programmation et Langage C (0/0/24)	Système d'exploitation UNIX (14/12/0) Langage C (0/0/8)		Système d'exploitation UNIX (9/12/18) Langage C (0/0/16)	Système d'exploitation UNIX (9/9/18)	Système d'exploitation UNIX (9/9/15)
IUP NTIC ISI 3						Interface Utilisateur (VB) (0/20/40) Gestion de Projets (4/0/6)	Interface Utilisateur (VB) (7/12/18)	Interface Utilisateur (VB) (4,5/18/24)	Interface Utilisateur (VB) (10,5/0/18)	Interface Utilisateur (VB.NET) (6/15/18)
MASTER 2 CCI									Info Logiciel 1 (Bases de l'algo) (15/9/18) Info Logiciel 2 Gestion de Projets (6/0/6) CPOO (6/6/16,5)	Info Logiciel 1 (Bases de l'algo) (9/9/18) Info Logiciel 2 Gestion de Projets (6/0/6) CPOO (VB.NET) (6/6/18)
DESS ISI								Projets (0/15/0)		
MASTER 2 ISI										Programmation Web1 (.NET) (6/6/6)
Licence Informatique 1ère Année									Unix (15/0/9)	Unix (15/4,5/6)
Total (C/TD/TP)	(0/0/45)	(0/28/48)	(14/9/99)	(41/27/165)	(53/59/84)	(56/60/96)	(52/61,5/87)	(37,5/79,5/98,5)	(61,5/24/85,5)	(57/49,5/87)
Total Eq. TD	**30**	**60**	**96**	**198,5**	**194,5**	**208**	**197,5**	**201,4**	**173,25**	**193**

3.3 - Description des cours de l'année 2005/2006 :

Système d'exploitation UNIX – IUP ISI 2

Objectif :

Ce cours est destiné à fournir une description claire des concepts qui sont à la base des systèmes d'exploitation ainsi qu'un exemple de système d'exploitation. Ce cours englobe également un enseignement sur le système d'exploitation UNIX : ses fonctions fondamentales et la programmation en langage Shell.

Plan du cours :

Introduction au système UNIX. Utilisation de l'environnement. Les commandes. Le Shell. Unix et les processus. La commande awk. Programmation Shell. Gestion des comptes.

Interface Utilisateur (VB.NET) – IUP ISI 3

Objectif :

L'objectif de cours est d'ouvrir la voie aux métiers spécifiques du domaine de la communication entre l'homme et la machine. Ce cours permettra aux étudiants de se familiariser avec un langage de développement orienté objet piloté par les événements : Visual Basic.NET.

lan du cours :

Introduction au framework .NET. Visual Studio .NET. Conception d'application avec VB.NET. Structures de décision. Structures de répétition. Procédures et Fonctions. Fichiers à accès séquentiels. Fichiers à accès direct. Accès aux données.

Info Logiciel 1 (Bases de l'algo) – MASTER 2 CCI

Objectif :

Fournir aux étudiants des bases nécessaires à l'algorithmique et à la programmation.

Plan du cours :

Historique de l'informatique. Structure d'un programme. Opérateurs et fonctions prédéfinies. Structures de contrôle (Instructions conditionnelles - Instructions répétitives). Types de données évoluées (Chaînes de caractères - Tableaux). Fonctions & Procédures. Variables globales et locales.

Info Logiciel 2(Gestion de Projets)

Objectif :

Offrir une culture de projet permettant de maîtriser le cycle de vie d'un développement logiciel mais aussi les principes du travail en équipe. Utilisation de MS Project.

Plan du cours :

Introduction à la gestion de projets. Planification de projets : méthode Gantt , Pert, Chemin critique. Les acteurs du projet. Gestion des ressources. Définition d'une ressource, affection de coût. Suivi de projets

CPOO (VB.NET)

Objectif :

Fournir aux étudiants des bases nécessaires au développement d'applications Windows avec .NET.

Plan du cours :

Introduction à Visual Basic Historique de Visual Basic et quelques définitions. Introduction à l'environnement de développement intégré Visual Studio 2005

et au processus de création des applications Feuilles, contrôles et menus. Présentation de la bibliothèque des classes. Éléments fondamentaux de programmation.

Programmation Web1 (ASP.NET)

Objectif :

Cet enseignement apporte aux étudiants les connaissances nécessaires pour construire et déployer des applications web à l'aide de la plateforme .NET. Les participants acquerront une expérience pratique à travers des séances de travaux dirigés et de travaux pratiques qui permettront la réalisation d'un projet.

Plan du cours :

Historique. Définition de .NET. Présentation de Microsoft .NET Framework. ASP.NET. Visual Studio .NET 2005. Création d'un formulaire Web ASP.NET. Le Code en ASP.NET. Contrôles serveurs avancés. Les Contrôles Utilisateurs

3.4 - Encadrement de stagiaires

Dans le cadre de leur cursus, les étudiants d'IUP ISI troisième année, de MASTER 2 CCI, et de MASTER 2 ISI ont à effectuer un stage en entreprise. Chaque étudiant doit avoir un tuteur coté université, j'ai donc été amené à suivre le déroulement de plusieurs stages. Le rôle du tuteur consiste à apporter éventuellement une aide technique à l'étudiant et surtout à veiller au bon déroulement du stage en lisant un pré-rapport envoyé par l'étudiant un mois après le début de son travail.

De plus, les MASTER 2 ISI et les IUP ISI 3 ont une unité d'enseignement intitulée projet professionnel : les étudiants, en groupe de deux ou trois, doivent choisir un sujet (proposé par une entreprise ou par un enseignant-chercheur) et réaliser ce travail de Novembre à Février. J'ai proposé et encadré trois sujets pour cette UE :

- Implémentation d'un parseur VHDL/Python
- Création d'un site web dynamique permettant la saisie de références bibliographiques
- Etude de la technologie RFID

3.5 - Tutorat de moniteur

Pour l'initiation à la pratique pédagogique, le moniteur bénéficie de l'encadrement d'un tuteur pédagogique. Il joue un rôle de conseiller pédagogique lors des premiers pas du moniteur comme enseignant, il l'aide à préparer et à évaluer ses enseignements et doit être sollicité par le moniteur pour tout problème pédagogique. Durant ces dernières années, j'ai assuré cette tâche auprès de Mr Capocchi (2002-2005), et je suis actuellement de Mr Paul-Antoine Bisgambiglia, moniteur depuis 2005.

4 - RAPPORT D'ACTIVITE EN MATIERE D'ADMINISTRATION

4.1 - Bref historique

Il est demandé aux maitres de conférences des universités de prendre en charge des responsabilités administratives en plus de leurs missions d'enseignement et de recherche. Dans ce cadre, j'ai été amené à m'investir dans des tâches de responsabilités pédagogiques et d'encadrement d'enseignants chercheurs. Le département informatique de l'UFR Sciences et Techniques de l'université de Corse a connu une forte croissance avec à l'heure actuelle 8 année-formations habilitées. Durant cette croissance, j'ai participé en particulier à la mise en place de la réforme Licence Master Doctorat au sein de ce département.

4.2 - Le DEUG MIAS

Mes premières fonctions de responsable pédagogique concernent le DEUG MIAS. Cette formation a ouvert en Septembre 1999, je l'ai eu en charge de Septembre 2000 à Septembre 2002.

4.3 - Le DESS CCI (MASTER 2 CCI)

De Septembre 2002 à Septembre 2005, j'ai géré la filière professionnalisante DESS Compétence Complémentaire en Informatique. J'ai en particulier assuré la transition de ce diplôme dans la nouvelle carte de formation basée sur la réforme LMD en rédigeant le dossier d'habilitation du MASTER 2 CCI. L'objectif de cette formation est de fournir en un an une solide compétence en informatique à des spécialistes issus d'autres domaines. Durant ces années j'ai notamment participé à la Validation d'Acquis par l'Expérience de ce diplôme par 2 candidats.

4.4 - Le DESS ISI (MASTER 2 ISI)

De Septembre 2003 à Septembre 2005, la gestion pédagogique de cette cinquième année universitaire m'a permis de la faire évoluer pour permettre aux étudiants diplômés d'être en adéquation avec le marché du travail, en particulier en l'axant développement web et systèmes distribués.

4.5 - L'IUP NTIC ISI

Depuis Juin 2006, je suis le directeur de l'IUP NTIC Intégration des Systèmes d'Information. Cette formation sur deux ans (IUP2 et IUP3) a pour objectif de former des cadres dans les domaines de l'informatique et des réseaux et plus largement dans le domaine des technologies de l'information et des communications.

4.6 – Le Département Informatique

Depuis Juin 2005, je suis directeur du département informatique. Ce département regroupe 9 enseignant-chercheurs et assure la gestion de deux masters, d'une licence et d'un IUP. Le fait d'avoir été responsable pédagogique des différentes filières du département est important pour mener à bien les différentes tâches qui incombent à un directeur de département :

- Administration le budget du département (environ 10000 €).
- Gestion de la répartition des services.
- Organisations de réunions pédagogiques.
- Recherche de vacataires professionnels.
- Communication avec les autres instances universitaires : la direction de l'UDR Sciences et Techniques, la présidence, le Centre de Ressources Informatiques.
- Organisation de manifestations pédagogiques pour les étudiants et les enseignants. Dans ce cadre j'ai organisé deux conférences en 2005 et en 2006, une relative à la plateforme .NET de Microsoft avec la venue d'un développeur de Microsoft France, et une portant sur les logiciels libres présentée par un chef de projet en informatique.

4.7 - Délocalisation de formations

En Mai 2005, j'ai mis en place la délocalisation de l'IUP ISI et du MASTER 2ISI à l'EHECT de Tanger au Maroc. Durant l'année 2005-2006, j'ai géré les missions des enseignants de Corté allant dispenser des cours à Tanger. Le nombre d'étudiants inscrits à Tanger est de 54 (42 pour l'iup 2ème année, et 12 pour le master 2 isi).

4.8 - Autres Responsabilités

- Membre élu du Conseil du laboratoire (UMR 6134 SPE) de 2003 à 2006.
- Membre élu de la commission de spécialistes de mathématiques/informatique de l'université de Corse depuis 2000.
- Membre de jury de concours d'Ingénieur d'Etude en 2004.
- Président de jury du baccalauréat pour les années 2000 et 2001.

5 - LISTE DES PUBLICATIONS

Ouvrage

1. D. Federici, "Simulation de Fautes Comportementales de Systèmes Digitaux Décrits à Haut Niveau d'Abstraction en VHDL", Thèse soutenue le 11 Janvier 1999 à l'Université de Corse.

Revues internationales à comité de lecture

2. L. Capocchi, F. Bernardi, D. Federici, P. Bisgambiglia, "BFS-DEVS: A General DEVS-Based Formalism For Behavioral Fault Simulation", Elsevier Simulation Pratice and Theory, Vol. 14, issue 7, pp. 945-970, 2006.

3. J.F. Santucci, P.A. Bisgambiglia, D. Federici, "Behavioral Fault Simulation", Embedded Design and Test, Chapitre 11, pp. 261-284, Ed. Kluwer Academic Publisher, 1998.

4. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Petri Net Modeling and Behavioral Fault Modeling Scheme for VHDL Descriptions". Advanced Technology Workshop 96, Hanscom UD Air-Force Base Boston, USA, publié dans Embedded System Applications, Ed. Kluwer Academic Publisher, 1997.

5. A. Rucinsky, N. Valverde, C. Baron, P.A. Bisgambiglia, D. Federici - J.F. Santucci, "Fault Modelling in Space-Borne Reconfigurable Microelectronic Systems", Embedded System Applications, Ed. Kluwer Academic Publisher, 1997.

Conférences internationales avec actes et comité de lecture

6. L. Capocchi, D. Federici, P.A. Bisgambiglia, H. Henao, G.A. Capolino, “A BFS-DEVS Approach for induction Generator Non Traditional Modelling”, Proceedings of the First International Symposium on Environment Identities and Mediterranean Area (IEEE ISEIM06), Corte-Ajaccio, France, July 2006.

7. L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, “A DEVS-based Concurrent and Comparative Fault Simulation Algorithm”, Proceedings of the SCS Summer Computer Simulation Conference (SCSC 2005), San Diego, CA, USA, 2005.

8. E.H. Khoumeri, J.F. Santucci, D. Federici, “Intégration de la notion de Hiérarchie d’abstraction de données spatiales dans un SIG; Application en Archéoastronomie“, Cassini 2005, Avignon. 20-23 juin 2005.

9. E.H. Khoumeri, J.F Santucci, D. Federici, “Hierarchical Multi-View Representation of Spatial Data; Application to the Analysis of Corsican Neolithic Tombs”, CAA 2005 (Computer Application for Archaeology). 21-24 Mars 2005. Tomar, Portugal.

10. L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, “A DEVS-based Modeling Behavioral Fault Simulator for RT-Level Digital Circuits”, Proceedings of the SCS Summer Computer Simulation Conference (SCSC 2004), p. 481-486, San José, CA, USA, 2004.

11. L. Capocchi, D. Federici, F. Bernardi, P.A. Bisgambiglia, “Behavioral Fault Simulation for VHDL Descriptions using the DEVS Formalism”, Fast Abstract at the IEEE Pacific Rim Dependable Computing International Conference (PRDC), Papeete, Tahiti, French Polynesia, 2004.

12. L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, “Transformation of VHDL Descriptions into DEVS Models for Fault Modeling and Simulation”, Proceedings of the IEEE Systems, Man and Cybernetics Conference (SMC03), p. 1205-1211, Washington D.C., USA, 2003.

13. D. Federici, J.F. Santucci, P.A. Bisgambiglia, “Behavioral Fault Simulation : Implementation and Experiments Results", First International Workshop on

Electronic Design, Test & Applications, Delta'2002, IEEE and TTTC conference, Christchurch, New Zealand, p.81-86, 2002.

14. P.A. Bisgambiglia, D. Federici, J.F. Santucci, "Fault Modeling and Simulation at Behavioral Level", Proceeding of the 2nd IEEE Latin-American Test Workshop, LATW'2001, IEEE and TTTC conference, Cancun, Mexico, p.45-52, 2001.

15. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "High Level Fault Simulation : Experiments and Results on ITC'99 Benchmarks", Fifth Annual IEEE International Workshop on High Level Design Validation and Test, HLDVT'00, Claremont Resort and Spa, Berkeley, California, p145-150, 2000.

16. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "VHDL Behavioural Fault Simulator : Experiments to the ITC'99 Benchmarks", 4th International ITC Workshop on System Test and Diagnosis, IEEE conference, New Atlantic City Convention Center, Atlantic City, New Jersey , USA, p3-8, 2000.

17. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Petri Net Modelling and Behavioral Fault-Modelling Scheme for VDHL Description", Proceedings of the Fifth Advanced Technology Workshop USA, Hanscom US Air-Force Base Boston, Massachusetts, p. 120-130, 6-9Août, 1996.

Conférences internationales avec actes sans comité de lecture

18. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "A Behavioral Fault Simulator : Algorithm, C++ Implementation, Results", Actes 6th European HCM Belsign Workshop, Aveiro, Portugal, p. 129-210, Octobre 1997.

19. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Behavioral Fault Simulation Software Implementation in C++", Actes 5th European HCM Belsign Workshop, Dresden, Allemagne, p. 115-123, Mai 1997.

20. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "A deductive approach to Behavioral Fault Simulation", Actes 4th European Belsign Workshop. Cantabria, Espagne, p. 115-123, Octobre 1996.

21. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Behavioral Fault Modelling and Simulation", Actes 3rd European HCM Belsign Workshop. Porticcio, France, p. 60-68, (11-12 Avril 1996).

Posters

22. E.H. Khoumeri, J.F Santucci, D. Federici, "Intégration de la notion de Hiérarchie d'abstraction de données spatiales dans un SIG : Application en Archéoastronomie", Cassini 2005, Avignon, 20-23 juin 2005.

23. E.H. Khoumeri, J.F Santucci, D. Federici, "Multi level Representation of spatial data : application in Archaeology and Anthropology ", ICC 2005, La Corogne, 9-16 Juillet 2005.

6 - RÉFÉRENCES

[Agr81] Agrawal, V.D, "Sampling techniques for determining fault coverage in LSI circuits", Jour. Digital Systems, volume 5, pages 189.202, (décembre 1981).

[Bar86] D.S. Barclay, J.R. Armstrong, "A heuristic chip-level test generation algorithm", 23th IEEE/ACM Design Automation Conference, pp.257-261, 1986.

[Bea00] Beazley D.M, "Advanced Python Programming", Department of Computer Science, University of Chicago, 2000.

[Bed02] Bédard Y., Bernier E., et Devillers R., "La métastructure VUEL et la gestion des représentations multiples", Généralisation et représentation multiple, p 149-161, édition hermès 2002.

[Bre76] Breuer M. A., Friedman A.D, "Diagnosis and Reliable Design of Digital Systems", Computer Science Press, 1976.

[Cap03] L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, “Transformation of VHDL Descriptions into DEVS Models for Fault Modeling and Simulation”, Proceedings of the IEEE Systems, Man and Cybernetics Conference (SMC03), p. 1205-1211, Washington D.C., USA, 2003.

[Cap04] L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, “A DEVS-based Modeling Behavioral Fault Simulator for RT-Level Digital Circuits”, Proceedings of the SCS Summer Computer Simulation Conference (SCSC 2004), p. 481-486, San José, CA, USA, 2004.

[Cap05a] L. Capocchi, “Simulation concurrente de fautes comportementales pour des systèmes à événements discrets : Application aux circuits digitaux”, Thèse de Doctorat, Université de Corse, 2005.

[Cap05b] L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, “A DEVS-based Concurrent and Comparative Fault Simulation Algorithm”, Proceedings of the SCS Summer Computer Simulation Conference (SCSC 2005), San Diego, CA, USA, 2005.

[Cap06a] L. Capocchi, F. Bernardi, D. Federici, P.A. Bisgambiglia, "BFS-DEVS: A General DEVS-Based Formalism For Behavioral Fault Simulation", Elsevier Simulation Pratice and Theory, Vol. 14, issue 7, pp. 945-970, 2006.

[Cap06b] L. Capocchi, D. Federici, P.A. Bisgambiglia, H. Henao, G.A. Capolino, “A BFS-DEVS Approach for induction Generator Non Traditional Modelling”, Proceedings of the First International Symposium on Environment, Identities and Mediterranean Area (IEEE ISEIM06), Corte-Ajaccio, France, July 2006.

[Cor00] Corno F., Manzone A., Pincetti A., Reorda M.S., et Squillero G, "Automatic test bench generation for validation of rt-level descriptions : an industrial experience", Proceedings of Design, Automation and Test in Europe (DATE'00), pp. 385.389. 2000.

[Cor97] Corno F., Prinetto P., et Reorda M.S., "Testability analysis and ATPG on behavioral RT-level VHDL", Proceedigns of IEEE International Test Conference (ITC'99), pp. 753.759, Washington D. C., USA, novembre 1997.

[Cou91] A-L. Courbis, "Contribution à l'étude et au développement d'un générateur de séquences de test comportemental", Thèse de Doctorat, Université Montpellier II, Décembre 1991.

[Dav92] R. David, H. Alla, "Du grafcet aux réseaux de Pétri", Edition Hermès, 1992.

[Fed96] D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Petri Net Modelling and Behavioral Fault-Modelling Scheme for VDHL Description", Proceedings of the Fifth Advanced Technology Workshop USA, Hanscom US Air-Force Base Boston, Massachusetts, p. 120-130, 6-9Août, 1996.

[Fed97] D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Petri Net Modeling and Behavioral Fault Modeling Scheme for VHDL Descriptions". Advanced Technology Workshop 96, Hanscom UD Air-Force Base Boston, USA, publié dans Embedded System Applications, Ed. Kluwer Academic Publisher, 1997.

[Fed99] D. Federici, "Simulation de Fautes Comportementales de Systèmes Digitaux Décrits à Haut Niveau d'Abstraction en VHDL", Thèse soutenue le 11 Janvier 1999 à l'Université de Corse.

[Fed00] D. Federici, J.F. Santucci, P.A. Bisgambiglia, "High Level Fault Simulation : Experiments and Results on ITC'99 Benchmarks", Fifth Annual IEEE International Workshop on High Level Design Validation and Test, HLDVT'00, Claremont Resort and Spa, Berkeley, California, p145-150, 2000.

[Fed02] D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Behavioral Fault Simulation : Implementation and Experiments Results", First International Workshop on Electronic Design, Test & Applications, Delta'2002, IEEE and TTTC conference, Christchurch, New Zealand, p.81-86, 2002.

[Gho91] S. Ghosh, T.J. Chakraborty, "On behavior fault modeling for digital designs", Journal of Electronic Testing : Theory and Applications, Vol. 2, pp. 135-151, 1991.

[Gua06] J.S. Gualtieri, C. Bellagamba, D. Verdoni, J.F. Santucci, T. Casalonga, N. Acquaviva, "Computer aided Characterization, and Synthesis of Corsican Songs", Proceedings of the First International Symposium on Environment, Identities and Mediterranean Area (IEEE ISEIM06), Corte-Ajaccio, France, July 2006.

[Kho84] R. Khorram, "Functional test pattern generation for integrated circuits", IEEE International Test Conference, pp. 246-249, Octobre 1984.

[Kho05a] E.H. Khoumeri, J.F. Santucci, D. Federici, "Intégration de la notion de Hiérarchie d'abstraction de données spatiales dans un SIG; Application en Archéoastronomie", Cassini 2005, Avignon. 20-23 juin 2005.

[Kho05b] E.H. Khoumeri, J.F Santucci, D. Federici, "Hierarchical Multi-View Representation of Spatial Data; Application to the Analysis of Corsican Neolithic Tombs", CAA 2005 (Computer Application for Archaeology). 21-24 Mars 2005. Tomar, Portugal.

[Kof00] Kofman, E., Giambiasi, N., et Junco, S., "FDEVS : A general DEVS based formalism for fault modeling and simulation", Proceedings of European SimulationSymposium, pages 77.82, Hamburg, Germany, 2000.

[Kof03a] Kofman E., Lapadula M., Pagliero E., "PowerDEVS: A DEVS-based environment for hybrid system modeling and simulation", tech. rep. LSD0306, Rosario National University., 2003.

[Kof03b] Kofman E, "Quantization-based simulation of differential algebraic equation systems", Trans. Soc. Comput. Simul. Int., vol. 79, no. 7, pp. 363–376, 2003.

[Kof05] Kofman E., "A third order discrete event simulation method for continuous system simulation. part i: Theory", tech. rep., 2005.

[Lev82] U.H. Levendel, P.R Menon, "Test generation algorithms for computer hardware description languages", IEEE Transactions on Computers, Vol. C-31 (7), pp. 577-588, 1982.

[Lin84] T.Lin, S.Y.H. Su, "Functional test generation of digital LSI/VLSI systems using machine symbolic execution technique", International Test Conference, pp.660-668, 1984.

[Max] http://www.cycling74.com/products/maxmsp

[McC76] McCabe T., "A software complexity measure", IEEE Transactions on Software Engineering, volume 2, pp. 308.320, 1976.

[Min82] Y. Min, S.Y.H. Su, "Testing functional faults in VLSI", 19th Design Automation Conference, pp.384-392 1982.

[Nor89] F.E. Norrod, "An automatic test generation algorithm for hardware description language", 26th Design Automation Conference, pp.76-85, July 1989.

[One90] M.D. Oneill, D.D. Jani, C.H. Cho, J.R. Armstrong, "BTG : a behavioral test generator", 9th Computer Hardware Description Languages and their Application, IFIP, pp.347-361, 1990.

[Pao04] Paoli C., Nivet M., Bernardi F., et Capocchi L., "Simulation-based validation of VHDL descriptions using constraints logic programming", Proceedings of 5th IEEE Workshop on RTL and High Level Testing (WRTLT'04). Osaka, Japan, 2004.

[Pet81] J.L. Peterson, "Petri net theory and the modeling of systems", Prenctice- Hall, Inc., Englewood Cliffs, New-York, 1981.

[Pla93] V. Pla, "Générateur de séquences de test comportemental", Thèse de Doctorat, Université Montpellier II, Décembre 1993.

[Rua99] Ruas A., "Modèle De généralisation de données géographiques à base de contraintes et d'autonomie", thèse à l'université de Marne la Vallée, 1999

[San98] J.F. Santucci, P.A. Bisgambiglia, D. Federici, "Behavioral Fault Simulation", Embedded Design and Test, Chapitre 11, pp. 261-284, Ed. Kluwer Academic Publisher, 1998.

[Spa00] Spaccapietra S, et all, “From multi-scale to Multi-Representation”, Chooueiry & T.Walsh Edition, Proceedings 4th International Symposium, Texas, USA, 2000

[Ulr94] Ulrich, E., Agrawal, V., et Arabian, J., "Concurrent and Comparative Discrete Event Simulation", Kluwer Academic publisher, 1994.

[Van 01] Vangenot C., "Multi-représentation dans les bases de données géographiques", école polytechnique fédérale de Lausanne, thèse n° 2430, 2001.

[Vhd87] "VHDL, language reference manual", IEEE Standard 1076, 1987.

[Yaz05] A. Yazidi, H. Henao, G. Capolino, D. Casadei, and F. Filippetti, "Doublefed three-phase induction machine abc model for simulation and control purposes", Proceedings of IEEE Industrial Electronics Conference (IECON’05), vol. 4, pp. 2560–2565, November 2005.

[Zei76] Zeigler B.P, "Theory of Modeling and Simulation", Academic Press, 1976.

[Zei00] Zeigler B.P., Praehofer H., et Kim, T.G, "Theory of Modeling and Simulation", Second Edition, Academic Press, 2000.

Partie 3 - Publications représentatives

---

1 - PREMIÈRE PUBLICATION

3. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "Behavioral Fault Simulation", Embedded Design and Test, Chapitre 11, pp. 261-284, Ed. Kluwer Academic Publisher, 1998.

11 BEHAVIORAL FAULT SIMULATION

Jean-François Santucci
Paul Bisgambiglia
Dominique Federici

11.1 Introduction

Due to the ever-increasing complexity of VLSI circuits, the use of VHDL [Vhd87] behavioral descriptions in the fields of test generation and fault simulation becomes advised. In order to understand this evolution and the context in which it is efficient, the increasing complexity of VLSI circuits must be considered. To better cope with the complexity of existing and future systems, higher abstraction levels must be taken into account. Furthermore, the only knowledge about the device being tested may come from data sheets or signal measurements. In this case the only way to generate test patterns is behavioral testing. Today, such a task is done manually and one of the main interests in Behavioral Test Pattern Generation (BTPG) is to automate it. Another point to consider is that the test generation process is an integral part of the design process and its implementation must begin during the behavioral design phase.

Motivated by these facts, various BTPG methods have been proposed in the recent past [Bar87,One90,Nor89,Hum91,Cou92,Stra93]. Most of these BTPG methods are based on a deterministic generation process using a formal fault model : each sequence is constructed in order to detect a given fault belonging to a Fault List determined from a Behavioral Fault Model.

Despite the use of heuristic criteria [Che91,San92,San93], to speed up the search for test sequences, these methods remain very time-consuming because of the deterministic nature of the BTPG processes and the lack of Behavioral Fault Simulation (BFS) methods.

In order to facilitate the definition of a powerful BFS method, it is essential to explicitly represent the concepts involved in behavioral descriptions. We have therefore defined an internal model which highlights on the one hand the sequential and concurrent aspects and on the other hand the separation and interaction between the control flow and the data flow.

The list of considered behavioral fault hypothesis is derived using two steps : (i) definition of an exhaustive fault model which collects all possible fault hypothesis in terms of the incorrect «functioning» of modeling items ; (ii) selection of a sub-set of conventional faults [Gho91].

Our approach for defining a BFS method leans on the resolution of the three following sub-problems in order to deal with high-level behavioral descriptions :

- 1- determination of the high level basic elements of the internal model which will have a key role in BFS.
- 2- definition of how lists of behavioral faults are propagated through the previous basic elements. The simulation of a basic element requires deducing the fault list at the basic element outputs from the input faults lists.
- 3- development of an overall algorithm allowing to propagate fault lists through the complex data and control structures involved in the internal model.

The BFS method has been implemented in C++ language and first results have been obtained.

In the next section of the chapter we deal with the main concepts involved in behavioral test pattern generation. Section 3 presents the motivation in behavioral testing. The interest of using of a behavioral fault simulator coupling with a deterministic and a pseudo-random behavioral test pattern generator is detailed in section 4. Our approach for performing BFS is given in section 5. Section 6 deals with implementation and results. Perspectives are overviewed in a concluding part.

11.2. What is Behavioral Testing

11.2.1 Concepts of Behavioral Testing

Testing is a major activity in CAD (Computer Aided Design) systems for digital circuits. For that reason, CAD systems include ATPG (Automatic Test Pattern Generation) tools. ATPG techniques can be divided into two families depending on the model type of the circuit under test : structural or behavioral.

The structural view or « white box » represents a potential structure of the studied system as an interconnection of basic elements. This structure is qualified as potential because there is not necessarily a complete mapping between the model and the actual circuit. The interconnected items model physical components which can be in turn modeled according either a structural or behavioral view. It should be pointed out that the item belonging to the lowest level of a structural description must be described according to a behavioral point of view.

The behavioral view or « black box » represents the studied system by defining the behavior of its outputs according to values applied on its inputs without knowledge of its internal organization. A behavioral model can be defined by means of two kinds of representation : alphanumeric representations or graph-based representations. Alphanumeric models are textual representations which describe the system behavior by means of declarative or procedural programming languages. Graph representations (binary decision diagrams [Ake78a] used in [Ake78b,Aba85,Cha86], transformation graphs [Lai83], Petri Nets [Pet81] used in [Olc91,Scho85]) are characterized by an interconnection of basic elements for which a behavior is predefined as context-free. It should be noted that this structure is not a potential structure of the modeled system as is the case in the structural view.

Whatever the view by which the circuit is modeled, an abstraction level is taken into account.

We have to point out that abstraction levels are not implied by a given view : the boolean level does not have to imply a structural view nor does system level have to imply a behavioral view.

The structural test refers to a test pattern generation methodology based on a structural model of the circuit under test.

The behavioral test refers to a test pattern generation methodology based on a behavioral model of the circuit under test.

11.2.2 Fault modeling

The aim of the generation process is to define patterns to apply on primary inputs in order to test eventual physical defects. These defects can be detected only if they induce an irregular behavior called a fault. The fault effect (or error) is measured by a difference between the state of the fault-free model (reference model) and the state of the faulty model (model in which a fault hypotheses is injected). Let us point out that the definition of test patterns usually leans on the definition of fault hypothesis depending on both the abstraction level and the view from which the circuit is modeled. A fault hypothesis is :

- either an hypothesis of an incorrect behavior of an item belonging to the model, but considered context-free
- or an hypothesis of modification of the initial global description by adding, suppressing or combining basic items, without modifying the predefined behavior of these items.

It should be pointed out that the modification of the global description of a model usually lead to take into account too large a number of possibilities. Thus this kind of fault hypothesis is not usually considered. In a similar way, overly complex faults concerning the modification of a basic item behavior will also be excluded.

11.2.3 Behavioral Test Pattern Generation

Behavioral Test Pattern Generation concerns the development of softwares dedicated to the determination of set of patterns allowing behavioral faults to be detected. Behavioral Fault Models are defined using the approach described in sub-section 2.2. Recently a set of work[Bar87,One90,Nor89,Hum91,Cou92,Stra93] has been developed in this area.

11.3 Motivation in Behavioral Testing

11.3.1 Main advantages of Behavioral Testing

Recent advances in VLSI technology have had a great impact on testing. For the last ten years, research work has been oriented towards Behavioral Test Pattern Generation. in contrast with conventional structural approaches [Kir87,Schu87,Fuj87]. In order to understand this evolution and the context in which it is efficient, the increasing complexity of the VLSI circuits must be considered. It is now well known that high abstraction levels must be taken into account when dealing with Design and Test of Digital Systems. One very important point is that the test generation process is an integral part of the design process and its implementation must begin during the behavioral design phase. The potential advantages include early estimates of the coverage rate of physical failures prior to the synthesis as well as a faster fault simulation.

11.3.2 Confidence in Behavioral Testing

The test quality depends on both the abstraction level and the view taken into account. Obviously, for a given abstraction level, generating from a structural view is better than generating from a behavioral model in terms of physical failures coverage.

However a set of work has proven that some measure of confidence could be highlighted for Behavioral Testing.

A first solution to address the validity of behavioral testing consists in performing a fault simulation of behavioral test sequences on a structural model described at a lower level of abstraction (i.e. the gate level single stuck-at fault model). This approach has been carried out in [Gho91] and encouraging results for Behavioral Testing have been pointed out.

A second solution is to establish a correlation between behavioral fault hypothesis and physical failures.

11.4 Behavioral Test Pattern Generation using Fault Simulation

11.4.1 Overview of a general BTPG System

Figure 11.1 shows a system overview allowing to perform an efficient BTPG. The inputs of such a system is a VHDL behavioral description of the circuit for which the BTPG has to be performed and the list of target faults. The outputs of the system comprises of the generated test patterns, three lists containing the detected faults, the redundant faults and the aborted faults.

A detected fault is a fault for which a test pattern exists. A redundant fault is a fault which can not be detected by any test pattern. An aborted fault is a fault for which no solution has been found using the deterministic BTPG within the backtrack limit ; obviously, this is a limit of the BTPG software. The BTPG process will require an important time consuming without this backtrack limit.

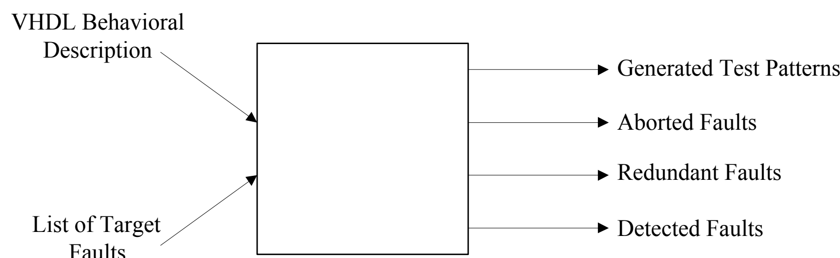


Figure 11.1. BTPG System

In order to speed up the behavioral test generation process, an efficient BTPG has to involve two distinct phases:

- Phase 1 : a Pseudo-Random T.P.G (PRTPG) ; the goal is to randomly generate test sequences using a process which is not time-consuming. The approach is successful if the random generation process allows the target fault list to be drastically reduced.
- Phase 2 : a deterministic T.P.G.

The BFS approach presented in section 6 has to be used for grading the generated test patterns in both phases of the ATPG process.

11.4.2 Behavioral Pseudo-random Test generation using BFS.

The pseudo-random test generation process is an inexpensive way to generate test sequences. By definition, in a pseudo-random method, tests are generated through a quite random process guided by two types of information : the number of patterns which have to be generated and a criterion that patterns have to fulfill.

In order to estimate the length of the test sequences we have been concerned by complexity-based metrics [Cou95] stemming from the Software Engineering field. Complexity-based metrics have been first proposed to evaluate the complexity of software in order not to design software which cannot be easily tested. An interesting application of the complexity metrics to the testing activity was developed in [Mca76]. In this work, McCabe defined the cyclomatic number of the control part of a software. He proved that this number is similar to a path coverage and represents the minimum number of test data to be generated to test the control part.

In order to evaluate the «quality » of the random generated test patterns the inexpensive way is to use behavioral fault simulation.

The reasons for choosing a PRTPG as test generation strategy in phase 1 can be summarized as follows :

- PRTPG is straightforward.
- The number of new faults detected per test is initially large. Thus after a few trials, the fault coverage achieved is sufficiently high to cause a significant decrease in the fault simulation cost per pattern.
- The deficiencies of the PRTPG are that the number of new faults detected per pattern decreases and that the fault simulation effort for the evaluation of those random patterns, which do not detect any additional fault, is wasted. So as soon as an acceptable fault coverage is obtained or the percentage of new detected faults is very low, the random generation of test patterns ends. Only faults which are not detected by the random test sequences are subjected to a deterministic generation process.

The approach involving BFS during a random generation process is summarized in figure 11.2.

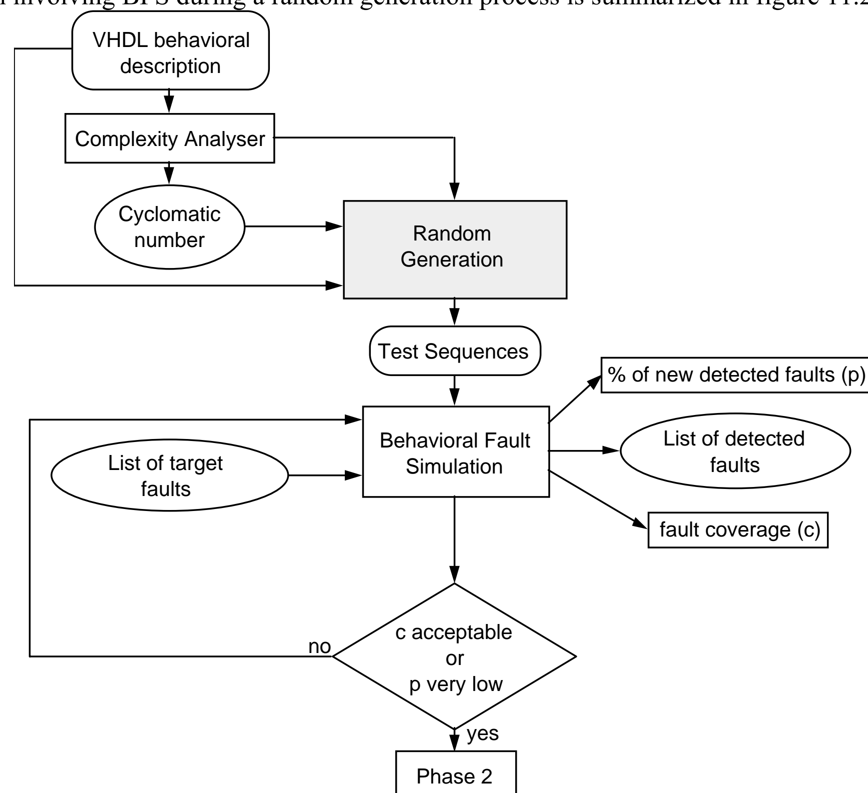


Figure 11.2. Phase one of the BTPG system

11.4.3 Deterministic BTPG using BFS.

The list of remaining faults not yet detected during phase 1 are concerned with phase 2. A deterministic behavioral test pattern generator derived from the path sensitization family introduced in section 2.3 is used during phase 2 in order to generate test patterns.

However instead of deterministically generate test patterns for each fault, once again we use the BFS in order to improve the generation process.

Our approach for phase 2 is illustrated in Figure 11.3

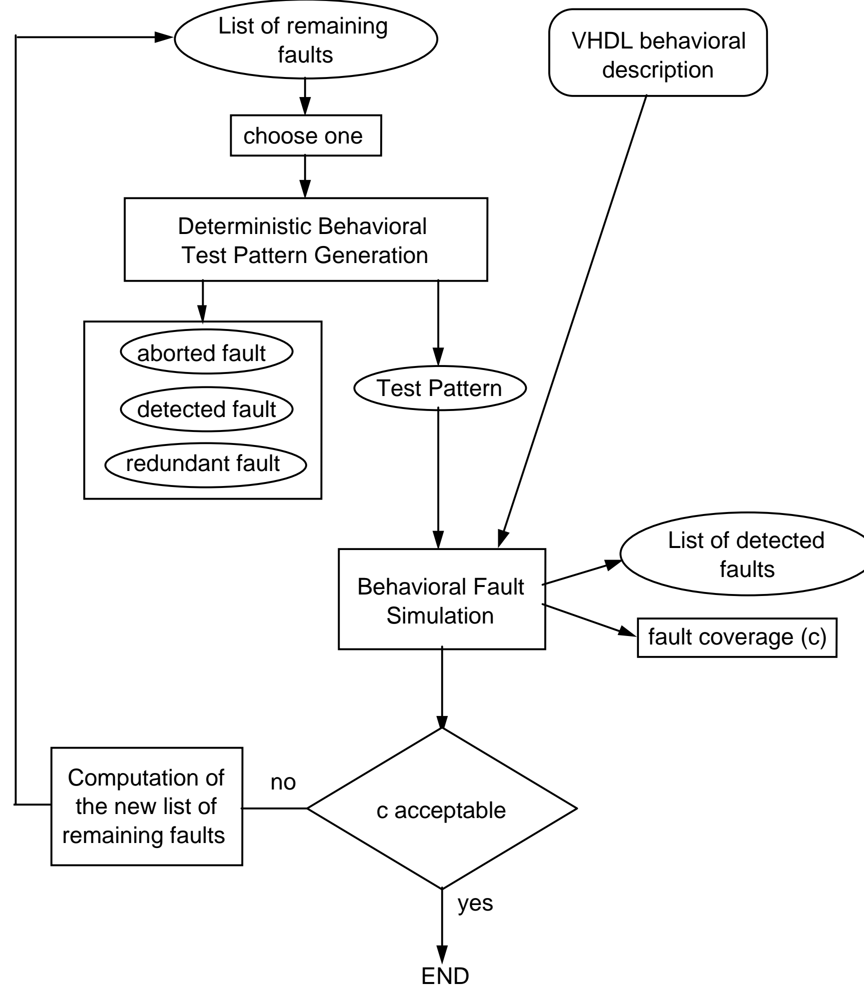


Figure 11.3. Phase two of the BTPG system

11.5 An approach for Behavioral Fault Simulation

11.5.1 Petri-Net Modeling of VHDL Behavioral Descriptions

In order to describe the circuit behavior the most commonly used language is VHDL [Vhd87]. A behavioral view allows a circuit to be described independently of its internal structure by defining how its outputs react when values are applied to its inputs. An internal model is associated with each behavioral description. This model must satisfy two requirements : it should be easy to handle, and able to highlight the concepts linked to behavioral descriptions for a high abstraction level. In order to satisfy the first requirement the definition of the internal model is based on graph concepts. In order to address the second requirement the internal model has been decomposed into two parts, the separation and the interaction of which are explicit : an control part representing the sequencing of operations and pointing out the different execution paths of the description and a data part pointing out the data handled and the operations which transform or test them.

The control part is modeled by a structural graph stemming from the formalism of Petri Nets [Pet81] involving two types of nodes : places and transitions. Structured sub-graphs are predefined for modeling the scanning phase, process activation, control structures (sequential, repetitive or selective) involved in VHDL descriptions.

The Data part is modeled using a graph with two types of nodes : data nodes and operation nodes. There are two classes of data nodes representing input/output signals or internal variables. Each data node has a value attribute allowing its current value to be memorized. Furthermore, in case of signal data nodes an additional attribute is used in order to memorize the previous value (required during the simulation cycles execution). For a signal called S, this attribute is noted S_{old}.

Lastly the interaction of the Control part and the Data part is achieved by associating an operation node with a place node. Two links are involved between the place and the operation node : an activation link of the operation node and an end report link.

Figure 11.4 represents the different entities involved in our modeling scheme.

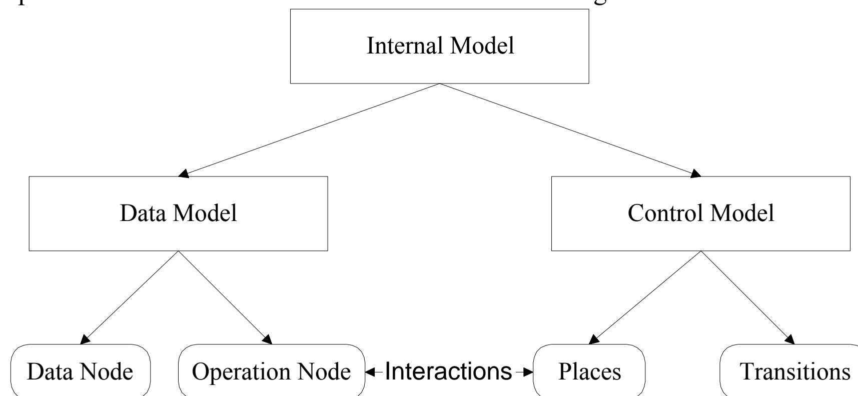


Figure 11.4. Basic entities of the internal modeling

This modeling scheme is illustrated by Figures 11.5 and 11.6. Figure 11.5 gives the VHDL behavioral description of a 8 bit register.

```
Entity Register IS
Port (DI : IN vlbit_1d(1 TO 8) ;
 STRB, DS1, NDS2 : IN vlbit ;
 DO : OUT vlbit_1d(1 TO 8)) ;
END Register
Architecture behavior of Register IS
SIGNAL reg : vlbi_1d(1 TO 8);
SIGNAL enbld : vlbit ;
BEGIN
   strobe: PROCESS (STRB)
   begin
      If (STRB =1) then reg <= DI;
      End If;
```

```
    END PROCESS ;
    enable : PROCESS (DS1,NDS2)
    begin
      enlbd <= DS1 AND NOT (NDS2);
    END PROCESS;
    output : PROCESS (reg,enbld)
    begin
      If (enbld=1) then DO<=reg
      Else DO <= 11111111;
      end If
    END PROCESS;
  END Behavior
```

Figure 11.5. VHDL behavioral description

The corresponding internal model is described in Figure 11.6. The data part is not represented in this Figure.

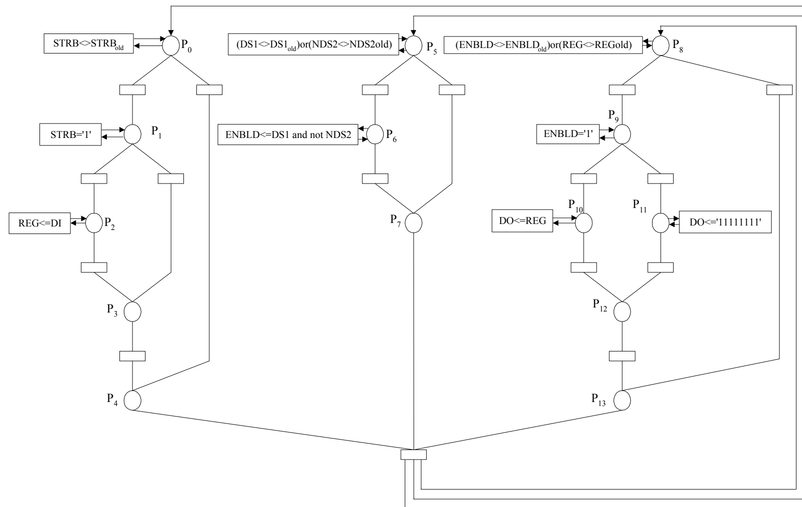


Figure 11.6. Internal Model of the 8-bit Register behavioral description

Figure 11.7 gives more details concerning the first process of the 8-bit register (Process STROBE) ; we have represented in this Figure both the control and data parts. We have to point out that our modeling scheme involves pure Petri Nets. The dynamical aspect of the interaction of the two models is supported by tokens. When a token arrives in a place, the associated operation is performed. In case of a decision node, the result associated with the end report link is used in order to select the next transition to fire.

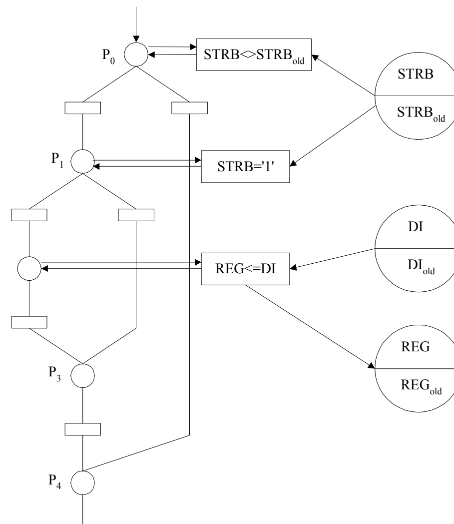


Figure 11.7. Process Strobe's internal modeling

11.5.2 Behavioral Fault Modeling Scheme

A fault modeling scheme allows to derive a set of fault hypothesis on the previously described model. Fault hypothesis are defined according to the elements involved in the internal model of the circuit under test. In order to have a behavioral fault model for which some measures of confidence are provided, we select fault hypothesis according to the fault model proposed in [Gho91].

The selected fault hypothesis are classified into the following three groups. The examples given in order to illustrate fault hypothesis belonging to the different groups are issued from the process Strobe (see section 5.1) :

Stuck-at fault on an element of the data model :

. **F1** : Stuck-at of a data node. It corresponds to the stuck-at of a signal (or a variable).

Example : REG_0 means that the value of the data node REG is stuck at 0.

Stuck-at fault of an element of the control model :

. **F2** : Faults on control elements. $F2_t$ (resp. $F2_f$) : the true branch (resp. the false branch) of the control element is always selected whatever the resulting value set up on the end reporting value.

Example : Let «a» be the selective place corresponding to the operation node STRB=«1». The fault $F2_{at}$ (resp. $F2_{af}$) is expressed by : whatever the results of the test «STRB=1?» is, the branch STRB=1 (resp. STRB=0) is selected.

Stuck-at fault of an element of the interaction of the control and data models :

. **F3** : Faults on the interaction data-control elements. F3 is the statement jump.

Example : $F3_1$ is expressed by the jump of the assignment number 1 (REG<=DI).

We have to point out that the focus of the paper is not the definition of a behavioral fault model. Furthermore the selected fault model may be modified but the BFS method we detail in the next section will still remain valid.

11.5.3 Overall Algorithm for BFS

The BFS technique described above is based on the gate level deductive fault simulation [Arm72]. We will describe in this sub-section the different phases of our method.

First, we give the two restrictions concerning the behavioral descriptions we want to test :

- Unique fault hypothesis.
- A given signal can not be assigned in two different processes.

The principle of our method is composed in two phases (see figure 11.8) : the Fault Free Simulation (FFS), and the Fault List Propagation (FLP).

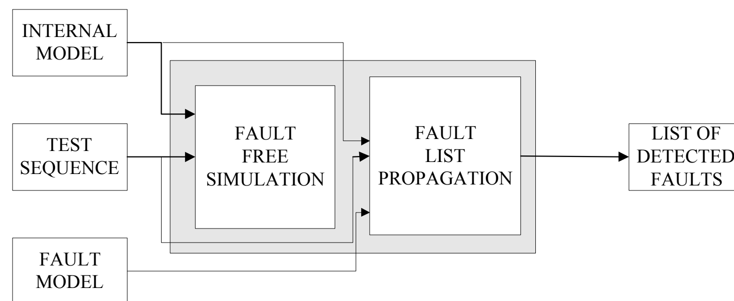


Figure 11.8. Overview of our methodology.

The Fault Free Simulation

The input data are : a test pattern, an internal model representing the behavioral description of our circuit. The goal of this step is to perform a fault free simulation of the circuit for the given test sequence in order to obtain two results :

- the fault free values of the primary output(s) of the system.
- the number of performed simulation cycles (nb_cycle)

The Fault List Propagation

In this sub-section, we define how fault lists are propagated through the internal model during the simulation.

Figure 11.9 represents the flow chart corresponding to the FLP step.

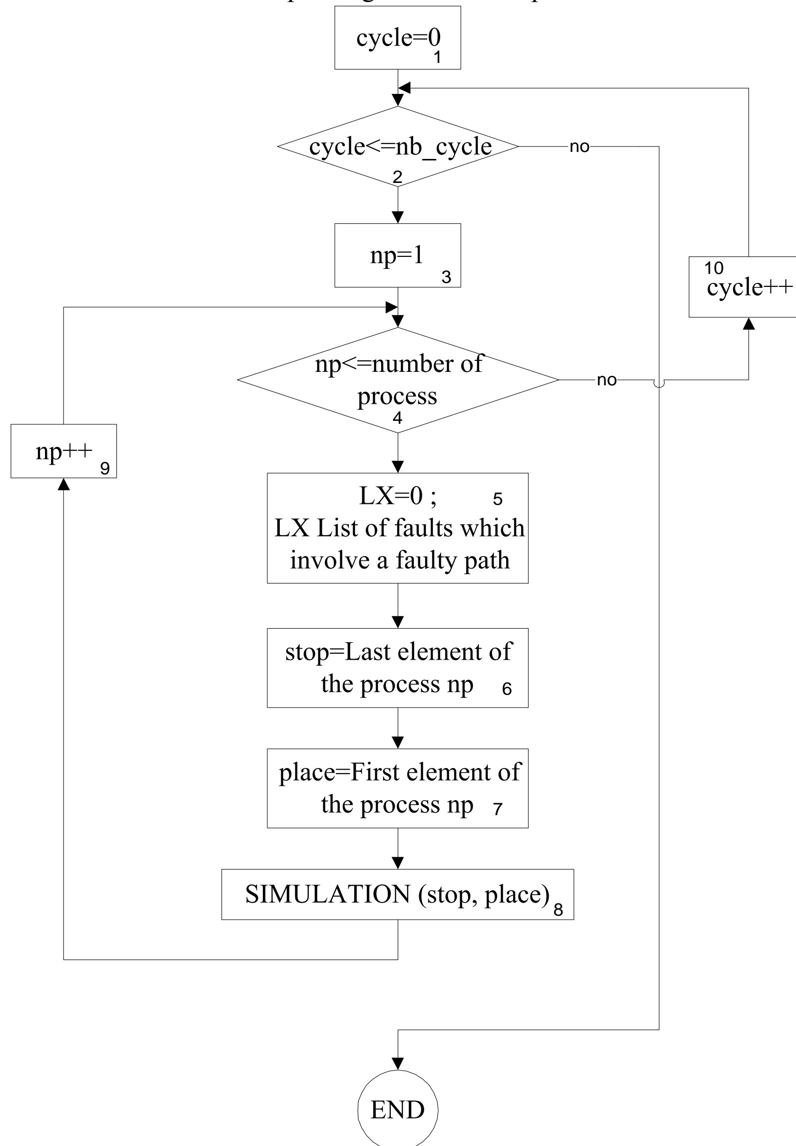


Figure 11.9. Flow chart of the FLP step

A first loop (boxes 1,2,3,4,10) allows the propagation of fault lists for the previously determined number of cycles.

A second loop (boxes 3,4,5,6,7,8,9) allows to achieve the fault list propagation through each process involved in the VHDL description.

The propagation through one process is performed using the procedure SIMULATION (box 8).

The flow chart of the procedure SIMULATION is given in Figure 11.10.

This procedure searches through the control model of the considered process until the last place is encountered (box 1). According to the type of the current place (boxes 2 and 5) , two different rules are applied :

if the place corresponds to an assignment operation, the following rule is used (box 6).

We consider the assignment : DATA1 <= E(DATAi) with 0<i<n and E a function.

We give the rule allowing to compute the fault list L_{DATA1} for an input vector T1 through an assignment element. This input vector allows to assign a value to the variable DATA1 using the algebraic or Boolean operation E(DATAi) where DATAi are represented by data nodes in the data part of the internal model.

$L_{DATA1} = \cup L_{DATAi} \cup$ Ljump-L

- L is the empty set if the value of the attribute old of the data node DATA1 is equal from the value computed using E(DATAi) given the input vector T1. In the other case, L is the fault list LDATA1 computed the last cycle.
- Ljump is the empty set if the value of the attribute old of the data node DATA1 is equal to the value computed using E(DATAi) given the input vector T1. In the other case, Ljump is equal to the singleton $\{F3_{assignment}\}$.
- $\cup L_{DATAi}$ is the union of the fault lists associated to the data nodes DATAi these lists are composed of the faults on the input signals of E(DATAi) which are locally observable on the output DATA1.

if the place corresponds to an selection operation, the rule is expressed by the procedure FAULTY_PATH (box 8).

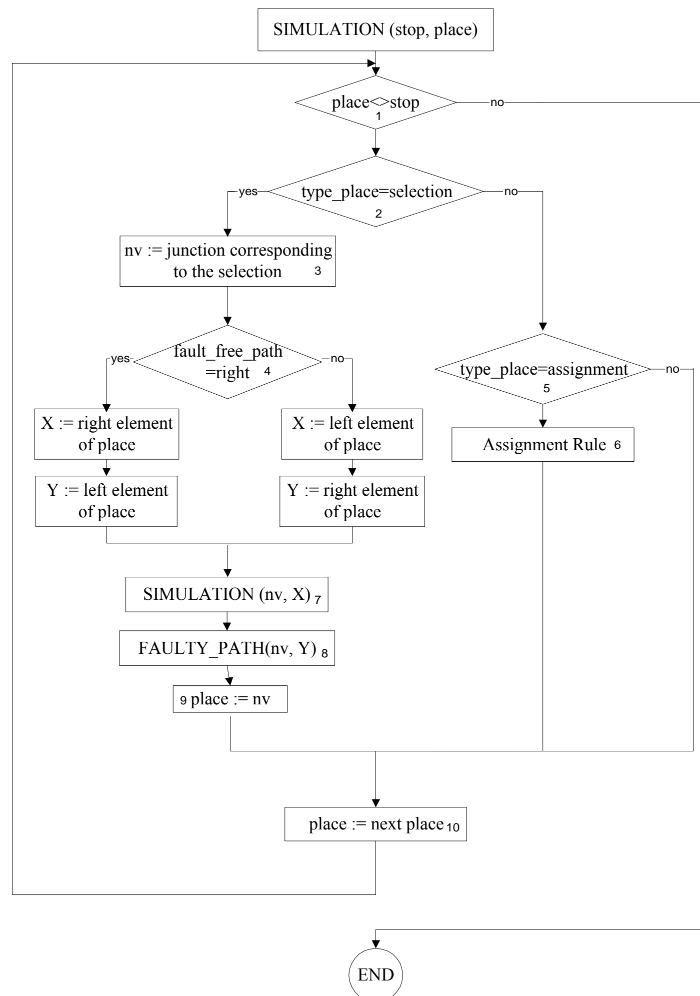


Figure 11.10. Flow chart of the procedure SIMULATION

The flow chart of this procedure is given in Figure 11.11.

The principle allowing to perform the fault list propagation is the following :

For each fault F belonging to the Fault List LF (box 1), we compute the values of the signals involved in the corresponding faulty path (boxes 6,7, 8).

The obtained values are compared with the ones found during the FFS (box 10)

If one of these values are different, the fault F is added to the list Lvar (list of faults locally detected on the signal var) (see box 11).

The detected faults are the ones associated with the primary output signals.

Figure 11.11. Flow chart of the procedure FAULTY_PATH

11.5.4 Pedagogical Example

This sub-section is dedicated to the description of the application of our technique on an example. We will describe a step by step fault simulation. The considered circuit is a 8-bit register, its VHDL Behavioral description and its internal model are given in the section 5.1.

The list of faults being simulated (derived from the fault model presented in 5.2) is the following :

L={ $STRB_0$, $STRB_1$, DI_0 , DI_1 , $DS1_0$, $DS1_1$, $NDS2_0$, $NDS2_1$, DO_0 , DO_1 , REG_0 , REG_1 , $ENBLD_0$, $ENBLD_1$, $F2_{POV}$, $F2_{P0F}$, $F2_{P1V}$, $F2_{P1F}$, $F2_{P5V}$, $F2_{P5F}$, $F2_{P8V}$, $F2_{P8F}$, $F2_{P9V}$, $F2_{P9F}$, $F3_{P2}$, $F3_{P6}$, $F3_{P6}$, $F3_{P11}$ }.

The test sequence we choose is : (***DI=00000000 , STRB=1 , DS1=1 , NDS2=0***)

The Fault Free Simulation

Table 11.1 resume the fault free simulation phase.

	Initial State	Test Sequence (Bold line)	State after the first cycle	Final State
REG	00000000	00000000	00000000	00000000
DI	00000000	**00000000**	00000000	00000000
STRB	0	**1**	1	1
ENBLD	0	0	1	1
DS1	0	**1**	1	1
NDS2	0	**0**	0	0
DO	00000000	11111111	11111111	***00000000***

Table 11.1. Fault Free Simulation

The two results which will drive the fault propagation phase are :

- The fault free value of the primary output : ***DO=00000000***.
- The number of cycles executed : ***nb_cycle=2***.

The Fault List Propagation

We describe in this sub-section the propagation of the fault hypothesis during the step by step simulation.

Nbcycle=0

This is the initialization phase (all the processes are activated).

Process STROBE

L_{P1V}={ $STRB_1$, $F2_{P1V}$ }

These faults are lost because there is no difference of the signals values between the fault free path and the faulty path.

Process ENABLE

L_{ENBLD}={ $DS1_1$, $ENBLD_1$ }

Process OUTPUT

L_{P9V}={ $ENBLD_1$, $F2_{P9V}$}

We find these faults on DO.

L_{DO}={DO_0 , $F3_{P11}$ } $\cup$ { $ENBLD_1$, $F2_{P9V}$}

We have in the fault list L_{DO} all the faults which involve a change on the value of DO with the one we found during FFS.

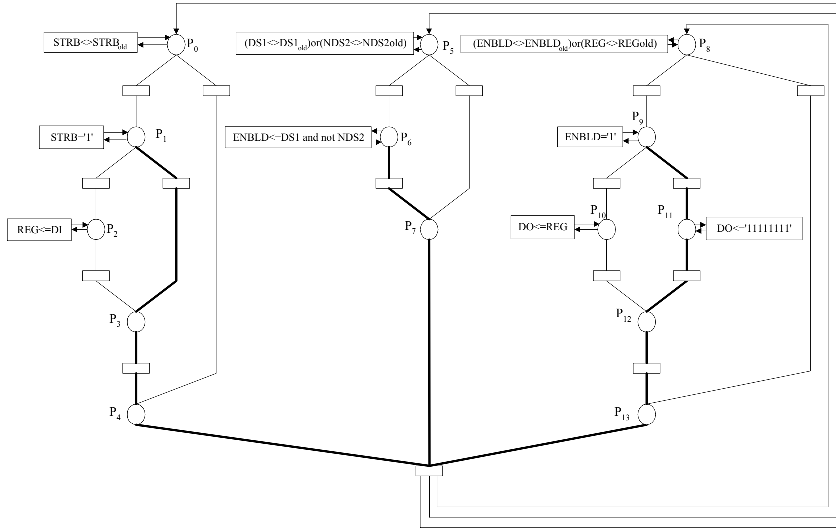


Figure 11.12. Initialization cycle

Nbcycle=1

This is the first cycle of the execution phase. Strobe and Enable are activated , Output is suspended .

Process STROBE

L_{P0F}={ $STRB_0$, $STRB_1$, $F2_{P0F}$ } and L_{P1F}={ $F2_{P1F}$ } are not locally detected.

L_{REG}={ REG_1 , DI_1 }

Process ENABLE

L_{P5F}={ $DS1_0$, $DS1_1$, $F2_{P5F}$ }(these faults are locally detected on ENBLD and added to L_{ENBLD}).

L_{ENBLD}={ $ENBLD_0$, $NDS2_1$, $F3_{P6}$ } $\cup$ { $DS1_0$, $DS1_1$, $F2_{P5F}$ }- L_{ENBLD}.

Process OUTPUT

L_{P8V}={ $DS1_1$, $F2_{P8V}$ }(these faults involve a faulty path).

Since $F2_{P8V}$ has no influence on ENBLD it is not locally detected (no change on the signal DO)

$DS1_1$ leads to ENBLD=1 ; as a result DO<=REG is performed under this fault hypothesis ; hence $DS1_1$ is locally detected on DO.

L_{DO}={ DO_0 , $F3_{P11}$, $ENBLD_1$, $F2_{P9V}$, $DS1_1$ }

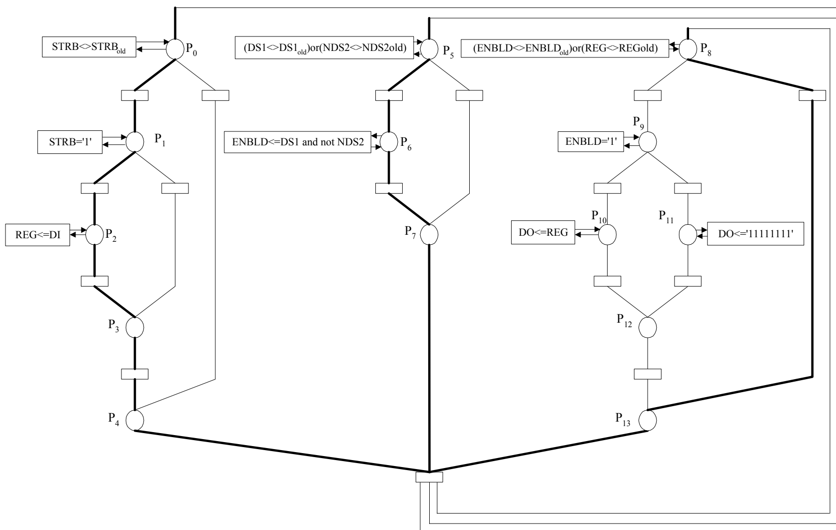


Figure 11.13. First cycle of the execution phase

Nbcycle=2

Process OUTPUT

L_{P8F}={ $ENBLD_1$, $ENBLD_0$, $F3_{P6}$, $NDS2_1$, $DS1_1$, $DS1_0$, $F2_{P5F}$, $F2_{P8F}$ }

L_{P9F}={ $F2_{P9F}$ }, this fault is locally observable on DO.

L_{DO}={ DO_1 , $F3_{P10}$, REG_1 , DI_1 }

$\mathbf{L_{DO\ FINAL} = L_{DO} \cup L_{P9F} \cup (L_{P8F} - L_{DO\ cycle\ 1(car\ DO\ change)})}$

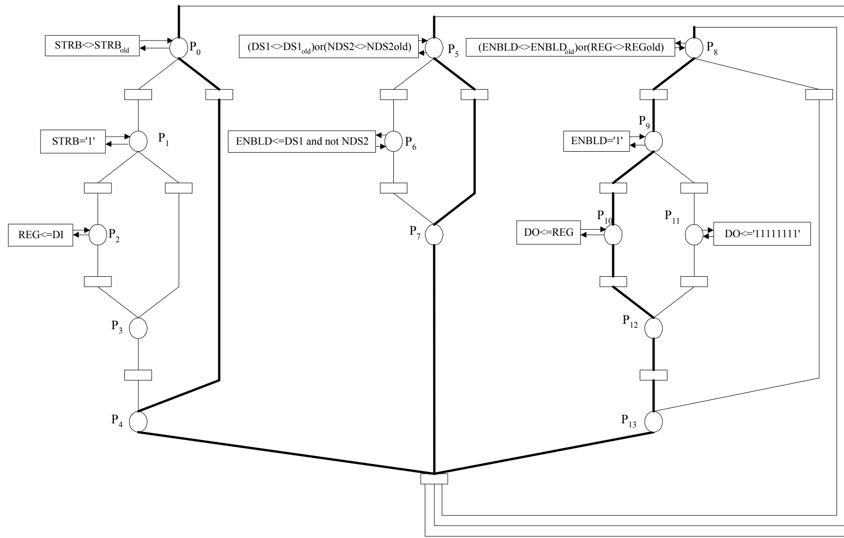


Figure 11.14. Second cycle of the execution phase

11.5.5 Implementation and First Results

The BFS method presented in section 4 was coded in C++ language on a Spark 5 Workstation according to object-oriented paradigms [Stro89].

The BFS software system is decomposed into three parts :

- generation of an ASCII file from a VHDL description ; this file allows to explicitly point out the control and data parts involved in the VHDL description (figure 11.15).
- generation of C++ instances allowing to describe both the internal and fault modeling schemes ; these instances are obtained from two C++ Classes files : control/data models Classes and fault hypotheses Classes (figure 11.15).
- the general algorithm allowing to perform the BFS is implemented in C++ language ; the fault lists propagation rules are executed using the messages sending notion (figure 11.16).

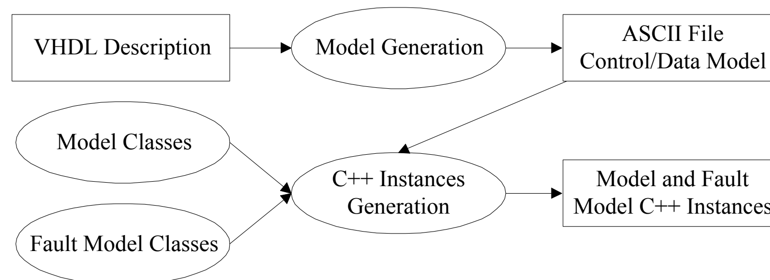


Figure 11.15. Generation of the C++ instances involved in our software

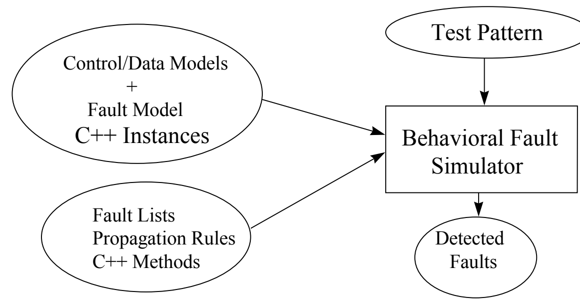


Figure 11.16. overview of our BFS software

Experiments were performed on experimental behavioral descriptions writen in VHDL. These experiments were conducted in order to validate the Behavioral Fault Simulation algorithm propose in section 5.3**.**

We give the final results of the BFS software applied to the VHDL description presented in section 5.1. The test pattern used for the experiment is the one presented in the pedagogical example (sub-section 5.4).

Test Pattern : (DI=00000000 , STRB=1 , DS1=1 , NDS2=0)

Type of Fault	Considered signal or place	Stuck at :
F_2	P_8	F
F_1	ENBLD	0
F_2	P_5	F
F_1	DS1	0
F_3	P_6	
F_1	NDS2	1
F_2	P_9	F
F_3	P_{10}	
F_1	DO	11111111
F_1	REG	11111111
F_1	DI	11111111

Table 11.2. Results of our BFS Software

11.6 Perspectives

In this chapter, we have presented a new approach for Behavioral Fault Simulation. By using an internal modeling allowing the main features of behavioral descriptions to be highlighted, we have been able to define an efficient BFS method. By transposing the fault model proposed in [Gho91] on the elements of the internal model, we make sure that the conclusion reported in [Gho91] about the confidence of the fault model are still characterized our corresponding fault model.

The approach we propose consists into three steps : (i) definition of basic elements allowing to guide the fault lists propagation, (ii) elaboration of simple rules allowing fault lists to be propagated through these previous basic elements, (iii) definition of an overall algorithm allowing the fault list propagation through the internal model using the previous rules.

This approach has been illustrated on an example by detailing each main point of the algorithm.

We have also presented a new approach for Behavioral Test Pattern Generation for digital circuits using Behavioral Fault Simulation.

We have defined the specifications of the BFS software using an Object Oriented Design Tool and implemented the BFS software in C++ language. We have presented the first results obtained using a set of experimental behavioral VHDL descriptions.

Our future investigations are summarized in the following :

- we envision to use a deterministic Behavioral Test Pattern Generator [Nor89,Hum91,Cou92,Stra93,Che91,San92] in order to obtain test patterns for faults remaining not detected after the pseudo-random test pattern generation.

- the experimental scheme has to be complemented with real VHDL behavioral descriptions.

Furthermore, the validation of the behavioral fault model is not addressed in this chapter. However, it is a crucial open problem which must be solved in order to see people from industry becoming confident in Behavioral Testing, even if it has been shown in [You96] that there was a correspondence between fault at the RT level and fault at the gate level.

References

[Vhd87] *VHDL Language reference Manual*, IEEE Standard 1076, 1987.

[Bar86] D.S. Barclay, J.R. Armstrong, "A chip-level Test Generation Algorithm", *23th IEEE/ACM Design Automation Conf. 1986*, pp.257-261.

[One90] M.D. Oneill, D.D. Jani, C.H. Cho, J.R. Armstrong, "BTG : a Behavioral Test Generator", *9th Computer Hardware Description Languages and their Application*, IFIP, pp.347-361, 1990.

[Nor89] F.E. Norrod, "An automatic test generation algorithm for hardware description language", *26th Design Automation Conference*, pp.76-85, July 1989.

[Hum91] H.D. Hummer, H. Veit, H. Toepfer, "Functional Tests for hardware derived from VHDL description", *CHDL 91*, pp. 433-445, 1991.

[Cou92] A.L. Courbis, J.F. Santucci, N. Giambiasi, "Automatic Test Pattern Generation for Digital Circuits", *1st IEEE Asian Symposium*, Hiroshima, pp. 112-118, 1992.

[Stra93] B. Straube, M. Gulbins, E. Fordran, "An approach to Hierarchical test Generation at Algorithmic Level", *IEEE Workshop on hierarchical test Generation* , Blackburg, Virginia, USA, 8-11 august 93.

[Che91] C. H. Chen, C. Wu, D. G. Saab, "BETA : Behavioral Testability Analysis", *IEEE ICCAD'91*, Santa Clara, pp. 202-205, 1991.

[San92] J. F. Santucci, G. Dray, N. Giambiasi, M. Boumédine, "Methodology to reduce computational cost of behavioral test pattern generation using testability measures", *IEEE/ACM 29th DAC*, pp. 267-272, 1992.

[San93]J. F. Santucci, A. L. Courbis, N. Giambiasi, "Speed up Behavioral Test Pattern Generation using an Heuristic Criterion", *IEEE/ACM 30th DAC*, pp. 92-96, 1993.

[Gho91]S. Ghosh, T.J. Chakraborty, "On behavior fault modeling for digital designs", *Journal of Electronic Testing : Theory and Applications*, Vol. 2, pp. 135-151, 1991.

[Ake78a]S.B. Akers, "Binary decision diagram", *IEEE Trans. on Computers*, Vol. C-27 (6), pp. 509-516 (1978).

[Ake78b]S.B. Akers, "Functional testing with binary decision diagrams", *8th Fault Tolerant Computing Symposium*, pp. 82-92 (1978).

[Aba85]M.S. Abadir, H.K. Reghbati, "Functinal test generation for LSI circuits described by binary decision diagrams", *International Test Conference*, pp. 483-492 (1985).

[Cha86]H.P. Chang, W.A. Rogers, J.A. Abraham, "Structured functional level test generation using binary decision diagrams", *IEEE International Test Conference*, pp. 97-104 (1986).

[Lai83]K.W. Lai, D.P. Siewiorek, "Functional testing of digital systems", *20th Design Automation Conference*, pp. 207-213 (1983).

[Pet81]J.L. Peterson, "Petri Net Theory and the Modeling of Systems", Prenctice-Hall, Inc., Englewood Cliffs, New York (1981).

[Olc91]S. Olco, J.M. Colom, "Petri Net based synthesis of VHDL programs", *Euro-VHDL'91*, Stockholm (September 1991).

[Scho85]P.D. Schotts, T.W. Pratt, "Hierarchical modeling of software systems with timed Petri Nets", *International workshop on Timed Petri Nets*, Turin, Italy, pp. 32-39 (June 1985).

[Kir87]T. Kirkland, M.R. Mercer, "A topological search algorithm for ATPG", *24th Design Automation Conference*, pp. 502-507 (1987).

[Schu87]M.H. Schulz, E. Trischler, T.M. Sarfert, "Socrates : A highly efficient automatic test pattern generation system", *International Test Conference*, pp. 1016-1026 (1987).

[Fuj87]H. Fujiwara, T. Shimono, "On the acceleration of test generation algorithms", *International Test Conference*, pp. 1016-1026 (1987).

[Cou95]A.L Courbis, J.F. Santucci, "Pseudo-Random Behavioral ATPG", *Great Lakes VLSI Symposium*, pp. 192-195, 1995.

[Mca76]T.J. Mcabe, "A Complexity Measure", *IEEE Trans. Software Engineering*, N°2, pp. 308-320, 1976.

[Arm72]D.B. Armstrong, "A Deductive Method for Simulating Fault in Logic Circuits", *IEEE Trans. on Computers*, Vol C-21, N°5, pp. 464-471, May 1972.

[Stro89]B. Stroustrup, "Le langage C++", Interedition, 1989.

[You96]C.R. Yount, D.P. Siewiorek, "A Methodology for the Rapid Injection of Transient Hardware Errors", *IEEE Trans.on Computers*, pp. 881-891, August 1996.

2 - SECONDE PUBLICATION

15. D. Federici, J.F. Santucci, P.A. Bisgambiglia, "High Level Fault Simulation : Experiments and Results on ITC'99 Benchmarks", Fifth Annual IEEE International Workshop on High Level Design Validation and Test, HLDVT'00, Claremont Resort and Spa, Berkeley, California, p145-150, 2000.

PETRI NET MODELING AND BEHAVIORAL FAULT MODELING SCHEME FOR VHDL DESCRIPTIONS [1]

Fédérici Dominique, Santucci Jean-François, Bisgambiglia Paul
Université de Corse
Quartier Grossetti
BP 52
20250 Corte
France
email : {federici,santucci,bisgambi}@univ-corse.fr

Keywords : Behavioral Descriptions, Behavioral Fault Modeling, Petri Nets, Fault Simulation

Abstract

This paper deals with the modeling of VHDL behavioral descriptions and the development of an efficient behavioral fault modeling scheme. A set of behavioral test pattern generation methods have been proposed in the recent past. One constraint having to be pointing out is the fact that circuits under test (CUT) are expressed using a high level behavioral VHDL description. We propose to define a behavioral fault simulation method own to (i) a behavioral modeling of CUT using Petri Nets and (ii) an efficient behavioral fault modeling scheme. In this paper, the emphasis is put on the modeling aspects.

Introduction

In the recent past a set of Behavioral Test Pattern Generation methods have been proposed in several international conferences [1, 2, 3, 4, 5, 6, 7, 8]. Motivated by this fact we are interested in defining an efficient behavioral fault simulation methodology. One important constraint within which we have to work is the fact that circuits under test (C.U.T.) are described using a high-level behavioral description. The previous work has focused on the A.T.P.G. method and not on the fault modeling and simulation tasks.

Our goal in this article is to present an efficient modeling and fault modeling scheme for behavioral descriptions.

The first part of the presentation is devoted to general definitions of modeling and fault modeling. The second part will deal with behavioral modeling of VHDL descriptions. The definition of an exhaustive behavioral fault modeling scheme is given in the third part. Future work concerning behavioral fault simulation is briefly introduced in the last section.

[1] this work is supported by the EEC : HCM network BELSIGN- contract N° CHRX-CT 94-0459

1. General definitions

The modeling of a circuit can be done according to two points of view : a structural view and a behavioral one. We focus on discrete models. The use of a discrete model to solve a given problem refers to a set of discrete variables called State Variables which define the State Space of the model. The behavior associated with the model is necessarily expressed through the alteration of the variable values.

In case of a behavioral view, the circuit is seen as a black box defining its output values according to input values by the use of algorithms, true tables, state tables or boolean equations.

Behavioral models can have two main representations :

. alphanumeric representations : they are textual representations involving objects such as variables, operators and control constructs.

. graph representations : they are based on transformation graphs, Petri Nets, etc... . These representations offer a structure, i.e. an interconnexion of basic elements for which a behavior is predefined out of context. We have to point out that this structure is not a potential structure but some elements can have a physical interpretation.

Whatever the general principle on which the Test Pattern Generation (T.P.G.) or Fault Simulation (F.S.) methods are based, structural or behavioral models of the circuit under test are considered as the reference or faultless model.

Given a description (structural or behavioral view) of a circuit, the T.P.G. or F.S. method has to operate on the information available by the description. In both cases fault models are needed in order to represent physical failures efficiently.

Before dealing in detail with behavioral fault modeling, we propose some definitions of a fault, a fault model, an error and a defect in order to guide the reader towards behavioral fault modeling.

A fault can be defined both on a structural and a behavioral model. In the case of the use of a discrete event model, a fault hypothesis is :

. either an hypothesis of a wrong item behavior belonging to the model, but considered out of context.

. or an hypothesis of modification of the initial global description by adding, suppressing or combining basic items, without modifying the predefined behavior of these items.

A fault model is the list of the selected fault hypothesis, i.e. the fault hypothesis taken into account for T.P.G. or F.S. There are several interests in using a fault model. The main one is to reduce all possible combinations of T.P. sequences at the input of the circuit under test. The definition of fault hypothesis allows the definition of fault classes, the reduction of the list of selected faults. Furthermore, a fault simulator can be used to determine all the faults set off by a given test pattern. Lastly, an hypothesis may be used to make a correlation between a fault and a physical defect which is useful for fault localization.

It should be pointed out that the modification of the global description of the model may lead to take into account a much too large number of possible combinations. In a similar way, the overly complex faults concerning the modification of a basic items behavior may also not be taken into account.

Whatever the given abstraction level, an error is the manifestation of a fault expressed as a difference between the state of the fault free model and the faulty model at a given time. If it is not posssible to obtain a difference between the two models, the fault is said to be redundant.

A defect refers to a physical anomaly of the actual circuit. It should be pointed out that a fault may or may not have a direct mapping with a physical defect.

2. Behavioral Modeling

In order to facilitate definition of formal methods based on graph structures, we have been interested in a first step to study an efficient graph modeling of behavioral descriptions. Behavioral descriptions are given using the VHDL language. In a first sub-section we highlight the main features of VHDL behavioral descriptions. We describe in a second sub-section how these features have been represented by an efficient graph modeling scheme.

2.1 Main features of VHDL Behavioral Descriptions

Four kinds of objects are involved in VHDL behavioral descriptions :

. Constants which have a predefined and unchangeable value.
. Variables which can be modified by an assignement statement.
. Signals which are the specificity of VHDL.
. Processes which are the fundamental objects manipulated by VHDL.

Variables are local to processes that is to say they can be read or affected only in the process where they are declared.
Signals are global to the overall description that is to say that they may be common to several processes.

A behavioral description leans on the description of a set of processes where each process is defined using a procedural description.

The key statement of a process is the WAIT statement.
syntax : WAIT {ON signal-list} {UNTIL boolean-condition}

This statement allows to suspend a process execution, that will restart when the next condition will be true : an event occurs on one of the signals specified in the signal_list and the evaluation result of the boolean_condition is true.

The simulation of a VHDL description is composed of two steps : an initialization phase and an execution phase.

The initialization phase consists in determining the initial value of each signal according to the following rules :

. the initial value is given explicitly during the signal declaration.

. the initial value is given implicitly. This value is defined like being the first of the signal definition domain.

With these initial values, each process of the description is executed without looking at the conditions associated with Wait statements.

The execution phase is performed by scanning of the sensitive signals and is driven by the events.

This modeling scheme will be described in sub-section 2.2 using the behavioral description given in Figure 1.

```
Entity Register IS
Port (DI : IN vlbit_1d(1 TO 8) ;
        STRB, DS1, NDS2 : IN vlbit ;
        DO : OUT vlbit_1d(1 TO 8)) ;
END Register

Architecture behavior of Register IS
SIGNAL reg : vlbi_1d(1 TO 8);
SIGNAL enbld : vlbit ;
BEGIN

  strobe: PROCESS (STRB)
  begin
  If (STRB =1) then reg <= DI;
  End If;
  END PROCESS Strobe;

  enable : PROCESS (DS1,NDS2)
  beign
  enlbd <= DS1 AND NOT (NDS2);
  END PROCESS;

  output : PROCESS (reg,enbld)
  Begin
  If (enbld=1) then DO<=reg
  Else DO <= 11111111;
  end If
  END PROCESS;

END Behavior
```

Figure 1 : VHDL behavioral description

2.2 Behavioral Modeling

This sub-section aims at describing the model used to highlight the main features of VHDL behavioral descriptions. In order to facilitate definition of formal methods based on graph structures for behavioral fault simulation, we have been interested in the development of two models :

• the input/output model allowing to represent existing links between signals involved in different processes of a description [9]. This model has been defined in [9] in order to perform Test Pattern Generation.

• the activation model pointing out the the execution of the processes involved in a VHDL description. This model is based on a model developed in [9]. However we have improved this previous model by defining a new modeling scheme involving pure Petri Nets. The use of Petri Nets will be useful when a behavioral fault simulation algorithm will be defined.

The VHDL behavioral description activation model is given in Figure 2. We have to mention that the activation model is made up of the following elements : a data model, a control model and their explicit interactions.

The Data Model represents the objects (variables and signals) and the handled operations involved in the description. It is composed of a graph involving two types of nodes : data nodes and operation nodes.

Let us point out that the data nodes have not been represented on the Figure 2.

Operation nodes are of two types :

. assignment nodes which represent the assignment of an object by an algebraic or boolean operation.

. decision nodes which represent an algebraic or boolean test the result of which is taken into account in a branch in the Control Model.

The Control Model represents the sequencing of the operations involved in the description. It is based on a Petri Net modeling.

The interaction between the Control Model and the Data Model is achieved by associating an operation node to a place. Two links are involved between the place and the operation node : an activation link of the operation node and an end report link.

The dynamical aspect of the interaction of the two models is supported by tokens. When a token arrives in a place, the associated operation is performed. In case of a decision node, the result associated with the end report link is used in order to select the next transition to fire. In Figure 2, we give the Petri Net associated with the description given in Figure 1.

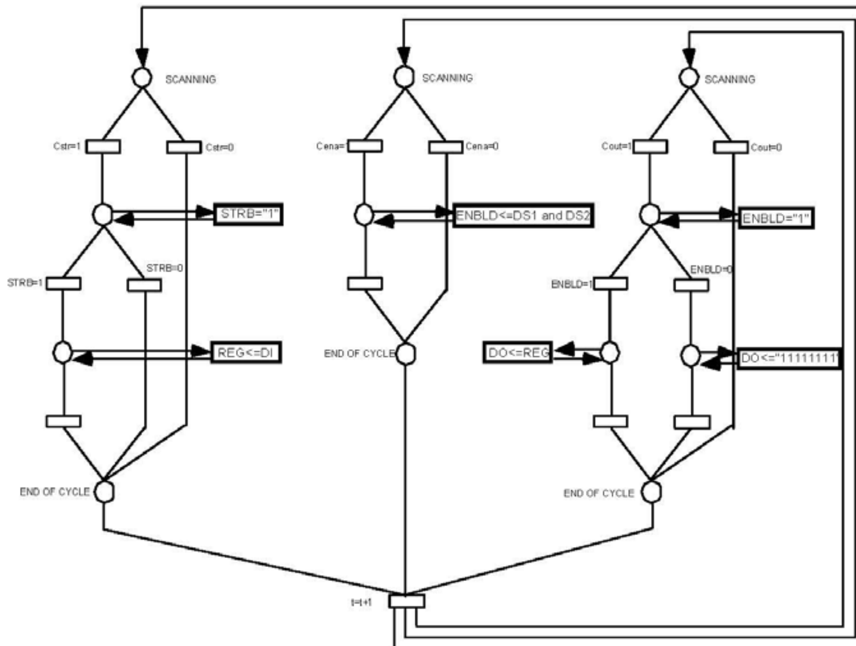


Figure 2 : Activation Model

3. Behavioral Fault Modeling

A fault modeling scheme based on the definition presented in section 1 allows to derive a set of fault hypotheses on the previously described model.

Fault hypotheses are defined according to the elements involved in the model of the circuit under test.

A General Fault Model (G.F.M.) can be generated using the two rules presented in section 1. This fault Model involves all possible fault hypotheses defined on the basic modeling element and with modification of the model structure.

Because of the too large number of faults involved in this model we may have to reduce it.

In order to have a behavioral fault model for which some measures of confidence are provided, we may select from among the G.F.M. some fault hypotheses according to the fault model proposed in [10].

The selected fault hypotheses may be classified as follows :

1. Stuck-at fault on an element of the data model :

   . The value attribute of a data node is stuck-at V1 or V2 where V1 and V2 express the lower and upper extremes of the domain definition of the represented signal or variable.

. The output of an operation node may fail such that it permanently returns V1 or V2, where V1 and V2 express the lower and upper extremes of the range of the operation.

2. Stuck-at fault of an element of the control model :
   . a control transition is always selected whatever the resulting value set up on the end report link may be
   . a process is never active
   . a process is always active

3. Stuck-at fault of an element of the interaction of the control and data models :
   . when a token arrives in a place, the corresponding operation is not performed. This means that the activation link is stuck-at 0.

Depending on the results given by these fault hypotheses in terms of gate-level fault coverage, this fault model can obviously be extended. However, having translated the behavioral fault model proposed in [10] by a set of fault hypotheses on the basic elements of the activation model defined in section 2, the quality of the test vectors generated for the previous fault model can be evaluated by the results presented in [10].

4. Current and Future work

Our current work deals with behavioral fault simulation [11]. Fault simulation is known to be an important step in the testing of digital systems. It is usually used to evaluate the fault coverage of each test vector generated by an Automatic Test Pattern Generation program. Traditional fault simulation algorithm [12, 13, 14] are not able to deal with behavioral description and behavioral fault hypotheses. We are interested in defining new algorithms allowing to perform fault simulation on VHDL behavioral descriptions.

Given a test sequence, a VHDL description, a set of behavioral fault hypotheses, our goal is to deduce the list of behavioral fault hypothesis detected by the given test sequence.

The first approach we have investigated is to define an algorithm based on the deductive fault simulation [12].

This approach consists in propagating a fault list through the basic elements of the activation model. Given a test sequence, this propagation is made until that we meet a primary output. To evolve this method, we have decomposed our activation model as an interconnexion of four basic elements shown in figure 3.

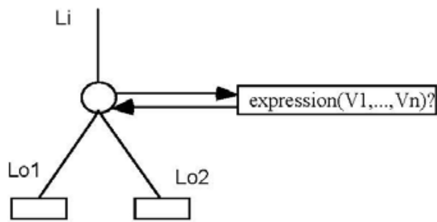


Figure 3a : Selection

Figure 3b : Assignment

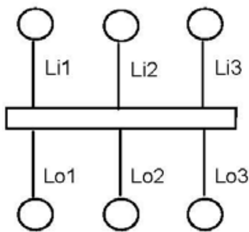


Figure 3c : Parallelism

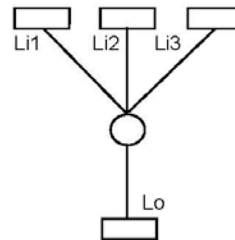


Figure 3d : Junction

Figure 3 : Basic elements involved in the activation model

We have defined how the list of detected faults Loi can be computed according to:

- each kind of basic element.
- Lii, list of detected faults.
- a given test pattern.

Our current work is to find out, how a fault list is propagated through each basic element.

Our future work will investigate the definition of a method based on a concurrent fault simulation[14].

References

[1] D.S. Barcaly, J.R. Armstrong, "A chip-level Test Generation Algorithm", 23th IEEE/ACM Design Automation Conf. 1986, pp.257-261.

[2] U.H. Levendel, P.R. Menon, "Test Generation algorithms for computer hardware description languages", IEEE Transactions on Computers, Vol.C-31, pp.577-588, 1982

[3] M.D. Oneill, D.D. Jani, C.H. Cho, J.R. Armstrong, "BTG : a Behavioral Test Generator", 9th Computer Hardware Description Languages and thier Application, IFIP, pp.347-361, 1990.
[4] F.E. Norrod, "An automatic test generation algorithm for hardware description language", 26th Design Automation Conference, pp.76-85, July 1989.
[5] H.D. Hummer, H. Veit, H. Toepfer, "Functional Tests for hardware derived from VHDL description", CHDL 91, pp. 433-445, 1991.
[6] A.L. Courbis, J.F. Santucci, N. Giambiasi, "Automatic Test Pattern Generation for Digital Circuits", 1st Asian Symposium, Hiroshima, pp. 112-118, 1992.
[7] J.F. Santucci, A.L. Courbis, N. Giambiasi, "Behavioral Testing of digital Circuits", Journal of MicroelectronicSystmes Integration, Vol.1, N°1,, March 1993, pp.55-78.
[8] B. Straube, M. Gulbins, E. Fordran, "An approach to Hierarchical test Generation at Algorithmic Level", IEEE Workshop on hierarchical test Generation , Blackburg, Virginia, USA, 8-11 august 93.
[9] V. Pla, J.F. Santucci, N. Giambiasi, "On the modeling and Testing of VHDL Behavioral Descriptions of Sequential Circuits, 3rd IEEE Euro-Dac/Euro-VHDL, Hamburg, September 93, pp.440-445.
[10] S. Ghosh, T.J. Chakraborty, "On behavior fault modeling for digital designs", Journal of Electronic Testing : Theory and Applications, Vol. 2, pp. 135-151, 1991.
[11] D. Fédérici, J.F. Santucci, P. Bisgambiglia, "Behavioral Fault Modeling and Simulation", 3rd BELSIGN Workshop Proceedings, 11-12 April 96, Porticcio, France.
[12] D.B. Armstrong, "A Deductive Method for Simulating Fault in Logic Circuits", IEEE Trans. on Computers, Vol C-21, N°5, May 1972, pp. 464-471.
[13] P. Goel, P.R. Moorby, "fault Simulation Techniques for VLSI Circuits", VLSI Design, July 1984, pp.22-26.
[14] E.G. Ulrich, T.Baker, "The Concurrent Simulation of Nearly Identical Digital Networks", Proc. 10th Design Automation Workshop, IEEE and ACM, New York, June 1973, pp.145-150.

3 - TROISIÈME PUBLICATION

7. L. Capocchi, F. Bernardi, D. Federici, P. Bisgambiglia, "A DEVS-based Concurrent and Comparative Fault Simulation Algorithm", Proceedings of the SCS Summer Computer Simulation Conference (SCSC 2005), San Diego, CA, USA, 2005.

A DEVS-based Concurrent and Comparative Fault Simulation Algorithm

Laurent CAPOCCHI Fabrice BERNARDI Dominique FEDERICI
Paul BISGAMBIGLIA
University of Corsica
UMR CNRS 6134, Quartier Grossetti, BP 52
20250 Corte, France
{capocchi, bernardi, federici, bisgambi}@univ-corse.fr

Abstract

Concurrent and Comparative Simulation (CCS) with Multi-List Propagation (MLP) provides a way to perform several simulations in a single execution run and Concurrent Fault Simulation (CFS) has been one of its first applications. The main obstacles to a wide use of this technique are the high complexity of the concurrent simulation algorithms, along with the difficulty to integrate them in a simulation kernel. We focus in this paper on the CFS with MLP of systems described in the BFS-DEVS formalism, an extension of the original DEVS simulator that integrates the CCS algorithm. Application is performed in the behavioral digital domain of systems described in the VHDL language.

Keywords: Discrete Event Simulation, Concurrent and Comparative Simulation, Fault Simulation, DEVS, VHDL.

1 Introduction

Over the last 40 years discrete event simulation has begun to replace physical experimentation, as modeling and simulation offer efficient alternatives to expensive and complex physical experiments. Discrete event modeling allows designing an easy-to-handle and reusable representation of a system. However classical discrete event simulation only permits one simulation at a time for a system. This solution can appear to be very time consuming when many successive simulations are required for the complete experiment, especially in terms of result analysis and observability. In order to escape from these limitations the CCS with MLP appears to be an adapted solution, by providing a way to perform several simulations or other tasks in a single execution run.

We focus in this paper on the CFS with MLP of systems described in the BFS-DEVS formalism (Behavioral Fault Simulation for Discrete EVent system Specification) based on the DEVS formalism introduced by Zeigler in the late 70's in [1]. DEVS provides a modular and modeling approach, and automatically defines a simulator directly from a model using formal elements and algorithms. It also allows the integration of concurrent simulation algorithms in a simple, evolutive and transparent fashion for the system modeler.

Discrete event simulation has been used in hundreds of different domains (graph analysis, economical studies, symbolic simulation,...) and CCS can be virtually used in each of them. However even if CFS is approximately 20 years old, it is the only real existing application of CCS, and is not or not much used in the discrete event domain. Indeed, for many years, fault simulation has been an essential basic tool of CAD (Computer Aided Design) systems for digital circuits (see [2], [3] and [4]), but has never been used in the discrete event domain where it could help to analyze faulty behaviors. [5] proposed a first approach for fault simulation of systems described using DEVS but did not formally integrate the concurrent algorithms. We present in this paper the BFS-DEVS formalism integrating the CFS with MLP algorithms in its kernel, and allowing the modeler to specify the faulty behavior of a system using a new faulty transition function.

The main objective of our work consists in defining a behavioral concurrent fault simulator implementing the BFS-DEVS formalism. We modify for that the original DEVS formalism by introducing a new transition function allowing the modeler to describe the

faulty behavior of a system. The BFS-DEVS simulator kernel is an evolution of the original DEVS simulator kernel that integrates the CCS algorithms. These lasts are based on fault lists propagated and directed by propagation rules inside BFS-DEVS models. Validation of our approach is performed on behavioral fault simulation of digital circuits described in high level description languages.

This article is structured as follows. We describe in a first section the modeling specifications including the DEVS formalism and its extension BFS-DEVS. In a second section, we introduce the fault simulation environment and the general architecture of this environment is described. We introduce the BFS-DEVS library concept and the fault model used by the BFS-DEVS simulation kernel. Finally, we provide some results, a conclusion and some perspectives of work.

2 Modeling Specifications

2.1 DEVS Formalism

Introduced by Zeigler in the early 70's and completed in [6], DEVS is a set-theoretic formalism that provides a mean of modeling discrete event systems in a hierarchical and modular fashion. Using this formalism, modeling can be easily performed by decomposing a large system into smaller component models with coupling specifications between them.

2.1.1 DEVS Modeling

As in General System Theory (see [7]), a DEVS atomic model contains a set of states and transition functions triggered by the simulator. It is represented by the following structure:

$$AM =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

where:

- $X : \{(p, v)|(p \in inputports, v \in X_p)\}$ is the input ports set and values for the reception of external events.
- $Y : \{(p, v)|(p \in outputports, v \in Y_p)\}$ is the output ports set and values for the emission of events.
- S: is the internal sequential states set.
- $\delta_{int} : S \rightarrow S$ is the internal transition function that will bring the system to the next state after the time returned by the time advance function.
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function that will schedule state changes in reaction to an input event.
- $\lambda : S \rightarrow Y$ is the output function that will generate external events just before the internal transition takes place.
- $ta : S \rightarrow \mathbb{R}^+_\infty$ is the time advance function, that will give the life time of the current state (returns the time to the next internal transition).

A DEVS coupled model CM is a structure:

$$CM =< X, Y, D, \{M_d \in D\}, EIC, EOC, IC >$$

where:

- X is the input ports set for the reception of external events.
- Y is the output ports set for the emission of external events.
- D is the components set (coupled or basic models).
- M_d is the DEVS model for each $d \in D$.
- EIC is the input links set, that connects the inputs of the coupled model to one or more of the inputs of the components that it contains.
- EOC is the output links set, that connects the outputs of one or more of the contained components to the output of the coupled model.
- IC is the set of internal links, that connects the output ports of the components to the input ports of the components in the coupled models.

In a coupled model, an output port from a model $M_d \in D$ can be connected to the input of another $M_d \in D$ but cannot be connected directly to itself.

2.1.2 DEVS Simulation

As described in [8], DEVS allows a hierarchical and modular modeling of decomposable discrete event systems thanks to these specifications. One the main advantages of DEVS is that the simulator is directly and automatically extracted from the model. This DEVS simulator uses the modeling hierarchy and associates a *simulator* component to each atomic model and a *coordinator* component to each coupled model. This association allows the control of the behavior of each atomic model and the synchronization of the whole set of models.

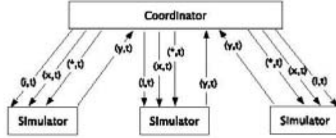


Figure 1: DEVS Simulator Message Exchange

Communication between elements drawn in Figure 1 is performed using four types of messages. The initialization message (i, t) is used to perform an initial temporal synchronization between all actors of the system (messages 1a and 1b on the figure). The internal transition message $(*, t)$ implies the execution of an internal event (message 2a), while the output message (y, t) allows the output communication to the parent elements, and results from the reception of a $(*, t)$ message (message 2b). Finally the external transition message (x, t) allows the execution of an external event (message 3). All these messages are more precisely discussed in the next sections.

2.2 BFS-DEVS Formalism

As [5] we consider a *behavioral fault* as a modification of the transition functions, and/or of the output functions of the DEVS models. Thus a fault influences the behavior (or the state) of an affected DEVS model and can lead to a faulty behavior.

2.2.1 BFS-DEVS Modeling

The BFS-DEVS formalism introduced in this paper allows the specification and the simulation of faulty discrete event models. A BFS-DEVS model is defined by a classical DEVS structure with the following additional features first introduced by [5]:

- Transition and/or output functions are modified if behavioral faults are present in the DEVS model.
- Faulty behavior of DEVS models is specified by a new faulty external transition function δ_{fault}.

Consider a DEVS atomic model whose *healthy* behavior can be represented by the following DEVS structure:

$$AM^h =< X^h, S^h, Y^h, \delta^h_{int}, \delta^h_{ext}, \lambda^h, ta^h >$$

We define the following structure to take the *faulty* behavior into account:

$$AM^f =< X, S, Y, \mathbf{F}, \delta_{int}, \delta_{ext}, \delta_{\mathbf{fault}}, \lambda, ta >$$

where,

- $X = X^h \bigcup X^f$ is the set of input values, with X^f representing a set of new faulty values.
- $S = S^h \bigcup S^f$ is the set of state values, and S^f is the set of new faulty states that the model can present due to the presence of a faulty input event.
- $Y = Y^h \bigcup Y^f$ is the set of output values, with Y^f representing the set of new faulty values that the output ports can take when a faulty input event occurs.
- $F = \{F_1, F_2, F_3\} \bigcup \oslash$ is the faults set, with $\{F1, F2, F3\}$ is the fault model described further.
- $\delta_{int} : S \times F \rightarrow S$ is the internal transition function modified by the presence of faults and presents the restriction $\delta_{int}(s, \oslash) = \delta^h_{int}(s)$.
- $\delta_{ext} : Q \times X \times F \rightarrow S$ is the external transition function modified by the presence of faults and verifies $\delta_{ext}(s, e, x, \oslash) = \delta^h_{ext}(s, e, x)$ if $x \in X^h$.
- $\delta_{fault} : Q \times X \times F \rightarrow S$ is the faulty external transition function providing the faulty behavior when a faulty event occurs.
- $\lambda : X \times F \rightarrow S$ is the output function, which verifies $\lambda(s, \oslash) = \lambda^h(s)$.
- $ta : S \times F \rightarrow \mathbb{R}^+_\infty$ is the time advance function,f with the restriction $ta(s, \oslash) = ta^h(s)$.

BFS-DEVS specifications are very similar with the original DEVS ones. The difference is that any time a faulty event x_f ($x_f \in X^f$) occurs, the new faulty state is calculated by the faulty external transition function δ_{fault}. If the healthy event x_h ($x_h \in X^h$) occurs, then the new healthy state is calculated by the healthy external transition function δ^h_{ext}.

2.2.2 BFS-DEVS Simulation

Communication between components is achieved in the original DEVS formalism using four types of messages. To distinguish a faulty behavior inside a simulation, we introduce a new message type that activates the model as soon as a *fault event* x_f is present on one of its ports at a given time t. Activation of the faulty behavior of a component is performed using a message representing a faulty external event as shown in Algorithm 1 and Algorithm 2.

Algorithm 1 BFS-DEVS Simulator Algorithm

Variables:
parent *(parent coordinator)*
t_l *(time of last event)*
t_n *(time of next internal event)*
$DEVS = \{X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a\}$
y *(current output value)*
Receives i-message(i,t) at time t:
  $t_l = t - e$
  $t_n = t_l + t_a(s)$
Receives *-message(*,t) at time t:
  if $(t != t_n)$ then
    Error: bad synchronisation
  $y = \lambda(s)$
  send y-message to coordinator parent
  $s = \delta_{int}(s)$
  $t_l = t$
  $t_n = t_l + t_a(s)$
Receives x-message(x,t) at time t:
  if $!(t_l \leq t \leq t_n)$ then
    Error: bad synchronisation
  $e = t - t_l$
  $s = \delta_{ext}(s, e, x)$
  $t_l = t$
  $t_n = t_l + t_a(s)$
Receives f-message(x_f,t) at time t:
  $e = t - t_l$
  $s = \delta_{fault}(s, e, x_f)$
  $t_l = t$
  $t_n = t_l + t_a(s)$

A coordinator, using the X set, is then able to send two types of external messages to its imminent children, a "x-message" for a healthy simulation, and a "f-message" for a faulty simulation. This specification is shown bold on the Algorithm 2.

The faulty simulation can only be performed if the healthy simulation has already been achieved, because values of the first are used as references for the second.

Algorithm 2 BFS-DEVS Coordinator Algorithm

Variables:
parent *(parent coordinator)*
t_l *(time of last event)*
t_n *(time of next internal event)*
$DEVS = (X, Y, D, \{Md\}, \{Id\}\{Z_{i,d}, select\})$
$list_{event}$ *(list of elements (d, t_{nd}))*
$d*$ *(selected imminent child)*
Receives i-message(i,t) at time t:
  for each d in D:
    send i-message to d
  update $list_{event}$ with t_{nd} and *select*
  $t_l = max\{t_{ld} \mid d \in D\}$
  $t_n = min\{t_{nd} \mid d \in D\}$
Receives *-message(*,t) at time t:
  if $(t != t_n)$ then:
    Error: bad synchronisation
  $d* = first(list_{event})$
  send y-message to $d*$
  for each d in D:
    send x and f-message to d if $x \in X_s$
    send f-message to d if $x \in X_f$
  update $list_{event}$ with t_{nd} and *select*
  $t_l = t$
  $t_n = min\{t_{nd} \mid d \in D\}$
Receives x-message(x,t) at time t:
  if $!(t_l \leq t \leq t_n)$ then:
    Error: bad synchronisation
  $receivers = \{r \mid r \in D, N \in I_r, Z_{N,r}(x) \neq \oslash\}$
  for each r in *receivers*:
    send x-message to r with $x_r = Z_{N,r}(x)$
  update $list_{event}$ with t_{nd} and *select*
  $t_l = t$
  $t_n = min\{t_{nd} \mid d \in D\}$
Receives f-message(x_f,t) at time t: if
  $!(t_l \leq t \leq t_n)$ then:
    Error: bad synchronisation
  $receivers = \{r \mid r \in D, N \in I_r, Z_{N,r}(x_f) \neq \oslash\}$
  for each r in *receivers*:
    send f-message to r with $x_{fr} = Z_{N,r}(x_f)$
  update $list_{event}$ with t_{nd} and *select*
  $t_l = t$
  $t_n = min\{t_{nd} \mid d \in D\}$

Thus the simulator associated to a DEVS model receiving an external event x ($x = x_h \in X^h$), and supposed to present a faulty behavior, receives in fact two types of messages. In a first step a x-message (x_s, t) implying the activation of the external transition for the healthy part of the simulation, followed in a second step by a f-message $(\oslash, t)$ implying the execution of the external fault transition function for the faulty part of the simulation. If the model receives an external faulty event x_f ($x_f \in X^f$), the associated simulator will only receive a f-message (x_f, t) from its parent coordinator. Thus it is the nature of the external events that allows the distinction on simulation paths.

3 The Fault Simulation Environment

3.1 General Architecture

Concurrent fault simulation is approximately 20 years old and is one of the only existing applications of the concurrent simulation. However discrete event simulation has been employed in many applications and CCS can virtually be applied to each of them. Figure 2 (a) extracted from [9] illustrates the relationships between the CCS simulation kernel and applications. We can see that this kernel implies the use of a mandatory modeling interface for a given application. However CCS allows the simulation of models created independently from their execution (concurrent) simulation environment. Thus CCS permits simplicity in writing simulation models and generality in how they are used.

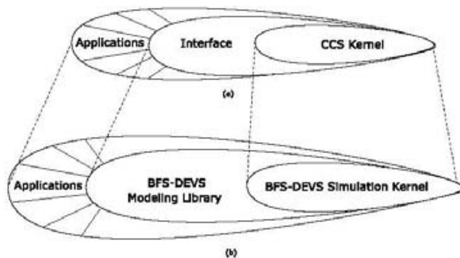


Figure 2: BFS-DEVS Positioning

To define the software interface, we propose the BFS-DEVS formalism allowing:

- Concurrently modeling and simulating faults of discrete event systems issued from one application.
- Distinguishing between the model of the system and the simulation kernel.
- Implementing the model and the concurrent algorithms inside the simulation kernel using an object-oriented approach.
- Plugging fault models on demand.
- An easy updating of models in a hierarchical and modular fashion.

Figure 2 (b) shows that an application will be represented by a network of BFS-DEVS components (atomic and/or coupled model) constituting the Library. This network is directly simulable by the simulation kernel. This modeling is the interface between the physical applications and the BFS-DEVS simulation kernel based on the CCS. Thus to simulate a given application the modeler would design a domain specific BFS-DEVS library without thinking about the simulation part.

3.2 BFS-DEVS Library and Fault Models

Each application owns a BFS-DEVS components library defined by the user. As shown in Figure 3, this library is composed by models (atomic and/or coupled) used to compose the final model to be simulated. Its construct implies the knowledge of a fault model mandatory for the definition of the faulty behaviors. The behavioral rules associated to the data and structure graphs of the application allow defining the models, their interfaces and behaviors.

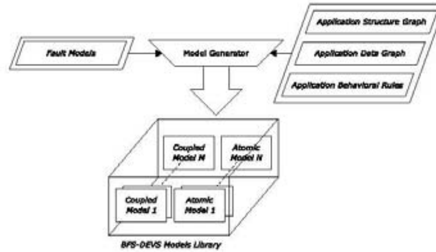


Figure 3: Building of a BFS-DEVS Library

Determining the control and data structures for a given application is not straightforward, and implies a strong collaboration with a studied domain specialist. Recurrent types of these models will constitute the BFS-DEVS library. Building a library for a given domain is not easy, but is the only "delicate" phase of our approach.

Inside the BFS-DEVS formalism, a new transition function δ_{fault} is introduced that permits to specify a faulty behavior for the model. This faulty behavior is obviously related to the fault type the model is affected by. Each fault type able to affect a BFS-DEVS model will change its state using the δ_{fault} function.

3.3 BFS-DEVS Simulation Kernel

A concurrent fault simulation inside a BFS-DEVS network (aka. BFS-DEVS simulation) is a healthy simulation of this network along with the concurrent execution of faulty simulations induced by multiple propagated fault lists.

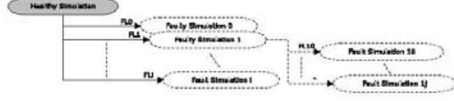


Figure 4: Concurrent Fault Simulation Scheme

Figure 4 shows how $i \in \mathbb{N}^+$ faulty simulations are generated starting from one unique healthy simulation. These faulty simulations result in faults contained in the FL_i propagated lists. Moreover a faulty simulation induced by the fault list FL_i can imply $j \in \mathbb{N}^+$ faulty simulations propagating the lists FL_{ij} such as $FL_{ij} \subset FL_i \forall i, j \in \mathbb{N}^+$. This propagation of faulty simulations by fault lists cutting up and re-orientation can be performed until one fault list is found, i.e. $FL_{ij} \geq 1$.

4 Experiments and Results: Application on Behavioral Digital Domain

Our approach has been applied in the digital circuits domain. We designed a BFS-DEVS library that allow us to model and simulate a circuit described in the VHDL hardware description language (see [10]). This transformation is fully described in [11].

In order to show the validity of our approach, we chose a sub-set of the VHDL ITC'99 benchmarks of [12] shown in Table 1. Columns in this table sum up the information at the Behavior level in terms of number of VHDL lines of code #Lines , number of processes #Proc , number of assignment #Assig and conditional statements a #Cond and number of signals #S and variables #V.

Figure 5 shows the architecture of the developed prototype. A parser allows obtaining the BFS-DEVS network representation file. This network is simulated to generate the fault list obtained from a test sequence provided by a *pseudo-random test pattern generator*. This generator provides several pseudo-random test vectors arranged in independent sequences, and each

Benchmark	#Lines	#Proc	#Assig	#Cond	#S	#V
b01	110	1	35	12	6	1
b02	70	1	19	7	4	1
b03	141	1	56	14	7	14
b04	102	1	40	12	7	13
b05	310	3	99	46	19	5
b06	128	1	50	11	8	1
b07	92	1	33	9	4	6
b08	89	1	22	8	8	4
b09	103	1	34	8	7	2
b10	167	1	74	19	13	8

Table 1: Benchmark Characteristics

Benchmark	#Vect	#Total Faults	#Detected Faults	FC [%]
b01	117	79	73	92,94
b02	209	48	42	87,50
b03	707	130	108	83,07
b04	103	105	89	84,76
b05	542	251	61	24.30
b06	200	95	78	82,10
b07	400	76	66	86,84
b08	373	64	63	98,43
b09	395	68	57	83,82
b10	4913	163	143	87,67

Table 2: VHDL BFS-DEVS Simulation Results

one of these sequences contains the "reset" signal with the value "1". The fault Model used is similar to [13] and is based on the bit and conditional coverage metrics.

Calculus of the fault coverage FC is obtained by the number of detected faults on the total number of faults given by the parser. Results are reported in Table 2 and show the number of detected faults along with the associated fault coverage.

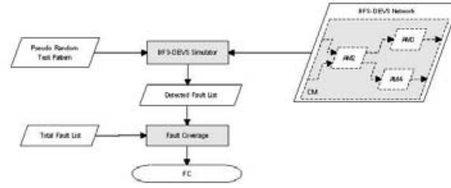


Figure 5: General prototype architecture

Experiments show the validity of the BFS-DEVS formalism and prototype presented in this paper. We can indeed determine the faults effects on the VHDL instructions of the behavioral descriptions. The use of a pseudo-random test pattern generator allows obtaining of significant fault coverages to show the validity of our approach. However the length of the test pattern is sufficiently large to detect the most easily observable faults.

5 Conclusion and Perspectives

We presented in this paper the BFS-DEVS formalism for the simulation of discrete event systems behavioral defaults in a simple and efficient fashion. We saw that this formalism is based on the CCS with MLP algorithms to simulate many input patterns against many faulty scenarios inside the BFS-DEVS network.

The proposed simulation environment is homogeneous and allows simply and generically integrating the domain-specific fault model. The design of a BFS-DEVS component library following the behavioral rules of a system is sufficient for the modeling and concurrent simulation of the defaults that can appear in the network. This simulation is also automatic and transparent for the user.

We validated this approach in the domain of behavioral faults for digital circuits. The fault coverages we obtained on the ITC'99 benchmarks are very satisfying.

More generally we have several perspectives concerning this research essentially about analysis and performance aspects. First distributed computing is more and more used in the discrete event simulation domain under the name of PDES (Parallel Discrete Event Simulation). We believe that our approach would take advantage of this technology and we plan to develop a Distributed BFS-DEVS based simulator. Second we think that this work can help testing and debugging classical software programs, and that integrating the CSS (Concurrent Simulation for Software) domain metrics would be easy to perform in our architecture.

References

[1] B. P. Zeigler, Theory of Modeling and Simulation, Academic Press, 1976.

[2] M. Larsson, Behavioral and structural model based approaches to discrete diagnosis, Ph.D. thesis, linkping University, Sweden (1999).

[3] N. Giambiasi, J. Santucci, A. Courbis, Test pattern generation for behavioral descriptions in VHDL, in: Proceedings of the Euro-VHDL Conference, 1991.

[4] J. Santucci, A. Courbis, N. Giambiasi, Behavioral testing of digital circuits, Journal of Microelectronic System Integration 1 (1).

[5] E. Kofman, N. Giambiasi, S. Junco, FDEVS: A general DEVS-based formalism for fault modeling and simulation, in: Proceedings of the European Simulation Symposium, 2000.

[6] B. P. Zeigler, H. Praehofer, T. G. Kim, Theory of Modeling and Simulation, Second Edition, Academic Press, 2000.

[7] B. P. Zeigler, An introduction to set theory, Tech. rep., aCIMS Laboratory, University of Arizona (2003).

[8] A. C. Chow, B. P. Zeigler, Abstract simulator for the parallel DEVS formalism, in: S. Editions (Ed.), Proceedings of AIS94, 1994.

[9] E. Ulrich, V. Agrawal, J. Arabian, Concurrent and Comparative Discrete Event Simulation, Kluwer Academic publisher, 1994.

[10] P. J. Ashenden, The designer guide to VHDL, Morgan Kaufmann Publishers, 2001.

[11] L. Capocchi, F. Bernardi, D. Federici, P. Bisgambiglia, Transformation of VHDL descriptions into DEVS models for fault modeling and simulation, in: Proceedings of the IEEE Systems, Man and Cybernetics Conference, 2003, pp. 1205–1211.

[12] High time for high-level test generation, 1999, pp. 1112–1119, panel at the IEEE International Test Conference.

[13] G. S. Fulvio Corno, Matteo Sonza Reorda, RT-level fault simulation techniques based on simulation command scripts, in: Proceedings of the XV Conference on Design of Circuits and Integrated Systems, 2000, pp. 825–830.

4 - QUATRIÈME PUBLICATION

2. L. Capocchi, F. Bernardi, D. Federici, P. Bisgambiglia, "BFS-DEVS: A General DEVS-Based Formalism For Behavioral Fault Simulation", Elsevier Simulation Pratice and Theory, Vol. 14, Issue 7, pp 945-970, 2006.

ELSEVIER

Available online at www.sciencedirect.com
ScienceDirect

Simulation Modelling Practice and Theory 14 (2006) 945–970

SIMULATION MODELLING PRACTICE AND THEORY

www.elsevier.com/locate/simpat

BFS-DEVS: A general DEVS-based formalism for behavioral fault simulation

Laurent Capocchi *, Fabrice Bernardi, Dominique Federici, Paul-Antoine Bisgambiglia

University of Corsica, SPE Laboratory, UMR CNRS 6134, 20250 Corte, France

Received 27 January 2005; received in revised form 19 May 2006; accepted 19 May 2006
Available online 12 July 2006

Abstract

Discrete event modeling allows designing an easy-to-handle and reusable representation of a system but, in its classical form, only permits one simulation at a time for a system. Concurrent and Comparative Simulation (CCS) with Multi-List Propagation (MLP) appears to be an adapted solution, by providing a way to perform several simulations in a single execution run. Concurrent Fault Simulation (CFS) has been one of the first applications of the CCS. The main obstacles to a wide use of this technique are the high complexity of the concurrent simulation algorithms, along with the difficulty to integrate them in a simulation kernel. We focus in this paper on the CFS with MLP of systems described in the new BFS-DEVS formalism, which is an evolution of the original DEVS simulator that integrates the CCS algorithm. The application is performed in the behavioral digital domain of systems described in the VHDL language.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Discrete event simulation; Concurrent and Comparative Simulation; Fault simulation; DEVS; VHDL

Author's personal copy

1. Introduction

Over the last 40 years discrete event simulation has begun to replace physical experimentation, as modeling and simulation offer efficient alternatives to expensive and complex physical experiments. Discrete event modeling allows designing an easy-to-handle and reusable representation of a system. However, classical discrete event simulation only permits one simulation at a time for a system. This solution can appear to be very time consuming when many successive simulations are required for the complete experiment, especially in terms of result analysis and observability. In order to escape from these limitations the Concurrent and Comparative Simulation (CCS) with Multi-List Propagation (MLP) appears to be an adapted solution, by providing a way to perform several simulations or other tasks in a single execution run.

* Corresponding author. Tel.: +33 495450208.
E-mail addresses: capocchi@univ-corse.fr (L. Capocchi), bernardi@univ-corse.fr (F. Bernardi), federici@univ-corse.fr (D. Federici), bisgambi@univ-corse.fr (P.-A. Bisgambiglia).

1569-190X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2006.05.002

Concurrent Fault Simulation (CFS) to simulate faults (mainly inside digital systems) has been one of the first applications of the CCS. The main obstacles to a wide use of this technique are the high complexity of the concurrent simulation algorithms, along with the difficulty to integrate them in a simulation kernel. Essentially for these reasons, fault simulation is not widely used in the domain of discrete event systems, whereas analysis of faulty behavior of discrete event systems could take profit from the established knowledge of digital domain.

We focus in this paper on the CFS with MLP of systems described in the BFS-DEVS formalism (Behavioral Fault Simulation for Discrete EVent system Specification) based on the DEVS formalism introduced by Zeigler in the late 1970's in [1]. DEVS provides a modular and modeling approach, and automatically defines a simulator directly from a model using formal elements and algorithms. It also allows the integration of concurrent simulation algorithms in a simple, evolutive and transparent fashion for the system modeler.

Discrete event simulation has been used in hundreds of different domains (graph analysis, economical studies, symbolic simulation, etc.) and CCS can be virtually used in each of them. However, even if CFS is approximately 20 years old, it is the only real existing application of CCS, and is not or not much used in the discrete event domain. Indeed, for many years, fault simulation has been an essential basic tool of CAD (Computer Aided Design) systems for digital circuits (see [2 4]), but has never been used in the discrete event domain where it could help to analyze faulty behaviors. Kofman et al. [5] proposed a first approach for fault simulation of systems described using DEVS but did not formally integrate the concurrent algorithms. We present in this paper the BFS-DEVS formalism integrating the CFS with MLP algorithms in its kernel, and allowing the modeler to specify the faulty behavior of a system using a new faulty transition function.

Many fault models (see [6 8]) and concurrent fault simulators for digital circuits (see [9 11]), essentially occurring at the gate level, have been implemented in past years. The increasing complexity of integrated circuits has led to the development of test tools occurring at higher abstraction levels than the gate level (see [12,13,8]). In order to sooner operate in the integrated circuits conception phase, commercial behavioral fault simulators such as HYPERFAULT or TURBOFAULT have been developed, allowing tests to be performed on circuits described in Hardware Description Languages (HDL). However, these simulators use heterogeneous simulation environments occurring on various description levels or do not allow performing the CFS with MLP. In this paper, we show that the proposed BFS-DEVS formalism permits the modeling of a behavioral HDL description control and data graphs to perform a CFS based on the MLP technique.

The main objective of our work consists in defining a behavioral concurrent fault simulator implementing the BFS-DEVS formalism. We modify for that the original DEVS formalism by introducing a new transition function allowing the modeler to describe the faulty behavior of a system. The BFS-DEVS simulator kernel is an evolution of the original DEVS simulator kernel that integrates the CCS algorithms. These lasts are based on fault lists propagated and directed by propagation rules inside BFS-DEVS models. Validation of our approach is performed on behavioral fault simulation of digital circuits described in high-level description languages.

This article is organized as follows. In a first section, we present a state of the art of the CCS domain. A second section is devoted to the presentation of the original DEVS and the new BFS-DEVS formalisms. We show in this section how DEVS is modified in order to integrate the concurrent simulation algorithm. We present in a third section the fault simulation environment along with the essential concepts of BFS-DEVS that are the component library and the fault model. The next two section show how BFS-DEVS is applied in the behavioral digital domain of systems described in the VHDL (VHSIC High-level Description Language, described for instance in [14]) language with some experiments and results. Finally, the last section concludes this paper and provides some perspectives of work.

2. Related work

2.1. Concurrent and comparative simulation

As claimed by Ulrich et al. [15], "based on the discrete event simulation, the serial simulation of single experiments is a widely used alternative for physical experimentation. For example, building a model and

serial simulation of a model is often superior to building, analyzing and testing physical prototypes of engineering designs. However, usually many similar experiments must be simulated, and each experiment requires manual work."

Since the early 1970's, many digital systems fault simulation methods have been simultaneously developed to become superior to the serial simulation: this include parallel, deductive and concurrent simulation approaches (respectively, proposed by Seshu and coworkers [16 18]). Many fault simulators have been implemented like DECSIM, MOZART, CREATOR, COSMOS or MOTIS, and these methods allow gains in generality, speed, observability and methodological power. Moreover, they can be used beyond digital circuits and fault simulation.

CCS is an algorithmic method that applies, and is limited, to systems simulated using discrete events. Its speed increases with the experiments number and similarity: results of a CCS execution are proportional to the number of simulated experiments. This method is parallel/concurrent and minimizes the manual work since it is more systematic, exhaustive and statistical than the serial approach. Moreover, its scope is greater because it permits the comparative experimentation of thousands of experiments.

Serial simulation involves a lot of manual work. Therefore initial experiments receive the maximum amount of user's attention, and later are often neglected and not simulated. Results arise in an arbitrary order in which experiments are simulated. For CCS no arbitrariness exists, all experiments are simulated, results appear in a race-like style, in time order and in parallel, and simultaneous results appear simultaneously. Thus, all results can be analyzed in a comparative/statistical fashion at any time.

CCS is similar to a multi-processor simulation with one processor by experiment. However, there is no need in a parallel computer and costly communication between processors are avoided. CCS obtains a similar efficiency to a parallel computer using only one processor and an equivalent number of virtual processors. In contrast with typical hardware, it is a precise time synchronous method. Being a software solution method, it is more general and flexible than hardware.

Other major properties of CCS are

(1) CCS is a general and precise method as it permits the simulation of complex sub-systems such as memories. This is essentially due to the MLP technique, which allows propagating several object lists into the system in a same time.
(2) CCS is fast in several ways. It minimizes the development time because it aggregates experiments into one simulation run, avoiding interruptions and manual work between successive serial experiments. It also minimizes the CPU time because it is based on similarity and on the unique initialization of the simulations.
(3) Observation, relative to the observation of the serial simulation, is easy to perform because experiments are performed in parallel. Based on experiment signatures handling and on an artificial experiment average, observation is largely automatic. Signatures may contain deterministic and statistical information about an experiment, its size (relative to a reference experiment) and a statistical distance (a similarity measure) starting from average experiments. Signatures can determine similarities between experiments allowing the elimination of non-informative ones.
(4) Modeling and simulation are tightly related and modeling requires the testing of sub-models. An optimal testing method is a systematic sub-models simulation, which is naturally and easily done with CCS.
(5) Many applications using CCS are available, and one particularly interesting is the Concurrent Software Simulation (CSS, used in [19]). CSS simulates variant executions of a computer program, and is useful for testing/debugging most kinds of computer programs, finding bugs more quickly and exhaustively than with non-simulating tools.
(6) Based on automatic observation, a CCS can be automatically controlled. Deletion or addition of experiments and run terminations can be performed automatically. While serial experiment requires much manual work, CCS is a more automatic, systematic, error-free method.
(7) Many scientific fields require similar physical experiments in parallel like biology, chemistry, meteorology, etc., but they are too costly due to the needed parallel resources. However, when discrete event simulation is a substitute for physical experimentation, then CCS is usually an alternative to serial simulation.

2.2. Concurrent fault simulation

Fault simulation for digital networks has been the first application of concurrent simulation and is widely used nowadays. Gate level simulation has been the first (and is today the most) implemented application in commercial tools (Mentor or Synopsys), while networks represented at other levels are less easily managed. For instance the DECSIM simulator described in [20] can simulate faults at four logic levels (switch, gate, register, and behavioral), but requires five distinct sub-systems to do this: the four logic levels plus an additional level to manage the information flow between them. The CREATOR simulator described in [21] reduces in a significant way the number of needed CCS sub-systems using MLP and other generalizations.

Originally concurrent fault simulation has not be defined to speed up the serial fault simulation. However, it appeared to be very powerful and the real experience speedups are typically above 1000:1. Fault simulation is characterized by the need to simulate thousands of faulty experiments. This is usually still impossible due to storage and CPU time limitations, but CSS runs with 10,000 to 50,000 are very efficient. However, simulating so many experiments is normally neither necessary nor desirable. For most preliminary experiments, it is sufficient to simulate approximately 1000 faulty experiments.

The most important works in the fault simulation domain are based on the MLP technique. Interesting approaches such as hierarchical concurrent fault simulations are presented by Lo et al. [22], but the most difficult for fault simulation appears to be fault detection and diagnosis, in conjunction with the execution of diagnostic programs as described by Breuer and Parker [23].

3. Modeling specifications

3.1. DEVS formalism

Introduced by Zeigler in the early 1970's and completed in [24], DEVS is a set-theoretic formalism that provides a mean of modeling discrete event systems in a hierarchical and modular fashion. Using this formalism, modeling can be easily performed by decomposing a large system into smaller component models with coupling specifications between them. DEVS defines two kinds of models: atomic and coupled models.

An atomic model is a basic model with specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. Coupled models tell how to couple several component models together to form a new model. This kind of model can be employed as a component in a larger coupled model, thus leading to the construction of complex models in a hierarchical fashion.

3.1.1. DEVS modeling

Fig. 1 shows a hierarchical structure of a coupled model CM_0 containing two atomic models AM_0, AM_1 and one coupled model CM_1. The *closer under coupling* property allows the inclusion of CM_1 inside CM_0.

As in general system theory (see [25]), a DEVS atomic model contains a set of states and transition functions triggered by the simulator. It is represented by the following structure:

$AM = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle$

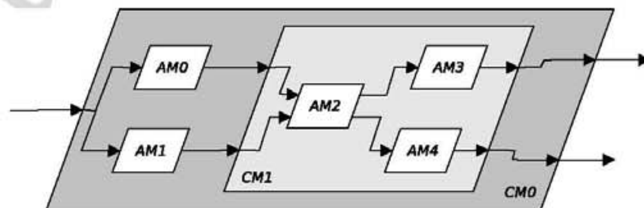


Fig. 1. DEVS model couplings example.

Author's personal copy

where

- $X: \{(p,v)|(p \in \text{input ports}, v \in X_p)\}$ is the input ports set and values for the reception of external events.
- $Y: \{(p,v)|(p \in \text{output ports}, v \in Y_p)\}$ is the output ports set and values for the emission of events.
- S: is the internal sequential states set.
- $\delta_{\text{int}}: S \to S$ is the internal transition function that will bring the system to the next state after the time returned by the time advance function.
- $\delta_{\text{ext}}: Q \times X \to S$ is the external transition function that will schedule state changes in reaction to an input event.
- $\lambda: S \to Y$ is the output function that will generate external events just before the internal transition takes place.
- $ta: S \to \mathbb{R}^{+}_{\infty}$ is the time advance function, that will give the life time of the current state (returns the time to the next internal transition).

We can sketch a dynamic interpretation of these definitions:

- $Q = \{(s,e)|s \in S, 0 < e < ta(s)\}$ is the total state set.
- e is the elapsed time since last transition, and s the partial set of states for the duration of $ta(s)$ if no external event occur.
- δ_{int}: the model being in a state s at t_i, it will go into s', $s' = \delta_{\text{int}}(s)$, if no external events occurs before $t_i + ta(s)$.
- δ_{ext}: when an external event occurs, the model being in the state s since the elapsed time e goes in s'. The next state depends on the elapsed time in the present state. At every state change, e is resetted to 0.
- λ: the output function is executed before an internal transition, and before emitting an output event the model remains in a transient state.
- A state with an infinite life time is a passive state (steady state), else, it is an active state (transient state). If the state s is passive, the model can evolve only with an input event occurrence.

A DEVS coupled model CM is a structure:

$$CM = \langle X, Y, D, \{M_d \in D\}, EIC, EOC, IC \rangle$$

where

- X is the input ports set for the reception of external events.
- Y is the output ports set for the emission of external events.
- D is the components set (coupled or basic models).
- M_d is the DEVS model for each $d \in D$.
- EIC is the input links set, that connects the inputs of the coupled model to one or more of the inputs of the components that it contains.
- EOC is the output links set, that connects the outputs of one or more of the contained components to the output of the coupled model.
- IC is the set of internal links, that connects the output ports of the components to the input ports of the components in the coupled models.

In a coupled model, an output port from a model $M_d \in D$ can be connected to the input of another $M_d \in D$ but cannot be connected directly to itself.

3.1.2. DEVS simulation

As described in [26], DEVS allows a hierarchical and modular modeling of decomposable discrete event systems thanks to these specifications. One the main advantages of DEVS is that the simulator is directly and automatically extracted from the model. This DEVS simulator uses the modeling hierarchy and associates a *simulator* component to each atomic model and a *coordinator* component to each coupled model. This

Author's personal copy

association allows the control of the behavior of each atomic model and the synchronization of the whole set of models. As shown in Fig. 2, the DEVS simulator is hierarchical and is composed by a tree composed by simulators and coordinators.

As shown in Fig. 3, each simulation cycle consists in three steps: first, the coordinator searches among its subordinates (coordinator and/or simulator) the smallest activation time of their next event. Next, the coordinator collects outputs from these imminent models (i.e., models presenting an activation time for the next event equal to the minimal time) and sends outputs on the models coupled to the imminent models. Finally, the coordinator calls each of the imminent models to execute their transition function.

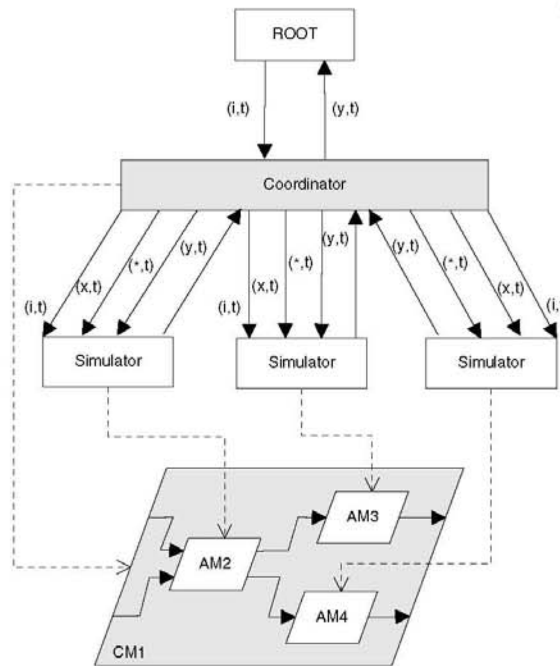


Fig. 2. Simulator hierarchy.

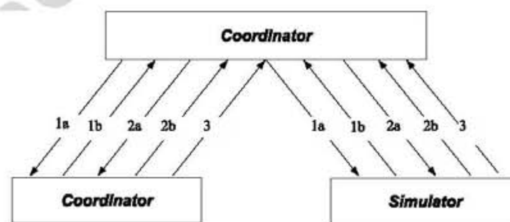


Fig. 3. DEVS simulator message exchange.

Communication between elements drawn in Fig. 3 is performed using four types of messages. The initialization message (i, t) is used to perform an initial temporal synchronization between all actors of the system (messages 1a and 1b on the figure). The internal transition message $(*, t)$ implies the execution of an internal event (message 2a), while the output message (y, t) allows the output communication to the parent elements, and results from the reception of a $(*, t)$ message (message 2b). Finally, the external transition message (x, t) allows the execution of an external event (message 3). All these messages are more precisely discussed in the next sections.

For [27] DEVS brings many advantages. First it uses a hierarchical and modular modeling approach allowing the description of the multiple levels of a system as shown by Zeigler and Vahie [28]. It also supports the definition of models defined in different paradigms, allowing the definition of multi-components, each defined using a different technique (see [29]). DEVS allows any existing model to be easily extended, and formal definitions of coupled or atomic models can be modified. Moreover, each model can be associated with an Experimental Framework (a set of DEVS atomic models that can be coupled with other DEVS models, designing an environment for conducting experiments) used as a testing module. This approach improves testing facilities.

3.2. BFS-DEVS formalism

3.2.1. BFS-DEVS modeling

The BFS-DEVS formalism introduced in this paper allows the specification and the simulation of faulty discrete event models. A BFS-DEVS model is defined by a classical DEVS structure with the following additional features first introduced by Kofman et al. [5]:

- Transition and/or output functions are modified if behavioral faults are present in the DEVS model.
- Faulty behavior of DEVS models is specified by a new faulty external transition function δ_{fault}.

Consider a DEVS atomic model whose *healthy* behavior can be represented by the following DEVS structure:

$$AM^h = \langle X^h, S^h, Y^h, \delta^h_{\text{int}}, \delta^h_{\text{ext}}, \lambda^h, ta^h \rangle$$

We define the following structure to take the *faulty* behavior into account:

$$AM^f = \langle X, S, Y, \mathbf{F}, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\textbf{fault}}, \lambda, ta \rangle$$

where,

- $X = X^h \cup X^f$ is the set of input values, with X^f representing a set of new faulty values.
- $S = S^h \cup S^f$ is the set of state values, and S^f is the set of new faulty states that the model can present due to the presence of a faulty input event.
- $Y = Y^h \cup Y^f$ is the set of output values, with Y^f representing the set of new faulty values that the output ports can take when a faulty input event occurs.
- $F = \{F_1, F_2, F_3\} \cup \emptyset$ is the faults set, with $\{F1, F2, F3\}$ is the fault model described further.
- $\delta_{\text{int}} : S \times F \rightarrow S$ is the internal transition function modified by the presence of faults and presents the restriction $\delta_{\text{int}}(s, \emptyset) = \delta^h_{\text{int}}(s)$.
- $\delta_{\text{ext}} : Q \times X \times F \rightarrow S$ is the external transition function modified by the presence of faults and verifies $\delta_{\text{ext}}(s, e, x, \emptyset) = \delta^h_{\text{ext}}(s, e, x)$ if $x \in X^h$.
- $\delta_{\text{fault}} : Q \times X \times F \rightarrow S$ is the faulty external transition function providing the faulty behavior when a faulty event occurs.
- $\lambda : X \times F \rightarrow S$ is the output function, which verifies $\lambda(s, \emptyset) = \lambda^h(s)$.
- $ta : S \times F \rightarrow \mathbb{R}^+_\infty$ is the time advance function,f with the restriction $ta(s, \emptyset) = ta^h(s)$.

BFS-DEVS specifications are very similar with the original DEVS ones. The difference is that any time a faulty event x_f ($x_f \in X^f$) occurs, the new faulty state is calculated by the faulty external transition function

Author's personal copy

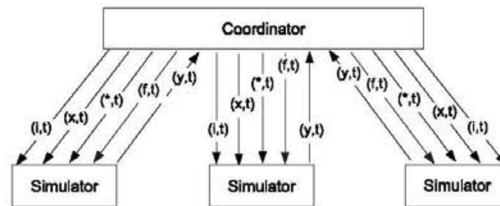


Fig. 4. Fault simulator architecture.

δ_{fault}. If the healthy event x_h ($x_h \in X^h$) occurs, then the new healthy state is calculated by the healthy external transition function δ^h_{ext}.

We note that BFS-DEVS models coupling is not changed. But if the fault model contains a structural fault type, this coupling becomes different from the original DEVS coupling. Moreover, we can prove that the property of closure under coupling allowing the hierarchical composition of model is preserved.

3.2.2. BFS-DEVS simulation

Communication between components is achieved in the original DEVS formalism using four types of messages. To distinguish a faulty behavior inside a simulation, we introduce a new message type message that activates the model as soon as a *fault event* x_f is present on one of its ports at a given time t. Thus we are able to activate the faulty behavior of a component using such a message representing a faulty external event as shown in Fig. 4.

A coordinator, using the X set, is then able to send two types of external messages to its imminent children, a "x-message" for a healthy simulation, and a "f-message" for a faulty simulation.

Thanks to this distinction we can concurrently perform the faulty and healthy simulations. Indeed the faulty simulation can only be performed if the healthy simulation has already been achieved, because values of the first are used as references for the second. Consequently, the healthy simulation precedes the faulty one.

Thus the simulator associated to a DEVS model receiving an external event x ($x = x_h \in X^h$), and supposed to present a faulty behavior, receives in fact two types of messages. In a first step a x-message (x_s, t) implying the activation of the external transition for the healthy part of the simulation, followed in a second step by a f-message $(\emptyset, t)$ implying the execution of the external fault transition function for the faulty part of the simulation. If the model receives an external faulty event x_f ($x_f \in X^f$), the associated simulator will only receive a f-message (x_f, t) from its parent coordinator. Thus it is the nature of the external events that allows the distinction on simulation paths.

4. The fault simulation environment

4.1. General architecture

Concurrent fault simulation is approximately 20 years old and is one of the only existing applications of the concurrent simulation. However, discrete event simulation has been employed in many applications and CCS can virtually be applied to each of them. Fig. 5(a) extracted from [15] illustrates the relationships between the CCS simulation kernel and applications. We can see that this kernel implies the use of a mandatory modeling interface for a given application. However, CCS allows the simulation of models created independently from their execution (concurrent) simulation environment. Thus CCS permits simplicity in writing simulation models and generality in how they are used.

To define the software interface, we propose the BFS-DEVS formalism allowing:

- Concurrently modeling and simulating faults of discrete event systems issued from one application.
- Distinguishing between the model of the system and the simulation kernel.

Author's personal copy

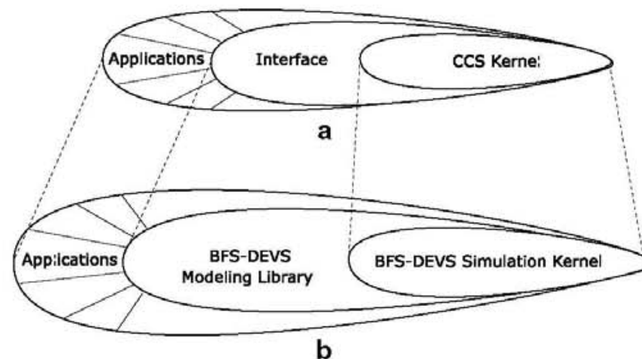


Fig. 5. BFS-DEVS positioning.

- Implementing the model and the concurrent algorithms inside the simulation kernel using an object-oriented approach.
- Plugging fault models on demand.
- An easy updating of models in a hierarchical and modular fashion.

Fig. 5(b) shows that an application will be represented by a network of BFS-DEVS components (atomic and/or coupled model) constituting the Library. This network is directly simulable by the simulation kernel. This modeling is the interface between the physical applications and the BFS-DEVS simulation kernel based on the CCS. Thus to simulate a given application the modeler would design a domain specific BFS-DEVS library without thinking about the simulation part.

4.2. BFS-DEVS library and fault models

Each application owns a BFS-DEVS components library defined by the user. As shown in Fig. 6, this library is composed by models (atomic and/or coupled) used to compose the final model to be simulated. Its construct implies the knowledge of a fault model mandatory for the definition of the faulty behaviors. The behavioral rules associated to the data and structure graphs of the application allow defining the models, their interfaces and behaviors.

Determining the control and data structures for a given application is not straightforward, and implies a strong collaboration with a studied domain specialist. This determination follows from the discrete event modeling and its result is translated into a network of models. Recurrent types of these models will constitute the BFS-DEVS library. Building a library for a given domain is not easy, but is the only "delicate" phase of our approach.

Before going further in our development we need to introduce some notions. First a *behavioral fault* is described by Kofman et al. [5] as a modification of the transition functions, and/or of the output functions of the DEVS models. Thus a fault influences the behavior (or the state) of an affected DEVS model and can lead to a faulty behavior we previously described. Inside the BFS-DEVS formalism, a new transition function δ_{fault} is introduced that permits to specify a faulty behavior for the model. This faulty behavior is obviously related to the fault type the model is affected by. Each fault type able to affect a BFS-DEVS model will change its state using the δ_{fault} function.

Second a *fault model* is the set of type of faults that can modify the behavior of a BFS-DEVS model. It strongly depends on the nature of the modeled and simulated physical system. Indeed behavioral fault types inside the BFS-DEVS network must be the most representative of the behavioral defaults of the physical system. The fault model strongly depends on the control and data structures of the application and is integrated into the behavior and the structure of each BFS-DEVS model.

Author's personal

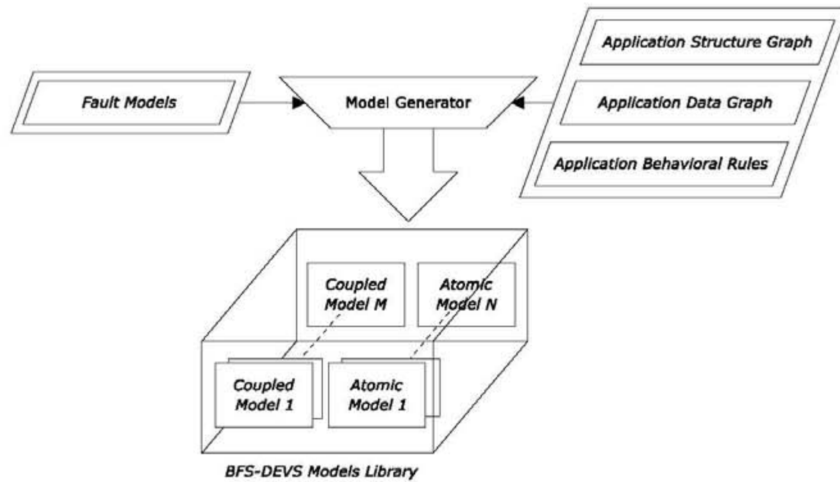


Fig. 6. Building of a BFS-DEVS library.

Another important notion is the *fault signature* which is the consequence of a fault on the behavior of the affected BFS-DEVS models. Each fault owns a signature in which the faulty behavior of all affected BFS-DEVS models is stored.

We call a *detectable fault* on a BFS-DEVS model a significant fault on a model, i.e., whose signature presents a faulty behavior distinct from the healthy behavior at the end of the simulation.

Finally, a *locally observable fault* is a fault that modifies the behavior of the model during the simulation. A fault is locally observable as long as the behavior is faulty.

4.3. BFS-DEVS simulation kernel

A concurrent fault simulation inside a BFS-DEVS network (aka. BFS-DEVS simulation) is a healthy simulation of this network along with the concurrent execution of faulty simulations induced by multiple propagated fault lists.

Fig. 7 shows how $i \in \mathbb{N}^+$ faulty simulations are generated starting from one unique healthy simulation. These faulty simulations result in faults contained in the FL_i propagated lists. Moreover, a faulty simulation induced by the fault list FL_i can imply $j \in \mathbb{N}^+$ faulty simulations propagating the lists FL_{ij} such as $FL_{ij} \subset FL_i$ $\forall i, j \in \mathbb{N}^+$. This propagation of faulty simulations by fault lists cutting up and re-orientation can be performed until one fault list is found, i.e., $FL_{ij} \geqslant 1$.

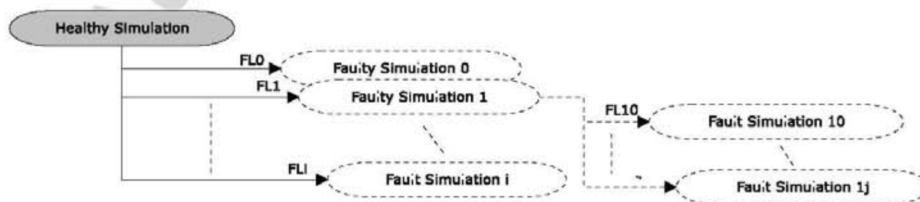


Fig. 7. Concurrent fault simulation scheme.

Before going further in our development we need to introduce some very essential notions. A *propagated fault list* groups all faults presenting same signatures. Indeed different faults belonging to a same fault model can imply the same faulty behavior on a BFS-DEVS network. This is one of the main reasons why we can detect different faults in a single BFS-DEVS simulation run. During the simulation the fault list propagation is performed using messages carried on the models ports.

A *healthy simulation* (aka. *reference simulation*) is a BFS-DEVS simulation in which no behavioral fault is present. A fault list is built for each model met on this path.

We use the *single fault hypothesis* defining that only one fault is present in a model at a time and that the effect of the fault is present during the whole simulation. However, we can still simulate many faults inside one BFS-DEVS model using the concurrent simulation based on fault lists propagation.

A *faulty simulation* (aka. *concurrent simulation*) is a BFS-DEVS simulation in which at least one model is affected by a unique fault list, which can modify the final result of the healthy simulation. However, a BFS-DEVS network can be composed by models presenting many possible faulty behaviors (models presenting many output ports). Consequently, faults present inside the propagated list can imply different faulty behaviors, leading to a fault list presenting different signatures. Since faults present in the propagated list must have unique signatures (cf. the propagated fault list definition), the initial list is decomposed in sub-lists with the same signature. As a faulty simulation propagates a unique fault list, previous sub-lists are propagated on new faulty concurrent simulations. Thus, a faulty "parent" simulation can give birth to "child" faulty simulations propagating sub-lists of the initial fault list. A BFS-DEVS concurrent fault simulation can be then viewed as several faulty simulations concurrencing the healthy one. This property allows us to simulate many faults inside a BFS-DEVS network while respecting the single fault hypothesis.

A *healthy* (resp. *faulty*) *message* is an object allowing the communication and the activation of BFS-DEVS models for a healthy simulation (resp. faulty simulation). It also allows the propagation of the fault lists, the simulation time and several other information inside the network. A faulty message containing an initial fault list can be divided and can give birth to messages containing sub-lists of the initial list.

A *healthy path* (aka. *reference path*) is the sequence of (atomic and/or coupled) BFS-DEVS models activated by a healthy simulation of the network.

A *faulty path* (aka. *concurrent path*) is the sequence of (atomic and/or coupled) BFS-DEVS models activated by a faulty simulation. This path is concurrent to and obviously different from a healthy path. A faulty path can be divided in some concurrent faulty paths.

In order to illustrate the previous notions, consider Fig. 8. Let us assume that the healthy simulation activates the models AM_0, AM_1, AM_4, AM_5, AM_6, AM_{15} and AM_{16} (in grey) thus defining the healthy path (bold lines). Each one of these models builds its own fault list: FL0 for AM_0, FL1 for AM_1, etc. In our approach, faulty simulations are executed concurrently to the healthy simulation, and activate the other atomic models (in white) to give birth to faulty paths (plain lines). Let us also consider that the dashed models and paths on the figure are not simulated because FL1_1 is not divided.

At the beginning of the simulation FL1 is propagated using a faulty message to build the fault signatures for each activated model (for instance AM_7 and AM_8). However, this list is cut up and is reoriented all along the main faulty path. Indeed all faults belonging to FL1 and evaluated by AM_2 do not imply the same faulty behavior for this last model. Consequently, the main fault list FL1 is divided into two sub-lists FL1_0 and FL1_1. These lists are composed by faults implying the same behavior for AM_2. This division ensures that it exists only one unique signature inside each propagated fault.

As soon as faulty paths join (AM_{14} and AM_{15}) propagated faults are merged and their signatures analyzed to build the detected fault list. Finally, the main fault list FL0 will be analyzed at the end of the healthy simulation, i.e., when all faulty behaviors have been simulated.

4.4. General object oriented implementation

We propose in this section an object oriented implementation based on the previously described notions, and that follows the original DEVS object architecture. The *DEVS* package shown in Fig. 9 is composed by two main abstract classes called *AtomicDEVS* and *CoupledDEVS* inheriting the properties of the other abstract class *BaseDEVS*. These two classes allow the implementation of the atomic and coupled DEVS

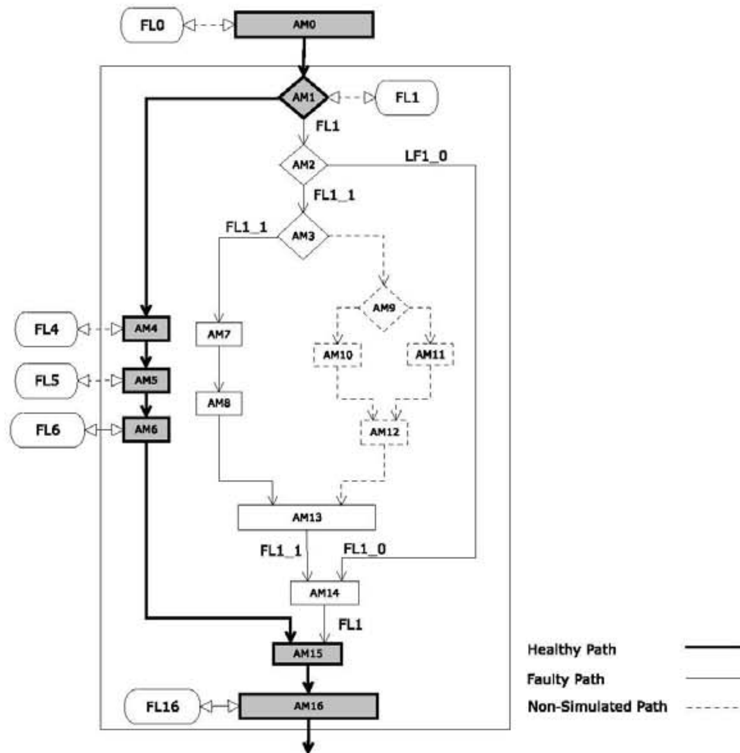


Fig. 8. Concurrent fault simulation of a sample BFS-DEVS network.

models and are associated with a *Port* class used to represent their ports. The whole structure of the system is then represented by an interconnection of instances of these classes. In the DEVS formalism atomic models represent the behavior of the model and coupled models the structure. Thus, to be generic, we define two new classes *DomainBehavior* and *DomainStructure* in an other package called *Domain*.

Classes belonging to this package are the basis of our architecture. The main class *Master* is a specialization of a *CoupledModel* class and shall represent the highest-level coupled model of the whole model. Only one instance of this class is allowed which will be statically accessed by the *DomainBehavior*, *DomainStructure* and *SDB* classes. The δ_{fault} function is defined inside the *DomainBehavior*. *DomainStructure* allows describing the structural elements used by the *Master* class, and presents methods like *check_rules*() allowing the validation of the structure using a domain parser. The *SDB* (Symbolic DataBase) class is introduced to facilitate access to the studied domain data structure. This class, used by a *Master* is used to store objects handled by the domain, and can also be instancied only one time.

As shown in Fig. 9, the *XDomainBehavior*, *XSDB* and *XDomainStructure* are specific to the studied domain, and for each domain these three classes need to be implemented.

To sum up, it will exist only one *Master* instance that will contain one instance of *XSDB* as a static attribute. BFS-DEVS atomic models will be represented by specialization instances of *XDomainBehavior*, and coupled models by class specialization instances of *XDomainStructure*. The set composed by these classes will define the BFS-DEVS library for the *X* domain.

We can note that we use a Singleton Design Pattern defined by Gamma et al. [30] to ensure the unicity of the Master and XSDB instances that do not appear on the diagram for simplicity reasons.

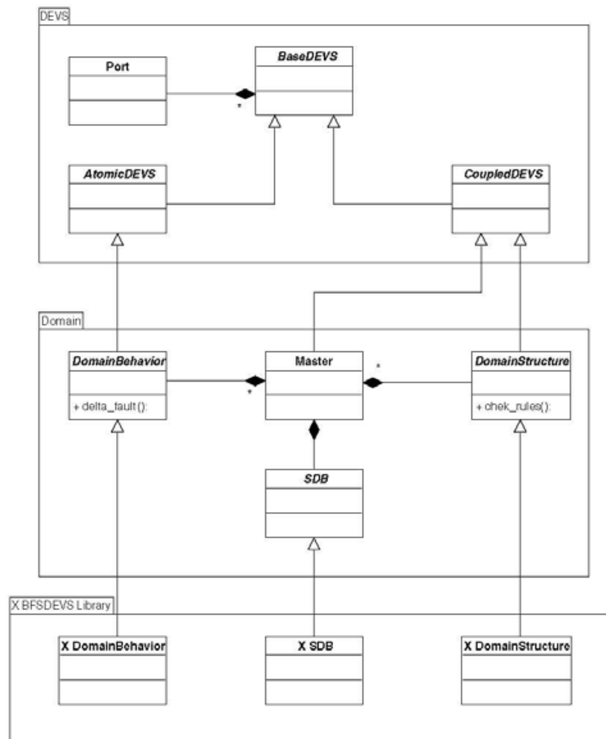


Fig. 9. Diagram of the main classes of the architecture.

5. Application on behavioral digital domain

5.1. Basics of the VHDL language

Hardware Description Languages are used in the digital circuits domain. These languages are classified following three types: structural, register transfer level (RTL) and behavioral. The VHDL language was created in the 1980's and has been normalized in 1987 (IEEE 1076). It allows capturing the circuit behavior using an algorithmic representation.

The VHDL model represents two aspects of the process: computation and control. Computation represented by *data flow* part of the model can be modeled by data flow graph. Control represented by *control flow* part of the model dictates the partial ordering of data operations and can be modeled by control flow graph. Both models are used for test generation.

A VHDL description is composed by two parts. First the description of the circuit interface with its environment called *entity* and composed by the signals list, their direction, their nature, etc.; Second the description of the circuit called *architecture* that can contain three types of description, the *structural* description of the sub-circuits interconnections, the *functional* description of the used boolean functions, and the *behavioral* description which is represented by algorithms allowing simulating the behavior of the system.

We only focus in this paper on the behavioral description, which is the most used today. This description is composed by a collection of parallel *processes* containing sequential statements (assignment, conditions, etc.), functions, procedures, signals and variables only used for calculus. *Signals* are used to allow processes to

communicate, and are stored in their respective *sensitive lists*. Each signal owns a *pilot* which is a 3-columns table containing the current and future values and the assignment time of the future value. A signal can be "multi-source", i.e., can be affected in several processes. But in order to avoid assignment conflicts we consider that a signal is "uni-source".

Fig. 10 presents a multi-process behavioral description of a 8 bits register containing three processes STROBE, ENABLE and OUTPUT along with their sensitive lists (STRB), (DS1, NDS2) and (REG, ENBLD).

5.2. Behavioral fault model

In the past years several high-level behavioral fault models associated with some fault simulation techniques have been proposed by Thaker and coworkers [8,6,31,4,7]. However, Goloubeva et al. [32] shows that none of them formally establishes the relationships existing between the high-level and gate level fault coverages, but considers that the closest fault model is constituted by the bit and conditional coverages. As in [13] the fault model we propose in this paper is based on techniques of test of software.

This fault model is mainly based on *stuck-at* signals and variables present in a VHDL behavioral description. To take into account faults inside the description control graph, we consider also the *conditional branch stuck-at* faults. Finally, we consider the *jump of assignment statements* fault since it can allow us analyzing the redundant code in a VHDL description. We can note that this fault model is evolutive and has not been conceived to validate a specific metric of the HDL. Moreover, it can use several fault models through the use of the fault transition function.

```
entity Register  is
Port(DI: in bit_vector(1 to 8);
     STRB, DS1, NDS2: in bit;
     DO: out bit_vector(1 to 8));
end Register
architecture Arch of Register is
  signal REG : bit_vector(1 to 8);
  signal ENBLD : bit;
begin

STROBE: process(STRB)
  begin
     if (STRB='1') then
       REG<=DI;
     end if;
end process STROBE;

ENABLE: process(DS1, NDS2)
  begin
    ENBLD<=DS1 and not NDS2;
end process ENABLE;

OUTPUT: process(REG, ENBLD)
  begin
    if (ENBLD='1') then
      DO<=REG;
    else
      DO<=''11111111'';
    end if;
end process OUTPUT;
end Arch;
```

Fig. 10. VHDL description of a 8 bits register.

As in [13] we adopt the single fault hypothesis. We use three behavioral fault types acting on the VHDL instructions present in the description to be simulated:

- *F*1 Fault Type: These faults are *value stuck-at (signals or variables) faults* present in the description. For instance, consider the assignment instruction $E : S1 <= (S2\ and\ S3)\ or\ S4$, where the type of $S1$, $S2$, $S3$ and $S4$ is *T:bit*. Evaluation of E gives a healthy value v_h to $S1$, and all $S1_i$ faults of type *F*1 on $S1$ implying a faulty value v_f will be the stuck-at values of its definition domain minus the healthy value: $\{S1_i | v_f \in T - \{v_h\}\}$. In this example if $v_h =$ '1', stuck-at faults on $S1$ will be stored in the list $L_{S1} = [S1_0]$. In the same way faults of type *F*1 implying a faulty evaluation of E will be determined following their healthy current values. Some of the fault models used by Corno et al. [6] convert VHDL objects such as "integer" into the equivalent bit vector according to synthesis conventions, and each element is considered separately as a bit. We can also perform this kind of translation, but the type *F*1 appears sufficient to show the validity of our approach. This type corresponds to the bit coverage described in [32].
- *F*2 Fault Type: These faults are *branch stuck-at faults* (stuck-at-true, stuck-at-false) of the conditional instructions. When a conditional instruction is evaluated, a healthy conditional branch is chosen between a finite set of possibilities. The stuck-at of other branches different from the healthy branch constitutes the *F*2 type. This type corresponds to the conditional coverage described in [32].
- *F*3 Fault Type: These faults prevent an assignment instruction to be evaluated. This fault type allows the deletion of redundant assignment instructions inside a VHDL description (see [33]).

5.3. VHDL to BFS-DEVS translation

The VHDL to BFS-DEVS translation allows describing the control and data flows of a VHDL description as a BFS-DEVS network. This translation brings several advantages. Indeed it provides an homogeneous and modular integration of the simulation algorithms inside the δ_{fault} transition functions. It also permits a simplification of the analysis and observability treatments and an easy adaptation of the behavioral (or even structural) fault models.

In VHDL a fault will be considered as a *bad evaluation of an instruction*. In BFS-DEVS a fault inside an atomic model representing a VHDL instruction modifies its behavior implying a faulty simulation. The three fault types previously described become in this case:

- The *F*1 Fault Type, a faulty state transition of a model.
- The *F*2 Fault Type, the selection of a faulty output port of a model.
- The *F*3 Fault Type, a non-activation of a model.

We use for this paper behavioral VHDL descriptions using sequential instructions composed by one or several *process* instructions. Each of them is composed by sequential instructions such as assignment, conditional ("if-then-else, case"), and loop statements. We show in [34] that these descriptions can be represented using four BFS-DEVS atomic models:

An Assignment atomic model represents a VHDL assignment statement. This association allows us giving a faulty or healthy behavior to the instruction. When such a model is found in the network a fault list of the *F*1 type is built.
A Conditional atomic model represents a VHDL conditional statement. This association allows us developing the functions used to determine the *F*1 and *F*2 faults.
A Junction atomic model is associated with end-code statements such as "endif, endcase". Algorithms for fault local observability are defined inside these components.
A Process coupled model is associated with the VHDL "process" statement. It is a coupled model for the structuring of the previous atomic models.

Author's personal copy

Two new atomic models are introduced:

A Generator atomic model that generates events for the activation of the components inside the BFS-DEVS network. A model of this type is mandatory for the simulation.
A ProcessEngine atomic model that manages the processes synchronization and the fault list propagation. This model participates also in the observability analysis, and in the detected fault list update.

These six BFS-DEVS components represent the BFS-DEVS library associated with the behavioral VHDL domain as shown in Fig. 11. The fault model (F_1, F_2, F_3) is integrated inside the transition functions of the components. The VHDL to BFS-DEVS translation is performed using a parser that uses the library, provides the BFS-DEVS network and the total fault list.

Fig. 12 shows the generated BFS-DEVS network starting from the 8 bits register description presented in Fig. 10.

5.4. VHDL BFS-DEVS simulation

In order to explain the VHDL BFS-DEVS simulation we need to introduce or complete some notions.

A *fault signature* corresponds to the trace $T_{F_i} = \{((S|V)_j, P_{(S|V)_j}) \mid i, j \in \mathbb{N}\}$ of the F_i fault on a healthy or faulty path. It is represented by the set of couples $((S|V)_{j\in\mathbb{N}}, P_{(S|V)_j})$ where: S_j *(or* V_i) is the influenced signal (or the influenced variable) by F_i, and $P_{(S|V)_j} = [v_{cf}, v_{ff}, time]$ is the pilot of the current v_{cf} and future v_{ff} faulty values of S_i (or V_i) by F_i at time cycle *time*. A fault can be associated to several signatures.

The *locally observable fault list* L_O is a list of faults implying a different result from the healthy simulation. This result is locally observable on signals and variables of the description.

The *detected fault list* L_D is a list of the detected faults visible on output signals during the concurrent fault simulation. We have $L_O \cap L_D = \emptyset$.

A *healthy simulation* is the parallel execution of the coupled models with no atomic model presenting a faulty behavior.

A *faulty simulation* is the parallel execution of the coupled models with one or more atomic models presenting a faulty behavior.

An *active process* is the activation of a coupled model for a healthy or faulty simulation.

An *inactive process* is the activation of a coupled model for a faulty simulation. However, if no atomic model presents a faulty behavior, the coupled model is not activated to speed the simulation up. A *healthy*

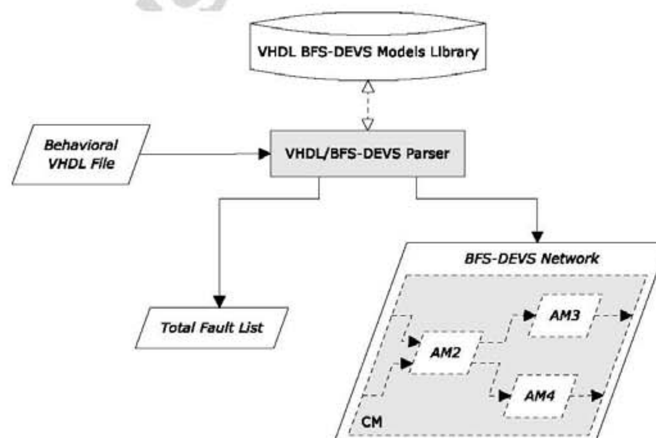


Fig. 11. VHDL to BFS-DEVS translation.

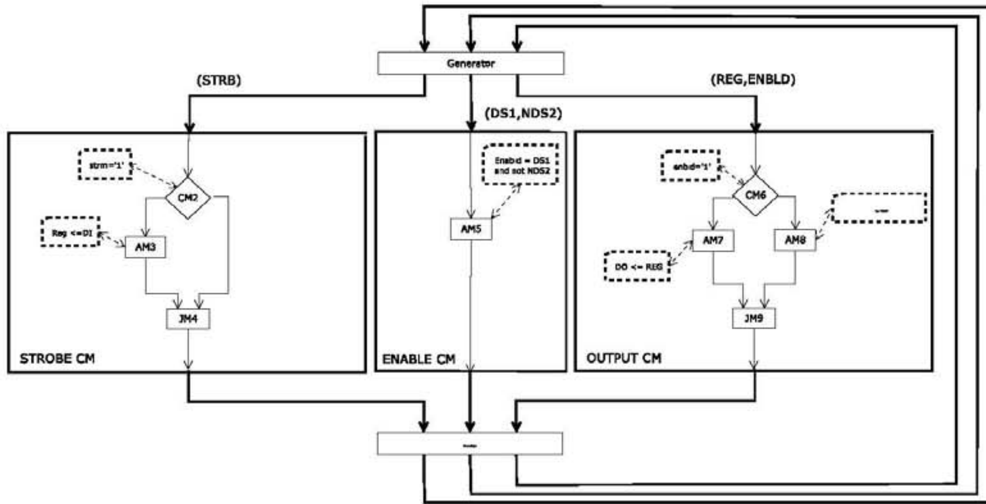


Fig. 12. BFS-DEVS network of 8 bit register.

path is composed by the set of BFS-DEVS models activated during a healthy simulation run. However, because of the concurrency a healthy path can obviously be divided in several independent healthy sub-paths.

A *faulty path* is composed by the set of BFS-DEVS models activated during a faulty simulation run.

A *reference database* is an object storing the VHDL constants and pilots of all signals, variables belonging to this description. This database is accessed by the four basic atomic models previously seen.

We can now define the BFS-DEVS concurrent fault simulation of VHDL systems represented by $N > 0$ coupled models, as the parallel execution of $x < N$ coupled models for a healthy simulation, and of $1 \leqslant i \leqslant N - x$ coupled models for i faulty simulations induced by L_i propagated fault lists inside the network.

To illustrate these definitions consider a BFS-DEVS network of a multi-processes behavioral description. During a simulation run, $n > 0$ coupled models are active for n healthy simulations, and $m > 0$ coupled models are active for m faulty simulations. We also consider that each coupled model has several paths, implied by the presence of conditional atomic models. Healthy simulations are executed inside the n coupled models, thus defining the n healthy paths. In order to highlight faults implying the faulty paths, faulty simulations are concurrently executed with the healthy simulation in the $n + m$ coupled models.

We can see on Fig. 13 that faulty paths derive directly (resp. indirectly) from the $n = 2$ healthy paths inside the $CM_{1,2}$ coupled models (resp. the CM_3 coupled model).

Fig. 13(a) shows the healthy paths (bold) resulting from a healthy simulation of the two coupled models CM_1 and CM_2 corresponding to two active processes AP_1 and AP_2. Since IP_1 is inactive CM_3 is not activated.

Fig. 13(b) shows the faulty paths (dashed) generated by the concurrent faulty simulations. We see that faulty paths inside the coupled models CM_1 and CM_2 are directly concurrent to the healthy paths. On the other side faulty paths inside CM_3 are indirectly concurrent to the healthy paths. Moreover, the propagation of the fault list $\#FL_3 \geqslant 4$ of the inactive CM_3 by cut up allows the simulation of the faults contained in the lists $\#FL_{30} \geqslant 1$, $\#FL_{31} \geqslant 2$, $\#FL_{310} \geqslant 1$ and $\#FL_{311} \geqslant 1$.

We can note that even if IP_1 is inactive for the VHDL simulation, CM_3 would become active whether a simulation of faults in FL_3 is activated.

5.5. VHDL simulation cycles

VHDL simulation is managed through events on the communication signals between processes. The simulation cycle called *symbolic cycle* (or *delta cycle*) is divided in three distinct phases (grey in Fig. 14):

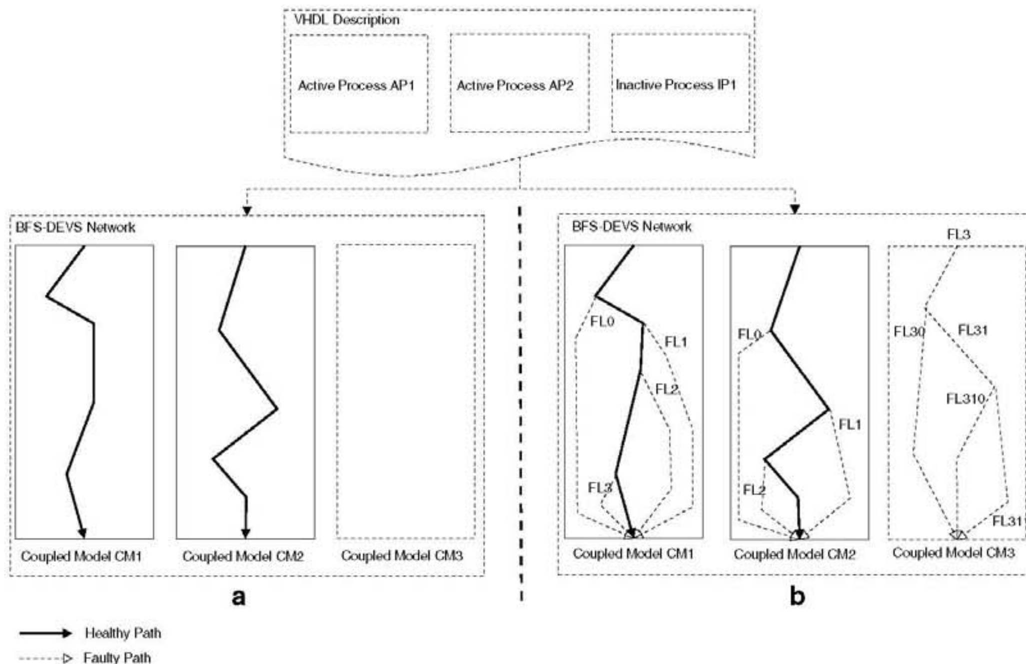


Fig. 13. Schematic view of a BFS-DEVS concurrent fault simulation.

the *EXECUTION* phase corresponding to the parallel execution of the processes, the *UPDATE* phase corresponding to the update of the signals pilot values attributes, and the *ANALYSIS* phase establishing the list of the processes to be activated following programmed events on sensitive signals.

As shown in Fig. 14 the concurrent fault simulation relies on these three phases and complete them with two new phases relative to the concurrent simulation technique used: the fault *OBSERVABILITY* phase to build the L_D list, and the *FAULT COVERAGE CALCULUS.*

Thus the simulation scheme can be divided in five phases:

(1) The *CONCURRENT FAULT SIMULATION* phase that consists in a healthy simulation whose results are stored in a reference database, and in concurrent faulty simulations to construct or update L_O.
(2) The local *OBSERVABILITY* phase of the propagated faults occurring as soon as all processes have been simulated. This phase consists in comparing each fault signature included in L_O with the results of the healthy simulation. This comparison will allow highlighting the locally observable faults on signals or variables of the VHDL description.
(3) The *UPDATE* phase is applied to the signals pilots values present in the reference database. If no process has been activated for a healthy simulation, no update is performed.
(4) The future process activity *ANALYSIS* that presents a supplementary procedure to manage the fault influence inside the sensitive signals. Indeed if a sensitive signal is affected by faults, these lasts can give birth to faulty simulations on the processes the signal belongs to.
(5) The *FAULT COVERAGE CALCULUS* only occurs when the future process activity analysis is negative. A fault belonging to L_O will be detectable if it implies, in its signature, a current faulty value different from the current pilot value of an output signal. All detected faults are transferred from L_O to L_D. The fault coverage is given by

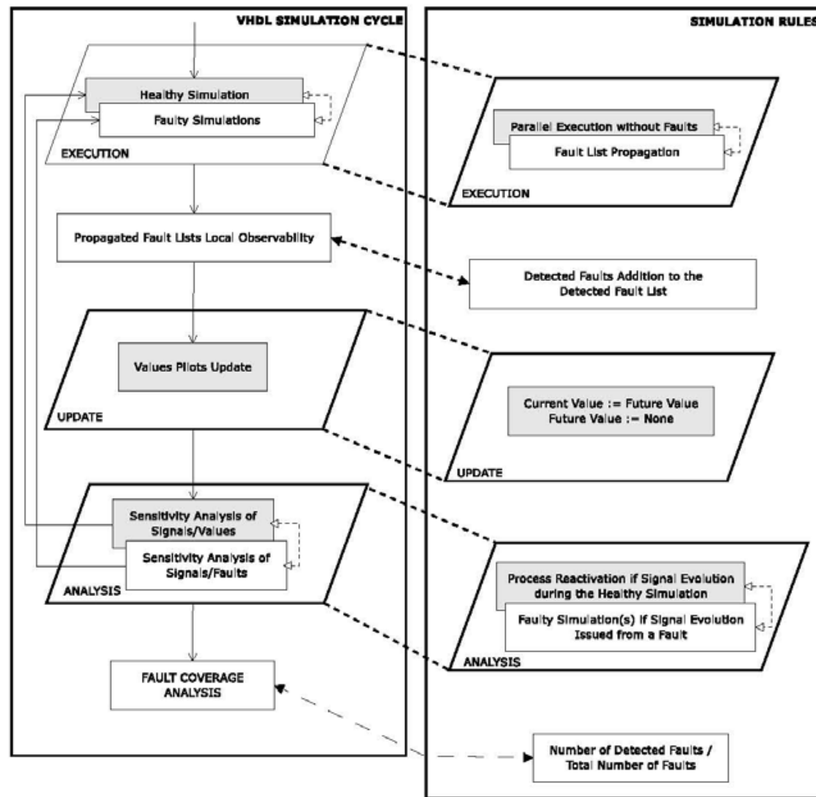


Fig. 14. Concurrent fault simulation scheme.

Author's

$$C(\%) = \frac{\text{number of detected faults}}{\text{total fault number}} * 100.$$

5.6. Fault list propagation

The BFS-DEVS concurrent fault simulation can be composed by several faulty simulations driving fault lists to obtain the signatures. Propagation of this list is achieved using a *cutting up* and an *initial list reorientation*. Fault list propagation can be divided in two parts: the *intra-process* (list propagation inside coupled models) and the *inter-process* (list propagation between coupled models) propagations. To describe these two propagations, we define:

- A VHDL description presenting N processes: $P_{0<n\leqslant N}$.
- The sensitive signals lists for processes activation: $L_n = \{S_{k\in\mathbb{N}}\}$.
- The propagated fault lists: $FL_n = \{F_i \mid i, n \in \mathbb{N}\}$.
- The locally observable fault list: L_O.
- The signatures describing the fault propagation during the concurrent simulation: $\forall F_i \in \mathbb{N}, T_{F_i} = \{((S|V)_j, P_{(S|V)_j}) \mid j \in \mathbb{N}\}$.

5.6.1. Intra-process propagation

At the beginning of a VHDL simulation cycle (except for the initialization cycle) each process can present an activity depending on the sensitivity of the signal belonging to L_n.

If a VHDL process P_n is active (cf. Fig. 15(a)) the corresponding coupled model CM is also activated by a Generator atomic model in order to run the healthy simulation along with:

- The simulation of the faults F_i implying a non-activation of CM, and stored in an initial fault list built by the Generator using L_O (FL_O in Fig. 15). Observability of these faults depends on the result of the healthy simulation. Faults will only be appended to L_O at the end of the simulation by a ProcessEngine atomic model.
- The simulation of the faults F_i able to appear inside the Assignment and Conditional atomic models activated by the healthy simulation (bold in Fig. 15(a)). These faults are stored in lists FL_n (FL_1 for AM_1, FL_4 for AM_4, etc.). In the case where these lists come from an Assignment atomic model (AM_1, AM_4, AM_5, AM_6), they are directly appended to L_O, the list of locally observable faults. If they come from a Conditional atomic model (AM_{16}) they will be appended to L_O by the corresponding Junction model only when once having been simulated on the faulty paths they imply (plain line on the figure). Indeed these lists FL_n can give birth to other sub-lists FL_{nm} ($FL_1 = FL_{1_0} + FL_{1_1}$ on the figure). L_O is acting as a reference. On the one hand, because if a fault is highlighted on a Assignment atomic model and is already in the fault list, an update of the trace is performed, otherwise the fault is appended. On the other hand because if a fault is highlighted on a Conditional model and is already in the fault list, it will be copied and inserted in the list FL_n that will be propagated on the faulty paths. Observability of the contents of the list will be analyzed at the junction of the faulty paths.

If a VHDL process P_n is inactive (cf. Fig. 15(b)), the associated coupled model is activated by a Generator atomic model in order to simulate the faults $F_i \in FL_n$ that would have activate it. These faults are essentially

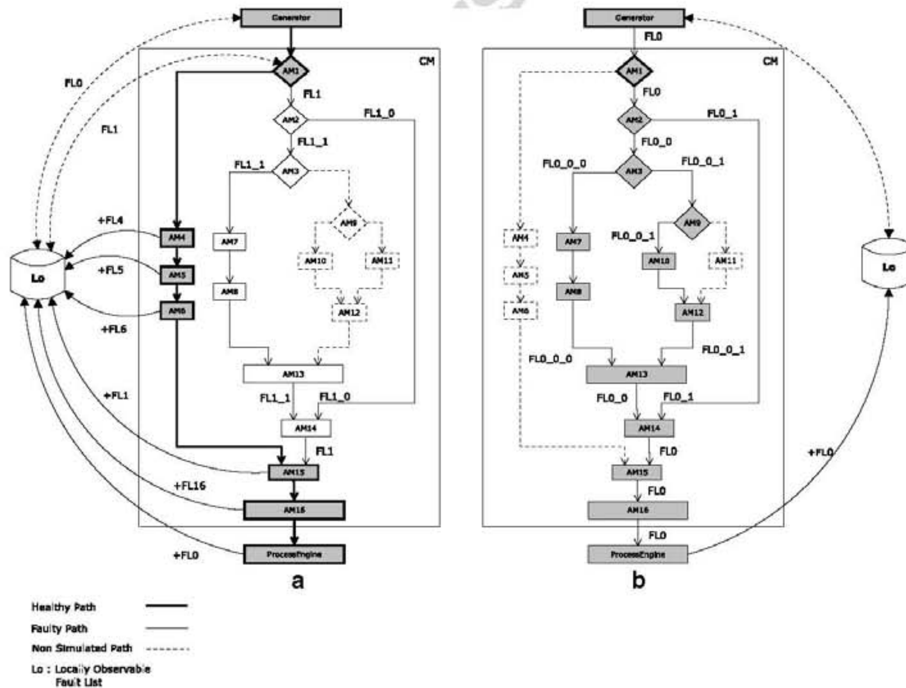


Fig. 15. Intra-process propagation. (a) Active process; (b) inactive process.

directed towards the signals of L_n. But contrary to the previous case, the faults F_i can present initial signatures during the list propagation. These signatures are issued from the reference list L_O and will be updated during the faulty simulation of P_n. All faults initially contained in FL_n cannot take the same path. In the example shown in Fig. 15(b), the initial fault list FL_1 is cut up in two sub-lists FL_{1_0} and FL_{1_1} such as $FL_1 = FL_{1_0} + FL_{1_1}$, since faults of FL_{1_0} do not have the same consequences than faults of FL_{1_1} in the AM_2 Conditional model. For the same reasons, FL_{0_0} is cut up in two sub-lists $FL_{0_0_0}$ and $FL_{0_0_1}$ such as $FL_{0_0} = FL_{0_0_0} + FL_{0_0_1}$.

Analysis of fault signatures is performed in the ProcessEngine model at the end of the faulty simulations. Every observable fault will be appended to L_O. If a fault is already present in the list a signature update is achieved. Following this update each fault presenting an empty signature is deleted from L_O.

5.6.2. Inter-process propagation

The VHDL language defines the *internal sensitive signals* in order to establish links and parallelization between the processes of a description. Thus faults which can be present on these signals are also linked with the processes. If a fault is locally observable on a sensitive signal it will be simulated inside one or several processes.

In order to establish the building rules for this kind of faults, it is necessary to recall the properties of the sensitive signals defined by VHDL:

(1) No (internal or input) sensitive signal can be affected in the process it activates (auto-activation). This property is illustrated in Fig. 16(a). This configuration can bring to a loop in the process activation.
(2) A (internal or input) sensitive signal can be affected in several processes. However, in order to avoid conflicts, a conflict resolution function must be implemented for such a signal. In our method, these functions are not taken into account. Thus we consider the configuration shown in Fig. 16(b).
(3) A sensitive signal can activate several processes. This property is illustrated in Fig. 16(c) and allows the parallelization of the processes activated by a same signal.

These properties allow us defining the following faults characteristics (Fig. 17):

(1) A fault can belong to several lists FL_n of the n inactive processes P_n. This characteristic comes from the property number 3 of the sensitive signals.
(2) A fault cannot imply different values for a same signal. This characteristic comes from the property number 2 of the sensitive signals.

We can now introduce the inter-process propagation rules for locally observable faults:

(1) If one fault belonging to L_O implies the faulty simulation of n different processes $P_{0\leqslant n<N}$, it will be duplicated and inserted in LF_n. At the end of the simulation run the fault becomes unique again with the fusion of the duplicated faults signatures (from the fault property number 1).

Author's personal copy

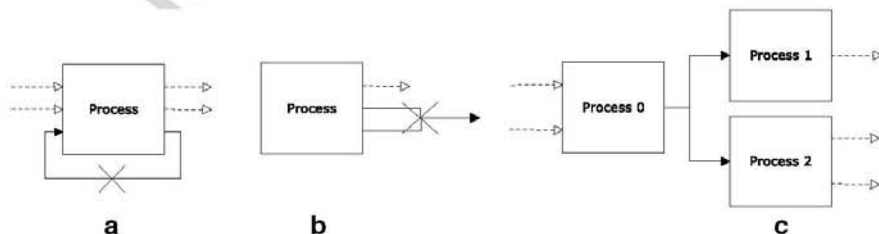


Fig. 16. Process auto-activation, assignment conflict and process multi-activation.

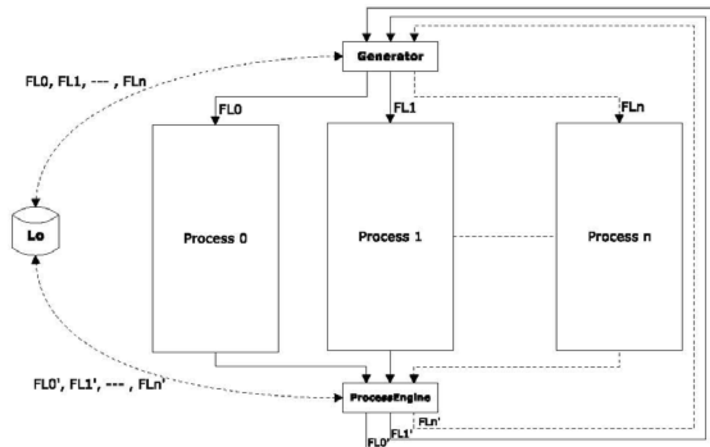


Fig. 17. Inter-process propagation.

(2) The analysis of the future processes activity is performed against the signatures of the faults in L_O. Every fault present in L_O, except those of $F1$ type on sensitive signals, can activate the associated processes.

We can find two origins for the fault lists FL_n. First if a model associated to a process is active for a physical cycle, these fault lists are created by the Generator. This last build these lists with reference to the locally observable fault list L_O. If a fault already belongs to L_O, it is copied and inserted in FL_n. It then will be simulated using the inter-process propagation and analyzed at the end of the simulation run inside the ProcessEngine model. If two faults belonging to different messages arrive at the ProcessEngine model in a same time, their signatures are merged in order to obtain only one signature for the fault.

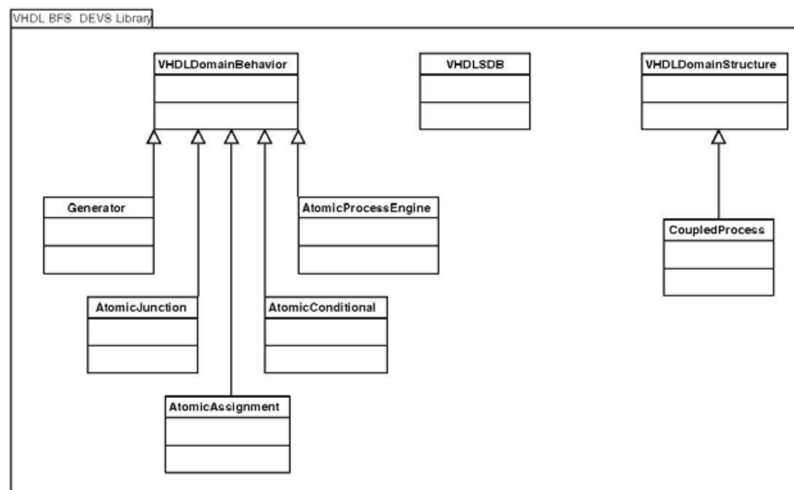


Fig. 18. Class diagram for the VHDL domain.

Second If a coupled model associated with a process is active for a symbolic cycle, the fault lists FL'_n are generated by the ProcessEngine model. Indeed if coupled model needs to be reactivated by a fault present in L_O and implying a variation of a signal of L_n, a faulty message is sent to the output port of the ProcessEngine model using the Generator. This last only transmits the message.

5.7. Object oriented implementation

Fig. 18 presents the object-oriented implementation we performed for this application of our formalism. We can see the four classes *AtomicJunction*, *AtomicAssignment*, *AtomicProcessEngine* and *CoupledProcess* implementing the VHDL instructions and specializing the classes *VHDLDomainBehavior* and *VHDLDomainStructure*. The three first cited classes implement their faulty behavior using a *delta_fault*() method.

The *VHDLSDB* class implements the *SDB* class and represents the data structure of the VHDL language. It is used to store pilots, attributes and update methods.

6. Experiments and results

The general CFS with MLP approach presented in this paper has been applied to digital systems described in the VHDL language using a BFS-DEVS simulator prototype and a library of BFS-DEVS VHDL components. This implementation derives from a simulation kernel conform to Zeigler's specifications. The whole prototype is composed by about 8000 lines of Python code.

In order to show the validity of our approach, we chose a sub-set of the VHDL ITC'99 benchmarks of [35] shown in Table 1. Columns in this table sum up the information at the RTL level in terms of number of VHDL lines of code, number of processes, number of assignment and conditional statements and number of signals and variables.

Fig. 19 shows the architecture of the developed prototype. A parser based on GENESI (see [36]) allows obtaining the BFS-DEVS network representation file. This network is simulated to generate the fault list obtained from a test sequence provided by a *pseudo-random test pattern generator*. This generator provides several pseudo-random test vectors arranged in independent sequences, and each one of these sequences contains the "reset" signal with the value "1".

Calculus of the fault coverage is obtained by the number of detected faults on the total number of faults given by the parser. Results are reported in Table 2 and show the number of detected faults along with the associated fault coverage.

Experiments show the validity of the BFS-DEVS formalism and prototype presented in this paper. We can indeed determine the faults effects on the VHDL instructions of the behavioral descriptions. The use of a pseudo-random test pattern generator allows obtaining of significant fault coverages to show the validity of our approach. However, the length of the test pattern is sufficiently large to detect the most easily observable faults.

The pseudo-random patterns testability can be anticipated by the analysis of the fault coverages shown in Table 2. We note that the b05 benchmark is random pattern-resistant because it presents low fault coverage.

Table 1
Benchmark characteristics

Benchmark	#Lines	#Proc	#Assignments	#Conditionals	#Signals	#Variables
b01	110	1	35	12	6	1
b02	70	1	19	7	4	1
b03	141	1	56	14	7	14
b04	102	1	40	12	7	13
b05	310	3	99	46	19	5
b06	128	1	50	11	8	1
b07	92	1	33	9	4	6
b08	89	1	22	8	8	4
b09	103	1	34	8	7	2
b10	167	1	74	19	13	8

Author's personal copy

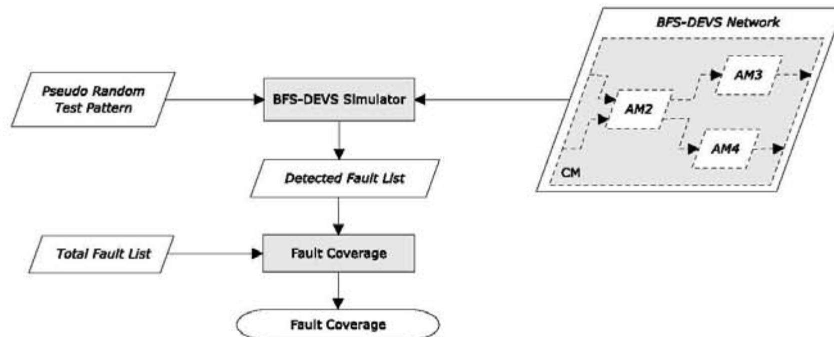


Fig. 19. General prototype architecture.

Table 2
VHDL BFS-DEVS simulation results

Benchmark	#Vect	#Total faults	#Detected faults	Fault coverage [%]
b01	117	79	73	92.94
b02	209	48	42	87.50
b03	707	130	108	83.07
b04	103	105	89	84.76
b05	542	251	61	24.30
b06	200	95	78	82.10
b07	400	76	66	86.84
b08	373	64	63	98.43
b09	395	68	57	83.82
b10	4913	163	143	87.67

Benchmarks b05, b03 and b10 are difficult to test because their fault coverages are obtained with a high number of test vectors. The fault model used in this test sequence is based on the *F*1, *F*2 and *F*3 fault types. Goloubeva et al. [32] shows that the fault model based only on the *F*1 and *F*2 types can be used for estimating the gate-level stuck-at fault coverage by reasoning on a behavioral model of the benchmarks. However, the majority of undetected faults on the benchmarks are of type *F*3. A way to improve the fault coverage would be to remove the *F*3 type from our global fault model. Considering the *F*3 type is however interesting since it allows removing redundant instructions in the VHDL code. Indeed an assignment model in which a *F*3 fault is not detected can be removed from the network. This implies a reduction of the number of lines of the description leading to a simulation speed-up.

7. Conclusion and perspectives

We presented in this paper the BFS-DEVS formalism for the simulation of discrete event systems behavioral defaults in a simple and efficient fashion. We saw that this formalism is based on the CCS with MLP algorithms to simulate many input patterns against many faulty scenarios inside the BFS-DEVS network.

The proposed simulation environment is homogeneous and allows simply and generically integrating the domain-specific fault model. The design of a BFS-DEVS component library following the behavioral rules of a system is sufficient for the modeling and concurrent simulation of the defaults that can appear in the network. This simulation is also automatic and transparent for the user.

The object architecture of our prototype is generic and extensible to several domains thanks to the abstract classes *DomainBehavior* and *DomainStructure*, respectively, used to describe the behavior and the structure of a new domain to be managed.

We validated this approach in the domain of behavioral faults for digital circuits. The fault model we used does not present an important correlation with the fault model at the gate level, but the genericity of our approach permits to specify some other fault models at lower abstraction levels. The fault coverages we obtained on the ITC'99 benchmarks are very satisfying.

In this digital domain the main perspective is to design an ATPG (Automatic Test Pattern Generator) to measure the effectiveness of the test patterns used to perform the fault simulation. We shall based this work on signature analysis and also on genetic algorithms for their proved efficiency.

More generally we have several perspectives concerning this research essentially about analysis and performance aspects. First distributed computing is more and more used in the discrete event simulation domain under the name of PDES (Parallel Discrete Event Simulation). We believe that our approach would take advantage of this technology and we plan to develop a Distributed BFS-DEVS based simulator. For instance when applied to the digital circuits domain, this distribution would allow simulating processes in parallel. Second, we think that this work can help testing and debugging classical software programs, and that integrating the CSS (Concurrent Simulation for Software) domain metrics (path coverage, statement coverage, branch coverage, etc.) would be easy to perform in our architecture.

Finally, we are currently applying this work to other domains in order to show its genericity of use. First applications are fire forest propagation and graph analysis.

References

[1] B.P. Zeigler, Theory of Modeling and Simulation, Academic Press, 1976.

[2] M. Larsson, Behavioral and structural model based approaches to discrete diagnosis, Ph.D. Thesis, Linkping University, Sweden, 1999.

[3] N. Giambiasi, J. Santucci, A. Courbis, Test pattern generation for behavioral descriptions in VHDL, in: Proceedings of the Euro-VHDL Conference, 1991.

[4] J. Santucci, A. Courbis, N. Giambiasi, Behavioral testing of digital circuits, Journal of Microelectronic System Integration 1 (1).

[5] E. Kofman, N. Giambiasi, S. Junco, FDEVS: A general DEVS-based formalism for fault modeling and simulation, in: Proceedings of the European Simulation Symposium, 2000.

[6] F. Corno, G. Cumani, M.S. Reorda, G. Squillero, An RT-level fault model with high gate level correlation, in: Proceedings of the IEEE International High Level Design Validation Workshop, 2000.

[7] G. Buonanno, L.F.F. Ferrandi, F. Fummi, D. Sciuto, How an "evolving" fault model improves the behavioral test generation, in: Proceedings of the IEEE Seventh Great Lakes Symposium on VLSI (GLS-VLSI '97), 1997.

[8] P.A. Thaker, V.D. Agrawal, M.E. Zaghloul, Register-transfer level fault modeling and test evaluation techniques for VLSI circuits, in: Proceedings of the IEEE International Test Conference, 2000.

[9] S. Gai, P. Montessoro, F. Somenzi, The performance of the concurrent fault simulation algorithms in MOZART, in: Proceedings of the Design Automation Conference, 1988, pp. 692–697.

[10] D. Machlin, D. Gross, S. Kadkade, E. Ulrich, Switch level concurrent fault simulation based on a general purpose list traversal mechanism, in: Proceedings of the International Test Conference, 1988, pp. 574–581.

[11] P. Montessoro, S. Gai, Creator: General and efficient multilevel concurrent fault simulation, in: Proceedings of the Design Automation Conference, 1991, pp. 160–163.

[12] A. Fin, F. Fummi, A VHDL error simulator for functional test generation, in: Design, Automation and Test in Europe (DATE '00), 2000.

[13] G.S. Fulvio Corno, Matteo Sonza Reorda, RT-level fault simulation techniques based on simulation command scripts, in: Proceedings of the XV Conference on Design of Circuits and Integrated Systems, 2000, pp. 825–830.

[14] P.J. Ashenden, The Designer Guide to VHDL, Morgan Kaufmann Publishers, 2001.

[15] E. Ulrich, V. Agrawal, J. Arabian, Concurrent and Comparative Discrete Event Simulation, Kluwer Academic publisher, 1994.

[16] S. Seshu, On a improved diagnosis program, IEEE Transactions on Electronic Computers 12 (1965) 76–79.

[17] D. Armstrong, A deductive method for simulating faults in logic circuits, IEEE Transactions on Computers 21 (1972) 464–471.

[18] M. Abramovici, M. Breuer, K. Kumar, Concurrent fault simulation and functional level modeling, in: Proceedings of the IEEE Design Automation Conference, 1977, pp. 128–137.

[19] S. Demba, E. Ulrich, K. Panetta, D. Giramma, Experiences with concurrent fault simulation of diagnostic programs, in: IEEE Transactions on CAD, vol. 9, 1990, pp. 621–628.

[20] M. Kearney, DECSIM: A multi-level simulation system for digital design, in: Proceedings of the International Conference on Computer Design, 1984, pp. 206–209.

[21] S. Gai, F. Somenzi, E. Ulrich, Advance in concurrent multilevel simulation, IEEE Transactions on CAD 6 (1987) 10006–10012.

[22] C.Y. Lo, H. Nham, A. Bose, Algorithms for an advanced fault simulation system in MOTIS, IEEE Transactions on CAD 6 (1987) 232–240.

[23] M.A. Breuer, A.C. Parker, Digital system simulation: current status and future trends, in: Proceedings of the IEEE Design Automation Conference, 1981, pp. 269–275.

[24] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, second ed., Academic Press, 2000.
[25] B.P. Zeigler, An introduction to set theory, Tech. Rep., aCIMS Laboratory, University of Arizona, 2003.
[26] A.C. Chow, B.P. Zeigler, Abstract simulator for the parallel DEVS formalism, in: S. Editions (Ed.), Proceedings of AIS94, 1994.
[27] G. Wainer, S. Daicz, A. Troccoli, Experiences in modeling and simulation of computer architectures in DEVS, Transactions of the Society for Computer Simulation 18 (4) (2001) 179–202.
[28] B.P. Zeigler, S. Vahie, DEVS formalism and methodology – unity of conception diversity of application, in: S. Editions (Ed.), Proceedings of the 1993 Winter Simulation Conference, 1993, pp. 573–579.
[29] B.P. Zeigler, DEVS theory of quantized systems, Tech. Rep., aCIMS Laboratory, University of Arizona, 2004.
[30] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 2002.
[31] F. Corno, P. Prinetto, M.S. Reorda, Testability analysis and ATPG on behavioral RT-level VHDL, in: Proceedings of the IEEE International Test Conference, 1997.
[32] O. Goloubeva, M.S. Reorda, M. Violante, Behavioral-level fault models comparison: an experimental approach, in: Proceedings of the IEEE Computer Aided Technologies in Applied Mathematics Conference, 2002.
[33] P. Bisgambiglia, D. Federici, J.-F. Santucci, Fault modeling and simulation at behavioral level, in: S. Editions (Ed.), Proceedings of LTAW01, 2001, pp. 45–50.
[34] L. Capocchi, F. Bernardi, D. Federici, P. Bisgambiglia, Transformation of VHDL descriptions into DEVS models for fault modeling and simulation, in: Proceedings of the IEEE Systems, Man and Cybernetics Conference, 2003, pp. 1205–1211.
[35] High time for high-level test generation, 1999, pp. 1112–1119, Panel at the IEEE International Test Conference.
[36] C. Paoli, M. Nivet, F. Bernardi, L. Capocchi, Simulation-based validation of VHDL descriptions using constraints logic programming, in: Proceedings of the 5th IEEE Workshop on RTL and High Level Testing (WRTLT'04), 2004, Osaka, Japan.

Author's personal copy

5 - Cinquième Publication

6. L. Capocchi, D. Federici, P.A. Bisgambiglia, H. Henao, G.A. Capolino, “A BFS-DEVS Approach for induction Generator Non Traditional Modelling”, Proceedings of the First International Symposium on Environment Identities and Mediterranean Area (IEEE ISEIM06), Corte-Ajaccio, France, July 2006.

A BFS-DEVS Approach for Induction Generator Non Traditional Modelling

L. Capocchi
University of Corsica
UMR CNRS 6134
Quartier Grossetti, BP 52
20250 Corte - FRANCE
capocchi@univ-corse.fr

D. Federici
University of Corsica
UMR CNRS 6134
Quartier Grossetti, BP 52
20250 Corte - FRANCE
federici@univ-corse.fr

P. A. Bisgambiglia
University of Corsica
UMR CNRS 6134
Quartier Grossetti, BP 52
20250 Corte - FRANCE
bisgambi@univ-corse.fr

H. Henao
University of Picardie
Department of Electrical Engineering
33, rue St Leu
80039 Amiens - FRANCE
Humberto.Henao@ieee.org

G. A. Capolino
University of Picardie
Department of Electrical Engineering
33, rue St Leu
80039 Amiens - FRANCE
Gerard.Capolino@ieee.org

Abstract— **The goal of this paper is to propose a behavioral modelling method of the electrical machine functioning, and more particularly of an induction generator machine. Our approach is based on VHDL-AMS (Very high speed integrated circuits Hardware Description Language for Analog and Mixed Signal) language and on the use of the Discrete EVent System specification: DEVS formalism. We show in this paper how, from the traditional circuit-oriented model, a simplified and realistic model of an induction generator machine can be defined. That model results from a transformation of those circuits described in VHDL-AMS language. It will allow to perform electrical default simulation for the induction machine diagnostics.**

I. Introduction

Nowadays, the models used to represent induction machine are based either on differential equations or on electrical circuits. The mathematical model allows a good representation of it but it involves non-linear effects leading to differential equations, which need a considerable resolution time. The schematical model of electrical circuits is handier, and it allows a simplified representation of the phenomena. It also leads to faster simulation. In the case of a modelling based on electrical circuits, software such as EMTDC and EMTP [1] can be used. Other software like SPICE are used but they are dedicated to the field of electronic simulation. However they are not well adapted to the fault simulation in electrical machines such as induction machines. The oriented-circuit model proposed in [2] is a good approach but it is heavy to manipulate when thousands of fault simulations must be performed for the machine diagnostics.

The aim of our works involves first the modelling of electrical machines with a simple method. That method allows an integration into a schematical model of all the parameters that can characterize the most usual faults of induction machines. The ultimate goal of that modelling is the fault automatic simulation within electrical machines with their diagnostics in mind.

A first step has been taken by [2], [3] with the proposal of a modelling method based on circuit-oriented approach which has shown excellent results. The interest of this solution is the calculation swiftness in contrast to the mathematical solution using differential equations which are often non-linear. However, that method is based on circuit-oriented models which are not well adapted for the quick execution of several experiments and with minimal manipulations and time.

We propose to use BFS-DEVS (Behavioral Fault Simulation for DEVS) [4] formalism in order to model, in a hierarchical and modular way, induction machines. Models are object-oriented and easily manageable when they must evolve or be re-used. Moreover, BFS-DEVS models can simulate automatically the faulty electrical machine behaviors. The circuits allowing the modelling of induction machines are composed of analogical elements showing purely physical phenomena.

The VHDL-AMS [5] language permits to model and simulate mixed-technology micro-systems which consist of electrical circuits connected to subsystems described by differential equations. VHDL-AMS allows the description of components and interconnections like conventional simulator input languages such as SPICE, and also allows expression of component model equations in the frequency domain. DEVS (Discrete EVent system Specification) [6] is a theoretical approach, which allows the definition of hierarchical modular models. In [7] the authors show that the use of DEVS formalism is an interesting solution which facilitates the modelling and simulation of continuous and hybrid systems that are described with VHDL-AMS descriptions.

BFS-DEVS formalism is based on DEVS and so permits to simulate the VHDL-AMS descriptions of circuits modelling the induction machines. It allows a simple model of complexe induction machines that can integrate all physical characteristics such as electromagnetic fields. Moreover, BFS-DEVS permits to specify the model's behavior if they are affected by behavioral faults.

This paper is organized in the following way: in the first part, we present the related works in the field of circuit-oriented modelling for electrical machines; we describe the Discrete Event systems Specification formalism (DEVS); we also present the Behavioral Fault Simulation for DEVS (BFS-DEVS) with its implementation concerning digital circuits testing domain. The second part describes our approach to model the induction machine from both the circuit-oriented model and the use of BFS-DEVS formalism. In the third part, we implement the method described by the example of a complete induction generator machine with its electrical model. Finally, we proposed some conclusions as well as some work prospects.

II. Related Works

A. *Circuit-oriented modelling for electrical machine*

The circuit-oriented model of an induction generator can be expressed with circuit elements such as resistances, inductances, capacitances and voltage or current sources [2], [8]. With this approach, the use of complex electrical circuits allows the time domain investigation without having to implement any transformation technique. Then, physical characteristics of the electri-

cal machine are taken into account. Moreover, the changes in electrical connections or circuit parameters can be easily implemented.

Starting from the concept of magnetically coupled circuits, an induction machine can be represented taking into account the distributed nature of the windings and stator and rotor equivalent circuits can be represented on a conductor-by-conductor basis. Resistive and inductive effects are taken into account and the capacitive effect is neglected to remain in the low frequency range.

Assuming a sinusoidal distribution of windings for both the three-phase stator and rotor circuits, the circuit modeling of this machine is described in figure 1.

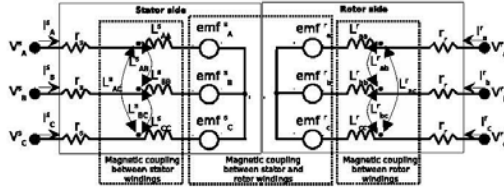


Fig. 1. Circuit-oriented modelling of an induction generator

In both stator and rotor sides, the conductor resistances (r_s and r_r), the self inductances (L^s_{AA}, L^s_{BB}, L^s_{CC} and L^r_{aa}, L^r_{bb}, L^r_{cc}) and the mutual inductances (L^s_{AB}=L^s_{BA}, L^s_{BC}=L^s_{CB}, L^s_{AC}=L^s_{CA} and L^r_{ab}=L^r_{ba}, L^r_{bc}=L^r_{cb}, L^r_{ac}=L^r_{ca}) are concentrated inside a single block to represent the constant parameters.

The magnetic coupling between stator and rotor coils are subject to the non-linear variation due to the stator-rotor relative position. To model this phenomenon, controlled voltage sources (emf^s_A, emf^s_B, emf^s_C, for the stator side and emf^s_a, emf^s_b, emf^s_c, for the rotor side) can be used. Moreover, (emf^s_A, emf^s_B, emf^s_C) depends on rotor currents (I^r_a, I^r_b, I^r_c) and (emf^r_a, emf^r_b, emf^r_c) depends on stator currents (I^s_A, I^s_B, I^s_C).

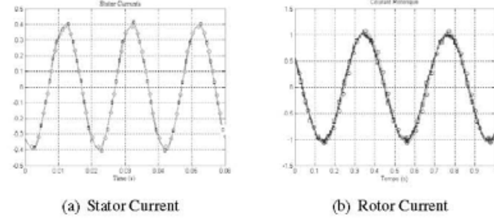
A specific experimental set-up has been designed in order to perform measurements on a lab-scaled three-phase 0.09kW, 50Hz, 220V/380V, 4-poles induction generator. The corresponding parameters are provided in table I.

Rated power	0.09kW
Rated line voltage	380V
Rated frequency	50Hz
Poles	4
Rated torque	0.6Nm
Inertia	0.05kg.m2
Stator resistance r_S	79.13Ω
Stator inductance L_S	2.83H
Stator magnetizing inductance L_{ms}	1.1H
Rotor resistance r_r	3.68Ω
Rotor inductance L_r	0.23H
Rotor magnetizing inductance L_{mr}	0.11H
Mutual inductance L_{sr}	0.68H

TABLE I
PARAMETERS VALUES FOR EXPERIMENTAL SIMULATION

In figures 2 (a) and (b), the experimental (square) and simulated (circle) stator and rotor currents are presented. On each figure, the currents are shown with a zoom to check the waveform magnitude and period with accuracy.

The simulated results obtained from this circuit-oriented model are very close to those obtained from experimental results. The approach proposed in [8] shows that the circuit-oriented model allows a good representation of an induction generator functioning.



(a) Stator Current (b) Rotor Current

Fig. 2. Experimental (square) and simulated (circle) stator and rotor currents at 1700 rpm

In the following part, we propose to exploit the oriented-circuit representation and to simplify the that model using DEVS formalism.

B. DEVS formalism

Based on systems theory, DEVS formalism was introduced by Professor B.P. Zeigler in the late 70s [9], [10]. It allows a hierarchical and modular way to model the discrete event systems. A system (or model) is called modular if it possesses the input and output ports permitting interaction with its outside environment. In DEVS, a model is seen as a "black box"S which receives and broadcasts messages on its input and output ports. DEVS defines two kinds of models: atomic models and coupled models, representing respectively the behavior and the internal structure of a part of a model.

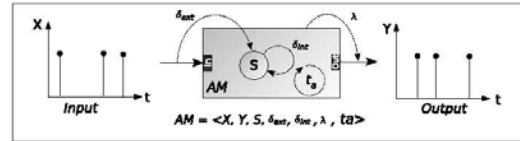


Fig. 3. DEVS atomic model

Figure 3 represents an *AM* atomic model with its output data Y calculated according to input data X. The *AM* atomic model has a state variable S that can be reached during the simulation. The functions δ_{ext}, λ, δ_{int} and t_a respectively allow the model's change of state when an external event occurs on one of those outputs (external transition function), the disposal of the output Y (output function), the model's change of state after having given an output (internal transition function) and finally the determination of the duration of the model's state (time advance function).

The coupled models are defined by a set of sub-models (atomic and/or coupled) and express the internal structure of the system's sub-parts thanks to the coupling definition between the sub-models.

Figure 4 shows an example of the hierarchical structure of coupled model CM_0 which has an input port IN and two output ports OUT_0 and OUT_1. It contains the atomic sub-models AM_0, AM_1 and also the coupled model CM_1. The latter can encapsulate other models such as atomic models AM_2, AM_3 and AM_4. A coupled model is specified through the list of its components (AM_0, AM_1,AM_2, AM_3, AM_4 and CM_1 in figure 4), the list of its internal couplings ($AM_0 \rightarrow CM_1$ and $AM_1 \rightarrow CM_1$ in figure 4), the list of the external input couplings ($IN \rightarrow AM_0$ and $IN \rightarrow AM_1$ in figure 4), the list of the external output couplings ($CM_1 \rightarrow OUT_0$ and $CM_1 \rightarrow OUT_1$ in figure 4) and the list of the sub-model's influence ($CM_1 = \{AM_0, AM_1\}$ or CM_1 and influenced by AM_0 and AM_1).

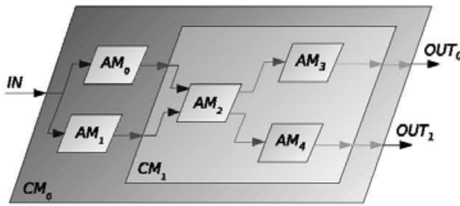


Fig. 4. DEVS coupled model

DEVS formalism is mainly used for the description of discrete event systems. It constitutes a powerful modeling and simulation tool permitting a system modelling on several levels of description as well as the definition of the models' behaviors.

One of DEVS formalism's important properties is that it automatically provides a simulator for each model. DEVS establishes a distinction between a system modeling and a system simulation so as any model can be simulated without the need for a specific simulator to be implemented. Each atomic model is associated with a simulator in charge of managing the component's behavior and each coupled model is associated with a coordinator in charge of the time synchronization of underlying components.

C. BFS-DEVS formalism

BFS-DEVS formalism *(Behavioral Fault Simulation for DEVS)* has the same modelling rules as DEVS but it also provides the possibility to define the faulty behavior of the models. It provides the possibility to construct the library of re-usable BFS-DEVS models that are specific to a field of study. After defining the system's fault model, the model faulty behavior can be specified through a new transition function called "faulty". As it is shown in figure 5, the user has to define a fault model, the control structure, the data structure as well as the behavioral rules of the system in order to construct the BFS-DEVS library.

A behavioral fault or state fault is represented by [11] as a change of the transition function and/or the output function of DEVS model. So a fault has an influence on the state of the model and can lead to a behavior that can be qualified as faulty. Thus we can define a behavioral fault model representing the whole of fault types leading to the unexpected behavior of a model. Faults strongly depend on the nature of the physical system we want to model and they must be the most representative of the real behavioral defaults. Fault model strongly depends on the control structure and the application data. Once the fault model is chosen it will be integrated into the behaviors of each BFS-DEVS model thanks to the faulty transition function δ_{fault}.

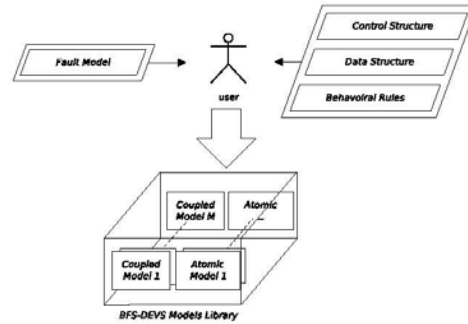


Fig. 5. BFS-DEVS models library

The determination of the control and data structure of an application needs experience. It ensues from the discrete event modelling and it will be integrated into the number, the behavior and the coupling of the BFS-DEVS models that compose the library. So, the construction of a library specific to a field is not an easy matter but it is the only delicate stage of BFS-DEVS modeling. Indeed once that task is performed, the formalism automatically ensures the simulation of the library models.

BFS-DEVS formalism constitutes an ideal interface to perform the behavioral fault simulation of a system belonging to an application field. The BFS-DEVS simulation kernel integrates algorithms which permit simulations of behavioral faults in a concurrent way within the models. This simulation is based on a fault list propagation technique through the BFS-DEVS components network. That technique allows the gathering of faulty simulations in order to obtain a better management of the simulation and it also makes the result's observability at the end of the simulation easier.

In [12], BFS-DEVS formalism has been used to make a behavioral fault simulation on high-level behavioral VHDL descriptions. The chosen approach consists in the transformation of the VHDL behavioral description corresponding to a gate-level circuit into a BFS-DEVS components network [13]. The latter is made thanks to the definition of the BFS-DEVS library composed of models describing all the basic statements of a VHDL language subset. As the representation is textual (circuit VHDL description), both the control and data structures are deduced from VHDL language. The chosen fault model is based on the behavioral rules of the VHDL subset statements. It takes into account the stuck-at of signals and VHDL variables ,the stuck-at of the conditional statement branches as well as the statement jump. The authors validate their approach on ITC'99 benchmarks [14] and they show how the concurrent and com-

parative algorithms accelerate the fault detection as soon as the beginning of the first simulation steps.

III. BFS-DEVS for Induction Machine Modelling

The goal of our modelling approach is to give a simplified model of induction machines from BFS-DEVS formalism in order to simulate the most usual defaults on those machines. The main idea, shown in figure 6, consists in the transformation of the circuit-oriented models proposed in [13] into VHDL-AMS descriptions and then in the translation of those descriptions into BFS-DEVS components networks.

According to [15] VHDL-AMS " is an IEEE standard and extension of a high-level hardware description digital language VHDL. VHDL-AMS is widely used in electronic design flow for modeling various mixed-signal (analog and digital) circuits and systems. It can also be used to model mixed-technology systems if their description is limited to differential algebraic equations". The analog circuits representing the induction machines make models based on differential equations intervene (mutual inductance for instance). The VHDL-AMS language permits a homogeneous textual representation of those analog circuits.

In [7], the authors show how DEVS formalism makes the simulation of the described models easier with a simplified version of the sAMS-VHDL language. This simulation is made after a transformation of models written in VHDL-AMS into equivalent DEVS models. The nature of DEVS formalism allows the integration of those models in component form.

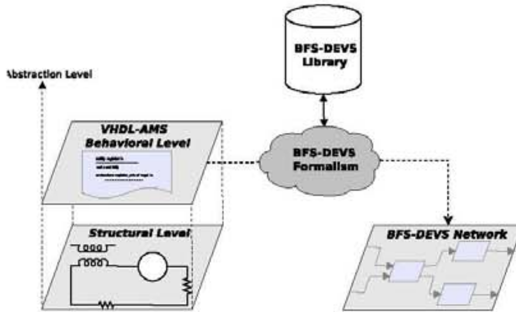


Fig. 6. BFS-DEVS for VHDL circuit modelling

BFS-DEVS formalism permits the transformation of VHDL-AMS textual model into BFS-DEVS model because it benefits from DEVS properties. Moreover, BFS-DEVS models belonging to the network integrate the system's faulty behaviors. The advantages of the construction of those networks are:

- a modular and hierarchical representation of electrical machines behavior,
- an easy way to define behavioral faults,
- an automatic simulation of behavioral faults.

Concerning induction machines, circuit-oriented modelling methods separate the rotor and stator models [16]. The stator model results from the representation of each coil turn. All the electrical parameters such as resistances and mutual inductances are obtained by applying the magnetic laws of the circuits. The model of the rotor cage is based on the same principle by identifying each electrical element according to each mesh. It is the connection of different meshes which gives the global representation of the rotor cage.

The two types of circuit representing the stator and rotor cage of the induction machine are transcribed into VHDL-AMS language by translating each equation of each electrical element. The voltage relationships of active elements such as mutual inductances or the resulting electromotive forces are implemented thanks to "Simultaneous" statements because they allow the description of algebraic differential equations [5]. Then their resolution calls on Euler or Runge-Kutta methods[7]. The passive elements such as resistances are represented by using the VHDL language basic sequential statements.

It is from VHDL-AMS language textual descriptions that we propose a construction of basic BFS-DEVS models. Each statement of the VHDL-AMS code in the circuit architecture is converted into an atomic or coupled model. The rules used in [13] for the transformation of behavioral VHDL description can be implemented. However, an atomic model has to be added in the case of the "Simultaneous" statement whose behavior is the digital integration of differential equations.

Now we are going to show how to construct the BFS-DEVS library specific to the field of induction machine in the case of a circuit which models a stator winding.

IV. Induction Generator Modelling Example

The three-phase electrical circuit modelling induction generator is shown in figure 7 coming from [8]. The representation of that circuit, simplified in VHDL-AMS language, goes through the behavioral definition of each passive and active element composing it.

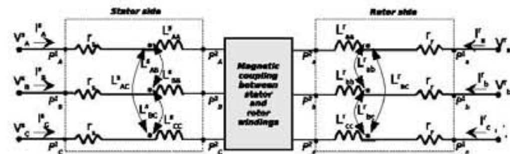


Fig. 7. Simplified circuit modeling of an induction generator

```
————————————Testbench————————————
LIBRARY IEEE; USE IEEE.electrical_systems.all; work.all;

ENTITY testbench IS
  PORT(TERMINAL P¹_A, P¹_B, P¹_C: ELECTRICAL;
       TERMINAL P¹_a, P¹_b, P¹_c: ELECTRICAL);
END testbench;

ARCHITECTURE arch OF testbench IS
BEGIN

  ENTITY stator(arch_stator)
   GENERIC map (r_s ⇒ 79.13, L^S_AA ⇒ 2.83, L^S_AB ⇒ 1.1,
                L^S_BB ⇒ 2.83, L^S_AC ⇒ 1.1, L^S_CC ⇒ 2.83,
                L^S_BC ⇒ 1.1)
   PORT map (v1 ⇒ P¹_A, v2 ⇒ P¹_B, v3 ⇒ P¹_C, v4 ⇒ P²_A,
             v5 ⇒ P²_B, v6 ⇒ P²_C);

  ENTITY rotor(arch_rotor)
   GENERIC map (r_r ⇒ 3.68, L^r_aa ⇒ 0.23, L^r_ab ⇒ 0.68,
                L^r_bb ⇒ 0.23, L^r_ac ⇒ 0.68 L^r_cc ⇒ 0.23,
                L^r_bc ⇒ 0.68)
   PORT map (v1 ⇒ P¹_a, v2 ⇒ P¹_b, v3 ⇒ P¹_c, v4 ⇒ P²_a,
             v5 ⇒ P²_b, v6 ⇒ P²_c);

  ENTITY magnetic_coupling(arch_magnetic_coupling)
   PORT map (v1 ⇒ P²_A, v2 ⇒ P²_B, v3 ⇒ P²_C, v4 ⇒ P²_a,
             v5 ⇒ P²_b, v6 ⇒ P²_c);
END arch;
```

Fig. 8. Example of VHDL-AMS testbench for stator model

The VHDL-AMS testbench allowing the simulation of the circuit is given in figure 8. It is composed of a global entity made of the following electrical ports $P^1_A = V^s_A$, $P^1_B = V^s_B$, $P^1_C = V^s_C$, $P^1_a = V^r_a$, $P^1_b = V^r_b$ and $P^1_c = V^r_c$. The "arch" architecture is composed of three main entities of the highest level: an entity corresponding to the stator model, an entity corresponding to the rotor model and an entity corresponding to the magnetic coupling between the stator and the rotor. Each entity allows the definition of the element's constants (resistance, inductance, ...) and the connection between the ports P. The "stator" entity has the "arch_stator" architecture, the line called GENERIC permits the definition and assignment of the resistance r_s, the inductances L_s and the mutual inductances L_{ms} thanks to the grid I.

The line called PORT permits the connection of the ports $v_{1\leq i\leq 6}$ to the ports of the element P^1_A,P^1_B,P^1_C,P^2_A,P^2_B, and P^2_C. Generally each element of the circuit, either passive or active, is represented by its entity which defines its characteristics (resistance, inductance, ...) and its interface (ports). Of course the modelling can be more explicit by considering each basic element (passive or active) as the full entity. We propose a model described at a high level of abstraction in order to present a handy and simple representation.

Each element of the circuit in figure 8 has an interface and a behavior that can be described thanks to an entity and a specific architecture. If we take the example of the stator, its VHDL-AMS description is given in figure 9. In the case of the description of an active element composed of differential equations we use the "Simultaneous" statement [5].

```
————————————stator————————————
LIBRARY IEEE; USE IEEE.electrical_systems.all; work.all;

ENTITY stator IS
  GENERIC (r_s, L^S_AA, L^S_AB, L^S_BB, L^S_AC, L^S_CC, L^S_BC: real);
  PORT(TERMINAL v1,v2,v3,v4,v5,v6: ELECTRICAL);
END stator;

ARCHITECTURE arch_stator OF stator IS
 QUANTITY U^s_A ACROSS I^s_A THROUGH v1 TO v4;
 QUANTITY U^s_B ACROSS I^s_B THROUGH v2 TO v5;
 QUANTITY U^s_C ACROSS I^s_C THROUGH v3 TO v6;
BEGIN
 U^s_A == f(r_s,I^s_A,L^s_AA,L^s_AB,L^s_AC)
 U^s_B == f(r_s,I^s_B,L^s_BB,L^s_BA,L^s_BC)
 U^s_C == f(r_s,I^s_C,L^s_CC,L^s_CB,L^s_CA)
END arch_stator;
```

Fig. 9. Example of VHDL-AMS testbench for stator model

The BFS-DEVS network of a generator machine includes the behavior of each circuit element presented in figure 7. The control and data structures of the modeled system are those of the VHDL-AMS code. The fault model can be based on the data values (resistance, inductance, mutual inductance, electromagnetic field, ...) in order to consider the stuck-at faults. The components library for the modelling of a generator machine can be constructed by considering a coupled model for each high-level element (stator, rotor and the stator/rotor coupling model). Each of those coupled models contains an interconnection of atomic models corresponding to the control graph of the statements in the element's VHDL-AMS architecture. The fault model is contained in the faulty transition function in each of those atomic models. So, the network can be directly simulated thanks to DEVS formalism properties.

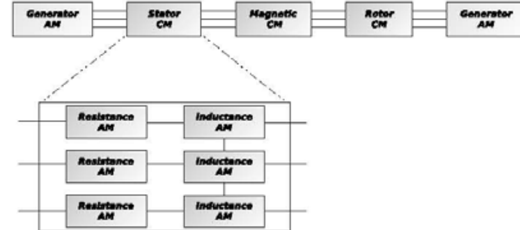


Fig. 10. BFS-DEVS network for generator induction circuit model

Figure 10 represents the BFS-DEVS network of a three-phase generator induction circuit. Each entity of the highest level such as the stator, the rotor and the magnetic coupling between the stator and the rotor is represented by a coupled model. Coupled models contain an interconnection of atomic models corresponding to the VHDL-AMS statements which describe the entity's behavior. Figure 10 shows that the stator coupled model

is composed of three atomic models "Inductance " which are interconnected. This interconnection allows the simulation of the mutual inductance effects within the stator. Moreover, the atomic models "Generator" are present in order to simulate the system's behavior.

V. Conclusion and Future Works

We have proposed a method for high-level behavioral modelling of the circuit-oriented induction generator. That is why we used the VHDL-AMS language to obtain a homogeneous description of electrical machines such as induction generators that can integrate representations based on differential equations. In order to take advantage from the re-use notion, we have transformed these textual descriptions into BFS-DEVS components network. DEVS formalism permitting a hierarchical and modular way to specify the behavior of systems, we show more particularly how, from circuit-oriented models, a realistic and simplified model of the induction generator stator can be defined.

The future works are mainly based on the possibility of obtaining a concurrent and automatic modelling and simulation environment. This environment will allow a simulation of the most usual defaults that can intervene on the induction machines. Indeed, the BFS-DEVS simulation kernel is based on concurrent and comparative algorithms which use the fault list propagation technique. After having entirely modeled the induction machine stator and rotor and having chosen a realistic fault model we plan to perform the concurrent fault simulation on the whole model.

References

[1] F. Soares and P. C. Branco, "Simulation of a 6/4 switched reluctance motor based on matlab/simulink environment," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 37, pp. 989–1009, 2001.

[2] H. Henao, C. Martis, and G. Capolino, "An equivalent internal circuit of the induction machine for advanced spectral analysis," in *Proceedings of IAS Annual Meeting (IAS'02)*, vol. 2, pp. 739–745, october 2002. Pittsburgh (USA).

[3] H.Hénao, G.A.Capolino, and M.Poloujadoff, "A circuit-oriented model of induction machine for diagnostics," in *Proceedings of International Symposium on Diagnostics for Electrical Machines, Power Electronics & Drives (SDEMPED'97)*, pp. 185–190, 1997. Carry-le-Rouet (France).

[4] L. Capocchi, *Simulation de Fautes Comportementales pour des Systèmes à Evénements Discrets: Application aux Circuits Digitaux*. PhD thesis, november 2005. University of Corsica.

[5] E. Moser and N. Mittwollen, "VHDL-AMS: The missing link in system design - experiments with unified modelling in automative angineering," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'98)*, pp. 59–65, 1998. Le Palais des Congrès de Paris, France.

[6] B. P. Zeigler and S. Vahie, "DEVS formalism and methodology - unity of conception diversity of application," in *Proceedings of 1993 Winter Simulation Conference* (S. Editions, ed.), pp. 573–579, 1993.

[7] S. Mehta and G. Wainer, "Devs for mixed-signal modeling based on VHDL," in *Proceedings of 2005 DEVS Integrative M&S Symposium, Spring Simulation Conference*, 2005. San Diego, CA. U.S.A.

[8] A. Yazidi, H. Henao, G. Capolino, D. Casadei, and F. Filippetti, "Double-fed three-phase induction machine abc model for simulation and control purposes," in *Proceedings of IEEE Industrial Electronics Conference (IECON'05)*, vol. 4, pp. 2560–2565, November 2005.

[9] B. P. Zeigler, *Theory of Modeling and Simulation*. Academic Press, 1976.

[10] B. P. Zeigler, "An introduction to set theory," tech. rep., 2003. ACIMS Laboratory, University of Arizona.

[11] E. Kofman, N. Giambiasi, and S. Junco, "FDEVS: A general DEVS-based formalism for fault modeling and simulation," in *Proceedings of European Simulation Symposium*, pp. 77–82, 2000. Hamburg, Germany.

[12] L. Capocchi, D. Federici, F. Bernardi, and P. Bisgambiglia, "Behavioral fault simulation for VHDL descriptions using the DEVS formalism," in *IEEE Pacific Rim Dependable Computing International Conference*, 2004.

[13] L. Capocchi, F. Bernardi, D. Federici, and P. Bisgambiglia, "Transformation of VHDL descriptions into DEVS models for fault modeling and simulation," in *Proceedings of IEEE Systems, Man and Cybernetics Conference (SMC'03)*, pp. 1205–1211, 2003. Washington D.C., USA.

[14] F. Corno, M. S. Reorda, and G. Squillero, "RT-level itc'99 benchmarks and first ATPG results," in *Proceedings of IEEE Design and Test of Computers*, pp. 44–53, 2000.

[15] P. Nikitin and C.-J. R. Shi, *VHDL-AMS Based Modeling and Simulation of Mixed-Technology Microsystems: A Tutorial*. Integration, the VLSI Journal, 2006.

[16] H.Hénao, G. Capolino, M. Melero, and M. Cabanas, "A new model of the induction motor rotor cage for diagnostics," in *Proceedings of International Symposium on Diagnostics for Electrical Machines, Power Electronics & Drives (SDEMPED'99)*, pp. 383–388, november 1999. Gijon (Spain).

Annexe 1 – Lettres d’appréciation

UMR CNRS 6134 - SPE
SYSTEMES PHYSIQUES DE L'ENVIRONNEMENT

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Objet : Habilitation à Diriger le Recherches de Mr Dominique Federici

Le candidat est membre de l'UMR 6134 SPE, que j'ai l'honneur de diriger, depuis 1995 : tout d'abord comme doctorant (sous la direction du professeur Jean-François Santucci) puis à partir de 1999 comme Maître de Conférences en Informatique. Je suis donc son activité scientifique depuis plus de dix ans.
Il a démarré cette activité dans l'équipe « Modélisation et Conception de Systèmes » qui elle même était en train de se constituer, donc dans des conditions en quelque sorte « pionnières ». Malgré cela, il a tout de suite manifesté son autonomie et son ouverture : dans le cadre du projet européen BELSIGN, il a effectué un séjour de recherche dans l'institut Fraunhofer de Dresde. Il a rapidement participé à l'encadrement de jeunes chercheurs (3 DEA, 3 codirections de thèse) ainsi que pris des responsabilités scientifiques dans différents projets (CATSAT avec les Etats-Unis et le Royaume-Uni, projet européen HCM-BELSIGN, INTERREG III – NTIC, GDR SEEDS).
Cette activité s'est traduite par un bon niveau de publications (6) et une vingtaine de communications dans des congrès bien connus.
Parallèlement, il n'a pas hésité à prendre plusieurs responsabilités pédagogiques (DEUG MIAS, MASTER 2 CCI, MASTER 2 ISI, IUP NTIC, Département Informatique).

En conclusion, ce collègue a largement fait ses preuves de compétences scientifiques mais aussi et c'est plus rare, d'engagement au service de sa structure d'enseignement et de recherche : il a parfaitement assumé ses différents rôles d'enseignant, de chercheur, de codirecteur de recherche et de responsable pédagogique et administratif (Conseil de laboratoire, Commission de spécialistes, …)

J'ai pu juger de près du sérieux et de la responsabilité dont il a su faire preuve dés le début de sa carrière. Je l'estime tout à fait prêt pour soutenir son Habilitation à Diriger les Recherches et il joue d'ores et déjà ce rôle de directeur de recherche.

Corté, le 4 Octobre 2006
Professeur Jacques-Henri Balbi
Directeur de l'UMR 6134

UNIVERSITE DE CORSE
B.P. 52 – CAMPUS GROSSETTI – F. 20250 CORTE
Tél.: (33) 04 95 45 01 65 – Fax.: (33) 04 95 45 01 62

UNIVERSITE DE CORSE
VIGNOLA – ROUTE DES SANGUINAIRES – F. 20000 AJACCIO
Tél.: (33) 04 95 52 41 41 – Fax.: (33) 04 95 52 41 42

UNIVERSITÉ DE CORSE
Laboratoire SPE
UMR CNRS n° 6134

Monsieur Dominique Federici a été membre du laboratoire *Systèmes Physiques de l'Environnement* (SPE) UMR CNRS n° 6134 de l'Université de Corse en tant que doctorant (bourse régionale), affecté au sein de l'équipe de recherche en informatique : *Modélisation et conception de systèmes* depuis 1995. Après des travaux de thèse brillants, il a été recruté comme Maître de Conférence en Informatique à l'Université de Corse en 1999.

Durant ces onze années, Dominique Federici a révélé des compétences scientifiques et un investissement exceptionnels, accomplissant ainsi un travail indispensable à l'évolution de notre recherche sur les systèmes complexes électroniques et énergétiques. Il a réalisé son étude de manière autonome, en faisant preuve d'une grande ouverture d'esprit au sein de l'équipe Modélisation et conception de système, de notre laboratoire et à l'extérieur.

En effet, l'implication forte de Dominique Federici au sein de notre laboratoire a démontré des qualités personnelles et professionnelles fortement appréciées par tous les membres de notre laboratoire. Cette implication peut être constatée à travers une riche production scientifique. De plus, Dominique Federici a su structurer son projet scientifique et l'organiser à travers différentes collaborations : locales, régionales, nationales et internationales, prouvant ainsi sa capacité à partager et à créer du savoir dans notre laboratoire. Par ailleurs, il a su trouver les contrats nécessaires au financement de ces collaborations. Enfin, il s'est impliqué avec dévouement et conscience dans les différentes instances de l'Université (conseil de laboratoire, membre de la commission de spécialistes mathématiques/informatique, direction du département informatique de la FST), où il a été élu.

En ce qui concerne la production scientifique de Dominique Federici, celle-ci atteste d'une réelle maturité. Cette maturité est attestée par : le nombre de publications, sa participation à des recherches en collaboration des universités ou instituts extérieurs (Université du New Hampshire, Institut Fraunhofer de Dresde et Université de Picardie), l'encadrement efficace et pleinement réussi de jeunes chercheurs (qui ont brillamment soutenu leur thèse ou leur Master recherche) et enfin par la reconnaissance témoignée par ses pairs, lesquels lui ont demandé d'être référé pour des conférences de rayonnement international.

Sur le plan des collaborations locales, Dominique Federici a réussi, en tant qu'informaticien, à développer une thématique originale s'intégrant au sein de l'axe Modélisation et conception de systèmes. Il a su de plus développer et participer des thèmes de recherche originaux et de dimension internationale avec des chercheurs de l'Université de Corse non informaticiens : énergies renouvelables, apprentissage de chants polyphoniques, SIG pour les SHS.

Sur le plan régional, il a été responsable du projet *Développement d'un outil de test de circuit et amélioration de la testabilité des circtuits*. Sur le plan national, il a collaboré avec des chercheurs confirmés (Gérard Capolino et Humberto Henao) du *Laboratoire* CREA - UPRES EA3299 de l'Université de Picardie à Amiens. Cette collaboration lui a permis d'être le responsable pour l'Université de Corse de la participation de l'UMR CNRS 6134 au GDR SEEDS. Enfin, sur le plan international, il collaboré et publié avec des chercheurs aux Etats-Unis (Prof. Andrzej Rucinski) à l'Université du New Hampshire où il a effectué un stage de recherche prédoctorale de deux mois, et collaboré dans le cadre d'un projet européen

avec des Universités espagnoles, portugaises, allemandes ainsi qu'avec l'Institut Fraunhofer de Dresde où il a effectué un stage de recherche doctorale de trois mois, prouvant ici encore sa motivation.

Pour toutes ces raisons, en tant que Professeur d'informatique, Directeur adjoint de l'UMR CNRS n° 6134 SPE et Directeur de l'Institut de l'environnement de l'Université de Corse, je pense que Dominique Federici a démontré qu'il possédait les qualités nécessaires pour remplir les fonctions de Professeur des Universités. L'affectation d'un tel chercheur, dans le domaine de la modélisation et de la simulation de systèmes, au sein de notre laboratoire – pluridisciplinaire, spécialisé dans l'étude des systèmes de l'environnement, permettrait d'apporter un soutien crucial à notre recherche dans tous les domaines (environnement, SHS et informatique) et plus particulièrement pour le développement de l'axe Energie Renouvelable, objectif de première importance de notre laboratoire dans le cadre du Pôle de Compétitivité PACA-Corse « CAP ENERGIE ».

Fait à Corté, le 4 octobre 2006
Professeur Jean-François SANTUCCI
Directeur adjoint de l'UMR SPE CNRS n° 6134
Directeur de l'Institut de l'environnement