ACADEMIE DE MONTPELLIER

UNIVERSITE MONTPELLIER II

SCIENCES ET TECHNIQUES DE LANGUEDOC

THESE

Présentée à l'Université de Montpellier II et Techniques du Languedoc
pour obtenir le **DIPLOME DE DOCTORAT**

Spécialité : **Microélectronique**

Formation Doctorale : **Systèmes Automatiques et Microélectroniques**

Ecole Doctorale : **Information, Structures, Systèmes**

Diagnostic de Pannes Electriques dans les Systèmes Logiques

par

Youssef BEN ABBOUD

Soutenue le **30 Avril 2010** devant le Jury composé de :

M. Abbas Dandache, Professeur, Université Paul Verlaine, LICM, Metz	Rapporteur
M. Emmanuel Simeu, Maître de Conférences, UJ F, TIMA, Grenoble	Rapporteur
Mme. Olivia Riewer, Sté STMicroelectronics, Crolles	Examinateur
M. Alberto Bosio, Maître de Conférences, Université Montpellier II, LIRMM	Examinateur
M. Patrick Girard, Directeur de Recherche CNRS, LIRMM	Directeur de Thèse
M. Serge Pravossoudovitch, Professeur, Université Montpellier II, LIRMM	Codirecteur de Thèse

Remerciements

Cette thèse a été effectuée au sein de la société STMicroelectronics (Crolles 38) dirigée par Monsieur Gérard Matheron et au Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM) dirigé par Monsieur Michel Robert, Professeur à l'Université de Montpellier II.

Je souhaite remercier Monsieur Philippe Magarshack, Directeur du département Central CAD et Design Solutions (CCDS) au sein de la société STMicroelectronics, et Monsieur Lionel Torres, Responsable du département Microélectronique du LIRMM, pour m'avoir accueilli au sein de leur département.

Toute ma gratitude va à mon directeur de thèse, Patrick Girard, Directeur de Recherche au CNRS, dont l'encadrement très professionnel, la présence et les encouragements constants m'ont permis d'évoluer en thèse en toute sérénité. Je voudrais qu'il trouve ici toute l'expression de ma reconnaissance.

Je tiens à remercier mon co-directeur de thèse, Serge Pravossoudovitch, Professeur à l'Université Montpellier II, pour le rôle important d'encadrement qu'il a joué tout au long de ma thèse, pour son soutien moral et technique.

Je voudrais également remercier Olivia Riewer, Manager de l'équipe de Diagnostic Logique au sein de CCDS, qui m'a beaucoup aidé tout au long de cette thèse tant sur l'aspect technique qu'industriel de mes travaux. Sa vision pragmatique et son esprit de synthèse ont été précieux pour mener à bien cette thèse. Je lui adresse ici mes remerciements les plus sincères.

Je remercie aussi Messieurs Alberto Bosio (Maître de Conférence Montpellier, LRMM), Arnaud Epinat (Ingénieur à STMicroelectronics) et Madame Isabelle Izaute (Ingénieur à STMicroelectronics) pour m'avoir encadré avec beaucoup d'enthousiasme durant ces trois années, mais aussi pour m'avoir apporté tant sur le plan scientifique que technique.

Je remercie sincèrement Monsieur Laroussi Bouzaida pour m'avoir donné l'opportunité de faire ma thèse au sein de la société STMicroelectronics, ainsi que pour tous les précieux conseils qu'il m'a donnés, pour la confiance qu'il m'a témoignée et sans qui ce travail n'aurait pas vu le jour.

Je remercie également Monsieur Abbas Dandache (Professeur à l'Université Paul Verlaine, Metz) et Monsieur Emmanuel Simeu (Maître de Conférences à l'Université Joseph Fourier, Grenoble), de s'être intéressés à ce travail de thèse et d'avoir accepté d'en être les rapporteurs. Qu'ils trouvent ici toute l'expression de ma reconnaissance.

Tous mes remerciements également à Madame Martin Dufils et Messieurs Nazir Hakim, Antonio Marques, Arnaud Virazel, Luigi Dilillo de m'avoir beaucoup soutenu durant ma thèse.

Mes remerciements vont tout droit à ma famille sans qui je ne serais jamais arrivé là où je suis maintenant, toujours là pour me soutenir, répondant à chaque appel.

Enfin, je terminerais en remerciant tous les thésards croisés durant ces quelques années passées au LIRMM qui ont joué un rôle de soutien, de confident ou un rôle technique pour moi, et tout particulièrement Messieurs Julien Vial, Olivier Brousse, Nicolas Houarche, Alexandre Ney, Amine Dehbaoui, Victor Lomné.

A tous merci…

Table des matières

Introduction générale 8

Chapitre I

Modèles de Fautes pour le Diagnostic 14

1.1 Modèles de fautes 16
1.1.1 Fautes de collage (stuck-at) 17
1.1.2 Fautes de court-circuit (bridge) 17
1.1.3 Fautes de circuit ouvert 25
1.1.4 Fautes de délai (transition) 26
1.1.5 Fautes de stuck-on/stuck-open 27
1.1.6 Fautes multiples 28
1.2 Conclusion 29

Chapitre II

Diagnostic de Pannes dans les Systèmes sur Puce 30

2.1 Diagnostic logique par une approche « Effet-à-Cause » 33
2.1.1 Principe de la méthode 33
2.1.2 Phase de localisation des sites potentiels de panne 36
2.1.3 Phase d'affectation des modèles de fautes 46
2.2 Adaptation de la méthode de diagnostic pour le traitement des systèmes sur puce 48
2.2.1 Description d'un système sur puce 48
2.2.2 Synoptique de la méthode de diagnostic adaptée 50
2.2.3 Modification du format des fichiers d'entrée 52
2.2.4 Modification du processus de diagnostic 59
2.3 Résultats expérimentaux 66
2.3.1 Processus expérimental 66
2.3.2 Les différents circuits étudiés 66
2.3.3 Résultats expérimentaux 67
2.3.4 Cas d'étude 69
2.4 Conclusion 71

Chapitre III

Développement d'une nouvelle méthode de diagnostic basée sur l'approche Cause-à-Effet 72

3.1 Nouvelle méthode de diagnostic basée sur l'approche Cause-à-Effet..... 74
3.1.1 Principe de la méthode de diagnostic développée 74
3.1.2 Phase de pré-traitement..... 76
3.1.3 Phase de localisation des sites potentiels de panne 80
3.1.4 Phase d'affectation des modèles de fautes..... 85
3.2 Adaptation de la méthode de diagnostic pour le traitement des blocs logiques séquentiels 89
3.2.1 Principe 89
3.2.2 Problématique liée aux circuits séquentiels 91
3.3 Résultats expérimentaux 95
3.3.1 Processus expérimental..... 95
3.3.2 Résultats sur les circuits de la famille ITC'99. 95
3.3.3 Cas d'étude industriel 97
3.4 Conclusion..... 103
Conclusion générale 105
Références personnelles.....110
Références bibliographiques.....113
Liste des figures 119
Liste des tableaux.....122
Annexe 125

Introduction générale

Les finesses de gravure atteintes par les procédés de fabrication CMOS, l'augmentation du nombre de niveaux de métallisation et l'élargissement de la surface des circuits intégrés sont autant de facteurs permettant l'accroissement régulier du nombre de transistors sur une même puce. Cette augmentation croît à une vitesse exponentielle, respectant ainsi la fameuse 'loi de Moore'. Dès lors, des circuits contenant plus d'un milliard de transistors voient le jour, dans lesquels il est possible d'intégrer un système complet sur silicium (*System-on-Chip* ou SoC en anglais).

La fabrication de tels circuits est une suite d'étapes très complexes. C'est pourquoi le bon fonctionnement de chaque circuit après fabrication n'est pas garanti. Le test de fabrication des circuits ou systèmes intégrés est donc un passage obligé, permettant d'effectuer le tri entre circuits correctement fabriqués et circuits défectueux.

Le test de fabrication est une opération complexe de part la diversité des blocs intégrés, l'inaccessibilité des nœuds internes, les jeux de test volumineux, etc., et requiert des testeurs (*Automated Test Equipment* ou ATE en anglais) sophistiqués et donc très coûteux. De façon générale, deux types de test sont mis en œuvre [Aza04]. Le premier consiste à effectuer des mesures paramétriques (courant, tension, temps de montée et de descente, ...). Ce *test paramétrique* fait intervenir des techniques de métrologie particulières. Le deuxième type de test, nommé *test logique*, consiste à inspecter les données logiques manipulées par le circuit et à les comparer avec les valeurs attendues.

Les défauts mis en évidence lors du test de fabrication sont de diverses natures et peuvent être classés en trois catégories. Les défauts aléatoires liés à la qualité du substrat et à la propreté de la salle blanche (pollution, poussière), …, les défauts associés à la lithographie comme par exemple la sur/sous gravure, l'électromigration dans les métaux, les radiations, l'instabilité du process de fabrication (la température), … et les défauts systématiques occasionnés par les pannes de collage, délai, court-circuit, circuit-ouvert, …..

Avec l'avancée des technologies, comme il est explicité sur la Figure 1, les défauts systématiques deviennent prépondérants et sont donc la cause principale des chutes de rendement observées en production. Le rendement de fabrication représente le rapport entre le nombre de circuits fonctionnels produits et le nombre total de circuits produits.

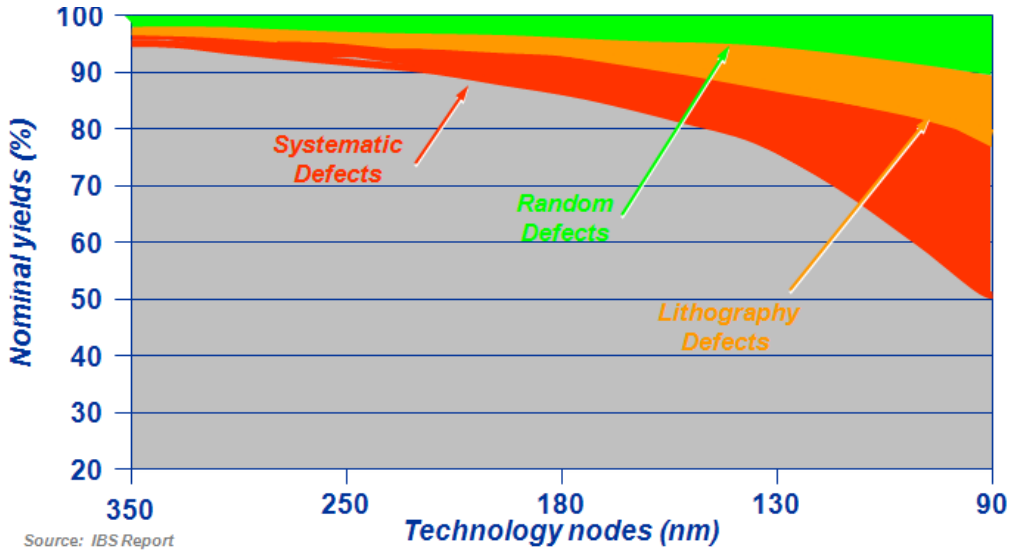


Figure 1 : Rendement vs Technologie

Afin de comprendre l'origine et la nature de défauts pouvant affecter un circuit, et permettre ainsi de corriger les processus de conception et/ou de fabrication, des analyses de défaillances sont systématiquement réalisées à l'issue des premières phases du test de production. Dans un premier temps, cette opération d'analyse des défaillances s'appuie sur des informations produites par un processus de diagnostic logique dont l'objectif est d'identifier un sous-ensemble de sites potentiellement fautifs justifiant les comportements erronés observés sur le Circuit Sous Test (CUT *Circuit Under Test*). Les informations fournies par le processus de diagnostic logique sont alors utilisées pour restreindre l'espace d'investigation et guider l'observation physique du circuit (à l'aide d'un Microscope Electronique à Balayage par exemple) durant l'analyse de défaillances proprement dite [Ber85]. Notons qu'à l'issue de ce processus, la résolution (ou précision) du diagnostic peut être améliorée par la génération de vecteurs de test spécifiques appelés *vecteurs de discernement* [Gir96] [Lin07][Wan06]. Dans ce cas, le résultat initial fourni par le diagnostic logique est utilisé comme entrée d'un processus de génération de vecteurs de test automatique

(ATPG *Automatic Test Pattern Generation*) permettant de générer des vecteurs de test additionnels capable de discriminer les pannes potentielles et ainsi d'améliorer la résolution du diagnostic.

Les algorithmes classiques de diagnostic logique sont basés soit sur une approche « *Cause-à-Effet* », soit sur une approche « *Effet-à-Cause* » [Abra90]. D'autres techniques de diagnostic récemment présentées dans la littérature combinent les deux approches (ou paradigmes) « *Effet-à-Cause* » et « *Cause-à-Effet* » afin de tirer parti des avantages de chacune d'elle [Liu05] [Amy06] [Bos09].

L'approche « *Cause-à-Effet* » s'appuie sur un processus de simulation de fautes. Il génère un dictionnaire de fautes contenant les réponses du circuit à une séquence de test en présence d'un ensemble donné de fautes [Wai89][Wan06]. Le diagnostic est alors exécuté en comparant les réponses du CUT (datalog) avec celles stockées dans le dictionnaire. L'avantage principal de cette approche est le fait de pouvoir traiter des circuits aussi bien combinatoires que séquentiels. Par contre, elle présente plusieurs inconvénients. Le premier d'entre eux est la nécessité de disposer de modèles de fautes représentant au mieux les comportements des défaillances pouvant affecter le circuit et de pouvoir faire exploiter ces modèles par un simulateur de fautes. En d'autres termes, il faut pouvoir disposer d'un dictionnaire de fautes de collage pour le diagnostic de fautes de collage, d'un dictionnaire de fautes de délai pour le diagnostic de fautes de délai, etc. Le second inconvénient est lié à la grande quantité de données qui doivent être générées par le simulateur de fautes (dictionnaire de fautes pour chaque modèle de faute), rendant cette approche difficilement adaptée pour des circuits de grande complexité.

L'approche « *Effet-à-Cause* » utilise un processus de recherche du site défaillant à partir des informations produites sur les sorties du circuit durant l'application du test. Cette approche, basée sur un processus de traçage de chemins critiques (*Critical Path Tracing* en anglais, ou CPT) a initialement été proposée dans [Abra80]. Ce processus démarre à partir d'une sortie fautive du circuit et, par une analyse systématique des possibilités de propagation des erreurs sur chacune des portes amont à cette sortie, en déduit les sites de pannes potentiels. Chaque ligne traversée par le processus de traçage est considérée comme une source possible de l'erreur observée en sortie. L'avantage principal de cette approche est la faible quantité d'informations manipulées, puisqu'aucun dictionnaire de fautes n'est

nécessaire [Ven01]. D'autre part, en n'utilisant pas de processus de simulation de fautes, mais uniquement un mécanisme d'analyse de la propagation des erreurs (CPT), les problèmes de modélisation de fautes de l'approche « *Cause-à-Effet* » sont à priori évités. Cette technique peut donc être utilisée indifféremment soit pour le diagnostic d'un type de fautes ciblé (diagnostic de fautes de collage [Abra83], diagnostic de fautes de délai [Gir92a/b], diagnostic de fautes de court-circuit [Gho00], soit pour le diagnostic multi-modèles de fautes dans lequel plusieurs modèles de fautes peuvent être manipulés simultanément. Une étude dans ce sens a d'ailleurs été menée au LIRMM dans le cadre de la thèse d'Alexandre Rousset [Rou08]. A l'inverse, l'inconvénient de l'approche « *Effet-à-Cause* » est qu'elle n'est pas facilement applicable pour le traitement des circuits séquentiels, ainsi que pour le traitement des fautes multiples [Kit91].

Une autre approche basée sur le concept de modèles de fautes composites a été proposée récemment pour permettre le traitement simultané de plusieurs modèles de fautes lors du processus de diagnostic. Dans cette approche, on considère que le comportement de plusieurs modèles de fautes complexes (par exemple le modèle de faute de court-circuit) peut être modélisé par des fautes de collage multiples [Lav98]. Un exemple est l'approche SLAT (*Single Location At a Time*) [Hui04] et l'extension présentée dans [Hol07]. Celle-ci est basée sur un processus d'analyse de la propagation des erreurs effectué par l'intermédiaire d'un simulateur de fautes de collage. L'analyse de la propagation d'erreurs à partir de chacune des lignes du circuit et la comparaison avec les réponses du circuit obtenues sur le testeur permettent de fournir une liste de lignes potentiellement fautives indépendamment du type de défaillance pouvant affecter cette ligne. Cette approche parait efficace et très prometteuse pour traiter les divers types de défaillances pouvant affecter les circuits, qu'il s'agisse d'ailleurs de pannes simples ou multiples. Cependant, elle présente encore les inconvénients inhérents aux approches « *Cause-à-Effet* » (volume de données à manipuler) mais surtout ne permet pas d'identifier le type de défaillance pouvant être à l'origine de l'erreur produite sur les lignes trouvées comme étant potentiellement fautives.

Ce mémoire sera donc organisé de la manière suivante. **Dans le premier chapitre**, nous présenterons les modèles de fautes qui sont pris en compte dans les techniques de diagnostic développées, ainsi que les conditions de sensibilisation associées à chacun de ces modèles.

Dans le deuxième chapitre, nous montrerons comment il est possible de traiter efficacement des circuits complexes (de type « système-sur-puce ») en utilisant une technique de diagnostic combinant les approches « *Cause-à-Effet* » et « *Effet-à-Cause* ». Dans cette étude, nous avons considéré comme point de départ la technique de diagnostic développée dans le cadre de la thèse d'Alexandre Rousset [Rou08]. Cette technique, qui reprend le principe d'analyse de la propagation des erreurs potentielles évoqué précédemment, s'affranchit d'une manipulation explicite des différents modèles de fautes. Cette technique a ensuite été modifiée et adaptée pour permettre le traitement des systèmes sur puce. Les résultats expérimentaux obtenus sur des circuits type fournis par la société STMicroelectronics seront présentés et commentés en fin de chapitre.

Dans le troisième chapitre, nous présenterons une nouvelle technique de diagnostic permettant de traiter à la fois les circuits combinatoires et les circuits séquentiels. Cette technique est basée sur une approche « *Cause-à-Effet* » et comprend deux phases : une phase de localisation des sites de pannes potentiels qui s'appuie sur le paradigme SLAT, et une phase d'affectation des modèles de fautes aux sites de pannes potentiels qui fait appel aux conditions de sensibilisation de pannes détaillées dans le Chapitre I. Dans la suite du chapitre, le flot d'adaptation de la technique proposée, ainsi que les résultats expérimentaux obtenus sur des circuits de référence pour des évaluations (appelés benchmark dans la suite de ce manuscrit) et sur des systèmes sur puce fournis par la société STMicroelectronics seront présentés et commentés.

Enfin, nous terminerons ce mémoire par une conclusion générale qui résumera l'ensemble des contributions des différents chapitres et qui proposera des perspectives de travail.

Notons que mes travaux de thèse ont été réalisés dans le cadre du projet Européen CEE 2A702 MEDEA+ NanoTEST. Ma présence à STMicroelectronics à Crolles (38) m'a permis d'une part de m'adapter au monde industriel et de maîtriser ainsi les différents outils indispensables pour le bon déroulement de ma thèse. D'autre part, j'avais à ma disposition plusieurs circuits industriels conçus avec des technologies récentes, que j'ai exploités dans le but de valider l'ensemble de mes travaux de recherches. Ces derniers ont fait l'objet de plusieurs publications dans des conférences internationales majeures.

Chapitre I

Modèles de Fautes pour le Diagnostic

Plan

1 Modèles de fautes 16
1.1 Fautes de collage (stuck-at) 17
1.2 Fautes de court-circuit (bridge) 17
1.2.1 Dominant Or 19
1.2.2 Dominant And 20
1.2.3 Strong Driver Wired 20
1.2.4 Wired Or 21
1.2.5 Wired And 22
1.2.6 Byzantine 22
1.2.7 Cas de Fautes de court-circuit résistif 23
1.3 Fautes de circuit ouvert 25
1.3.1 Circuit ouvert franc (open) 25
1.3.2 Circuit ouvert résistif (resistive open) 25
1.4 Fautes de délai (transition) 26
1.5 Fautes de stuck-on/stuck-open 27
1.6 Fautes multiples 28
2 Conclusion 29

Les informations généralement disponibles et exploitées par les processus de diagnostic logique sont : (i) les vecteurs de test appliqués au circuit, (ii) les réponses « pass/fail » issues du testeur et obtenues pour chaque vecteur de test appliqué au circuit et (iii) la description structurelle du circuit.

Afin d'obtenir des indications suffisamment précises sur l'origine et la nature des défaillances à diagnostiquer dans un circuit, d'autres informations sont nécessaires. En particulier, les modèles de fautes à manipuler et les conditions de sensibilisation associées à ces modèles sont des informations que les outils de diagnostic doivent posséder.

L'objectif de ce chapitre est donc de présenter l'ensemble des modèles de fautes qui seront pris en compte et manipulés par les techniques de diagnostic proposées dans les chapitres suivants. Les conditions de sensibilisation associées à chacun de ces modèles de fautes sont également détaillées.

1.1 Modèles de fautes

Les défaillances affectant les circuits peuvent être de natures très diverses (défauts localisés ou répartis, variations des paramètres des transistors dues à des variations du processus de fabrication, ...) et entraîner des comportements également divers (erreurs logiques, retards, déviations de paramètres électriques, …). Afin de pouvoir être traités par les outils de simulation de fautes, de génération de vecteurs de test, ou de diagnostic, les comportements engendrés par ces défauts sont généralement représentés par des modèles. Ces modèles traduisent des déviations du comportement du circuit qui peuvent être logiques (collages, courts-circuits, circuits ouverts), temporelles (pannes de délai de porte/chemin) ou paramétriques (courts-circuits résistifs, circuits ouverts résistifs). D'autre part, certaines défaillances affectent le comportement statique du circuit, d'autres affectent le comportement dynamique. Les pannes affectant le comportement dynamique telles que pannes temporelles ou pannes de type « transistor stuck-open » nécessitent l'application d'une séquence de vecteurs pour être détectées. Ainsi, afin de pouvoir réaliser un diagnostic le plus fiable et précis possible, il est fondamental de pouvoir considérer le plus large éventail possible de défaillances et d'effets induits par ces défaillances, et par conséquent de prendre en compte dans l'outil de diagnostic, un ensemble de modèles de fautes représentant au mieux tous ces comportements. Dans le cadre de cette étude, nous considérons l'ensemble des comportements représentés par les modèles de pannes les plus couramment utilisés [Abra83, Che06, Gho00], à savoir les modèles de :

- Collages
- Courts-circuits
- Courts-circuits résistifs
- Circuits ouverts
- Circuits ouverts résistifs
- Délais de porte (de type lent-à-monter et/ou lent-à-descendre) et par extension, de chemins ou de segments de chemins

- Stuck-on et Stuck-open (transistors N ou P)

Ces modèles peuvent affecter le circuit de manière plus ou moins différente et sont surtout sensibilisés par des conditions particulières. Dans la suite de ce chapitre, nous étudierons le comportement de ces modèles de fautes, et en particulier les conditions de sensibilisation de chacun d'entre eux.

1.1.1 Fautes de collage (stuck-at)

Une faute de collage (simple) affecte une connexion (ou une partie de connexion) unique lorsque cette connexion (ou une partie de connexion) est figée à une valeur logique (0 ou 1) quelle que soit la valeur imposée par la porte amont (Figure 1.1). Deux types de collage, sont à différencier, le collage à 0 (Ca0) et le collage à 1 (Ca1). **Condition de sensibilisation :** La condition de sensibilisation d'une telle panne est l'application d'une valeur logique opposée au collage sur le site de la panne.

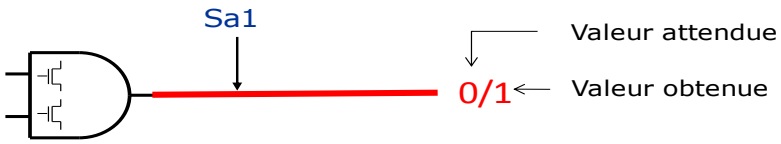


Figure 1.1 : Exemple de collage à 1 (Ca1) d'une sortie de porte ET

1.1.2 Fautes de court-circuit (bridge)

Plusieurs modèles de court-circuit ont été reportés dans la littérature (Dominant OR/AND, Strong Driver Wired, Wired AND/OR, Byzantine, courts-circuits résistifs). Chacun de ces modèles représente un comportement particulier pouvant être induit par la mise en court-circuit de deux lignes. Parmi ces modèles nous pouvons différencier :

- Ceux qui provoquent une erreur sur une seule des deux lignes en court circuit (erreur simple) pour une condition de sensibilisation unique (Dominant Or, Dominant And). Dans ce cas, l'erreur produite sur la ligne est systématiquement soit de polarité 0/1, soit 1/0, où la première valeur représente la valeur attendue et la deuxième représente la valeur obtenue.

- Ceux qui provoquent une erreur sur une seule des deux lignes en court circuit (erreur simple) pour deux conditions de sensibilisation différentes (Strong Driver Wired). Dans ce cas l'erreur produite sur la ligne peut être de polarité 0/1 et 1/0.
- Ceux qui provoquent une erreur sur l'une ou l'autre des deux lignes en court circuit mais pas sur les deux (erreur simple) pour deux conditions de sensibilisation différentes (Wired Or, Wired And [Lo00]).
- Ceux qui provoquent une erreur sur les deux lignes en court circuit (erreur double) (Byzantine [Han03]).

Le Tableau 1.1 résume les conditions de sensibilisation ainsi que les effets de chacun des modèles de court-circuit cités précédemment. Ces conditions seront explicitées à la suite de ce tableau.

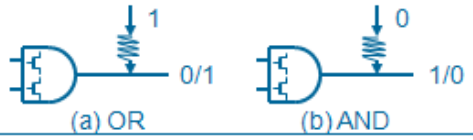
Dominant		**1 site d'erreur** 1 polarité
Strong Driver Wired		**1 site d'erreur** 2 polarités
Wired-OR		**2 sites d'erreurs** 1 polarité
Wired-AND		**2 sites d'erreurs** 1 polarité
Byzantine		**2 sites d'erreurs** 2 polarités

Tableau 1.1 : Différents modèles de courts-circuits

Un modèle général de court-circuit permettant de préciser les conditions de sensibilisation ainsi que les effets induits (erreurs) par les courts-circuits est proposé sur la Figure 1.2. x et y représentent les valeurs respectives que devraient avoir les deux lignes en court circuit dans le

circuit sain. La fonction réalisée par le court-circuit est notée Z(x,y). Les sorties (x' et y') de cette fonction représentent les valeurs effectives qu'ont les deux lignes en présence du court-circuit considéré.

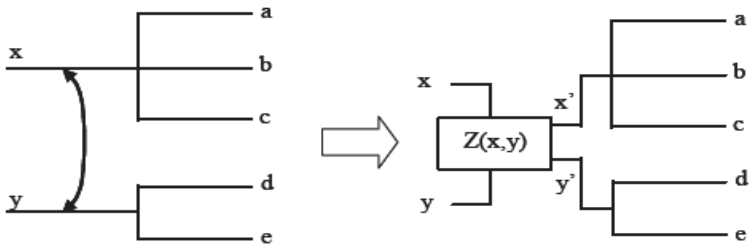


Figure 1.2 : Modèle générale de court-circuit

1.1.2.1 Dominant Or

Un court-circuit de type ***Dominant Or*** correspond à une panne telle qu'une ligne x positionnée à '1' impose sa valeur à une ligne y positionnée à '0' (Figure 1.3). On dit que la ligne x est l'*agresseur* et que la ligne y est la *victime*.

Conditions de sensibilisation : avoir des valeurs opposées entre l'agresseur et la victime. La ligne victime doit avoir la valeur logique '0' et la ligne agresseur la valeur logique '1'.

Les conditions de sensibilisation et le comportement de ce type de court-circuit peuvent être modélisés de la manière suivante : soient la fonction Z(x,y) avec x et y, deux lignes distinctes du circuit :

Si (x,y) = (1,0), alors Z(x,y) = (1,1) sinon Z(x,y) = (x,y)

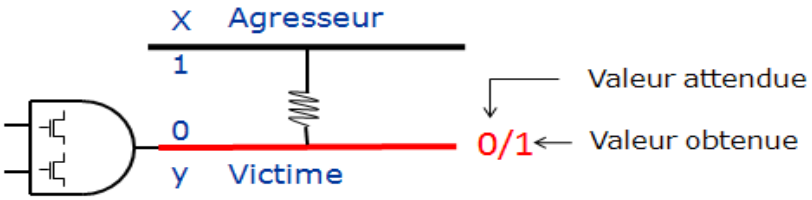


Figure 1.3 : Court-circuit de type Dominant Or

1.1.2.2 Dominant And

Un court-circuit de type ***Dominant And*** correspond à une panne telle qu'une ligne x positionnée à '0' impose sa valeur à une ligne y positionnée à '1' (Figure 1.4)

On dit que la ligne x est l'*agresseur* et que la ligne y est la *victime.*

Conditions de sensibilisation : avoir des valeurs opposées entre l'agresseur et la victime. La ligne victime doit avoir la valeur logique '1' et la ligne agresseur la valeur logique '0'.

Les conditions de sensibilisation et le comportement de ce type de court-circuit peuvent être modélisés de la manière suivante : soient la fonction Z(x,y) avec x et y, deux lignes distinctes du circuit :

Si (x,y) = (0,1), alors Z(x,y) = (0,0) sinon Z(x,y) = (x,y)

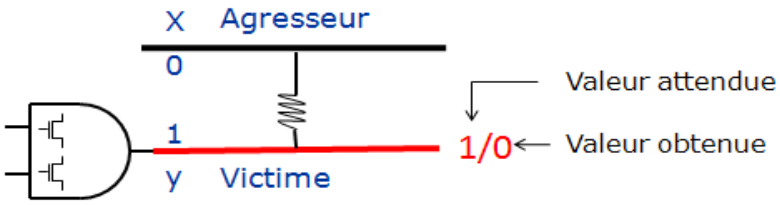


Figure 1.4 : Court-circuit de type Dominant And

1.1.2.3 Strong Driver Wired

Le court-circuit de type ***Strong Driver Wired*** **(SDW)** (Figure 1.5) produit une erreur double sur une seule des deux connexions en court-circuit mais l'erreur peut être de polarité différente (0/1 ou 1/0) en fonction des valeurs respectives des deux lignes. La valeur de la ligne *agresseur* est toujours imposée à la ligne *victime.*

Condition de sensibilisation : avoir des valeurs logiques opposées entre l'agresseur et la victime.

Les conditions de sensibilisation et le comportement de ce type de court-circuit peuvent être modélisés de la manière suivante : soient la fonction Z(x,y) avec x et y, deux lignes distinctes du circuit :

Si $x \neq y$, alors Z(x,y) = (x,x) sinon Z(x,y) = (x,y)

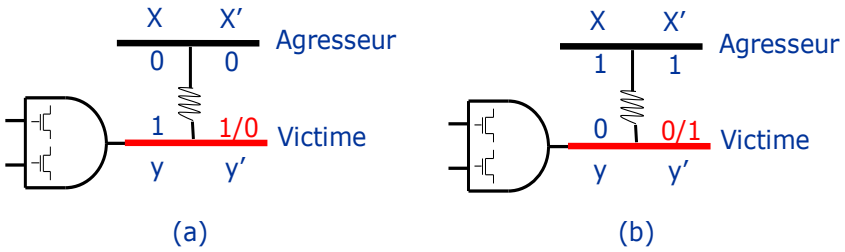


Figure 1.5 : Court-circuit de type Strong Driver Wired

1.1.2.4 Wired Or

Le court-circuit de type ***Wired Or*** (Figure 1.6) produit une erreur simple et sur une seule des deux connexions à la fois. Néanmoins, ce type de court-circuit peut affecter chacune des deux connexions. Dans ce cas là, il n'y a pas de lignes victime et agresseur. Si les deux lignes court-circuitées ont des valeurs logiques opposées, alors la ligne ayant la valeur logique '1' imposera sa valeur à la ligne ayant la valeur logique '0'.

Condition de sensibilisation : avoir des valeurs logiques opposées entre l'agresseur et la victime.

Les conditions de sensibilisation et le comportement de ce type de court-circuit peuvent être modélisés de la manière suivante : soient la fonction Z(x,y) avec x et y, deux lignes distinctes du circuit :

Si (x,y) = (1,0), alors Z(x,y) = (1,1) sinon Z(x,y) = (x,y)

Ou si (x,y) = (0,1), alors Z(x,y) = (1,1) sinon Z(x,y) = (x,y)

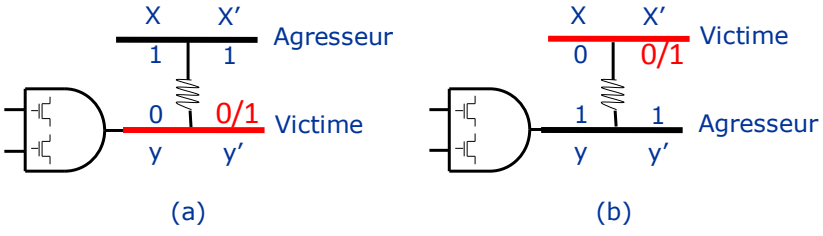


Figure 1.6 : Court-circuit de type Wired Or

1.1.2.5 Wired And

Le court-circuit de type ***Wired And*** (Figure 1.7) produit une erreur simple et sur une seule des deux connexions à la fois. Néanmoins ce type de court-circuit peut affecter chacune des deux connexions. Dans ce cas là, il n'y a pas de lignes victime et agresseur. Si les deux lignes court-circuitées ont des valeurs logiques opposées, alors la ligne ayant la valeur logique '0' imposera sa valeur à la ligne ayant la valeur logique '1'.

Condition de sensibilisation : avoir des valeurs logiques opposées entre l'agresseur et la victime.

Les conditions de sensibilisation et le comportement de ce type de court-circuit peuvent être modélisés de la manière suivante : soient la fonction Z(x,y) avec x et y, deux lignes distinctes du circuit :

Si (x,y) = (1,0), alors Z(x,y) = (0,0) sinon Z(x,y) = (x,y)

Ou si (x,y) = (0,1), alors Z(x,y) = (0,0) sinon Z(x,y) = (x,y)

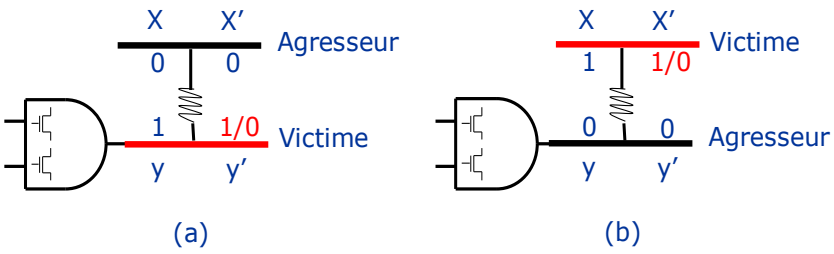


Figure 1.7 : Court-circuit de type Wired And

1.1.2.6 Byzantine

Le court-circuit de type ***Byzantine*** (Figure 1.8) produit une erreur double sur les deux lignes court-circuitées. Les valeurs logiques de chaque ligne court-circuitée sont respectivement inversées, c'est-à-dire que pour un même vecteur une des deux lignes prendra la valeur '0' (la ligne « a » par exemple) alors qu'à l'origine un '1' était attendu, et inversement pour la seconde ligne (la ligne « b »). Mais une analyse électrique de ce type de panne permet de montrer que cette panne n'est valable que dans un seul sens. Admettons

maintenant que la ligne « b » ait la valeur '1' et la ligne « a » ait la valeur '0', aucune erreur n'apparaîtra sur ces lignes. La Figure 1.8 montre donc les 2 cas possibles.

Condition de sensibilisation : avoir des valeurs logiques opposées entre l'agresseur et la victime.

Les conditions de sensibilisation et le comportement de ce type de court-circuit peuvent être modélisés de la manière suivante : soient la fonction Z(x,y) avec x et y, deux lignes distinctes du circuit :

$$\text{Si } x \neq y, \text{ alors } Z(x,y) = (y,x) \text{ sinon } Z(x,y) = (x,y)$$

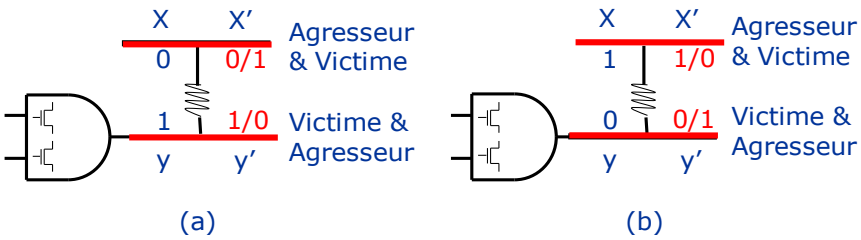


Figure 1.8 : Court-circuit de type Byzantine

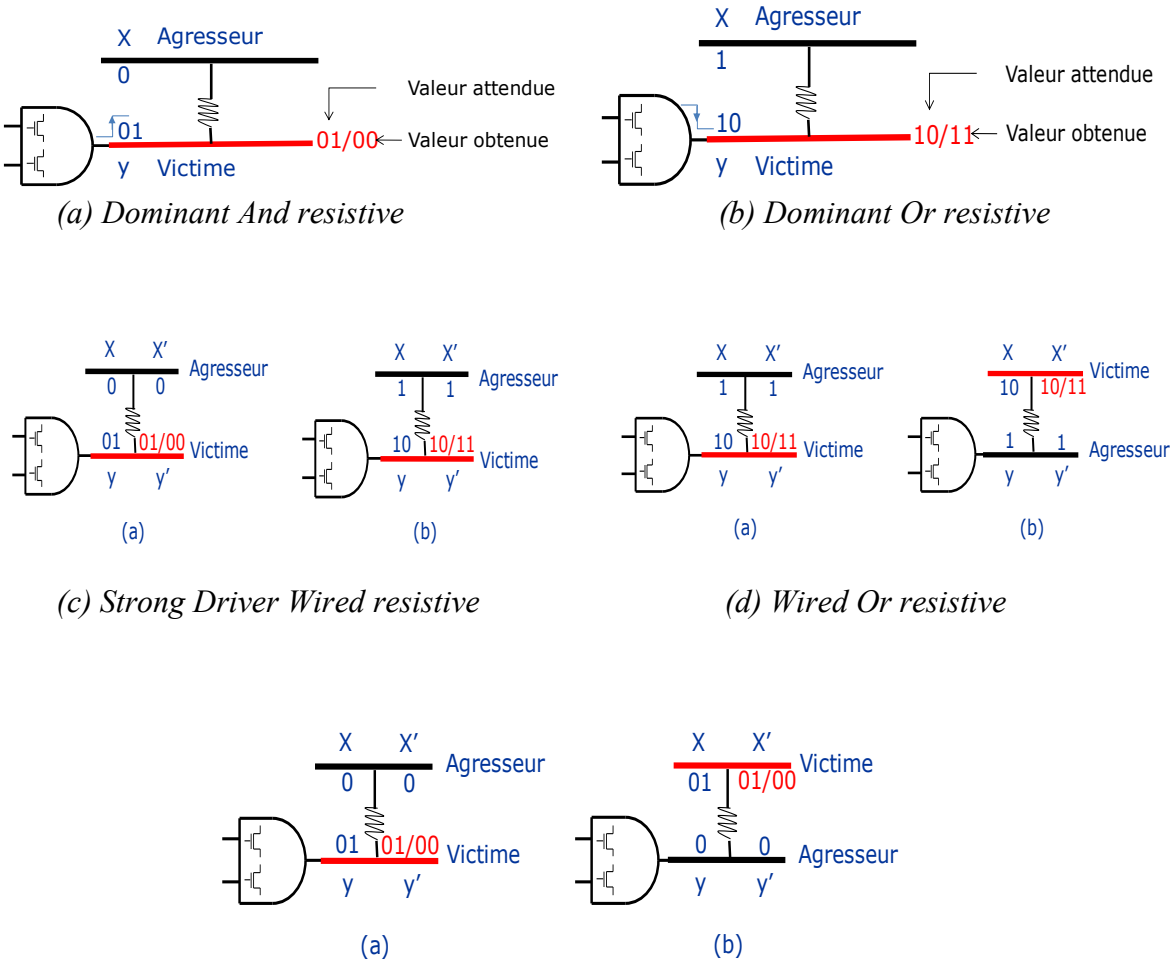
1.1.2.7 Cas de Fautes de court-circuit résistif

Un court-circuit peut être franc ou résistif. Lorsqu'il est résistif, il peut avoir des comportements différents en fonction de la valeur de la résistance. En effet, lorsque la résistance est nulle (faible), nous sommes dans le cas des courts circuits francs traités précédemment. Lorsqu'elle est infinie (importante), nous pouvons considérer qu'il n'y a pas de panne ou que cette panne ne peut pas se manifester lors du test de production. Entre ces deux extrêmes, une panne de court-circuit peut affecter le fonctionnement du circuit sur sa fonctionnalité dynamique, c'est-à-dire en retardant certaines transitions. Dans ce cas, ces pannes sont assimilables à des pannes de délai.

Ainsi, par rapport aux conditions de sensibilisation développées pour les courts circuits francs, la seule différence à introduire pour les courts-circuits résistifs concerne la sensibilisation de la ligne «victime ». En effet, pour que ce court-circuit résistif ait un effet

(différent de celui du court-circuit franc) une transition doit être présente sur la ligne « victime ». L'effet du court-circuit sera alors de retarder cette transition.

Les différents cas sont représentés sur la Figure 1.9. Nous pouvons voir que ce sont les mêmes cas que pour les courts-circuits francs à la seule différence que la ligne victime est sensibilisée par une transition et que l'observation de la panne se traduira par un retard de commutation.



(a) Dominant And resistive

(b) Dominant Or resistive

(c) Strong Driver Wired resistive

(d) Wired Or resistive

(e) Wired And resistive

Figure 1.9: Cas des circuits résistifs

1.1.3 Fautes de circuit ouvert

1.1.3.1 Circuit ouvert franc (open)

Les fautes de circuit ouvert franc sont assimilables aux fautes de collage en terme de comportement. En effet, une ligne ouverte est ramenée à un potentiel qui peut être la masse, équivalent à un '0' logique permanent, quelle que soit la valeur appliquée sur cette ligne, ou à Vdd, équivalent à un '1' logique permanent (Figure 1.10).

Condition de sensibilisation : application d'une valeur logique opposée à celle engendrée par la panne.

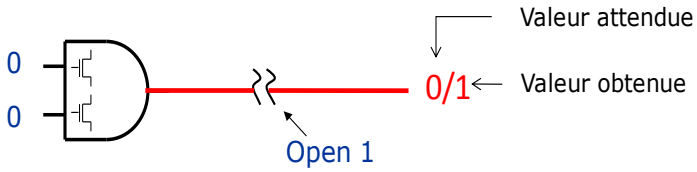


Figure 1.10 : Exemple d'un circuit ouvert à 1

1.1.3.2 Circuit ouvert résistif (resistive open)

Comme pour les fautes de courts-circuits résistifs, les fautes de circuits ouverts résistifs peuvent induire un dysfonctionnement dynamique du circuit en retardant des transitions. En effet, une ligne ouverte peut être vue comme une résistance qui ralentit la transition. Elle est ainsi assimilable à une faute de délai. Ce type de faute peut affecter les transitions montantes (lent-à-monter) (Figure 1.11), les transitions descendantes (lent-à-descendre) ou les deux à la fois.

Condition de sensibilisation : application d'une paire de vecteurs provoquant une transition (montante pour lent-à-monter, **ou/et** descendante pour lent-à-descendre) sur le site de la panne.

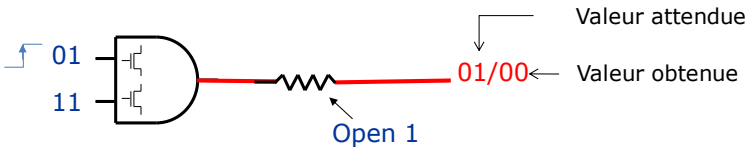


Figure 1.11 : Exemple d'un circuit ouvert résistif entraînant une transition « lent-à-monter »

1.1.4 Fautes de délai (transition)

Lorsqu'une panne temporelle est à l'origine du mauvais fonctionnement d'un circuit, cette panne peut soit être issue d'un défaut physique localisé sur un site particulier, soit être la conséquence de plusieurs défauts accumulés le long d'un chemin de propagation. Par conséquent, la modélisation des défauts d'origine temporelle a donné lieu à une classification suivant l'aspect local ou global du défaut. Les deux modèles locaux sont le modèle de *faute de délai de porte* [Bar83, Car87, Hsi77, Kis86, Pra88], et le modèle de *faute de transition* [Lev86, Sch87, Wai86].

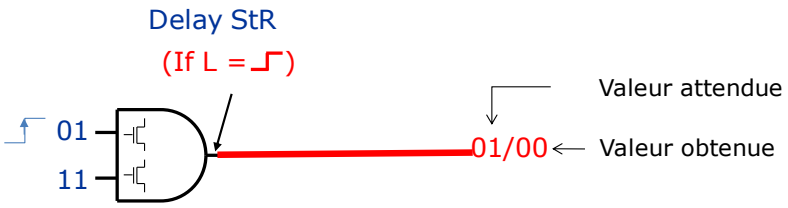


Figure 1.12 : Exemple d'un délai « lent-à-monter » (Slow-to-Rise) sur une porte ET

Le modèle global est le modèle de *faute de délai de chemin* [Smi85]. Quel que soit le modèle de faute de délai de chemin utilisé, il y a toujours deux fautes de délai associées à un même site : une faute pour la transition montante (de type « *lent à monter* » (Slow-to-Fall: StF), Figure1.12) et une autre sur la transition descendante (de type « *lent à descendre* » (Slow-to-Rise: StR), Figure 1.13).

Condition de sensibilisation : application d'une paire de vecteurs provoquant une transition (montante pour « lent-à-monter », et descendante pour «lent-à-descendre ») sur le site de la panne.

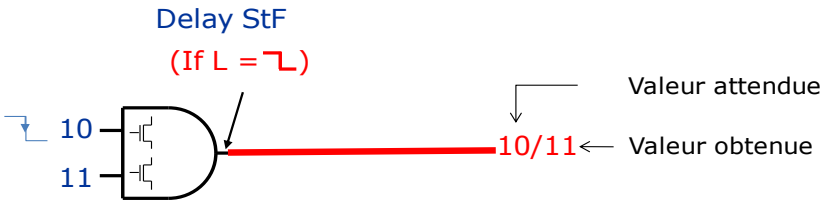


Figure 1.13 : Exemple d'un délai « lent-à-descendre » sur une porte ET

1.1.5 Fautes de stuck-on/stuck-open

Le modèle de stuck-on/open suppose que l'un des transistors de la porte logique se trouve dans un état permanent passant (stuck-on) ou ouvert (stuck-open) [Meno93, Wood87]. La Figure 1.14 montre le cas d'une porte logique NAND dont le transistor T1 est stuck-open. Les fautes de stuck-on/open peuvent engendrer un comportement séquentiel et requièrent donc une séquence de vecteurs pour être détectées. Considérant le cas de la porte NAND dont le transistor T1 est stuck-open (Figure 1.14), une séquence de deux vecteurs doit être appliquée. En effet, considérons tous les vecteurs possibles qui peuvent être appliqués et qui sont donnés dans le Tableau 1.2.

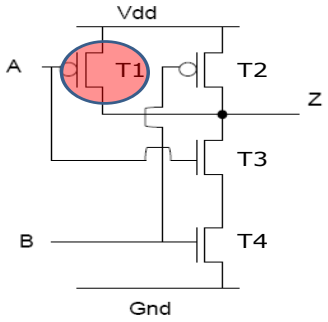


Figure 1.14 : Transistor T1 stuck-open

Vecteur	Valeur Précédente de Z	A	B	Z attendu	Z observé
1	0	0	0	1	1
2	0	0	1	1	**0**
3	0	1	0	1	1
4	0	1	1	0	0
5	1	0	0	1	1
6	1	0	1	1	1
7	1	1	0	1	1
8	1	1	1	0	0

Tableau 1.2 : Comportement logique en présence d'un Stuck-open

La deuxième colonne représente la valeur logique précédente de la sortie Z. La cinquième colonne donne les valeurs logiques attendues à la sortie de la porte NAND, tandis que la

sixième colonne fournit les valeurs logiques obtenues en réalité sur la sortie Z en présence de la faute. Ce tableau montre bien qu'une paire de vecteurs est nécessaire à la sensibilisation et à la propagation d'une faute de stuck-open puisque dans ce cas précis la valeur logique précédente du vecteur 2 (ligne 2 du Tableau 1.2), est nécessaire pour mettre en évidence le stuck-open du transistor T1. Par contre, si la valeur précédente du Z = 1 (ligne 6 du Tableau 1.2), il n'y a aucun changement observé sur la sortie Z.

Condition de sensibilisation : Application d'une paire de vecteurs pour provoquer une transition sur la sortie (effet séquentiel).

1.1.6 Fautes multiples

L'ensemble des modèles de fautes considérés montre bien que chaque modèle est sensibilisé de façon différente, en effet :

- Les modèles de fautes assimilables à des fautes de délai (court-circuit résistif, circuit ouvert résistif, délai) ou ayant un comportement séquentiel (stuck-on/open) nécessitent l'application d'une paire de vecteurs pour sensibiliser la faute.

- Les modèles statiques tels que court-circuit, collage, circuit ouvert, ne nécessitent qu'un seul vecteur pour être sensibilisés, mais parfois d'autres conditions sont nécessaires comme dans le cas des modèles de courts-circuits où l'agresseur doit avoir une certaine valeur logique.

- L'ensemble des modèles de courts-circuits étudié nécessite aussi d'avoir des valeurs logiques qui soient opposées entre les deux lignes court-circuitées.

Lorsque plusieurs pannes sont à l'origine du mauvais fonctionnement d'un circuit, ces pannes peuvent soit être issue d'un ou plusieurs défaut physique localisé sur un ou plusieurs sites particulier, soit être la conséquence de plusieurs défauts statiques et/ou dynamiques.

Condition de sensibilisation : application de toutes les conditions de sensibilisation vues auparavant. La complexité d'identifier les défauts multiples est relative à la propagation, certains défauts peuvent en masquer d'autres. La propagation de défauts multiples est différente par rapport à celle d'un défaut simple.

1.2 Conclusion

Dans ce chapitre, nous avons présenté les modèles de fautes qui seront pris en compte et manipulés par les techniques de diagnostic proposées dans les chapitres suivants. Ces modèles de fautes représentent des comportements variés de défaillances. La diversité de ces comportements implique un mode de sensibilisation approprié selon le modèle de faute considéré.

Nous avons pu constater que certains modèles de fautes ne provoquent qu'une seule erreur dans le circuit, et n'affectent qu'une seule ligne (cas d'une panne simple), alors que d'autres peuvent soit provoquer sur une même ligne une erreur pouvant avoir plusieurs polarités en fonction des conditions de sensibilisation, soit affecter plusieurs lignes du circuit (cas des erreurs multiples).

Nous pouvons en conclure que lorsque une erreur est observée en sortie du circuit, cette erreur peut être provoquée par un certain nombre de modèles de fautes en fonction des conditions de sensibilisation de chaque ligne suspecte (transition, valeur constante), mais aussi des valeurs sur les autres lignes (agresseur pour les courts-circuits). Lors du processus de diagnostic, il sera donc important de tenir compte de toutes ces conditions de sensibilisation.

Chapitre II

Diagnostic de Pannes dans les Systèmes sur Puce

Plan

1 Diagnostic logique par une approche « Effet-à-Cause » 33
1.1 Principe de la méthode 33
1.2 Phase de localisation des sites potentiels de panne 36
1.2.1 Simulation multi-valuée du circuit sain 36
1.2.2 Traçage de chemins critiques 39
1.2.3 Processus d'intersection 44
1.1.1.1 Règles d'intersection 44
1.3 Phase d'affectation des modèles de fautes 46
2 Adaptation de la méthode de diagnostic pour le traitement des systèmes sur puce 48
2.1 Description d'un système sur puce 48
2.2 Synoptique de la méthode de diagnostic adaptée 50
2.3 Modification du format des fichiers d'entrée 52
2.3.1 Modification de la description d'un système sur puce 52
2.3.2 Modifications du format des vecteurs de test 55
2.3.3 Modifications du format des réponses du testeur 57
2.4 Modification du processus de diagnostic 59
2.4.1 Modification du codage des signaux 59
2.4.2 Modification du processus de traçage de chemins critiques 62
2.4.3 Modification du processus d'intersection 63
3 Résultats expérimentaux 66
3.1 Processus expérimental 66
3.2 Les différents circuits étudiés 66

3.3 Résultats expérimentaux 67
3.4 Cas d’étude 69
4 Conclusion 71

Ce chapitre est consacré à la description d'une méthode de diagnostic de pannes dédiée au traitement des systèmes sur puce pouvant contenir des blocs de natures diverses.

Dans la première partie, nous rappellerons le principe de la méthode de diagnostic développée durant la thèse de Mr. Alexandre Rousset [Rou08], et qui a servi de point de départ à cette étude. Dans un premier temps, nous donnerons les grands principes de cette méthode basée sur l'approche « Effet-à-Cause » et destinée à traiter des blocs logiques combinatoires. Nous décrirons par la suite les différentes étapes du processus de diagnostic. Nous verrons que ce processus comprend deux phases principales : une phase de localisation des sites de pannes potentiels et une phase d'affectation des modèles de fautes.

Dans la deuxième partie, nous présenterons les modifications apportées au processus de diagnostic initial pour permettre le traitement des systèmes sur puce. Nous verrons que ces modifications interviennent à la fois au niveau du format des fichiers d'entrée manipulés par l'outil eDERRIC issu de la méthode de diagnostic initiale, ainsi qu'au niveau du processus de diagnostic proprement dit.

Enfin, dans la troisième partie, nous détaillerons les résultats expérimentaux obtenus sur différents circuits type fournis par la société STMicroelectronics. Des comparaisons avec un outil de référence seront réalisées, et nous montrerons ainsi l'intérêt de notre méthode en terme de résolution du diagnostic.

2.1 Diagnostic logique par une approche « Effet-à-Cause »

2.1.1 Principe de la méthode

Les erreurs observées lors du test d'un circuit sont le résultat de défaillances physiques affectant certains composants du circuit. Comme indiqué dans le chapitre précédent, ces défaillances sont généralement représentées par divers modèles de fautes qui peuvent modéliser soit une déviation logique (collage, courts-circuits, circuits ouverts, transistors stuck-open), soit une déviation temporelle ou paramétrique (pannes de délais, courts-circuits résistifs, circuits ouverts résistifs, transistors stuck-on). Certaines fautes (collage, courts-circuits, circuits ouverts, transistors stuck-on) affectent le comportement statique du circuit, d'autres (transistors stuck-open, pannes de délais, courts-circuits résistifs, circuits ouverts résistifs) modifient le comportement dynamique et, par conséquent, nécessitent une transition pour être sensibilisées et ainsi produire une erreur.

Quoi qu'il en soit, l'effet de ces défaillances se traduit, lors du test, par l'observation de valeurs logiques erronées sur les sorties du circuit. A partir de l'observation de ces valeurs erronées sur les sorties, le but est de remonter au site défaillant et, éventuellement, au modèle de défaillance. Cela est le principe de l'approche Effet-à-Cause. Lorsqu'une erreur est observée sur une sortie 'S_i' du circuit, cette erreur provient soit d'une défaillance sur cette sortie, soit de la propagation d'une erreur provoquée par une défaillance sur une connexion située en amont. Le résultat de diagnostic est composé de plusieurs endroits potentiels de la panne.

A partir de cette constatation, nous avons établi un processus de diagnostic scindé en deux phases. Le synoptique du processus de diagnostic ainsi développé est présenté dans la Figure 2.1. La phase de « localisation » a pour entrée la description structurelle du circuit (CUT), la séquence de vecteurs de test et les réponses du circuit à ces vecteurs. Elle fournit une liste de suspects (lignes suspectées d'être à l'origine de l'erreur observée) ainsi que des informations spécifiques concernant les valeurs propagées dans le circuit. La liste des suspects et ces

informations sont exploitées lors de la phase « d'affectation des modèles de fautes » qui produit le rapport du diagnostic final.

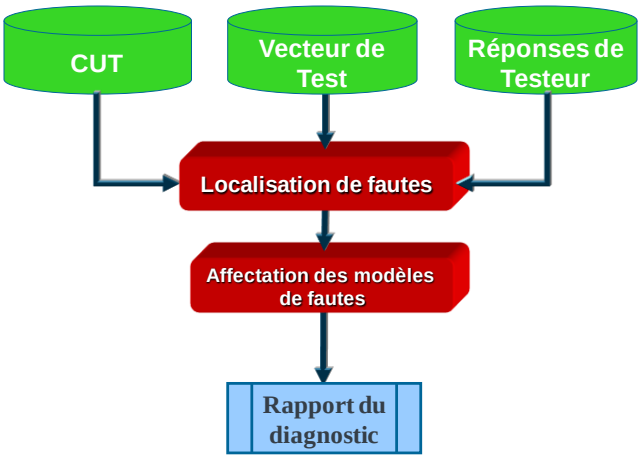


Figure 2.1 : Synoptique du processus de diagnostic développé

La première phase du processus de diagnostic, appelée « phase de localisation » est une phase de recherche des sites potentiels d'erreurs ayant pu produire les erreurs observées en sortie (Figure 2.2). Cette phase requiert la connaissance (i) de la description du circuit exprimée au niveau porte, (ii) de vecteurs de test appliqués au circuit, (iii) de réponses du circuit à ces vecteurs de test (réponses fournies par le testeur). Le résultat de cette première étape est une liste de sites potentiels des erreurs pouvant expliquer les erreurs observées en sortie ainsi que des informations concernant les valeurs logiques propagées dans le circuit.

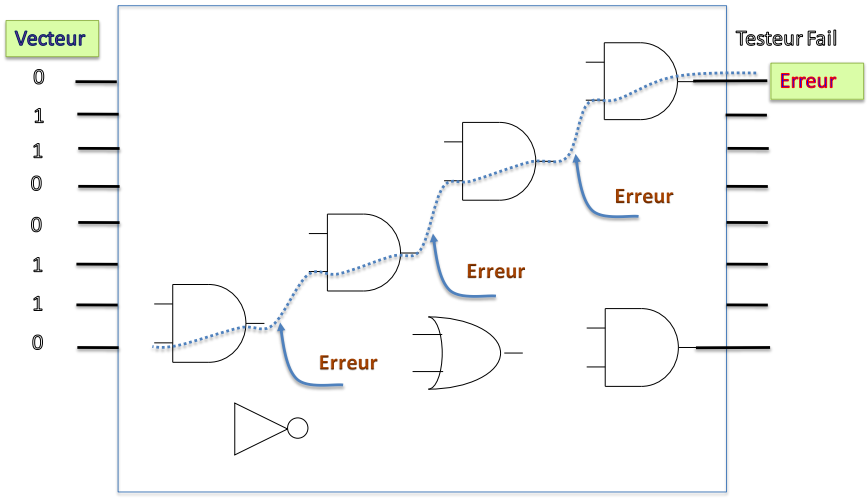


Figure 2.2 : Sites potentiels des erreurs ayant pu produire l'erreur observée en sortie

La seconde phase du processus de diagnostic développé appelée « phase d'affectation de modèles » est une phase de définition des modèles de pannes potentiels associés à ces sites. Lorsqu'une erreur potentielle est localisée sur la ligne L, cette erreur peut être engendrée par divers défauts qui peuvent être identifiés grâce à des informations telles que : (i) la valeur logique attendue sur la ligne L, (ii) les valeurs logiques attendues sur les autres lignes du circuit (pannes de court-circuit), (iii) les transitions produites sur la ligne L (pannes dynamiques telles que pannes de délai et stuck-open). Ainsi, il apparaît que ces modèles de fautes peuvent être déterminés à partir de la connaissance du fonctionnement du circuit sain.

A titre d'exemple, si un '1' est observé sur une ligne L alors qu'un '0' est attendu (Figure 2.3), cette erreur peut provenir :

- D'un collage à 1 (Stuck-at 1 « Sa1 ») de L,
- D'une coupure de type (Open 1) de la ligne L,
- D'un Stuck-on sur un transistor P de la porte amont,
- D'une panne de délai « *lent à descendre* » (Delay StF) sur L (uniquement si une commutation se produit sur L),
- D'un court-circuit (Bridge: Wired Or) entre L et toute ligne à 1 (si une commutation se produit sur L),
- D'un circuit ouvert résistif (Resistive Open) sur L (uniquement si une commutation se produit sur L),
- D'un Stuck-open sur un transistor N de la porte amont.

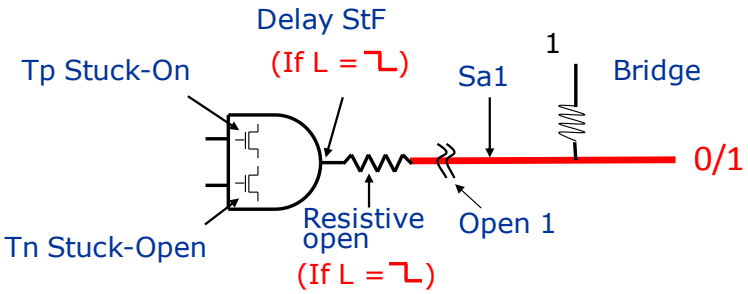


Figure 2.3 : Défauts susceptibles d être à l'origine d'une erreur

2.1.2 Phase de localisation des sites potentiels de panne

L'objectif de cette première phase du processus de diagnostic est de déterminer l'ensemble de sites de pannes potentiels, c'est-à-dire l'ensemble de lignes potentiellement suspectes.

Une analyse « Effet-à-Cause » basée sur le principe de traçage de chemins critiques est réalisée, dans un premier temps, pour déterminer les sites de pannes potentiels. Cette analyse comporte les trois étapes suivantes :

- Simulation mutli-valuée du circuit sain,
- Traçage de chemins critiques (Critical Path Tracing CPT [Abra80] [Abra83]) à partir des sorties déclarées fautives (fail) lors du test,
- Intersection des listes de suspects obtenues par le processus de traçage de chemins.

2.1.2.1 Simulation multi-valuée du circuit sain

La simulation multi-valuée est une simulation symbolique qui considère deux vecteurs de test successifs pour traiter plusieurs modèles de fautes à la fois (statiques et dynamiques). Elle permet de déterminer, pour chaque vecteur de test conduisant à l'observation d'une erreur en sortie, les informations logiques utiles au processus de traçage de chemins critiques. Afin de faire apparaître, outre les valeurs logiques, les commutations ou possibilités de commutations sur les diverses lignes du circuit, cette simulation met en œuvre une algèbre multi-valuée.

Nous avons vu dans le chapitre précédent que les conditions de sensibilisation variaient en fonction du modèle de faute incriminé. En effet, pour les fautes statiques telles que le collage, court-circuit, circuit ouvert, un seul vecteur est nécessaire à la sensibilisation de la panne. Par contre, les fautes dynamiques (délai, court-circuit résistif, circuit ouvert résistif, stuck-open) ont besoin d'une paire de vecteurs pour les sensibiliser. L'algèbre à six valeurs (C0, C1, F0, R1, P0, P1) développé dans [Gira92a, Gira92b] permet de satisfaire toutes les exigences de sensibilisation précédemment citées. Dans le cadre de cette étude, nous utiliserons donc cette algèbre à six valeurs :

- C0 (C1) représente un signal stable à 0 (1) quels que soient les délais présents dans le circuit (délais inhérents au circuit ou délais provenant d'une panne).

- R1 (F0) modélise un signal dont la valeur initiale est 0 (1) et la valeur finale est 1 (0). Ce signal peut présenter plusieurs transitions mais la dernière est nécessairement une transition montante (descendante).

- P0 (P1) représente un signal de valeur initiale et finale à 0 (1) mais présentant une possibilité d'aléas logiques à 1 (0) en fonction des valeurs des paramètres temporels du circuit ou des pannes temporelles affectant ce circuit.

Ce codage permet de représenter les valeurs logiques produites dans le circuit par le vecteur de test ayant produit l'erreur (0, 1), les transitions générées par ce vecteur (R1, F0) ou les possibilités de transitions (P0, P1), sans avoir à réaliser l'analyse temporelle du circuit.

Nous pouvons remarquer qu'une algèbre à quatre valeurs (C0, C1, F0, R1) aurait pu être suffisante pour représenter les transitions. Néanmoins, disposer des possibilités de transition peut être intéressant, notamment pour affiner le traitement des pannes du délai. Les signaux à appliquer sur les entrées du circuit lors de la simulation sont déterminés à partir du vecteur de test V_i ayant produit l'erreur et du vecteur V_{i-1} appliqué sur le circuit immédiatement avant le vecteur V_i. Ainsi, pour une entrée E donnée, la valeur à appliquer est déterminée de la manière suivante :

- $V_{i-1}(E) = 0, V_i(E) = 0 \Rightarrow E = C0$
- $V_{i-1}(E) = 1, V_i(E) = 1 \Rightarrow E = C1$
- $V_{i-1}(E) = 0, V_i(E) = 1 \Rightarrow E = R1$
- $V_{i-1}(E) = 1, V_i(E) = 0 \Rightarrow E = F0$

Une fois les entrées déterminées, le processus de simulation revient à propager ces signaux dans le circuit sain en utilisant les tables de transfert associées aux portes logiques. Les tables de transfert des portes ET (AND), OU (OR), Inverseur (NOT) sont présentées dans le Tableau 2.1. Celles, associées aux autres types de portes, se déduisent aisément de celles-ci.

AND	**C0**	**C1**	**F0**	**R1**	**P0**	**P1**
C0	C0	C0	C0	C0	C0	C0
C1	C0	C1	F0	R1	P0	P1
F0	C0	F0	F0	P0	P0	F0
R1	C0	R1	P0	R1	P0	R1
P0	C0	P0	P0	P0	P0	P0
P1	C0	P1	F0	R1	P0	P1

OR	C0	C1	F0	R1	P0	P1
C0	C0	C1	F0	R1	P0	P1
C1	C1	C1	C1	C1	C1	C1
F0	F0	C1	F0	P1	F0	P1
R1	R1	C1	P1	R1	R1	P1
P0	P0	C1	F0	R1	P0	P1
P1	P1	C1	P1	P1	P1	P1

NOT	IN	OUT = Not (IN)
	C0	C1
	C1	C0
	F0	R1
	R1	F0
	P0	P1
	P1	P0

Tableau 2.1 : Tables de transfert des portes élémentaires

Exemple : Considérons la Figure 2.4 qui comporte cinq entrées (E_1, E_2, E_3, E_4, E_5) et deux sorties (Z_1, Z_2) ainsi que quatre portes logiques.

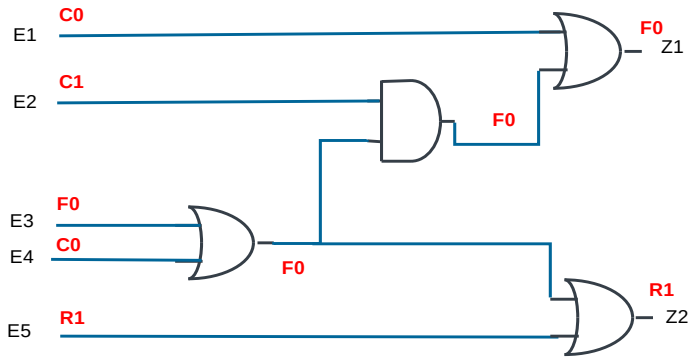


Figure 2.4 : Simulation du vecteur (E1, E2, E3, E4, E5) = (C0, C1, F0, C0, R1)

La première paire de vecteurs appliquée sur les entrées du circuit est V_1= (0, 1, 1, 0, 0), V_2= (0, 1, 0, 0, 1), sachant que, lors du test, c'est le vecteur V_2 qui avait propagé des erreurs en sortie(s). D'après ce que nous avons observé précédemment, nous pouvons déduire que les vecteurs d'entrées codés sont V_{21}= (C0, C1, F0, C0, R1) (Figure 2.4), et les valeurs seront propagées selon les Tableaux 2.1.

2.1.2.2 Traçage de chemins critiques

Le processus de traçage de chemins critiques consiste, pour chaque sortie fautive, à remonter dans le circuit à travers les portes logiques afin de déterminer les lignes susceptibles d'être à l'origine de l'erreur observée sur la sortie considérée. Cette remontée s'effectue en suivant les entrées sensibles des portes.

Le processus de traçage de chemins critiques développé s'inspire de ceux utilisés dans [Abra80] et [Gira92a] sans être pour autant parfaitement identique. En effet, dans ces deux cas, un modèle de faute précis était considéré ; le modèle de collage pour [Abra80] et le modèle de délai pour [Gira92a]. Bien que l'algèbre que nous utilisons ici soit celle développée dans [Gira92a], la notion d'entrée sensible, qui est utilisée dans [Gira92a] pour assurer la remontée à travers le circuit, n'est pas utilisable directement dans notre cas. En effet, dans [Gira92a], les entrées des portes ayant des valeurs stables (C0, C1) ne sont pas considérées comme sensibles puisque ces valeurs stables ne peuvent pas remplir les conditions de sensibilisation de fautes de délai. Les lignes portant ce type de symbole sont donc ignorées dans le processus de traçage de chemins critiques. Dans notre cas, nous cherchons à remonter une erreur logique, que cette erreur soit induite par des fautes statiques ou dynamiques. Comme nous avons à manipuler des erreurs logiques, nous pouvons par contre, en l'adaptant à la logique mutli-valuée utilisée, exploiter directement les principes développés dans [Abra80a]. Pour cela, il suffit de considérer la valeur finale de chaque symbole (correspondant au vecteur fautif) pour déterminer les entrées sensibles de chaque porte et remonter ainsi dans le circuit. Les entrées sensibles de chaque porte (marquée par un point noire, Figure 2.5) sont définies selon les règles [Abra80]. Généralement, une entrée de porte « i » est sensible lorsque la modification de la valeur finale de « i » provoque un changement de la valeur de la sortie de la porte. Dans certains cas, le processus de traçage de chemins critiques s'arrête à une porte, qui ne possède pas des entrées sensible, donc il reprend à partir de point de divergence (fanout). En effet, lorsqu'une porte non sensible était rencontrée, le processus de traçage de chemins critiques stoppait et un point de divergence était recherché [Abra80].Deux cas étaient alors possibles :

- Soit aucun point de divergence n'était trouvé et alors le processus de traçage de chemins critiques s'arrêtait à la porte non sensible

- Soit un point de divergence était déterminé et alors le processus de traçage de chemins critiques reprenait à partir de ce point de divergence. Les portes et lignes se trouvant entre cette porte non sensible et le point de divergence étaient alors ignorées.

Du fait que le traçage de chemins critiques s'arrêtait à la porte non sensible et reprenait à partir d'un point de divergence, il ne permettra pas de traiter les pannes de court-circuit. Une amélioration du processus de traçage de chemins critiques était nécessaire.

Par exemple, dans le cadre d'un diagnostic adapté au cas de la panne byzantine, il n'est plus possible d'ignorer les portes logiques se trouvant dans une boucle de reconvergence (fanin). En effet, avec ce type de pannes, il est possible qu'au même moment, deux erreurs apparaissent dans une boucle de reconvergence et se propagent au-delà de la porte non sensible, puis jusqu'aux sorties primaires du circuit. Ainsi, durant le processus de traçage de chemins critiques, si une porte de reconvergence non sensible est trouvée, les lignes en amont de celle-ci devront être considérées comme si elles étaient des lignes sensibles. Dans tous les autres cas, c'est-à-dire en présence uniquement d'entrées sensibles ou en présence d'entrées sensibles et d'autres non sensibles, le processus de traçage n'est pas modifié. Considérons la Figure 2.5, ce circuit a trois entrées (**a**,**c**,**b**) et une sortie (**s**). Le point de divergence est la ligne ***c***. La porte de reconvergence non sensible est la porte *5*. Lors du test avec le vecteur (1,1,0) correspondant respectivement aux entrées (**a**,**c**,**b**), la sortie **s** a été déclarée fautive, la valeur logique '0' était attendue, et nous avons obtenu un '1'. Dans l'hypothèse de la panne simple et avec le processus de traçage de chemins utilisé jusqu'à présent, nous nous serions arrêtés à la porte 5 non sensible et aurions repris le processus de traçage de chemins critiques à partir du point de divergence ***c***, en ignorant les lignes et portes intermédiaires (**h**, **i**, porte 3, porte 4, **d**, **e**, **f**, **g**, porte 1, porte 2). Or nous pouvons voir dans cet exemple que la cause des erreurs est le modèle de faute byzantine. Ce modèle provoque une erreur simultanément sur deux lignes (**e** et **g**). Ces deux erreurs se propagent via les entrées sensibles des portes 3 et 4, atteignant la porte *5*. Ces erreurs provoquent alors simultanément un changement de valeur sur les entrées **h** et **i** de la porte non sensible *5*, changeant ainsi la valeur de sortie de celle-ci. Il est donc impératif, maintenant avec la prise en compte des possibilités de erreurs multiples, de considérer les chemins internes aux boucles de reconvergence

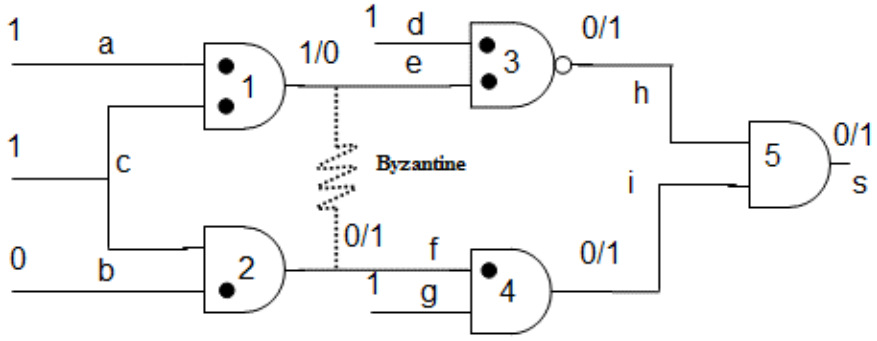


Figure 2.5 : Exemple de porte non sensible

Par ce processus, nous pouvons ainsi déterminer, pour chaque sortie fautive, une liste de lignes suspectées d'être à l'origine de l'erreur observée. D'autre part, à chaque ligne suspecte, nous associons le symbole porté par celle-ci au moment du traçage de chemins. Ainsi, le processus de traçage de chemins fournit une ou plusieurs listes de suspects.

La liste de suspects LS est définie comme suit : LS = { (LC_0, S_0, CNT_0), (LC_1, S_1, CNT_1), … (LC_n, S_{n-1}, CNT_{n-1})} avec,

- LC_i est l'identificateur de la ligne critique i
- S_i est le signal porté par cette ligne i pour le vecteur considéré
- CNT_i est le paramètre de suspects de la ligne i. Ce dernier est en fait, un compteur qui est incrémenté chaque fois que le processus de traçage de chemins critiques passe sur la même ligne i.

C'est l'association entre la ligne et le symbole (LC_i, S_i, CNT_i) qui nous permettra, lors de la dernière étape de ce processus de diagnostic, d'associer les modèles de fautes correspondant à chaque ligne critique.

Exemple :

Considérons à nouveau l'exemple précédent (Figure 2.4) après la simulation de la paire de vecteurs V_{21}. Supposons maintenant que, lors du test, l'application du vecteur V_2 a provoqué une erreur sur la sortie Z_1 (Figure 2.6). Le processus de traçage de chemins critiques débute donc à partir de cette sortie.

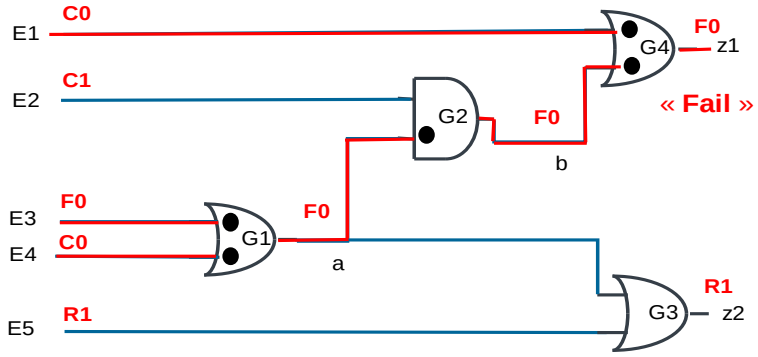


Figure 2.6 : Traçage de Chemins Critiques pour le vecteur V_{21}

La connexion de sortie Z_1, pouvant être à l'origine de l'erreur observée, est directement ajoutée à la liste des suspects. La remontée à travers le circuit s'effectue ensuite à travers la porte logique OU. Les deux entrées de cette porte sont dites sensibles, car ces deux entrées ont pour valeur finale la valeur non prioritaire '0' de la porte OU. Autrement dit, la modification de la valeur finale d'une de ses deux entrées provoquerait un changement de la valeur de la sortie. Ainsi, les lignes E_1 et *b* doivent être ajoutées à la liste des suspects avec le symbole associé à chacune d'entre elles. Cette opération est répétée pour toutes les entrées sensibles des portes (entrées marquées par un point noir). La liste des suspects ainsi obtenue à partir de la sortie Z_1 est la suivante (Tableau 2.2) :

L_{Z1_V21} = {(Z_1,F0,1), (E_1,C0,1), (b,F0,1), (a,F0,1), (E_3,F0,1), (E_4,C0,1)}

Ligne	S	CNT
Z_1	F0	1
E1	C0	1
b	F0	1
a	F0	1
E_3	F0	1
E_4	C0	1

Tableau 2.2 : Liste des suspects pour le vecteur V_{21}

Cette première liste des suspects est composée de six lignes pouvant être la cause de l'erreur observée sur la sortie Z_1. Ce processus de traçage de chemins critiques est répété pour tous les vecteurs ayant propagé des erreurs en sortie (Tableau 2.3). Si nous supposons que deux vecteurs (V_{21}, V_{65}) ont généré des erreurs, toutes sur la sortie Z_1 (F→Fail), le processus

de traçage de chemins critiques doit donc être répété pour la sortie fautive correspondante au vecteur V_{65}.

Vecteur fautif	E1	E2	E3	E4	E5	Z1	Z2
V21	C0	C1	F0	C0	R1	Fail	Pass
V65	F0	C1	C1	C0	C0	Fail	Pass

Tableau 2.3 : Vecteurs et réponses associés

Nous obtenons ainsi la liste suivante des suspects supplémentaires (Figure 2.7 et Tableau 2.4) : $\mathbf{L_{Z1_V65} = \{(Z_1,C1,2), (b,C1,2), (a,C1,2), (E_3,C1,2), (E_2,C1,1)\}}$

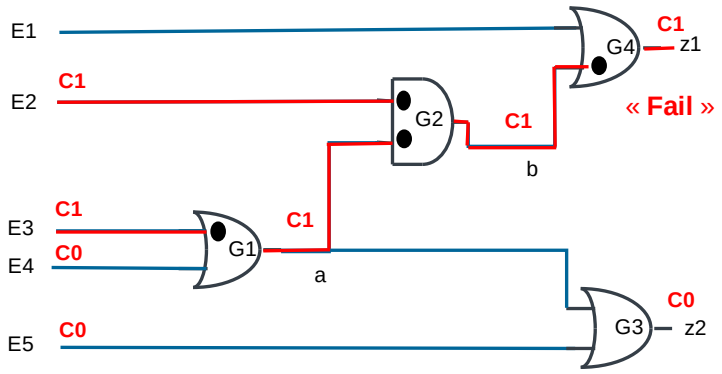


Figure 2.7 : Traçage de Chemins Critiques pour le vecteur V_{65}

Ligne	S	CNT précédent	CNT actualisé
Z_1	C1	1	2
b	C1	1	2
a	C1	1	2
E_3	C1	1	2
E_2	C1	0	1

Tableau 2.4 : Liste des suspects pour le vecteur V_{65} et actualisation du compteur CNT

A la fin de cette première étape, nous avons donc deux listes de suspects composées, chacune de plusieurs lignes pouvant être la cause des erreurs observées sur la sortie Z_1. Le paramètre CNT a été mis à jour dans le Tableau 2.4. Nous remarquons que certaines lignes ont été tracées deux fois.

2.1.2.3 Processus d'intersection

Le processus de traçage de chemins critiques est réalisé à partir de toutes les sorties déclarées « fail » lors du test. Ce processus fournit, pour chacune de ces sorties, une liste de lignes critiques auxquelles sont associées le symbole représentant le signal et le compteur CNT. Les sites des pannes ayant conduit aux erreurs observées sont nécessairement présents dans ces listes. Si nous nous plaçons maintenant dans l'hypothèse de la panne simple, le site de la panne est nécessairement présent dans toutes les listes. Dans cette hypothèse, les lignes critiques du circuit peuvent être ainsi déterminées, par intersection des listes produites par le processus de traçage de chemins critiques. Le compteur CNT nous permet aussi de distinguer entre une panne simple et une panne multiple, dans l'hypothèse de la panne simple, les lignes suspectes étaient celles dont la valeur du compteur était égale au nombre de fois que le processus de traçage de chemin avait été lancé (nombre de sortie produisant une erreur lors de l'application de la séquence de test).Par contre, si aucune ligne suspecte dont la valeur du compteur n'a pas été égale au nombre de fois que le processus du traçage de chemins critiques, nous considérons que nous avons une panne multiple.

En effet, chacune des lignes dont la valeur du compteur CNT est supérieure à 0 doit être considérée comme étant potentiellement une des sources possibles des erreurs observées en sortie du circuit. Néanmoins, l'information fournie par la valeur du compteur associé à chacune de ces lignes, peut donner une indication sur la probabilité que la panne vienne de telle ou telle connexion. En effet, il parait plus probable que la panne provienne de plusieurs ou seule une ligne qui aurait été rencontrée autant de fois qu'il y a de sorties fautives plutôt que d'une ligne qui aurait été rencontrée une seule fois.

2.1.1.1.1 Règles d'intersection

Les listes générées par le processus de traçage de chemins critiques comprennent des éléments constitués d'un site de panne potentiel et de la valeur du signal associé à ce site. Ces deux paramètres sont à prendre en compte par le processus d'intersection. Soient les deux listes, L1 et L2, produites par le processus de traçage de chemins critiques, soit LS = L1 $\cap$ L2. Ls résulte de l'intersection entre chaque élément $(LC_i,S_i,CNT_i)_1$ de L1 et chaque élément $(LC_j,S_j,CNT_j)_2$ de L2. L'intersection entre ces éléments est définie de la manière suivante :

- Si $LC_i \neq LC_j$ alors $(LC_i,S_i,CNT_i)_1 \cap (LC_j,S_j,CNT_j)_2 = \varnothing$
- $S_i \cap_s S_j$ non défini alors $(LC_i,S_i,CNT_i)_1 \cap (LC_j,S_j,CNT_j)_2 = \varnothing$
- Sinon $(LC_i,S_i,CNT_i)_1 \cap (LC_j,S_j,CNT_j)_2 = (LC,S_i \cap_s S_j,CNT)$

Où $\cap_s$ est l'opération d'intersection portant sur les signaux associés à une ligne critique. Pour les pannes considérées dans ce chapitre, l'opération d'intersection est présentée dans le Tableau 2.5.

$\cap_s$	**C0**	**C1**	**F0**	**R1**	**P0**	**P1**	**D**	**SDW**
C0	C0	SDW	C0	SDW	C0	SDW	SDW	SDW
C1	SDW	C1	SDW	C1	SDW	C1	SDW	SDW
F0	C0	SDW	F0	D	P0	D	D	SDW
R1	SDW	C1	D	R1	D	P1	D	SDW
P0	C0	SDW	P0	D	P0	D	D	SDW
P1	SDW	C1	D	P1	D	P1	D	SDW
D	SDW	SDW	D	D	D	D	D	SDW
SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW

Tableau 2.5 : Opération d'intersection entre signaux

L'intersection entre les symboles représentant une transition (F0, R1, P0, P1) avec des valeurs finales opposées (0 et 1) donne comme résultat le symbole « D » représentant une panne de type délai. Ce symbole est ajouté à l'ensemble des symboles utilisés durant la simulation mutli-valuée, permettant ainsi de représenter des retards sur les fronts (montant et/ou descendant). De la même façon, nous avons besoin d'expliquer une situation où le signal sur une ligne suspecte peut être stable et peut prendre des valeurs différentes pour différents vecteurs fautifs. Ceci se produit dans le cas de pannes court-circuit. Le symbole « SDW » symbolise une panne de type court-circuit « Strong Driver Wired », il permet de décrire plusieurs situation dans laquelle la valeur de suspect dépend de la valeur d'une autre ligne. Dans ce cas, le suspect est considéré comme victime de court-circuit (cf. explication au chapitre I). Par exemple, si nous considérons le symbole C0 qui sensibiliserait un collage à '1' et le symbole C1 qui sensibiliserait un collage à '0', l'intersection de ces deux symboles conduit à un court-circuit.

Exemple : Reprenons les deux listes des suspects de l'exemple précédent :

- **L_{Z1_V21} = {(Z_1,F0,1), (E_1,C0,1), (b,F0,1), (a,F0,1), (E_3,F0,1), (E_4,C0,1)}**

- **L_{Z1_V65} = {(Z_1,C1,2), (b,C1,2), (a,C1,2), (E_3,C1,2), (E_2,C1,1)}**

Le résultat de l'intersection de ces deux listes est (Tableau 2.6):

L = {(Z_1,SDW,2), (b,SDW,2), (a,SDW,2), (E_3,SDW,2)}

Ligne	S	CNT
Z_1	SDW	2
b	SDW	2
a	SDW	2
E_3	SDW	2
~~**E_1**~~	~~C0~~	~~1~~
~~**E_2**~~	~~C1~~	~~1~~
~~**E_4**~~	~~C0~~	~~1~~

Tableau 2.6 : Classement et résultat d'intersection entre les deux listes des suspects

Les lignes E_5 et Z_2 n'ont pas été conservées car elles n'étaient pas présentes dans les deux listes des suspects. Les lignes E_1, E_2 et E_4 conservent les mêmes symboles car ils ne sont pas présentes dans les deux listes, donc il n y a pas d'intersection de symbole. Enfin, dans l'hypothèse de la panne simple, les connexions susceptibles de porter la panne sont uniquement les lignes dont le paramètre CNT est égale au nombre d'erreurs observées (nombre de fois où le processus de traçage de chemins critiques a été activé). Dans cet exemple, nous conserverons uniquement les lignes ayant un CNT=2, c'est-à-dire les lignes Z_1, b, a, E_3 (Tableau 2.6).

2.1.3 Phase d'affectation des modèles de fautes

Les modèles de fautes considérés sont les fautes de collage (stuck-at 0/1), de transistors Tn/Tp stuck-on, de transistors Tn/Tp stuck-open, de circuit ouvert (open 0/1), de circuit ouvert résistif (resistive open), de court-circuit de type Strong Driver Wired, Dominant And/Or, Wired And/Or, Byzantine, de court-circuit résistif de type resistive And/Or, de délai de type StF/StR (Tableau 2.7).

Nous avons vu que les modèles de fautes associés à chaque ligne suspecte sont déduits grâce à la simulation multi-valuée. Les modèles de fautes associés avec une ligne suspecte peuvent être déterminés en fonction du symbole associé avec la ligne. Les lignes marquées avec un symbole stable (C0 ou C1) portent seulement des fautes statiques, puisqu'aucune

transition n'est apparue sur la ligne et donc aucune faute dynamique n'a été sensibilisée (colonnes 2 et 3 du Tableau 2.7).

Les lignes marquées avec un symbole représentant une transition (F0, R1, P0, P1) peuvent représenter aussi bien des fautes statiques que dynamiques (colonnes 4 et 5 du Tableau 2.7).

Enfin, les colonnes 5 et 6 du Tableau 2.7 présentent les modèles de fautes associées avec les symboles 'D' et 'SDW'. Ces symboles sont les résultats d'intersections vues précédemment. Le symbole 'D' représente une situation dans laquelle les lignes suspectes présentent des retards sur les fronts (montant et/ou descendant), cela signifie que les lignes marquées avec ce symbole sont associées à des fautes dynamique ou de types byzantine. Le symbole 'SDW' présente une ligne avec des valeurs opposée '0' et '1' venant des deux symboles statiques et dynamiques. Si le symbole associé avec une ligne suspecte est marquée avec le symbole 'SDW', cela signifie que la ligne est associée à un court-circuit de types Strong Driver Wired ou Byzantine [Rou07$_2$].

	C0	C1	F0,P0	R1,P1	D	SDW
Stuck at 0		X		x		
Stuck at 1	x		x			
Tn Stuck open	x		x			
Tn Stuck on		x		x		
Tp Stuck open		x		x		
Tp Stuck on	x		x			
Open 0		x		x		
Open 1	x		x			
Resistive open			x	x	x	
Dominant Or Bridge	x		x			
Dominant And Bridge		x		x		
Resistive Or			x			
Resistive And				x		
Byzantine	x	x	x	x	x	x
Strong Driver Wired	x	x	x	x	x	x
Wired Or	x		x			
Wired And		x		x		
Delay StF			x			
Delay StR				x		
Delay StR & StF			x	x	x	

Tableau 2.7 : Modèles de fautes associés aux lignes critiques en fonction du symbole

2.2 Adaptation de la méthode de diagnostic pour le traitement des systèmes sur puce

2.2.1 Description d'un système sur puce

La conception d'un système sur puce se fait par assemblage de différents blocs, appelés IPs (*Intellectual Property*), qui sont en général de type processeur (CPU), DSP, RAMs, ROMS, DRAM, PLL, CAN, CNA, contrôleurs USB ou Ethernet, blocs logiques, etc (Figure 2.8). Les IPs d'un système sur puce peuvent communiquer entre elles via un bus, un cross-bar, un réseau sur puce (*Network-on-Chip* ou NoC en anglais), etc.

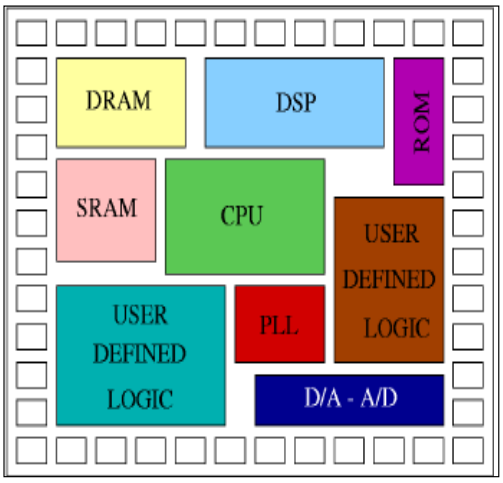


Figure 2.8 : Architecture classique d'un système sur puce

Ces IPs sont généralement disponibles sous forme d'une description logique écrite en langage 'VHDL' ou 'Verilog', ou sous forme de description physique appelé 'layout'. La flexibilité d'utilisation de l'IP dépend de la forme sous laquelle elle est fournie. Trois catégories de l'IP existent :

- *Soft IPs :* Ces IPs sont disponibles sous forme d'une description RTL (Register Transfert Level) synthétisable. Ceci implique que l'utilisateur de l'IP est responsable de son implémentation physique.
- *Hard IPs :* Ces IPs sont disponibles sous forme d'une description des niveaux physiques de 'layout' (par exemple, au format GDSII). elles sont livrées pour une

technologie donnée en étant optimisés en performance, et/ou en consommation, et/ou en surface. Les droits de propriété intellectuelle (*Intellectual Property* ou 'IP' en anglais) ne permettent pas de fournir plus de détails sur leur implémentation.

- *Firm IPs :* est une catégorie intermédiaire. Ces IPs sont disponibles sous forme d'une combinaison de code RTL synthétisable et d'une description au niveau porte logique. Les estimations quant à la complexité en nombre de portes, aux performances et à la consommation sont disponibles.

Dans notre étude, nous possédons uniquement des systèmes sur puce composés d'IPs de type « Soft IPs » et « Firm IPs ». Ces derniers sont livrés par les ingénieurs de conception numérique. Les systèmes sur puce de type « Soft IPs » ont été synthétisés en technologies 90 nm, alors que les systèmes sur puce de type « Firm IPs » ont été fournis au niveau porte logique. Pour chaque circuit étudié, nous avons considéré des IPs logiques de type processeurs, DSP, etc. Les IPs analogiques de type PLL, CAN, CNA, etc, ont été remplacées par des boîtes noires dans les différentes netlists.

Nous avons traité les mémoires dans deux cas possibles durant le processus du diagnostic. Dans le premier cas, les IPs mémoires qui ne possèdent pas des descriptions comportementales ont été aussi remplacées par des boîtes noires (Black Box) (Figure 2.9). Par contre, dans le deuxième cas, les IPs mémoires qui possèdent des descriptions comportementales ont été prise en compte. Notons que pour le deuxième cas, nous traitons uniquement les entrées, sorties, bus et blocs logiques autour de la mémoire (décodeur, adressage lignes/colonnes, circuit E/S, etc). Nous ne nous intéressions pas à la cellule mémoire.

Une boîte noire (Black Box) représente une IP dont les informations sur sa fonction et sur son contenu interne sont inconnues. Seuls les noms des entrées et sorties primaires sont connus. Chaque sortie primaire a, par défaut, la valeur 'X' ou 'Z'.

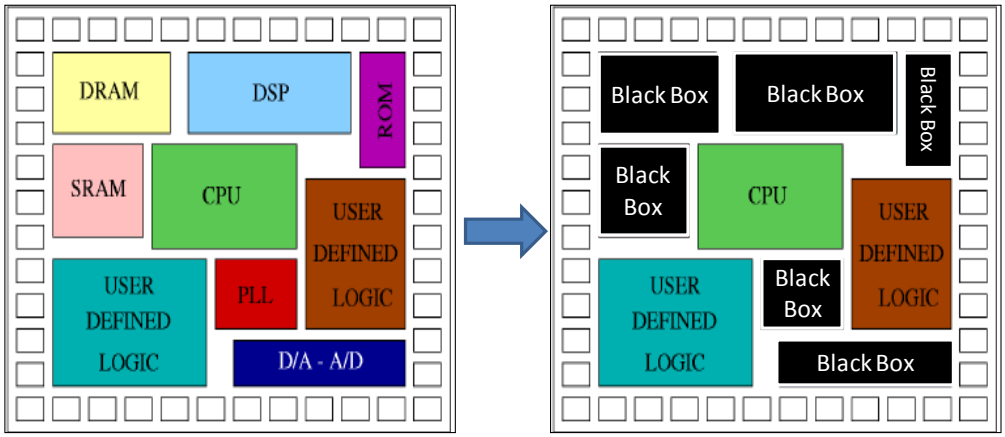


Figure 2.9 : Exemple de représentation d'un système sur puce pour le diagnostic

Lors du processus de diagnostic, l'idée générale est de pouvoir retrouver l'origine et la nature exacte du défaut lorsqu'il se trouve dans une IP logique, ou d'identifier la sortie du bloc représenté par une boîte noire lorsque le défaut provient d'une IP analogique ou d'une mémoire.

2.2.2 Synoptique de la méthode de diagnostic adaptée

Dans la première partie de ce chapitre, nous avons vu que la méthode de diagnostic proposée dans [Rou08] sous l'acronyme eDERRIC traite uniquement les circuits séquentiels de type « full scan », assimilables à des circuits combinatoires. Cependant, il existe aussi des circuits séquentiels de type « partial-scan » dans lesquels certaines bascules ne sont pas de type « scan » et ne peuvent pas être contrôlées et observées directement par le biais des chaînes de scan. Par ailleurs, la méthode de diagnostic initiale a été développée pour traiter une IP logique à la fois, alors qu'un système sur puce possède à priori plusieurs IPs de natures diverses. Il était donc nécessaire de modifier la méthode initiale (eDERRIC) afin de prendre en compte ces éléments lors du diagnostic de pannes dans les systèmes sur puce.

Le synoptique de la méthode de diagnostic adaptée au traitement des systèmes sur puce pouvant contenir des blocs séquentiels de type « partial-scan » est représenté sur la Figure 2.10. Les modifications à apporter à la méthode initiale sont de deux types : i) modifications concernant le format des fichiers d'entrée manipulés par l'outil eDERRIC, et ii) modifications concernant le processus de diagnostic proprement dit.

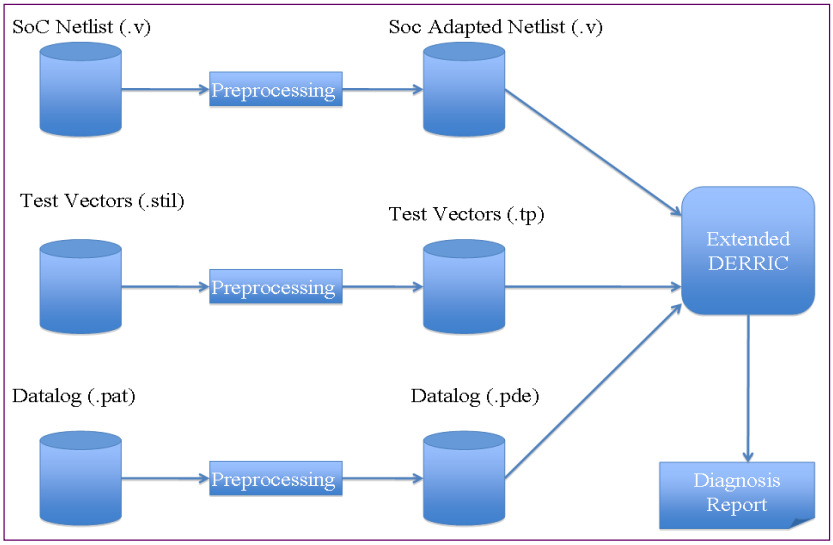


Figure 2.10 : Synoptique d'adaptation de la méthode de diagnostic

En ce qui concerne le format des fichiers d'entrée manipulés par eDERRIC, les modifications concernent la description au niveau porte du système, les vecteurs de test et les réponses fournies par le testeur (datalog). Une phase de pré-traitement a donc été mise en place dans ce but afin de réaliser les tâches suivantes. Tout d'abord, la description du système initial (SoC netlist) au format verilog '.v' est transformée en une description de système de type combinatoire (SoC Adapted Netlist) au format verilog '.v' compatible avec l'outil eDERRIC. La deuxième étape consiste à transformer les vecteurs de test au format '.Stil' [Stil07] en vecteurs de test au format '.tp' compatible avec eDERRIC. La dernière étape consiste à transformer les réponses du testeur initialement au format '.pat' (pattern-based, [Tmax07]) dans un format '.pde' compatible avec celui de l'outil eDERRIC. Cette phase de pré-traitement fournit donc trois fichiers utilisables par l'outil eDERRIC pour effectuer le diagnostic d'un système sur puce. Notons que les réponses du testeur de type « cycle (.cyc)» [Tmax07] ne sont pas supportées dans cette étude.

En ce qui concerne le processus de diagnostic proprement dit, les modifications à apporter concernent i) le codage des signaux, et par conséquent l'algèbre utilisée, durant le processus de simulation multi-valuée, ainsi que ii) le processus de traçage de chemins critiques (CPT).

Toutes ces modifications sont détaillées dans les paragraphes suivants.

2.2.3 Modification du format des fichiers d'entrée

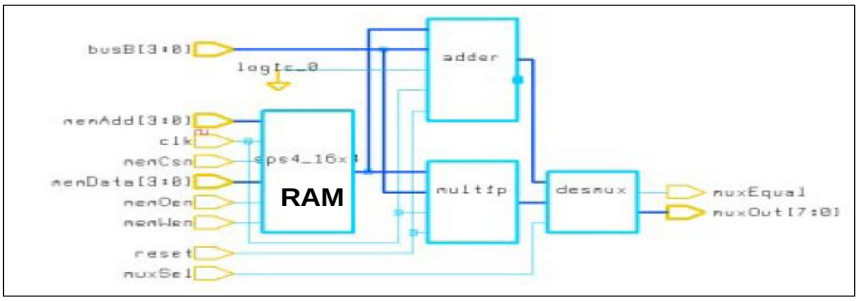
2.2.3.1 Modification de la description d'un système sur puce

La première étape de la phase de pré-traitement consiste à modifier la description du système sur puce à diagnostiquer. Cette description peut contenir les éléments suivants, qui ne sont pas pris en compte par la méthode de diagnostic initiale et l'outil eDERRIC :

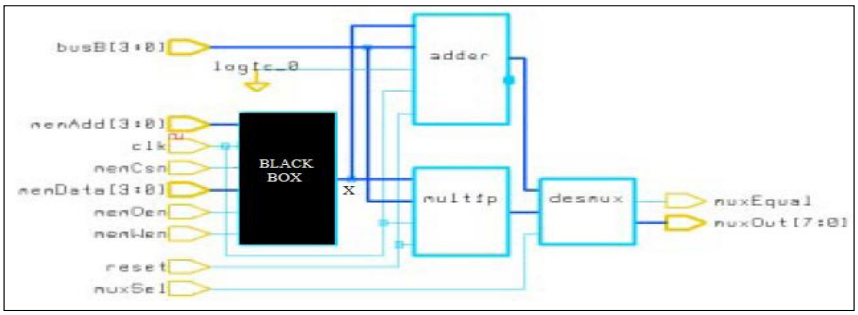
- Les éléments logiques autres que les portes logiques de base : registre, multiplexeur, démultiplexeur, buffer bidirectionnel, tri-state, OA, AO etc.
- Les éléments séquentiels (bascule D, latch, RS, RST, JK, T, etc.)
- Les bus
- Les niveaux de hiérarchie entre IPs (une seule IP est considérée dans la version initiale de l'outil eDERRIC).

La phase de pré-traitement consiste donc, dans un premier temps, à rendre ces éléments compréhensibles par l'outil eDERRIC. Pour cela, un flot défini à l'aide de l'outil Design Compiler [DC07] a été mis en place. La première partie de ce flot consiste à supprimer les hiérarchies (regrouper toutes les IPs dans une seule IP) et les bus, et dans un deuxième temps, à remplacer tous les éléments logiques de type registre, multiplexeur, OA, AO, etc. par des portes logiques de base de type NOT, AND, OR, NAND, NOR, XOR, XNOR, et Buffer.

Par ailleurs, la présence de boîtes noires dans la description d'un système doit également être prise en compte et nécessite pour cela certaines modifications. Ces modifications consiste à figer les sorties de ces boîtes noires à la valeur 'X' ou 'Z'. La valeur 'X' sera utilisée dans le cas où les sorties sont inconnues, cela correspond aux blocs analogiques et mémoires. La valeur 'Z' sera utilisée dans le cas où certaines sorties des blocs mémoires sont en hautes impédances. Dès lors, ces valeurs indéterminées seront utilisées lors des processus de simulation multi-valuée et de traçage de chemins critiques explicités dans la partie 2.4. A titre d'exemple, considérons le circuit représenté sur la Figure 2.11.a et composé de quatre IPs (additionneur, multiplieur, multiplexeur et mémoire RAM). Si la mémoire RAM est remplacée par une boîte noire, la valeur 'X' assignée à sa sortie pourra se propager à travers les blocs additionneur, multiplieur et multiplexeur jusqu'aux sorties primaires du système (Figure 2.11 (b).



a) Netlist fonctionnelle



b) Netlist avec une boîte noire

Figure 2.11 : Exemple de propagation de la valeur 'X'

Une autre attribution du pré-processeur chargé de modifier la description initiale du système est le traitement des éléments séquentiels, puisque l'outil de base eDERRIC ne traite que des IPs logiques combinatoires. Dans le troisième chapitre de ce manuscrit, nous verrons comment il est possible de traiter les IPs purement ou partiellement séquentielles lors du diagnostic, et nous présenterons la méthode que nous avons développée pour cela. Dans l'immédiat, la solution pour laquelle nous avons opté consiste à ne pas prendre en compte ces éléments séquentiels, en les transformant pour cela en éléments combinatoires. Ce choix peut avoir des conséquences sur la résolution du diagnostic, comme nous le verrons en fin de chapitre.

La deuxième partie du flot consiste donc à remplacer chaque élément purement séquentiel (bascule, latch, RS, JK, etc.) par un ensemble composé de buffers, de pseudo-entrées primaires et de pseudo-sorties primaires, comme illustré sur la Figure 2.12. Par le biais d'une simulation logique, nous déterminons ensuite, pour chaque vecteur de test, les valeurs qui correspondent à chaque pseudo-entrée primaire, et nous mémorisons l'ensemble de ces valeurs dans un fichier de données. Les pseudo-sorties primaires sont, quant à elles, considérées comme des sorties du système. Notons qu'en ce qui concerne les bascules de type

scan et non-scan, de façon très classique, les entrées de bascules sont considérées comme des pseudo-sorties primaires, et les sorties de bascules sont considérées comme des pseudo-entrées primaires. Dans ce cas, les valeurs de chaque pseudo-entrée primaire sont définies directement dans chaque vecteur de test.

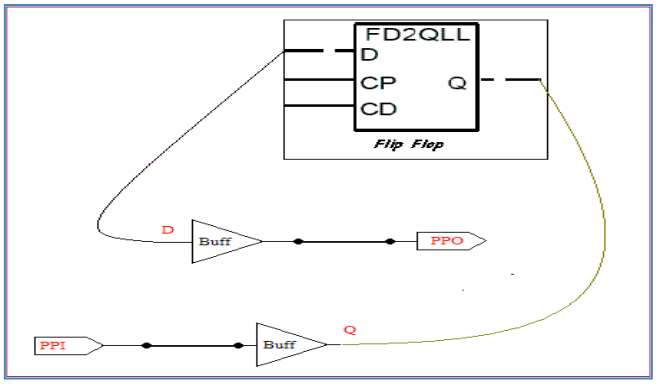


Figure 2.12 : Remplacement de flip flop par des buffers

Considérons l'exemple de l'IP de la Figure 2.13, contenant des éléments séquentiels. La transformation de cette IP séquentiel en IP combinatoire (Figure 2.14) se fait comme suit :

1- Nous remplaçons les bascules de type scan et non-scan par des buffers et des pseudos entrées/sorties primaires (Pseudo Primary Input (PPI)/Pseudo Primary Output (PPO)) (Figure 2.14).
2- Pour chaque bascule, nous déconnectons l'entrée 'D' et la sortie 'Q', puis nous ajoutons deux buffers. D'une part, l'entrée du premier buffer sera connectée à l'entrée 'D' de la bascule, et sa sortie sera connectée à une pseudo-sortie primaire (PPO). D'autre part, l'entrée du deuxième buffer sera connectée à une pseudo-entrée primaire (PPI) et sa sortie sera connectée à la sortie 'Q' de la bascule (Figure 2.12).
3- Nous supprimons la bascule concernée.
4- Nous répétons la même procédure pour toutes les bascules du circuit.

Notons que nous ignorons les défauts affectant les lignes d'horloge (clock), de remise à '0' (reset), de remise à '1' (set), d'entrée scan (scan in), de mode de fonctionnement (scan enable) et de sortie scan (scan out).

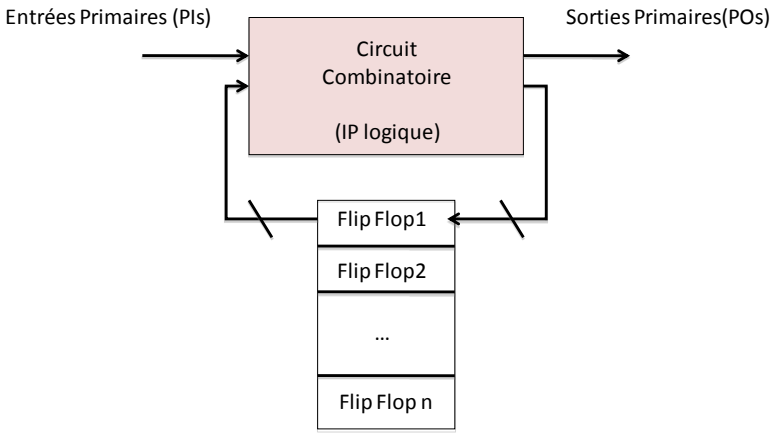


Figure 2.13 : IP contenant des éléments séquentiels (bascules)

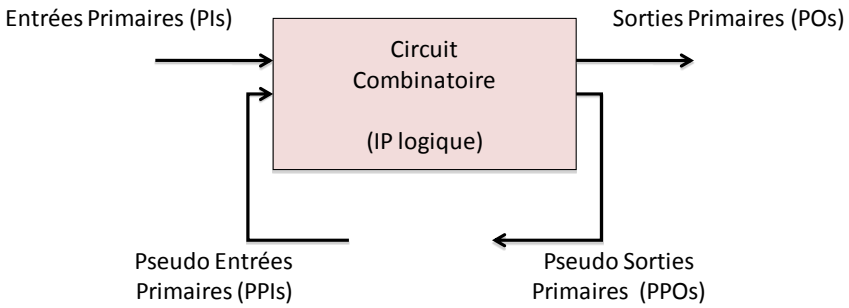


Figure 2.14 : IP combinatoire (suppression des bascules)

2.2.3.2 Modifications du format des vecteurs de test

La deuxième étape de la phase de pré-traitement consiste à modifier le format des vecteurs de test pour la version combinatoire du système à considérer. Sachant que les vecteurs de test fournis avec la description du système sur puce sont de type 'Stil' [STIL07]. Nous avons donc écrit un programme en langage script [Tcl07] à l'aide de l'outil TetraMax [Tmax07] permettant de transformer les vecteurs de test de type 'Stil' [STIL07] en vecteurs de test de type '.tp'. utilisables par l'outil eDERRIC.

Les vecteurs de test de type '.tp' sont sous la forme d'une série de chiffres de 0 et/ou 1 (exemple : vecteur '1' = *010000100110101*), le nombre de chiffres égale au nombre des entrées et pseudo-entrées primaires du système sur puce.

La Figure 2.15 présente un exemple de fichier STIL décrivant une séquence de vecteurs de test pour un circuit de type « partial-scan ».

```
…
"pattern 0": Call "load_unload" {
        "scan_in"=01000; }
        Call "capture_clock" {
        "_pi"=000001; "_po"=LLL; }
"pattern 1": Call "load_unload" {
        "scan_out"=LLLLL; "scan_in"=00110;
…
```

Figure 2.15 : Exemple de vecteurs de test de type 'Stil'

Après la modification de la description du système sur puce détaillée précédemment, nous avons obtenu des entrées primaires supplémentaires (sorties de bascules scan et non scan, entrées des boîtes noires). Dans ce cas, pour chaque vecteur de test, nous devons affecter la valeur logique correspondante à chaque entrée primaire supplémentaire.

En effet, d'une part, nous utilisons le fichier d'entrée de type 'Stil' [STIL09] pour extraire les valeurs des entrées primaires (_pi) et les valeurs des sorties des bascules scan (scan_out). D'autre part, nous ne nous intéressons pas aux valeurs des entrées primaires des boîtes noires, car les sorties seront figées à la valeur 'X' ou 'Z'. Enfin, nous faisons une simulation du système sain pour récupérer les valeurs des sorties des bascules non-scan. Le vecteur de test transformé de type '.tp' est représenté sous la forme suivante :

$$TV_j = _pi_j + scan_out_j + CPPIs_j + nsFF_out_j$$

Avec

- « + » = concaténation des séquences de '0'/'1'
- TV_j : vecteur de test numéro j
- pi_j : les valeurs des entrées primaires
- $scan_out_j$: les valeurs des pseudos sorties primaires, c'est-à-dire les valeurs des sorties (Q) des bascules scan.
- $CPPIs_j$: les valeurs des entrées primaires des boîtes noires

- nsFF_out$_j$: les valeurs des pseudos sorties primaires, c'est-à-dire les valeurs des sorties (Q) des bascules non-scan.

Prenons l'exemple du vecteurs de test présentés sur la Figure 2.15, ils correspondent à un circuit contenant 6 entrées primaires (Pi$_j$), 5 bascules de chaîne de scan (scan_in$_j$), 6 entrées primaires de la mémoire (remplacée par une boîte noire, non figurée dans la figure 2.15) (CPPIs$_j$) et une bascule non-scan (nsFF_in$_j$) (non figurée dans le fichier).

Le vecteur de test '0' (pattern 0, Figure 2.15) transformé et compatible avec l'outil eDERRIC est le suivant :

TV_0 = 000001 + 00000+ 000000 + 1 = 000001000000010111

La valeur de la sortie de la bascule non-scan (nsFF_in$_j$) a été déduite grâce à la simulation du système sain avec le vecteur de test '0'. Pour cet exemple, nous avons obtenu Q_nsFF_in$_j$= 1. Nous pouvons assigner la valeur '0' ou '1' à toutes les entrées de la mémoire (CPPIs$_j$), pour une raison arbitraire, nous avons choisit la valeur '0'.

2.2.3.3 Modifications du format des réponses du testeur

La troisième étape de la phase de pré-traitement consiste à modifier le format des réponses du testeur pour la version combinatoire du système à considérer. Sachant que les réponses du testeur considérées dans cette étude, fournies avec la description du système sur puce sont de type 'pattern-based' [Tmax07]. Nous avons donc écrit un programme en langage script [Tcl07] à l'aide de l'outil TetraMax [Tmax07] ; permettant de transformer les réponses du testeur de type 'pattern-based' [Tmax07] en réponses de testeur de type '.pde'. utilisable par l'outil eDERRIC.

Les réponses de testeur de type '.pde' sont sous la forme d'une série de lettre de P et/ou F (exemple : réponses du testeur du vecteur '1' = *PPFFPFPPPPPPPF*), le nombre de lettres égale au nombre des sorties et pseudo-sorties primaires du système sur puce.

Après la modification de la description système sur puce détaillée précédemment, nous avons obtenu des sorties primaires supplémentaires (entrées de bascules scan et non scan, sorties des boîtes noires). Dans ce cas, pour chaque vecteur de test, nous devons affecter la lettre correspondante à chaque sortie primaire supplémentaire.

En effet, dans un premier temps, nous mettons la lettre 'P' à toutes les sorties primaires et pseudo primaires. Ensuite, pour chaque vecteur de test fautif (*failing test vector* ou FTV) figurant dans les réponses du testeur, nous remplaçons la lettre 'P' correspondante à la sortie fautive (*failing primary output* ou FPO) par la lettre 'F'. Nous répétons le processus de modification pour tous les vecteurs de test fautifs.

Nous représentons les réponses du testeur sous la forme suivante :

$$TR_j = _po_j + scan_in_j + CPPOs_j + nsFF_in_j$$

Avec

- « + » = concaténation des séquences de 'P'/'F'
- TR_j : réponses du testeur numéro j
- $_po_j$: les sorties primaires.
- $scan_in_j$: les pseudo-entrées primaires, c'est-à-dire si la sortie fautive de datalog est une bascule dans la chaîne de scan.
- $CPPOs_j$: les sorties primaires des boîtes noires.
- $nsFF_in_j$: les pseudos entrées primaires, c'est-à-dire les valeurs des sorties des bascules, qui ne se trouvent pas dans la chaîne de scan.

Pour mieux comprendre cette transformation, prenons toujours le même exemple de la Figure 2.15, le circuit contient 3 sorties primaires (Po_j : S1, S2, S3), 5 bascules de chaîne de scan ($scan_in_j$), une sortie primaires de la mémoire (remplacée par une boîte noire, non figurée dans la figure 2.15) ($CPPOs_j$) et une bascules non-scan ($nsFF_in_j$) (non figurée dans la Figure 2.15).

FTV	FPO
0	S2
3	S1, S3
5	Scan_out 0

Tableau 2.8 : Réponses du testeur de l'exemple (Figure 2.15)

Le circuit a été défaillant lors de l'application du vecteur de test '0, 3 et 5' durant la séquence du test comme cela indiqué au Tableau 2.8. Ce dernier a rapporté les sorties fautives (FPO) correspondantes à chaque vecteur de test fautif (FTV).

Notons que dans cet exemple, nous affectons la lette ‘P’ pour les sorties primaires des boîtes noires, car elles ne sont pas concernées dans le processus diagnostic logique (la mémoire est une boîte noire).

$$TR_j = _po_j + scan_in_j + CPPOs_j + nsFF_in_j$$

Pour le FTV numéro 0, seulement la sortie S2 est fautive :

- Les réponses du testeur numéro ‘0’ : TR_0 = PFP +PPPPP + P + P =PFPPPPPPPP

Pour le FTV numéro 3, uniquement les sorties S1 et S2 sont fautives:

- Les réponses du testeur numéro ‘3’ : TR_3 = FPF +PPPPP + P + P =FPFPPPPPPP

Pour le FTV numéro 5, la bascule scan numéro ‘0’ est considéré comme sortie fautive:

- Les réponses du testeur numéro ‘5’ : TR_5 = PPP +PPFPP + P + P =PPPPPFPPPP

Les réponses de testeur utilisables par l’outil eDERRIC sont :

- TR_0 = PFPPPPPPPP
- TR_1 = PPPPPPPPPP
- TR_2 = PPPPPPPPPP
- TR_3 = FPFPPPPPPP
- TR_4 = PPPPPPPPPP
- TR_5 = PPPPPPPPPP
- TR_6 = PPPPPFPPPP

2.2.4 Modification du processus de diagnostic

2.2.4.1 Modification du codage des signaux

Pour prendre en compte la présence de boîtes noires dans les systèmes à diagnostiquer, et de ce fait considérer les valeurs ‘X’ et ‘Z’ lors de la simulation multi-valuée et du traçage de chemins critiques, des modifications ont été apportées au processus de diagnostic initial.

Tout d’abord, l’algèbre à six valeurs, utilisée pour le codage des signaux et détaillée au début de ce chapitre, a été modifiée pour prendre en compte la présence des valeurs ‘X’ et ‘Z’

sur certains nœuds du circuit. Une nouvelle algèbre à 18 valeurs a donc été développée, dans laquelle 12 valeurs nouvelles ont été définies, les six valeurs restantes correspondant aux anciens symboles. Nous avons considéré toutes les combinaisons possibles des signaux (0, 1, X, Z). L'algèbre à 18 valeurs permettra donc d'étudier les circuits combinatoires, séquentiels et les systèmes sur puce. Cette nouvelle algèbre est illustrée sur la Figure 2.16.

Symbole	Vi-1	Vi	Description
CX	X	X	Signal stable à 'X'
CZ	Z	Z	Signal stable à 'Z'
XZ	X	Z	Signal dont la valeur initiale est 'X' et la valeur finale est 'Z'
ZX	Z	X	Signal dont la valeur initiale est 'Z' et la valeur finale est 'X'
0X	0	X	Signal dont la valeur initiale est '0' et la valeur finale est 'X'
X0	X	0	Signal dont la valeur initiale est 'X' et la valeur finale est '0'
1X	1	X	Signal dont la valeur initiale est '1' et la valeur finale est 'X'
X1	X	1	Signal dont la valeur initiale est 'X' et la valeur finale est '1'
0Z	0	Z	Signal dont la valeur initiale est '0' et la valeur finale est 'Z'
Z0	Z	0	Signal dont la valeur initiale est 'Z' et la valeur finale est '0'
1Z	1	Z	Signal dont la valeur initiale est '1' et la valeur finale est 'Z'
Z1	Z	1	Signal dont la valeur initiale est 'Z' et la valeur finale est '1'

Figure 2.16 : Les 12 nouveaux symboles de l'algèbre à 18 valeurs

Les tables de transfert associées aux portes logiques ET (AND), OU (OR), et Inverseur (NOT) ont été également modifiées pour prendre en compte ce nouveau codage à 18 valeurs. Ces tables de transfert sont représentées sur la Figure 2.17.

OR	CZ	CX	C0	C1	F0	R1	P0	P1	X0	X1	Z0	Z1	0X	1X	0Z	1Z	XZ	ZX
CZ	CZ	CX	CX	C1	1X	X1	1X	X1	CX	X1	ZX	Z1	CX	1X	XZ	1Z	XZ	ZX
CX	CX	CX	CX	C1	1X	X1	1X	X1	CX	X1	CX	X1	CX	1X	CX	1X	CX	CX
C0	CX	CX	C0	C1	F0	R1	P0	P1	X0	X1	X0	X1	0X	1X	0X	1X	CX	CX
C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1	C1
F0	1X	1X	F0	C1	F0	P1	F0	P1	F0	C1	F0	C1	1X	1X	1X	1X	1X	1X
R1	X1	X1	R1	C1	P1	R1	R1	P1	X1	X1	X1	X1	R1	C1	R1	C1	X1	X1
P0	1X	1X	P0	C1	F0	R1	P0	P1	F0	C1	F0	C1	1X	1X	1X	1X	1X	1X
P1	X1	X1	P1	C1	P1	P1	P1	P1	X1	X1	X1	X1	R1	C1	R1	C1	X1	X1
X0	CX	CX	X0	C1	F0	X1	F0	X1	X0	X1	X0	X1	CX	1X	CX	1X	CX	CX
X1	X1	X1	X1	C1	C1	X1	C1	X1	X1	X1	X1	X1	X1	C1	X1	C1	X1	X1
Z0	ZX	CX	X0	C1	F0	X1	F0	X1	X0	X1	Z0	Z1	CX	1X	CX	1X	CX	ZX
Z1	Z1	X1	X1	C1	C1	X1	C1	X1	X1	X1	Z1	Z1	X1	C1	X1	C1	X1	Z1
0X	CX	CX	0X	C1	1X	R1	1X	R1	CX	X1	CX	X1	0X	1X	0X	1X	CX	CX
1X	1X	1X	**1X**	C1	1X	C1	1X	C1	1X	C1	1X	C1	1X	1X	1X	1X	1X	1X
0Z	XZ	CX	0X	C1	1X	R1	1X	R1	CX	X1	CX	X1	0X	1X	0Z	1Z	XZ	CX
1Z	1Z	1X	1X	C1	1X	C1	1X	C1	1X	C1	1X	C1	1X	1X	1Z	1Z	1Z	1X
XZ	XZ	CX	CX	C1	1X	X1	1X	X1	CX	X1	CX	X1	CX	1X	XZ	1Z	XZ	CX
ZX	ZX	CX	CX	C1	1X	X1	1X	X1	CX	X1	ZX	Z1	CX	1X	CX	1X	CX	ZX

AND	CZ	CX	C0	C1	F0	R1	P0	P1	X0	X1	Z0	Z1	0X	1X	0Z	1Z	XZ	ZX
CZ	CZ	CX	C0	CX	X0	0X	X0	0X	X0	CX	Z0	ZX	0X	CX	0Z	XZ	XZ	ZX
CX	CX	CX	C0	CX	X0	0X	X0	0X	X0	CX	X0	CX	0X	CX	0X	CX	CX	CX
C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0	C0
C1	CX	CX	C0	C1	F0	R1	P0	P1	X0	X1	X0	X1	0X	1X	0X	1X	CX	CX
F0	X0	X0	C0	F0	F0	P0	P0	F0	X0	X0	X0	X0	C0	F0	C0	F0	X0	X0
R1	0X	0X	C0	R1	P0	R1	P0	R1	C0	R1	C0	R1	0X	0X	0X	0X	0X	0X
P0	X0	X0	C0	P0	P0	P0	P0	P0	X0	X0	X0	X0	C0	F0	C0	F0	X0	X0
P1	0X	0X	C0	P1	F0	R1	P0	P1	C0	R1	C0	R1	0X	0X	0X	0X	0X	0X
X0	X0	X0	C0	X0	X0	C0	X0	C0	X0	X0	X0	X0	C0	X0	C0	X0	X0	X0
X1	CX	CX	C0	X1	X0	R1	X0	R1	X0	X1	X0	X1	0X	CX	0X	CX	CX	CX
Z0	Z0	X0	C0	X0	X0	C0	X0	C0	X0	X0	Z0	Z0	C0	X0	C0	X0	X0	Z0
Z1	ZX	CX	C0	X1	X0	R1	X0	R1	X0	X1	Z0	Z1	0X	CX	0X	CX	CX	ZX
0X	0X	0X	C0	0X	C0	0X	C0	0X	C0	0X	C0	0X	0X	0X	0X	0X	0X	0X
1X	CX	CX	C0	1X	F0	0X	F0	0X	X0	CX	X0	CX	0X	1X	0X	1X	CX	CX
0Z	0Z	0X	C0	0X	C0	0X	C0	0X	C0	0X	C0	0X	0X	0X	0Z	0Z	0Z	0X
1Z	XZ	CX	C0	1X	F0	0X	F0	0X	X0	CX	X0	CX	0X	1X	0Z	1Z	XZ	CX
XZ	XZ	CX	C0	CX	X0	0X	X0	0X	X0	CX	X0	CX	0X	CX	0Z	XZ	XZ	CX
ZX	ZX	CX	C0	CX	X0	0X	X0	0X	X0	CX	Z0	ZX	0X	CX	0X	CX	CX	ZX

NOT	CZ	CX	C0	C1	F0	R1	P0	P1	X0	X1	Z0	Z1	0X	1X	0Z	1Z	XZ	ZX
	CX	CX	C1	C0	R1	F0	P1	P0	X1	X0	X1	X0	1X	0X	1X	0X	CX	CX

Figure 2.17 : Tables de transfert des portes élémentaires avec l'algèbre à 18 valeurs

2.2.4.2 Modification du processus de traçage de chemins critiques

Le processus de traçage de chemins critiques a été modifié pour traiter les 18 symboles de la nouvelle algèbre. La Figure 2.18 représente le pseudo-code de l'algorithme adopté pour le traçage de chemins critiques. Au début, la liste initiale de lignes suspectes et le compteur de traçage CNT valent zéro (ligne 1 de la Figure 2.18). Pour chaque vecteur de test fautif (FTV) et pour chaque sortie primaire fautive (FPO) (lignes 2 et 3 de la Figure 2.18), nous commençons le traçage de chemins critiques (ligne 4 de la Figure 2.18). Les informations venant de FTV et de FPO obtenues durant la phase de test se trouvent dans le fichier de réponses de testeur (datalog). Si nous trouvons une ligne sensible non tracée (CNT=0) ou bien une ligne sensible avec la valeur 'X' ou 'Z' (ligne 5 de la Figure 2.18), nous considérons cette ligne comme ligne suspecte (ligne 6 de la Figure 2.18). Par contre, si la ligne a été déjà tracée (CNT!=0) (ligne 7 de la Figure 2.18), nous incrémentons seulement le compteur CNT (ligne 8 de la Figure 2.18). La fonction 'get_susp' permet de rassembler toutes les lignes tracées qui peuvent expliquer le FPO correspondant au FTV (ligne 10 de la Figure 2.18).

```
1: Curr_susps = ∅;  CNT_i = 0
2: For each failing test vector FTV do
3:  For each failing primary output FPO do
4:   Start CPT
5:   if (((LC_i==sensitive)||(LC_i==sensitive&&'X|Z'))&&(CNT_i==0))then
6:      Susps = LC_i, S_i, CNT_i
7:   Elseif (((LC_i==sensitive)||(LC_i==sensitive&&'X|Z'))&&(CNT_i!=0))then
8:      incr CNT_i
9:   Endif
10:  Curr_susps = get_susp (Susps,FTV, FPO)
11:  End CPT
12:  Done
13: Done
14: Print Curr_susps
```

Figure 2.18 : Algorithme de CPT en tenant compte des valeurs X et Z

Afin d'illustrer le processus de CPT en présence d'une valeur 'X' en sortie d'une boîte noire et sa propagation jusqu'au sorties primaires, considérons le circuit de la Figure 2.19. Dans cet exemple, on suppose que l'entrée E5 est connectée à la sortie d'une boîte noire. Les signaux obtenus sur chaque ligne du circuit après la simulation multi-valuée sont représentés sur la figure. Si on suppose que le vecteur de test TV = (C1, C1, C0, F0, CX, R1) a propagé des erreurs sur les sorties primaires S1 et S2, le processus de CPT trace toutes les lignes

sensibles à partir de S1 et S2 jusqu'au entrées primaires. Ces lignes critiques tracées par le processus de CPT sont respectivement {S_1, a, c, b, E_3, E_4} et {S_2, c, b, E_3, E_4}.

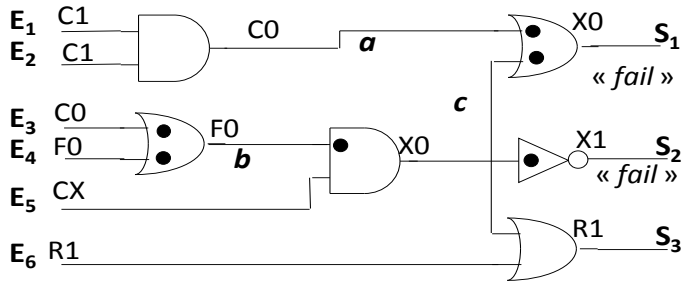


Figure 2.19 : Exemple de l'application de CPT

Comme dans la méthode de diagnostic initiale, nous mémorisons le nombre de fois où une ligne a été tracée lors du CPT. Nous obtenons cette information grâce au compteur CNT qui est associé à chaque ligne tracée. A la fin du CPT, comme dans la méthode initiale, nous classons les lignes suspectes en fonction de la valeur du CNT. Pour l'exemple de la Figure 2.19, le Tableau 2.9 donne la liste des lignes suspectes en fonction du CNT de chaque ligne.

Line	Symbole	CNT
c	X0	2
b	F0	2
E_3	C0	2
E_4	F0	2
S_2	X1	1
S_1	X0	1
a	C0	1

Tableau 2.9 : La liste des suspects

2.2.4.3 Modification du processus d'intersection

Nous avons étendu le processus d'intersection portant sur les symboles associés à une ligne critique pour prendre en charge les deux valeurs 'X' et 'Z'. L'opération d'intersection des listes produites par le CPT à partir de chaque sortie primaire fautive est donnée par le Tableau 2.10 (au lieu du Tableau 2.5).

∩s	C0	C1	F0	R1	P0	P1	D	SDW	0*	*0	CX	CZ
C0	C0	SDW	C0	SDW	C0	SDW	SDW	SDW	C0	C0	CX	CZ
C1	SDW	C1	SDW	C1	SDW	C1	SDW	SDW	SDW	SDW	CX	CZ
F0	C0	SDW	F0	D	P0	D	D	SDW	SDW	F0	CX	CZ
R1	SDW	C1	D	R1	D	P1	D	SDW	R1	SDW	CX	CZ
P0	C0	SDW	P0	D	P0	D	D	SDW	D	P0	CX	CZ
P1	SDW	C1	D	P1	D	P1	D	SDW	P1	D	CX	CZ
D	SDW	DW	D	D	D	D	D	SDW	D	D	CX	CZ
SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW
1*	SDW	C1	F0	SDW	P0	D	D	SDW	SDW	-	CX	CZ
***1**	SDW	C1	SDW	R1	D	P1	D	SDW	-	SDW	CX	CZ
CZ	CZ	CZ	CZ	CZ	CZ	CZ	CZ	SDW	CZ	CZ	-	CZ

Tableau 2.10 : Opération d'intersection entre signaux avec l'algèbre à 18 valeurs

Dans ce tableau, le symbole « ***0** » correspond aux symboles **X0, Z0** et le symbole « **0*** » correspond aux symboles **0X**, **0Z** (de même pour **1*** and ***1**). Par ailleurs, le symbole « - » représente une intersection non définie. Si ce symbole est rencontré lors du processus de traçage de chemins critiques, l'erreur sur la connexion ne peut en aucun cas venir d'une défaillance affectant la connexion en question. Elle vient nécessairement de la propagation d'une erreur amont. La défaillance ne pouvant être localisée sur la connexion en question, cette connexion sera donc à exclure de la liste des lignes suspectes.

L'affectation des modèles de fautes est ainsi étalée pour prendre en compte les symboles relatifs aux 'X' et 'Z'. En particulier, les symboles *0, 0*, *1, 1* conduisent à des défauts à la fois statiques et dynamiques.

Les modèles de fautes associées à chaque ligne suspecte peuvent être déterminés en fonction du symbole associé à la ligne (Tableau 2.11). Les lignes marquées d'un symbole stable (C0 ou C1) conduisent à des défauts statiques (colonne 1 et 2), tandis que les lignes marquées d'un symbole représentant une transition ou une possible transition liée à une faute statique et dynamique simultanément (colonne 3 et 4). Le symbole 'D' représente des retards sur les fronts montant et/ou descendant (colonne 5), et le symbole 'SDW' symbolise le cas des court circuits (colonne 6).

La phase d'affectation des modèles de fautes n'est pas en mesure d'associer tout type de modèles de fautes pour un suspect tracé par le symbole CX ou CZ, car il n'y a aucune information déterministe sur la valeur logique affectée par le suspect.

	C0	C1	F0,P0 *0,1*	R1,P1 *1,0*	D	SDW
Stuck at 0		X		x		
Stuck at 1	x		x			
Tn Stuck open	x		x			
Tn Stuck on		x		x		
Tp Stuck open		x		x		
Tp Stuck on	x		x			
Open 0		x		x		
Open 1	x		x			
Resistive open			x	x	x	
Dominant Or Bridge	x		x			
Dominant And Bridge		x		x		
Resistive Or			x			
Resistive And				x		
Byzantine	x	x	x	x	x	x
Strong Driver Wired	x	x	x	x	x	x
Wired Or	x		x			
Wired And		x		x		
Delay StF			x			
Delay StR				x		
Delay StR & StF			x	x	x	

Tableau 2.11 : Modèles de fautes associés en tenant compte des symboles 'X' et 'Z'

Reprenons l'exemple de la Figure 2.19, et la liste de suspects présentés au Tableau 2.9. Le Tableau 2.12 expose le rapport de diagnostic obtenu après l'opération d'intersection et la phase d'affectation des modèles de fautes. Il illustre l'ensemble complet des modèles de défauts qui peuvent être associés à chaque ligne suspecte.

Ligne	Symbole	Modèles de fautes
c	X0	Stuck-at 1, Tn stuck-open, Tp stuck-on, open 1, OR-bridge, resistive open, resistive OR, byzantine, SDW, wired OR, StF,StF &StR
b	F0	Stuck-at 1, Tn stuck-open, Tp stuck-on, open 1, OR-bridge, resistive open, resistive OR, byzantine, SDW, wired OR, StF,StF &StR
E_3	C0	Stuck-at 1, Tn stuck-open, Tp stuck-on, open 1, OR-bridge, byzantine, SDW, wired OR
E_4	F0	Stuck-at 1, Tn stuck-open, Tp stuck-on, open 1, OR-bridge, resistive open, resistive OR, byzantine, SDW, wired OR, StF,StF &StR
S_2	X1	Stuck-at 0, Tn stuck-on, Tp stuck-open, open 0, AND-bridge, resistive open, resistive AND, wired OR, StR,StF &StR
S_1	X0	Stuck-at 1, Tn stuck-open, Tp stuck-on, open 1, OR-bridge, resistive open, resistive OR, byzantine, SDW, wired OR, StF,StF &StR
a	C0	Stuck-at 1, Tn stuck-open, Tp stuck-on, open 1, OR-bridge, byzantine, SDW, wired OR

Tableau 2.12 : Rapport de diagnostic

2.3 Résultats expérimentaux

2.3.1 Processus expérimental

Le but des expérimentations est de valider la méthode de diagnostic présentée dans la partie précédente, et de comparer la résolution du diagnostic obtenue avec à celle obtenue avec un outil commercial. Rappelons que le terme « résolution » du diagnostic représente le nombre de lignes suspectes fourni par le processus de diagnostic.

Le processus expérimental a été réalisé de la façon suivante. Chaque séquence de vecteurs de test a été générée par un ATPG commercial en considérant le modèle de fautes de collage. Pour chaque circuit, nous avons ensuite injecté de façon aléatoire une faute dans la netlist, et nous avons simulé le circuit fautif pour obtenir la liste des vecteurs de test fautifs (*failing test vectors* ou FTVs) et les sorties primaires fautives correspondantes (*failing primary outputs* ou FPOs). Les fautes que nous avons injectées sont de type collage à 0 (à 1), faute de délai de type lent-à-monter (lent-à-descendre) et faute de court-circuit (Strong Driver Wired, Dominant Or/And). Pour chaque modèle de faute, nous avons réalisé un ensemble fixe et aléatoire (du point de vue de la localisation) d'injection de fautes.

2.3.2 Les différents circuits étudiés

Nous avons considéré deux systèmes sur puce de petites tailles synthétisés à partir d'une librairie STMicroelectronics (STM) en technologie 90 nm. Ces deux cas d'études contiennent les éléments réels d'un circuit industriel de STM. L'objectif ici est de mettre en évidence les avantages de notre approche par rapport à celle de l'état de l'art. Par la suite, nous montrerons des résultats expérimentaux obtenus sur des systèmes plus complexes afin de prouver la faisabilité de notre approche.

Les caractéristiques des deux systèmes sur puce expérimentés sont présentées dans le Tableau 2.13. Pour chaque circuit, la première colonne indique le nom de celui-ci. Les colonnes suivantes indiquent : le nombre de portes logiques (Logic Gates), le nombre de chaînes de scan (#Scan Chain), le nombre de bascules scan (#FFs), le nombre d'entrées primaires (#PIs) et de sorties primaires (#POs). Les deux dernières colonnes donnent des

indications sur la composition de chaque système en terme de nombre des IPs mémoires (#Mem) et de blocs logiques (#Logic blocks).

Circuit	Logic Gates	#Scan Chain	#FFs	#PIs	#POs	#Mem	#Logic Blocks
P1	2336	2	81	88	69	1	2
P2	237	1	13	20	10	1	4

Tableau 2.13 : Description des systèmes sur puce expérimentés

2.3.3 Résultats expérimentaux

Le Tableau 2.14 résume les résultats obtenus sur les deux systèmes décrits précédemment, dans le cas où la description de la mémoire est connue et donc remplacée par un bloc fonctionnel. La première colonne indique le nom du système. Les colonnes suivantes donnent : le type de faute injecté (Scenario), le nombre moyen de vecteurs de test fautifs (#FTV) et le nombre moyen de sorties primaires fautives (#FPO). La cinquième colonne (Reference Tool, version 2007.12) donne la résolution moyenne du diagnostic obtenue par l'outil de référence (sous-colonne 'Res'). La dernière colonne (eDERRIC) donne la résolution moyenne du diagnostic obtenue par eDERRIC (sous-colonne Res). Pour les deux outils de diagnostic, nous indiquons si l'outil fournit un succès complet 'Y' ou partiel 'P'. Un outil fournit un succès complet lorsque la faute injectée dans le système est toujours incluse dans la liste de suspects fournie par l'outil de diagnostic. Un outil fournit un succès partiel lorsque la faute injectée dans le système n'est pas toujours comprise dans la liste de suspects.

Circuit	Scenario	#FTV	#FPO	Reference Tool		eDERRIC	
				Res	Succ	Res	Succ
P1	Stuck-at	178.61	40.28	4.23	Y	25.28	Y
	Transition	12.71	87.71	5.4	P	19.14	Y
	Bridging	50.6	545.6	20.6	P	8.8	Y
P2	Stuck-at	13.5	13.5	1	Y	3	Y
	Transition	4	8	1	P	2	Y
	Bridging	15.17	15.17	3.3	P	7	Y

Tableau 2.14 : Résultats expérimentaux en utilisant des blocs mémoires fonctionnels

Le premier commentaire relatif à ces résultats concerne la différence en terme de résolution entre l'outil eDERRIC et l'outil de référence (en particulier dans le cas du système P1 avec l'injection de fautes de collage (stuck-at)). La résolution obtenue avec l'outil de référence est meilleure par rapport à celle obtenue avec eDERRIC (moins de suspects fournis). La différence de résolution s'explique par le fait que l'outil de référence considère uniquement le modèle de faute de collage dans son analyse. Il ne fournit donc que des suspects pouvant être liés à des fautes de collage. Comme eDERRIC considère un ensemble plus complet de modèles de fautes, il ne peut éliminer certaines lignes suspectes qui pourraient être à l'origine des erreurs observées. C'est par exemple le cas des lignes qui sont éliminées par l'outil de référence quand il existe un chemin sensible vers une sortie déclarée bonne. Dans notre cas, ces lignes ne peuvent pas être systématiquement éliminées notamment, par exemple, pour la prise en compte de pannes de délai, qui en fonction du retard introduit par la panne, ne produisent pas nécessairement une erreur sur chaque sortie sensibilisée.

Un deuxième commentaire concernant ces résultats est lié au cas de fautes de délai (transition). Dans ce cas, la résolution obtenue avec l'outil de référence est meilleure que celle obtenue avec eDERRIC. Comme précédemment, ceci provient du fait que l'outil de référence ne considère qu'un seul modèle de fautes. Cependant, on peut observer que dans ce cas là, le succès est partiel, ce qui signifie que la faute de transition injectée ne se retrouve pas toujours dans l'ensemble final de suspects fournit par l'outil de référence. Il apparaît donc que dans certains cas, le diagnostic fournit par l'outil de référence est erroné. A l'inverse, eDERRIC obtient toujours un succès complet, ce qui signifie que la faute injectée est toujours présente dans l'ensemble de suspects fourni. Le diagnostic avec eDERRIC est donc toujours complet.

Le troisième commentaire concernant ces résultats est lié au cas de fautes de court-circuit (bridging). Dans ce cas, les résultats obtenus par eDERRIC démontrent une meilleure résolution par rapport à ceux obtenus avec l'outil de référence. De plus, eDERRIC obtient toujours un succès complet, tandis que l'outil de référence n'obtient qu'un succès partiel et conduit donc, dans certains cas, à un diagnostic erroné.

Notons que si le nombre de suspects devient conséquent, cela pourrait potentiellement agrandir l'espace de recherche dans le circuit (localisation de faute) et donc ralentir le processus d'analyse de défaillances.

Le Tableau 2.15 présente les résultats obtenu sur le circuit P1 dans le cas où l'IP de la mémoire est remplacée par une boîte noire. L'objectif de cette expérience est d'analyser l'impact dû à la présence des symboles 'X' et 'Z' sur la résolution de diagnostic.

Circuit	Scenario	#FTV	#FPO	Reference Tool		eDERRIC	
				Res	Succ	Res	Succ
P1	Stuck-at	188.27	37.45	9.63	Y	50.59	Y
	Transition	11.06	87.81	43.75	P	35.68	Y
	Bridging	52.2	547.4	9.01	P	10.01	Y

Tableau 2.15 : Résultats expérimentaux en utilisant une boîte noire pour la mémoire

Dans cet exemple, comme prévu, nous remarquons une diminution de la résolution obtenue par rapport à celle obtenue dans le cas du Tableau 2.14. Cette remarque est valable quel que soit l'outil de diagnostic utilisé. Cela étant, on observe que, comme précédemment, eDERRIC obtient toujours un succès complet, alors que l'outil de référence n'obtient à nouveau qu'un succès partiel.

2.3.4 Cas d'étude

Pour démontrer à nouveau l'efficacité de notre approche, nous avons considéré un système sur puce d'une plus grande complexité que précédemment. Ce système a été proposé dans [App06] et est composé de trois IPs (Figure 2.20) :

(i) un microcontrôleur 8-bit, équipé d'une infrastructure *IP* (SBST-IIP)

(ii) une mémoire SRAM 32Kx8 bits, équipée d'un programme BIST

(iii) un multiplieur parallèle 16x16, équipé d'un IBIST

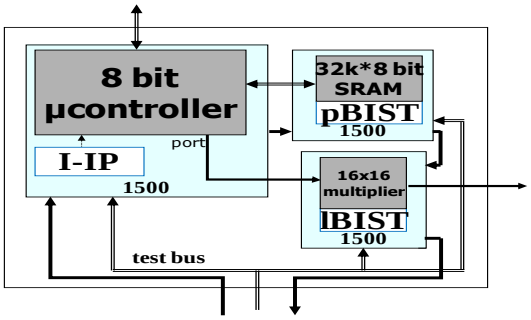


Figure 2.20 : Architecture de système sur puce

Les caractéristiques de ce système, synthétisé à partir d'une librairie STMicroelectronics en technologie 90 nm, sont données dans le Tableau 2.16. Notons que dans le cas de ce système, certaines bascules sont des bascules non-scan, et que le bloc mémoire a été remplacé par une boîte fonctionnelle. La génération des vecteurs de test et l'injection de fautes ont été réalisées de la même façon que celle décrite dans la partie 3.1.

#PIs	**#POs**	**#Logic Gates**	**#FFs**	**#FFs scan**	**#FFs non-scan**	**#Mem**	**#Logic blocs**
10	3	33989	2850	2844	6	1	2

Tableau 2.16 : Caractéristiques du système sur puce

Les résultats obtenus avec eDERRIC sont donnés dans le Tableau 2.17. Pour chaque injection de faute (colonne 1), nous donnons le nombre moyen de vecteurs de test fautifs (colonne 2) et le nombre moyen de sorties primaires fautives (colonne 3). Les deux dernières colonnes nous donnent la moyenne du nombre de suspects trouvés par l'outil de référence et eDERRIC (sous-colonne 'Res') et nous renseignent sur le succès complet (Y) ou partiel succès (P) obtenu par chaque outil (sous-colonne 'Succ').

Scenario	**#FTV**	**#FPO**	**Reference Tool (version 2007.12)**		**eDERRIC**	
			Res	**Succ**	**Res**	**Succ**
Stuck-at	40	77.48	4.8	Y	17.8	Y
Transition	21.45	22.63	9.09	Y	16.9	Y
Bridging	46.5	56	21	P	13.3	Y

Tableau 2.17 : Résultats expérimentaux

Dans un premier temps, nous pouvons observer que ces résultats conduisent aux mêmes remarques que celles faites précédemment pour le circuit P1, sauf le cas de fautes de transition où l'outil de référence fournit bien un succès complet. La résolution obtenue avec l'outil de référence peut être meilleure que celle obtenue avec eDERRIC, mais les résultats du diagnostic peuvent être erronés avec l'outil de référence alors qu'ils sont toujours corrects et complets avec eDERRIC.

2.4 Conclusion

Les expérimentations menées ont permis de valider toutes les étapes de développement de notre méthode de diagnostic basée sur l'approche « Effet-à-Cause ». Dans un premier temps, nous avons pu constater l'utilité de développer une méthode de diagnostic basée sur un ensemble complet de modèles de fautes, car là où l'outil de référence a échoué, notre méthode contenait toujours le bon site avec le bon modèle de faute associé, même si la résolution est élevée notamment dans le cas de fautes de collage. Dans un deuxième temps, nous avons adapté et validé la méthode telle qu'elle a été développée dans la première partie de ce chapitre sur des systèmes sur puce. Quel que soit le système sur puce considéré (avec une boîte noire ou sans), nous avons remarqué que notre méthode de diagnostic fournit toujours un succès complet.

En résumé, la méthode de diagnostic développée permet de prendre en compte différents types de fautes (collage «stuck-at», délai «transition», court-circuit «bridge»), alors que celle de l'outil de référence ne les traite pas de façon complète (Figure 2.21).

Tool	Stuck-at	Transition	Bridge
Reference tool (version 2007.12)	☺		
eDERRIC tool	☺	☺	☺

Figure 2.21 : Comparaison entre l'outil de référence et notre approche de diagnostic

Les avantages de la méthode de diagnostic proposée sont les suivants :

- Elle ne nécessite pas de simulation de fautes
- Elle ne nécessite pas de dictionnaire de fautes
- Plusieurs modèles de fautes peuvent être traités simultanément

L' inconvénient majeur de cette méthode est qu'elle ne permet pas de traiter les IPs purement séquentielles. Afin de remédier à ce problème, nous proposons dans le chapitre suivant une autre méthode de diagnostic basée sur une approche « Cause-à-Effet ».

Chapitre III

Développement d'une nouvelle méthode de diagnostic basée sur l'approche Cause-à-Effet

Plan

3.1 Nouvelle méthode de diagnostic basée sur l'approche Cause-à-Effet..... 74
3.1.1 Principe de la méthode de diagnostic développée 74
3.1.2 Phase de pré-traitement..... 76
3.1.3 Phase de localisation des sites potentiels de panne 80
3.1.4 Phase d'affectation des modèles de fautes..... 85
3.2 Adaptation de la méthode de diagnostic pour le traitement des blocs logiques séquentiels 89
3.2.1 Principe 89
3.2.2 Problématique liée aux circuits séquentiels 91
3.3 Résultats expérimentaux 95
3.3.1 Processus expérimental..... 95
3.3.2 Résultats sur les circuits de la famille ITC'99. 95
3.3.3 Cas d'étude industriel 97
3.4 Conclusion..... 103

L'objectif de ce chapitre est de présenter une nouvelle méthode de diagnostic basée sur une approche « Cause-à-Effet ».

Dans la première partie de ce chapitre, nous décrivons les deux phases principales de cette méthode de diagnostic : une phase de localisation des sites de pannes potentiels, qui s'appuie sur le paradigme SLAT, et une phase d'affectation des modèles de fautes aux sites de pannes potentiels, qui fait appel aux conditions de sensibilisation de pannes détaillées dans le premier chapitre.

Dans la deuxième partie, nous montrons comment cette méthode a été adoptée afin de traiter les circuits séquentiels. L'intérêt est de valider la technique développée sur des circuits utilisés dans un environnement industriel.

La troisième partie de ce chapitre est consacrée au processus expérimental et aux résultats obtenus sur des circuits benchmark de la famille ITC'99 et sur des systèmes sur puce fournis par la société STMicroelectronics. Des comparaisons seront effectuées avec un outil de référence pour montrer l'efficacité et la fiabilité de l'approche.

3.1 Nouvelle méthode de diagnostic basée sur l'approche Cause-à-Effet

3.1.1 Principe de la méthode de diagnostic développée

Dans l'introduction générale de ce manuscrit, nous avons mentionné l'existence de méthodes de diagnostic basées sur le paradigme SLAT et le concept de modèles de fautes composites [Hui04] [Hol07]. Nous avons indiqué que ces méthodes sont basées sur un processus d'analyse de la propagation des erreurs qui est effectué par l'intermédiaire d'un simulateur de fautes de collage et qui permet, malgré tout, le traitement simultané de plusieurs modèles de fautes.

Bien que cette approche permette de prendre en compte plusieurs modèles de fautes à la fois, et bien qu'elle permette le traitement de pannes simples ou multiples, deux inconvénients majeurs des techniques basées sur ce principe subsistent. D'une part, elles ne permettent de prendre en compte qu'un ensemble limité de modèle de fautes (collage et court-circuit pour [Hol07]). D'autre part, elles ne permettent pas d'identifier le type de défaillance associé à chaque ligne suspecte trouvée par le processus de diagnostic.

Afin de remédier aux inconvénients des techniques existantes, nous proposons dans ce chapitre une nouvelle méthode de diagnostic. Cette méthode est basée sur une approche «*Cause-à-Effet*» et comprend deux phases : une phase de localisation des sites de pannes potentiels qui s'appuie sur le paradigme SLAT, et une phase d'affectation des modèles de fautes aux sites de pannes potentiels qui fait appel aux conditions de sensibilisation de pannes détaillées dans le Chapitre I. Les avantages principaux de cette méthode sont l'utilisation d'un seul dictionnaire de fautes de collage pour prendre en compte différents modèles de fautes, ainsi que le traitement des fautes simples et des fautes multiples. Un autre avantage majeur de cette méthode est la possibilité de diagnostiquer à la fois les circuits combinatoires et les circuits séquentiels.

La Figure 3.1 présente le flot d'implémentation de la méthode de diagnostic proposée dans ce chapitre. Comme on peut le constater, une phase de pré-traitement est nécessaire pour fournir les informations requises par le processus de diagnostic proprement dit.

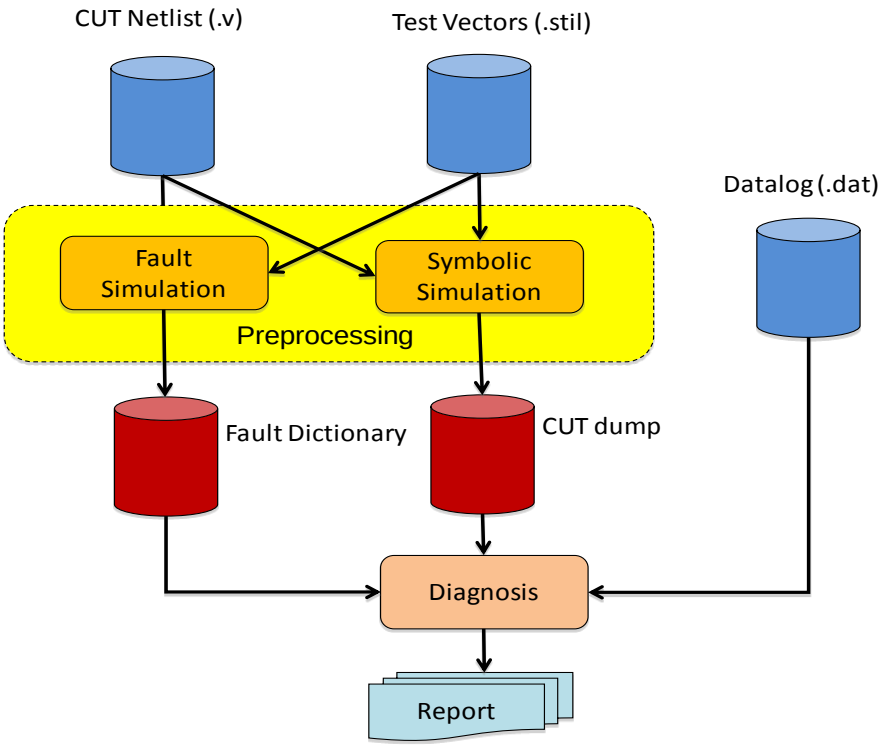


Figure 3.1 : Flot d'implémentation de la méthode de diagnostic

Les entrées principales de ce flot sont la description structurelle du circuit (CUT Netlist), la séquence de vecteurs de test (Test Vectors) et les réponses du circuit à ces vecteurs de test (Datalog) fournies par le testeur. La phase de pré-traitement consiste en une simulation de fautes de collage (Fault Simulation) et en une simulation multi-valuée (Symbolic Simulation). La simulation de fautes génère le dictionnaire de fautes (Fault Dictionary) contenant, pour chaque vecteur de test, les réponses attendues du circuit en présence de chaque faute de collage. La simulation multi-valuée permet d'obtenir, pour chaque vecteur de test, le symbole (par exemple, C0 ou C1) associé à chaque ligne du circuit. L'ensemble des symboles associés à chaque ligne pour tous les vecteurs de test est stocké dans un fichier 'CUT Dump' utilisé comme entrée du processus de diagnostic.

La sortie de ce flot est un rapport de diagnostic contenant une ou plusieurs lignes suspectes avec, pour chaque ligne suspecte, un ensemble de modèles de fautes associé.

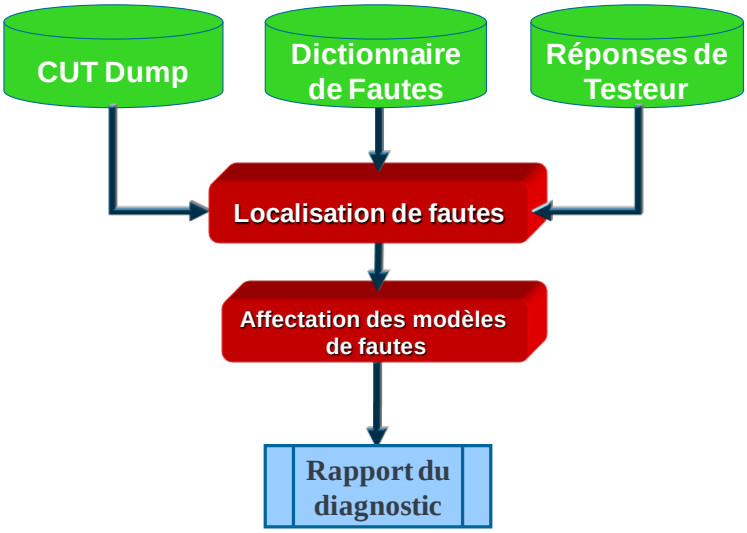


Figure 3.2 : Synoptique du processus de diagnostic développé

Comme indiqué précédemment, les entrées utilisées par la technique de diagnostic sont celles issues des blocs 'CUT Dump', 'Dictionnaire de Fautes' et 'Réponses de Testeur'. Le flot de diagnostic est composé de deux étapes (Figure 3.2) : (i) localisation de fautes et (ii) l'affectation des modèles de fautes.

La première partie de ce processus consiste à identifier les lignes suspectes du circuit permettant d'expliquer complètement ou partiellement les erreurs observées. L'identification des lignes suspectes est basée sur le paradigme SLAT qui analyse les données stockées dans le dictionnaire de fautes par rapport aux erreurs observées (venant du Datalog). Le résultat de ce processus est un ensemble de fautes de collage associé aux lignes suspectes.

La deuxième étape consiste à associer un ensemble de modèles de fautes à chaque ligne suspecte obtenue durant la phase précédente, en s'appuyant pour cela sur la connaissance du symbole associé à chaque ligne suspecte pour chaque vecteur de test fautif (et stockée dans le fichier 'CUT Dump'). Nous considérons tous les modèles de fautes vus au chapitre I.

3.1.2 Phase de pré-traitement

La phase de pré-traitement permet de générer deux fichiers de données nécessaires à la mise en œuvre du processus de diagnostic : (i) un dictionnaire de fautes contenant les réponses du circuit pour chaque vecteur de test en présence de chaque faute de collage simple

et (ii) un fichier contenant l'ensemble des symboles associés à chaque ligne du circuit pour tous les vecteurs déclarés fautifs à l'issue du test.

Le dictionnaire de fautes de collage est généré en utilisant le simulateur de fautes 'LIFTING' proposé dans [Bos08]. Comme indiqué dans [Hui04], seul le modèle de faute de collage est considéré pour construire ce dictionnaire puisque, dans un premier temps, seule la propagation des erreurs logiques est prise en compte par cette méthode de diagnostic basée sur le paradigme SLAT. Comme le montre la Figure 3.3, on peut constater que quel que soit le type de défaut présent sur une porte ou sa sortie, la ligne affectée se comporte toujours comme une ligne affectée par une faute de collage si une ou plusieurs conditions de sensibilisation sont respectées.

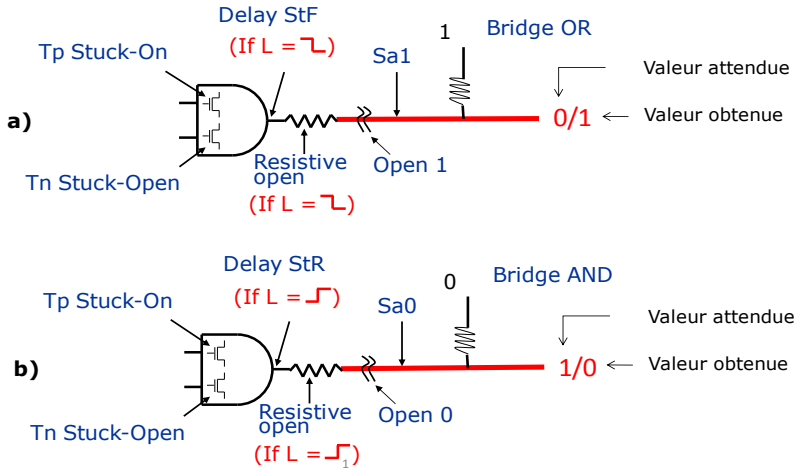


Figure 3.3 : les différents types de défauts (statiques et dynamiques)

Par ailleurs, pour traiter les systèmes sur puce et ainsi tenir compte des valeurs 'X' et 'Z', la simulation multi-valuée considère l'algèbre à dix huit valeurs vue dans le chapitre II.

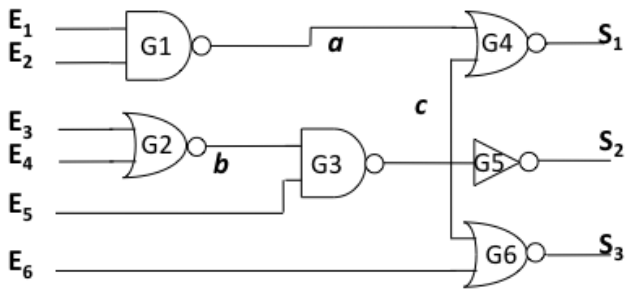


Figure 3.4 : Un exemple du circuit

Considérons l'exemple du circuit de la Figure 3.4. Ce circuit est composé de six entrées primaires, six portes logiques et trois sorties primaires. Neuf vecteurs de test sont considérés dans cet exemple et représentés sur la Figure 3.5 au format STIL.

```
"tv0": Call "capture" {"_pi"=110011; }
"tv1": Call "capture" {"_pi"=111000; }
"tv2": Call "capture" {"_pi"=010000; }
"tv3": Call "capture" {"_pi"=000011; }
"tv4": Call "capture" {"_pi"=100011; }
"tv5": Call "capture" {"_pi"=010011; }
"tv6": Call "capture" {"_pi"=011011; }
"tv7": Call "capture" {"_pi"=100111; }
"tv8": Call "capture" {"_pi"=100010; }
```

Figure 3.5 : Un exemple de séquence de test en format STIL

Le dictionnaire de fautes obtenu pour cet exemple est présenté dans le Tableau 3.1. La première colonne donne le nom de chaque ligne du circuit (Net). La deuxième colonne indique le modèle de faute considéré sur cette ligne (Collage-à- 0/1). Dans les colonnes suivantes, les réponses du circuit sont stockées pour chaque vecteur de test. La réponse est stockée sous la forme d'un mot binaire de longueur égale au nombre de sorties primaires (# PO). Si le $i^{ème}$ bit est égal à 1, cela signifie que la $i^{ème}$ sortie primaire est fautive, sinon ($i^{ème}$ bit égal à 0) la $i^{ème}$ sortie primaire est saine.

Net	FM	TV0	TV1	TV2	TV3	TV4	TV5	TV6	TV7	TV8
E1	Ca0	100	000	000	000	000	000	000	000	000
E1	Ca1	000	000	000	000	000	100	000	000	000
E2	Ca0	100	000	000	000	000	000	000	000	000
E2	Ca1	000	000	000	000	100	000	000	000	100
E3	Ca0	000	000	000	000	000	000	010	000	000
E3	Ca1	110	000	000	010	010	010	000	000	011
E4	Ca0	000	000	000	000	000	000	000	010	000
E4	Ca1	110	000	000	010	010	010	000	000	011
E5	Ca0	110	000	000	010	010	010	000	000	011
E5	Ca1	000	000	011	000	000	000	000	000	000
E6	Ca0	001	000	000	001	001	001	000	000	000
E6	Ca1	000	000	000	000	000	000	000	000	001
a	Ca0	000	000	000	100	100	100	000	000	100

a	Ca1	100	000	000	000	000	000	000	000	000
b	Ca0	110	000	000	010	010	010	000	000	011
b	Ca1	000	000	000	000	000	000	010	010	000
c	Ca0	000	111	011	000	000	000	010	010	000
c	Ca1	110	000	000	010	010	010	000	000	011
S1	Ca0	100	000	000	000	000	000	000	000	000
S1	Ca1	000	100	100	100	100	100	100	100	100
S2	Ca0	010	000	000	010	010	010	000	000	010
S2	Ca1	000	010	010	000	000	000	010	010	000
S3	Ca0	000	000	000	000	000	000	000	000	001
S3	Ca1	001	001	001	001	001	001	001	001	000

Tableau 3.1 : Le dictionnaire de fautes

Le Tableau 3.2 présente le fichier 'CUT Dump' généré pour l'exemple de la Figure 3.5. La simulation multi-valuée permet d'associer une valeur symbolique par vecteur de test à chaque ligne du circuit.

Net	TV0	TV1	TV2	TV3	TV4	TV5	TV6	TV7	TV8
E1	C1	C1	F0	C0	R1	F0	C0	R1	C1
E2	C1	C1	C1	F0	C0	R1	C1	F0	C0
E3	C0	R1	F0	C0	C0	C0	R1	F0	C0
E4	C0	C0	C0	C0	C0	C0	C0	R1	F0
E5	C1	F0	C0	R1	C1	C1	C1	C1	C1
E6	C1	F0	C0	R1	C1	C1	C1	C1	F0
a	C0	C0	R1	C1	C1	P1	C1	C1	C1
b	C1	F0	R1	C1	C1	C1	F0	C0	R1
c	C0	R1	C1	F0	C0	C0	R1	C1	F0
S1	C1	F0	C0	C0	C0	P0	C0	C0	C0
S2	C1	F0	C0	R1	C1	C1	F0	C0	R1
S3	C0	C0	C0	C0	C0	C0	C0	C0	R1

Tableau 3.2: Le fichier 'CUT Dump'

Notons que la génération du dictionnaire de fautes se fait une seule fois pour chaque circuit et pour chaque vecteur de test. Le temps nécessaire pour la simulation de fautes peut varier d'un circuit à l'autre en fonction de sa taille. L'espace disque occupé par le dictionnaire de fautes peut devenir un problème lors de sa lecture. Il y a plusieurs manières de représenter un

dictionnaire de fautes en utilisant plus au moins d'espace disque. Nous avons choisi la plus simple des représentations (présentation matricielle), mais aussi celle qui occupe le plus d'espace disque. Il est cependant possible de procéder plus efficacement en utilisant un arbre équilibré [Bop96].

3.1.3 Phase de localisation des sites potentiels de panne

La phase de localisation de fautes est la première étape du processus de diagnostic. Elle consiste à identifier les lignes suspectes dans le circuit. Pour permettre cette localisation de fautes, les informations utilisées sont celles issues de la phase de test et stockées dans le fichier Datalog : les vecteurs de test fautifs (FTVs), et les sorties primaires fautives (FPOs) associées à chaque vecteur de test fautif. Ces informations sont ensuite utilisées pour extraire une ou plusieurs de lignes suspectes à partir du dictionnaire de fautes.

La Figure 3.6 représente le pseudo-code de l'algorithme développé pour la localisation de fautes. La liste initiale de suspects correspond à l'ensemble total des lignes du circuit (ligne 1 de la Figure 3.6). Ensuite, pour chaque vecteur de test fautif (FTV) et pour chaque sortie primaire fautive (FPO), nous déterminons la liste courante de suspects (ligne 6 de la Figure 3.6). La fonction 'get_susp' recherche toutes les lignes dans la liste de suspects qui peuvent expliquer la sortie primaire fautive (FPO) correspondant au vecteur de test fautif (FTV).

```
1: Susps = Fault_Dictionary
2: Curr_susps = ∅
3: For each failing test vector FTV do
4:  For each failing primary output FPO do
5:
6:   Curr_susps = get_susp (Susps, FTV, FPO)
7:     If (Curr_susps <> ∅) then
8:      Susps = Curr_susps
9:     Else
10:     Print Susps
11:     Susps = Fault_Dictionary
12:     Curr_susps = get_susp (Susps,FTV, FPO)
13:    Endif
14:  Done
15: Done
16: Print Susps
```

Figure 3.6 : Algorithme de localisation de fautes

Si la liste courante de suspects n'est pas vide (ligne 7 de la Figure 3.6), alors nous la considérons comme une nouvelle liste de suspects (ligne 8 de la Figure 3.6). La Liste de suspects LS est définie comme suit : LS = {LC_0, LC_1, … LC_n} où LC_i représente l'identificateur (le nom) de la ligne suspecte « i ».

À la fin de ce processus, l'algorithme peut fournir une ou plusieurs listes de suspects. Chaque liste de suspects fournie est capable d'expliquer un sous-ensemble de vecteurs de test fautifs, sauf dans le cas où une seule liste de suspects est fournie, qui explique alors tous les vecteurs de test fautifs FTVs.

En analysant le nombre de listes de suspects obtenues, nous pouvons formuler plusieurs hypothèses sur la nature du défaut affectant le circuit, comme indiqué ci-dessous :

1. Une seule liste de suspects est générée à l'issue du processus de localisation de fautes. Dans ce cas, nous pouvons supposer qu'un seul défaut avec un seul effet est à l'origine des erreurs observées;

2. Plusieurs listes de suspects sont générées et chaque liste contient des suspects qui fournissent les mêmes sorties primaires fautives FPOs que celles fournies par le testeur. Dans ce cas, nous pouvons supposer qu'un seul défaut à effet multiple est à l'origine des erreurs observées;

3. Plusieurs listes de suspects sont générées et une liste au moins contient des suspects qui fournissent des sorties primaires fautives FPOs différentes de celles fournies par le testeur. Dans ce cas, on peut supposer que des défauts multiple à effet unique ou multiple, ou qu'un défaut unique à effets multiples, sont à l'origine des erreurs observées lors du test.

Un défaut unique est un défaut affectant une seul ligne du circuit, tandis qu'un défaut multiple est un défaut affectant plus d'une ligne du circuit. Un effet unique correspond à un défaut qui se comporte toujours comme un collage à '0' ou toujours comme un collage à '1' sur la même ligne de circuit. A l'inverse, un effet multiple correspond à un défaut qui se comporte comme un collage à '0' au cours de l'application d'un vecteur de test TV_i et comme un collage à '1' pendant le vecteur de test TV_j. Dans le cas général, on considère à la fois le cas où $TV_i = TV_j$ et le cas où $TV_i != TV_j$. Par exemple, une faute de transition de type «lent à monter (StR)» est un défaut unique avec un seul effet. Elle affecte une seule ligne du circuit et se comporte comme un collage à 0. La faute de court-circuit 'SDW' est un défaut unique à

effets multiple, car la ligne victime du court-circuit se comporte comme un collage à 0 si la ligne 'agresseur' porte la valeur logique '0', ou bien comme un collage à 1 si la ligne 'agresseur' porte la valeur logique '1'.

Notons que même dans le premier cas (une seule liste de suspects), nous ne pouvons pas exclure le fait que l'erreur observée soit causée par un défaut multiple qui se comporte comme un défaut unique (par exemple, à cause de phénomènes de masquage ou de sensibilisation multiples). Inversement, si on obtient plusieurs listes de suspects, nous pouvons exclure le fait que l'erreur observée soit causée par un défaut unique à effet unique.

Exemple 1 : Considérons à nouveau le circuit de la Figure 3.4 dans lequel une faute de transition de type «lent à descendre (StF)» affecte la ligne « c ». Le datalog fourni après la phase de test réel est représenté par le Tableau 3.3. Il indique, dans la première colonne, les vecteurs de test fautifs (FTV) et dans la deuxième colonne, les sorties primaires fautives (FPO) correspondantes.

FTV	**FPO**
3	S2
8	S2, S3

Tableau 3.3 : Datalog de la faute de transition « StF » sur la ligne « c »

Nous analysons le dictionnaire de fautes pour déterminer toutes les lignes qui sont à l'origine des erreurs rapportées dans le datalog. Pour le vecteur de test fautif FTV = 3, nous recherchons toutes les lignes qui, lors de la simulation de fautes, ont conduit à la réponse S1 **S2** S3 = 0 **1** 0 pour le vecteur de test TV3. Ces lignes sont reportées sur le Tableau 3.4.

Net	FM	TV0	TV1	TV2	**TV3**	TV4	TV5	TV6	TV7	TV8
E3	Ca1	110	000	000	**010**	010	010	000	000	011
E4	Ca1	110	000	000	**010**	010	010	000	000	011
E5	Ca0	110	000	000	**010**	010	010	000	000	011
b	Ca0	110	000	000	**010**	010	010	000	000	011
c	Ca1	110	000	000	**010**	010	010	000	000	011
S2	Ca0	010	000	000	**010**	010	010	000	000	010

Tableau 3.4 : Liste des suspects du vecteur de test TV3

De même, pour le vecteur de test fautif FTV = 8, nous recherchons dans le dictionnaire de fautes toutes les lignes ayant comme réponse le mot S1 **S2** **S3** = 0 **1** **1** pour le vecteur de test

TV8. Ces lignes sont reportées sur le Tableau 3.5. Les lignes restantes dans le dictionnaire de fautes (*i.e.* le Tableau 3.5) après avoir discriminer toutes les autres lignes correspondent à la liste finale de suspects. Elles sont les seules lignes du circuit en commun qui conduisent aux mêmes sorties primaires fautives que celles données par le datalog. La liste finale de suspects pour cet exemple est LS = {E3, E4, E5, b, c}.

Net	FM	TV0	TV1	TV2	**TV3**	TV4	TV5	TV6	TV7	**TV8**
E3	Ca1	110	000	000	**010**	010	010	000	000	**011**
E4	Ca1	110	000	000	**010**	010	010	000	000	**011**
E5	Ca0	110	000	000	**010**	010	010	000	000	**011**
b	Ca0	110	000	000	**010**	010	010	000	000	**011**
c	Ca1	110	000	000	**010**	010	010	000	000	**011**

Tableau 3.5 : Liste des suspects des vecteurs de test TV3 et TV8

Dans le cas d'un défaut simple avec un seul effet, nous pouvons affiner le diagnostic en tenant compte également des informations provenant des vecteurs déclarés sains lors du test réel. Soit # FTV_S le nombre de vecteurs de test fautifs obtenues à l'issue de la simulation de fautes pour une faute de collage unique considérée. Soit # FTV_D le nombre de vecteurs de test fautifs obtenus à l'issue du test réel en présence de ce même défaut dans le circuit sous test. Nous avons deux cas possibles :

1. # FTV_S> # FTV_D: le nombre de vecteurs de test fautifs obtenu en présence du défaut pendant le test réel est plus petit que le nombre de vecteurs de test fautifs obtenus par simulation. Le défaut simple ne peut pas être un collage simple.
2. # FTV_S = # FTV_D: le nombre de vecteurs de test fautifs obtenu en présence du défaut pendant le test réel est le même que le nombre de vecteurs de test fautifs obtenus par simulation. Le défaut simple peut être un collage, ou bien des défauts multiples ayant le comportement d'un défaut simple.

Dans le premier cas, nous pouvons assurer que le défaut ne correspond pas à une faute de collage simple, tandis que dans le second cas, on ne peut que supposer que le comportement observé peut être un défaut de collage ou des défauts différents qui se comportent exactement comme une faute de collage.

Dans l'exemple du Tableau 3.5, chaque ligne suspecte a un nombre de vecteurs de test fautifs obtenus par simulation (# FTV_S) égal à 5 (TV0, TV3, TV4, TV5, TV8). A l'inverse, le nombre de vecteurs de test fautifs obtenu lors du test réel (# FTV_D) est égal à 2 (FTV3, FTV8). Par conséquent, dans cet exemple, l'origine de l'erreur observée ne peut pas être une faute de collage.

Exemple 2 : Afin d'illustrer le cas où l'on a plusieurs listes de suspects, reprenons le circuit de la Figure 3.4 affecté cette fois d'une faute de court-circuit de type « Wired Or » entre la ligne E1 et la ligne E6. Le Tableau 3.6 représente les réponses fournies par le testeur (datalog) pour cet exemple là.

FTV	FPO
5	S1
8	S3

Tableau 3.6 : Datalog de l'exemple d'une faute de court-circuit de type « Wired Or »

L'algorithme de localisation de fautes consiste à extraire toutes les listes possibles de suspects. Nous suivons la même procédure que celle détaillée dans l'exemple 1 précédent. Nous obtenons cette fois deux listes de suspects LS1 = {E1, S1, a} et LS2 = {E6, S3} présentées respectivement dans le Tableau 3.7 et le Tableau 3.8. La liste LS1 représente l'ensemble de lignes qui, lors de la simulation de fautes, ont conduit à la réponse **S1** S2 S3 = **1** 0 0 (colonne TV5, Tableau 3.7) pour le vecteur de test fautif 5 (FTV = 5, Tableau 3.6). La liste LS2 représente l'ensemble de lignes qui ont conduit à la réponse S1 S2 **S3** = 0 0 **1** (colonne TV8, Tableau 3.8) pour le vecteur de test fautif 8 (FTV = 8, Tableau 3.6). Nous avons trouvé deux listes de suspects car, lors de la recherche dans le dictionnaire de fautes, nous n'avons obtenu aucune ligne suspecte en commun entre les vecteurs de test FTV5 et FTV8. Tous les suspects de chaque liste fournissent la même sortie fautive que celle observée durant le test réel : S1 pour FTV5 et S3 pour FTV8. On en conclut que dans ce cas, la faute peut être un seul défaut à effet multiple.

Pour préciser la nature de ce défaut, nous déterminons toutes les combinaisons possibles entre les éléments des deux listes LS1 et LS2, ce qui nous conduit à la liste finale de suspects suivante :

LS12 = {(E1, E6); (E1, S3); (S1, E6); (S1, S3); (a, E6), (a, S3)}

Net	FM	TV0	TV1	TV2	TV3	TV4	**TV5**	TV6	TV7	TV8
E1	Ca1	000	000	000	000	000	**100**	000	000	000
a	Ca0	000	000	000	100	100	**100**	000	000	100
S1	Ca1	000	100	100	100	100	**100**	100	100	100

Tableau 3.7 : Première liste de suspects

Net	FM	TV0	TV1	TV2	TV3	TV4	TV5	TV6	TV7	**TV8**
E6	Ca1	000	000	000	000	000	000	000	000	**001**
S3	Ca0	000	000	000	000	000	000	000	000	**001**

Tableau 3.8 : Deuxième liste de suspects

Cette liste sera ensuite utilisée par la phase suivante du processus de diagnostic (affectation des modèles de fautes) pour déterminer plus précisément le type et la nature de la faute.

3.1.4 Phase d'affectation des modèles de fautes

Le but de cette phase est d'analyser chaque ligne suspecte lors de l'application de vecteur de test fautif (FTV). La Figure 3.7 met en évidence l'algorithme utilisé pour associé à chaque ligne suspecte un symbole correspondant.

Pour chaque liste de suspects (ligne 1 de la Figure 3.7), nous considérons les dix huit symboles (ligne 2 de la Figure 3.7). Pour chaque ligne suspecte L, nous faisons une intersection entre tous les symboles affectés à cette ligne à chaque vecteur de test fautif (FTV) (ligne 4 et 5 de la Figure 3.7). Le résultat est stocké dans Curr_sym (ligne 7 de la Figure 3.7). L'opérateur d'intersection est défini dans le Tableau 3.9.

```
1: for each set of suspect Si
2:Curr_symb={C0,C1,R1,F0,P1,P0,CX,CZ,XZ,ZX,0X,X0,1X,X1,0Z,Z0,1Z,Z1}
3:
4:  For each line L do
5:   For each failing test vector FTV do
6:
7:    Curr_symb = Curr_symb ∩ get_v (L, FTV)
8:
9:   Done
10: Done
11:Done
```

Figure 3.7 : l'algorithme d'affectation des modèles de fautes

∩s	C0	C1	F0	R1	P0	P1	D	SDW	0*	*0	CX	CZ
C0	C0	SDW	C0	SDW	C0	SDW	SDW	SDW	C0	C0	CX	CZ
C1	SDW	C1	SDW	C1	SDW	C1	SDW	SDW	SDW	SDW	CX	CZ
F0	C0	SDW	F0	D	P0	D	D	SDW	SDW	F0	CX	CZ
R1	SDW	C1	D	R1	D	P1	D	SDW	R1	SDW	CX	CZ
P0	C0	SDW	P0	D	P0	D	D	SDW	D	P0	CX	CZ
P1	SDW	C1	D	P1	D	P1	D	SDW	P1	D	CX	CZ
D	SDW	DW	D	D	D	D	D	SDW	D	D	CX	CZ
SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW	SDW
1*	SDW	C1	F0	SDW	P0	D	D	SDW	SDW	-	CX	CZ
***1**	SDW	C1	SDW	R1	D	P1	D	SDW	-	SDW	CX	CZ
CZ	CZ	CZ	CZ	CZ	CZ	CZ	CZ	SDW	CZ	CZ	-	CZ

Tableau 3.9 : Opération d'intersection entre signaux avec l'algèbre à 18 valeurs

Le résultat final est une liste de suspects définie comme suit :

LS = {(LC_0/ S_0), (LC_1/ S_1), … (LC_n/ S_{n-1})}, où :

- LC_i est l'identificateur (le nom) de la ligne suspecte « i » trouvée durant la phase de localisation
- S_i est le symbole final (après intersection de signaux) porté par la ligne LC_i pour le vecteur considéré

Exemple 1 :

Net	TV3	TV8	∩
E3	**C0**	**C0**	**C0**
E4	**C0**	**F0**	**C0**
E5	**R1**	**C1**	**C1**
b	**C1**	**R1**	**C1**
c	**F0**	**F0**	**F0**

Tableau 3.10 : Intersection de symboles entre les vecteurs de test fautifs (exemple 1)

Revenons à l'exemple 1 avec la faute de type « lent à descendre (StF) » dans le circuit de la Figure 3.4. Le Tableau 3.10 illustre l'intersection de symboles entre les vecteurs de test fautifs pour chaque ligne suspecte. Le symbole (en bleu) associé à chaque ligne suspecte est déduit à l'aide des Tableaux 3.3 et Tableau 3.9. La liste finale est S1 = (E3/C0, E4/C0, E5/C1, B/C1, c/F0). Nous avons vu que pour cet exemple, la faute ne peut pas être une faute de collage,

donc seule la ligne « c » affectée par une faute de transition « lent à descendre (StF) » peut être à l'origine de l'erreur observée.

Exemple 2 :

Net	TV5
E1	**F0**
a	**P1**
S1	**P0**

Net	TV8
E6	**F0**
S3	**R1**

Tableau 3.11 : Intersection de symboles entre les vecteurs de test fautifs (exemple 2)

Dans le cas de l'exemple 2 (Tableau 3.7 et Tableau 3.8), nous obtenons S2 = ((E1/F0, E6/F0); (E1/F0, S3/R1); (S1/P0, E6/F0); (S1/P0, S3/R1); (a/P1, E6/F0), (a/P1, S3/R1)) (Tableau 3.11).

Dans le cas où chaque suspect correspond à un couple de lignes, nous pouvons en déduire que deux types de fautes peuvent être à l'origine des erreurs observées durant la phase de test. Les fautes peuvent être de type collage multiple ou bien de type court-circuit. Nous déterminons le type de court-circuit en fonction de la valeur finale du symbole associé à chaque ligne :

- Si les deux symboles ont la même valeur finale, la faute de court-circuit est de type Wired AND (valeur finale du symbole égale à 1) ou Wired OR (valeur finale du symbole égale à 0).
- Si les deux symboles ont une valeur finale différente, la faute de court-circuit est de type « byzantine ».

Dans le cas où chaque suspect correspond à un couple de ligne composé de la même ligne (la même ligne se trouve dans deux listes différentes), on peut retrouver le même symbole dans les deux listes ou bien deux symboles différents. Nous déterminons le type de court-circuit en fonction de la valeur finale du symbole associé à cette ligne.

- Si les deux symboles ont la même valeur finale, la faute de court-circuit est de type Dominant AND (valeur finale du symbole égale à 1) ou Dominant OR (valeur finale du symbole égale à 0).
- Si les deux symboles ont une valeur finale différente, la faute de court-circuit est de type « Strong Driver Wired ».

Dans l'exemple 2, nous avons trouvé que parmi tous les suspects (uniquement des fautes de court-circuit), les lignes E1 et E6 étaient affectées par une faute de court-circuit de type « Wired Or », qui correspond à la faute injectée.

Le Tableau 3.12 illustre l'ensemble de modèles de fautes associés à chaque symbole, qui peuvent être à l'origine des erreurs observées lors de la phase du test.

	C0	C1	F0,P0 *0,1*	R1,P1 *1,0*	D	SDW
Stuck at 0		X		x		
Stuck at 1	x		x			
Tn Stuck open	x		x			
Tn Stuck on		x		x		
Tp Stuck open		x		x		
Tp Stuck on	x		x			
Open 0		x		x		
Open 1	x		x			
Resistive open			x	x	x	
Dominant Or Bridge	x		x			
Dominant And Bridge		x		x		
Resistive Or			x			
Resistive And				x		
Byzantine	x	x	x	x	x	x
Strong Driver Wired	x	x	x	x	x	x
Wired Or	x		x			
Wired And		x		x		
Delay StF			x			
Delay StR				x		
Delay StR & StF			x	x	x	

Tableau 3.12 : Modèles de fautes associés en tenant compte des symboles 'X' et 'Z'

3.2 Adaptation de la méthode de diagnostic pour le traitement des blocs logiques séquentiels

Dans la première partie de ce chapitre, nous avons étudié un exemple de l'application de la méthode de diagnostic proposée sur un circuit combinatoire. Cependant, il existe aussi les circuits séquentiels de type « partial scan » qui contiennent des bascules qui ne sont pas scanables et ne peuvent pas être contrôlées et observées directement par le biais des chaînes de scan ou des entrées et sorties primaires. Il était donc nécessaire d'adapter la méthode initiale afin de prendre en compte cette problématique lors du diagnostic de pannes.

3.2.1 Principe

Du point de vue de son fonctionnement, un circuit séquentiel synchrone peut être vu comme une succession de blocs combinatoires indépendants de la notion de temps. Chaque intervalle de temps, appelé « time frame », représente le fonctionnement du circuit pendant une période d'horloge. Deux intervalles de temps sont séparés par un front d'horloge qui permet de mettre à jour les variables d'état (*i.e.* les bascules) du système. La Figure 3.8 donne une représentation de ce type. Les entrées primaires (PIs) et les sorties primaires (POs) sont représentées pour chaque intervalle de temps. Les pseudo-entrées primaires (PPIs) ont une valeur stockée dans les bascules durant l'intervalle de temps précédent ($PPIs_i = PPOs_{i-1}$). PPOs représente les pseudo-sorties primaires.

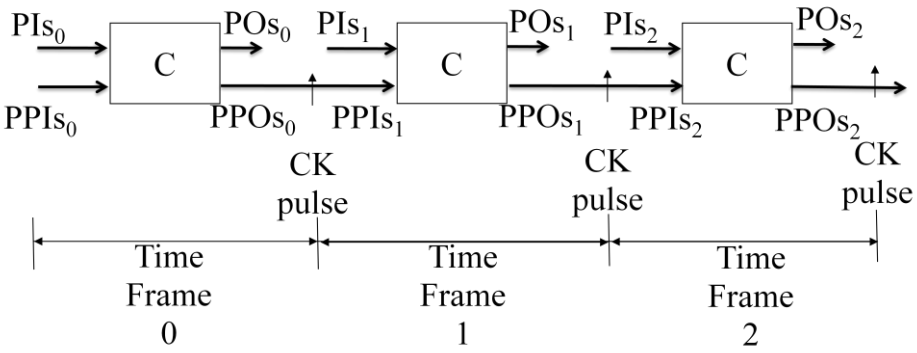


Figure 3.8 : Représentation d'un circuit séquentiel dans les intervalles de temps

Dans un circuit séquentiel, les effets d'une faute F sensibilisée durant l'intervalle de temps *j* peuvent être propagés à travers le circuit pour atteindre les sorties primaires POs et/ou les pseudo-sorties primaires PPOs. Si l'effet de la faute est observé sur une ou plusieurs sorties primaires, la faute F est alors détectée par le vecteur de test courant. Si l'effet de la faute est mémorisé dans une ou plusieurs bascules, cet effet sera ensuite propagé dans les intervalles de temps suivants jusqu'à atteindre une sortie primaire.

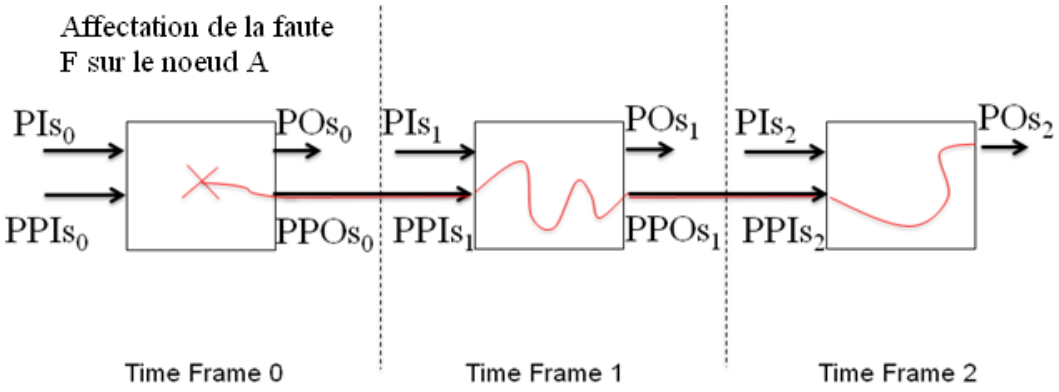


Figure 3.9 : Un exemple de propagation de la faute F sur le nœud A vers une sortie primaire

A titre d'exemple, la Figure 3.9 montre le cas d'une faute F sur le nœud A qui est sensibilisée durant l'intervalle de temps 0 (« time frame 0 »). Ici, l'effet de la faute est propagé à travers le circuit vers la pseudo-sortie $PPOs_0$. Durant l'intervalle de temps suivant, cet effet est à nouveau propagé à travers le circuit et atteint la pseudo-sortie $PPOs_1$. Au final, c'est dans l'intervalle de temps 2 que l'effet de la faute F atteint une sortie primaire (POs_2) et peut donc être observé.

Dans la méthode de diagnostic proposée dans ce chapitre, le dictionnaire de fautes est construit à partir de la simulation de chaque faute de collage avec l'ensemble complet des vecteurs de test. Les informations stockées dans le dictionnaire sont, pour chaque faute de collage, le vecteur de test qui détecte la faute et les sorties primaires sur lesquelles la faute est observée. Dans le cas d'un circuit séquentiel, ce processus ne change pas.

Considérons à nouveau l'exemple de la Figure 3.9. Le dictionnaire de fautes contiendra l'information suivante pour la faute F sur le nœud A :

Fault **F** net **A** ; TF2 ; POk

Où "Fault F Net A" indique le type F de faute (par exemple, collage à 0 ou à 1) et sa localisation (sur le nœud A), TF2 précise l'intervalle de temps au cours duquel la faute a été observée (ce qui donne donc une indication sur le vecteur de test qui a détecté la faute), et POk est la sortie primaire sur laquelle la faute a été observée.

En ce qui concerne la localisation de fautes, l'algorithme proposé dans le paragraphe 3.1.3 (« Phase de localisation des sites potentiels de panne ») est appliqué simplement, en considérant les erreurs observées durant le test et stockées dans le datalog, puis en comparant ces données avec celles du dictionnaire de fautes. Dans l'exemple de la Figure 3.9, la phase de localisation fournira donc, parmi tous les suspects, le nœud A, même si la faute correspondante a été sensibilisée dans un intervalle de temps précédent celui de l'observation.

La conclusion que l'on peut dresser est que le diagnostic de fautes pour les circuits séquentiels peut être réalisé de façon très simple avec la méthode proposée, puisque celle-ci s'appuie sur l'utilisation d'un dictionnaire de fautes qui est construit de la même façon que dans le cas d'un circuit combinatoire. Comme on peut le constater, le problème de la propagation de l'effet d'une faute à travers plusieurs intervalles de temps est traité de façon implicite avec cette méthode. Notons au passage que le traitement des circuits séquentiels avec une méthode de diagnostic de type « Effet à Cause », comme celle proposée dans le chapitre II, serait beaucoup plus complexe. En effet, dans ce cas, le traçage de chemins doit s'effectuer à partir de chaque sortie primaire fautive, en remontant jusqu'à l'intervalle de temps 0 correspondant au premier vecteur de test, et ceci indépendamment de l'intervalle de temps au cours duquel la faute a pu être observée. Le temps d'exécution nécessaire à cette opération, ainsi que le nombre final de suspects obtenus, deviendraient donc rapidement prohibitif dans ce cas.

Si la détermination de l'ensemble des suspects est relativement aisée avec cette méthode, il subsiste un problème qui est celui de la détermination des modèles de fautes associés à chacun de ces suspects. Ce problème est expliqué dans le paragraphe suivant.

3.2.2 Problématique liée aux circuits séquentiels

La méthode de diagnostic présentée dans ce chapitre, qui est basée sur une approche « Cause-à-Effet », permet de traiter les circuits combinatoires ainsi que les circuits

séquentiels. L'utilisation du paradigme SLAT et la construction d'un dictionnaire de fautes sont des éléments qui peuvent être utilisés indépendamment de la nature (combinatoire ou séquentiel) du circuit (ou bloc) ciblé pour le diagnostic.

Le problème qui se pose dans le cas des circuits séquentiels se situe au niveau de la phase d'affectation des modèles de fautes. Dans le cas d'un circuit combinatoire, nous pouvons facilement déterminer le symbole associé à chaque ligne suspecte lors de la simulation d'un vecteur de test fautif, et par la suite déduire le modèle de faute associé à chacune de ces lignes suspectes. A l'inverse, dans le cas d'un circuit séquentiel, une faute (erreur) observée durant l'application d'un vecteur de test peut avoir été sensibilisée par ce vecteur (qui est le vecteur déclaré fautif par le testeur), ou par un vecteur de test précédent. Pour illustrer cette situation, considérons le circuit b02 de la famille des benchmarks ITC'99. Nous avons injecté dans la description du circuit une faute de transition de type « lent-à-descendre (StF) » sur la ligne « U26/Q ».

FTV	**FPO**
10	u
11	u
12	u
65	u

Tableau 3.13 : Datalog du circuit b02

Le Tableau 3.13 donne les réponses obtenues lors de la simulation du circuit fautif, indiquant que quatre vecteurs de test (TV10, TV11, TV12 et TV65) ont été déclarés fautifs sur la même sortie « u ». La procédure de localisation de fautes a extrait une liste de suspects contenant 17 lignes, parmi lesquelles la ligne « U26/Q » (ce qui indique à priori un diagnostic correct). Le Tableau 3.14 fournit une partie du fichier 'CUT Dump' allant du vecteur de test TV7 au vecteur TV10. Nous observons que le symbole associé à la ligne « U26/Q » pour le vecteur TV10 (en réalité le vecteur fautif FTV10) est le symbole C1, ce qui indique qu'une faute de type « StF » n'a pas pu être sensibilisée par ce vecteur (auquel cas on aurait eu le symbole F0 associé à cette ligne). Par conséquent, on peut en déduire que la faute de transition de type « StF » a été sensibilisée non pas par le vecteur TV10, mais plutôt par les vecteurs TV7 ou/et TV8 comme le montre le Tableau 3.14. Ces vecteurs sont en effet les

seuls à avoir pu sensibiliser ce type de faute sur la ligne « U26/Q », faute qui a ensuite été propagée par le vecteur TV9 puis observée par le vecteur TV10.

TV	U26/Q
.	.
.	.
.	.
7	F0
8	F0
9	C0
10	**C1**

Précédent intervalle de temps

Vecteur de test fautif (FTV)

Tableau 3.14 : 'CUT Dump' du circuit b02

Dans ce qui suit, l'objectif est de montrer comment il est possible de prendre en compte ce genre de situation lors du processus d'affectation des modèles de fautes.

Une solution possible est de modifier le processus de simulation de fautes pour faire en sorte que les informations nécessaires soient enregistrées durant cette phase. Ces informations nécessaires concernent la connaissance de l'intervalle de temps au cours duquel une faute observée dans un « time frame » *j* a été sensibilisée. Dans l'exemple de la Figure 3.9, la faute observée au cours de l'intervalle de temps 2 a été sensibilisée dans l'intervalle de temps 0. En modifiant le simulateur de fautes pour faire en sorte que le « time frame » de sensibilisation soit enregistré pour chaque faute du dictionnaire en même temps que le « time frame » d'observation (ou vecteur de test) et les sorties primaires sur lesquelles la faute est observable, il est alors possible de résoudre le problème mentionnée ci-dessus. En effet, dans ce cas, il devient aisé de déterminer le symbole associé à la ligne potentiellement fautive (nœud A dans l'exemple de la Figure 3.9) dans le « time frame » de sensibilisation, et d'en déduire ainsi le modèle de faute correspondant au suspect.

En réalité, ce type d'information est contenu de manière implicite dans le simulateur. Le seul problème est qu'en utilisant un simulateur de fautes industriel, comme nous l'avons fait dans notre étude, l'information n'est pas accessible directement. La solution consisterait donc à utiliser un simulateur de fautes que l'on aurait modifié pour qu'il fournisse ce type d'information.

Cette solution possible au problème mentionné ci-dessus n'a pas été implémentée et par conséquent validée de façon expérimentale. Il s'agit donc d'une extension possible à mettre en œuvre dans un travail futur. Néanmoins, il est important de noter que, même si la solution que nous avons implémentée ne permet pas, à l'heure actuelle, de fournir une information parfaitement fiable sur le modèle de fautes associé à chaque suspect dans le cas d'un circuit séquentiel, cela ne remet pas en cause la fiabilité de notre approche quant à la localisation des suspects fournis. Dans tous les cas, la liste de suspects fournis par notre méthode de diagnostic est toujours correcte (tous les suspects fournis peuvent être à l'origine des erreurs observées).

Pour conclure cette section, nous renvoyons le lecteur à l'Annexe (en fin de document) dans laquelle nous détaillons cette méthode de diagnostic dans le cas de circuits séquentiels affectés de fautes de transition. Il s'agit donc d'un cas d'étude limité au traitement d'un certain type de fautes, sachant que tous les modèles de fautes peuvent être pris en compte par l'approche proposée. Le résultat principal de l'article reporté dans cette Annexe est que la méthode de diagnostic proposée est capable de localiser avec précision la ligne ou la porte affectée par une faute de transition, même si le modèle de faute ne peut pas être déterminé avec certitude.

3.3 Résultats expérimentaux

3.3.1 Processus expérimental

Le but des expérimentations est de valider la méthode de diagnostic présentée dans la partie précédente, et de comparer la résolution du diagnostic obtenue avec à celle obtenue avec un outil commercial. Rappelons que le terme « résolution » du diagnostic représente le nombre de lignes suspectes fourni par le processus de diagnostic.

Les expérimentations ont été effectuées en deux étapes. La première étape consiste à valider notre méthode de diagnostic sur des circuits benchmark de la famille ITC'99. La deuxième étape vise à montrer l'applicabilité de la technique de diagnostic sur des circuits industriels.

Le processus expérimental a été réalisé de la façon suivante. Chaque séquence de vecteurs de test a été générée par un ATPG commercial en considérant le modèle de fautes de collage. Pour chaque circuit, nous avons ensuite injecté de façon aléatoire une faute dans la netlist (entre 10 à 20 injections par modèle de faute), et nous avons simulé le circuit fautif pour obtenir la liste des vecteurs de test fautifs (*failing test vectors* ou FTVs) et les sorties primaires fautives correspondantes (*failing primary outputs* ou FPOs). Les fautes que nous avons injectées sont de type collage à 0 (à 1), faute de délai de type lent-à-monter (lent-à-descendre) et faute de court-circuit (Strong Driver Wired, Dominant Or/And). Pour chaque modèle de faute, nous avons réalisé un ensemble fixe et aléatoire (du point de vue de la localisation) d'injection de fautes.

3.3.2 Résultats sur les circuits de la famille ITC'99.

Le Tableau 3.15 résume les résultats obtenus sur les circuits benchmark de la famille ITC'99 (B08, B09, B10, B11 et B12). La première colonne (circuit) indique le nom du circuit benchmark. Les colonnes suivantes donnent : le type de faute injecté (Scenario), le nombre moyen de vecteurs de test fautifs (#FTV) et le nombre moyen de sorties primaires fautives (#FPO). La cinquième colonne (Reference Tool) donne la résolution moyenne du diagnostic obtenue par l'outil de référence (version 2007.12) (sous-colonne 'Res'). La dernière colonne (Our Approach) donne la résolution moyenne du diagnostic obtenue par notre technique de

diagnostic (sous-colonne 'Res'). Pour les deux outils de diagnostic, nous indiquons si l'outil fournit un succès complet 'Y' ou partiel 'P' (sous-colonne 'Succ'). Un outil fournit un succès complet lorsque la faute injectée dans le système est toujours inclue dans la liste de suspects fournie par l'outil de diagnostic. Un outil fournit un succès partiel lorsque la faute injectée dans le système n'est pas toujours comprise dans la liste de suspects.

Circuit	Scenario	#FTV	#FPO	Reference Tool		Our Approach	
				Res	Succ	Res	Succ
B08	Stuck-at	4.8	5.2	5.4	Y	5.4	Y
	Transition	1.9	2.5	6.7	P	7.6	Y
	Bridging	4.8	5.3	9.5	P	8.3	Y
B09	Stuck-at	6.1	6.1	7.6	Y	7.6	Y
	Transition	2.5	4.4	8.5	P	10.9	Y
	Bridging	4.6	5.8	21.3	P	24.3	Y
B10	Stuck-at	4.9	4.9	7.4	Y	7.4	Y
	Transition	2.6	4.5	12.7	P	11.1	Y
	Bridging	5.8	8.4	13.5	P	10.5	Y
B11	Stuck-at	10.1	10.1	8	Y	8	Y
	Transition	4	9.6	9.2	P	15.5	Y
	Bridging	6.2	9.4	12	P	22.2	Y
B12	Stuck-at	31.5	49.6	6.3	Y	6.3	Y
	Transition	3.8	9.9	10	Y	11.8	Y
	Bridging	30.3	33.4	15.1	P	19	Y
B13	Stuck-at	12	17	7	Y	7	Y
	Transition	12.3	7.5	9.9	P	6.7	Y
	Bridging	10.4	12.6	8.6	P	10.1	Y

Tableau 3.15 : Résultats expérimentaux sur des circuits benchmark de la famille ITC'99

Le premier commentaire relatif à ces résultats concerne la similitude en terme de résolution entre notre méthode de diagnostic et l'outil de référence (en particulier dans le cas avec l'injection de fautes de collage « stuck-at »). Par contre, il y a une légère différence de résolution concernant les fautes de délai « transition » et de court-circuit « bridging ». Cette différence s'explique par le fait que l'outil de référence considère uniquement le modèle de faute de collage dans son analyse. Il ne fournit donc que des suspects pouvant être liés à des fautes de collage. Comme notre technique de diagnostic considère un ensemble plus complet de modèles de fautes, elle ne peut éliminer certaines lignes suspectes qui pourraient être à l'origine des erreurs observées. Elle propose donc un ensemble de suspects plus grand mais plus réaliste.

Un deuxième commentaire concernant ces résultats est lié au cas de fautes de délai « transition ». Dans ce cas, la résolution obtenue avec l'outil de référence est légèrement meilleure que celle obtenue avec notre technique de diagnostic. Comme précédemment, ceci provient du fait que l'outil de référence ne considère qu'un seul modèle de fautes. Cependant, on peut observer que dans ce cas là, le succès est partiel, ce qui signifie que la faute de transition injectée ne se retrouve pas toujours dans l'ensemble final de suspects fournit par l'outil de référence. Il apparaît donc que dans certains cas, le diagnostic fournit par l'outil de référence est erroné, car la faute injectée n'est pas inclue dans la liste de suspects. A l'inverse, notre technique de diagnostic obtient toujours un succès complet, ce qui signifie que la faute injectée est toujours présente dans l'ensemble de suspects fourni. Le diagnostic avec notre technique de diagnostic est donc toujours complet.

Le troisième commentaire concernant ces résultats est lié au cas de fautes de court-circuit « bridge ». Dans ce cas, la résolution obtenue avec l'outil de référence est également légèrement meilleure que celle obtenue avec notre technique de diagnostic. Néanmoins, notre technique de diagnostic obtient toujours un succès complet, tandis que l'outil de référence n'obtient qu'un succès partiel et conduit donc, dans certains cas, à un diagnostic erroné.

3.3.3 Cas d'étude industriel

Pour démontrer à nouveau l'efficacité de notre approche, nous avons considéré un système sur puce industriel fourni par la société STMicroelectronics appelé « SoC STM » par la suite. Comme dans le cas d'étude du chapitre II, les IPs de ce système autres que les IPs logiques ont été considérées comme des boîtes noires. Synthétisé en technologie 90 nm, fabriqué et testé par la société STMicroelectronics [App08], il est composé de trois IPs :

(i) un microcontrôleur 8-bit, équipé d'une infrastructure *IP* (SBST-IIP)

(ii) une mémoire SRAM 4 * 16Kx8 bits, équipée d'un micro programme BIST (MPB)

(iii) un multiplieur parallèle 16x16, équipé d'un BIST

L'architecture de ce système sur puce et son layout sont représentées respectivement sur la Figure 3.10 et la Figure 3.11.

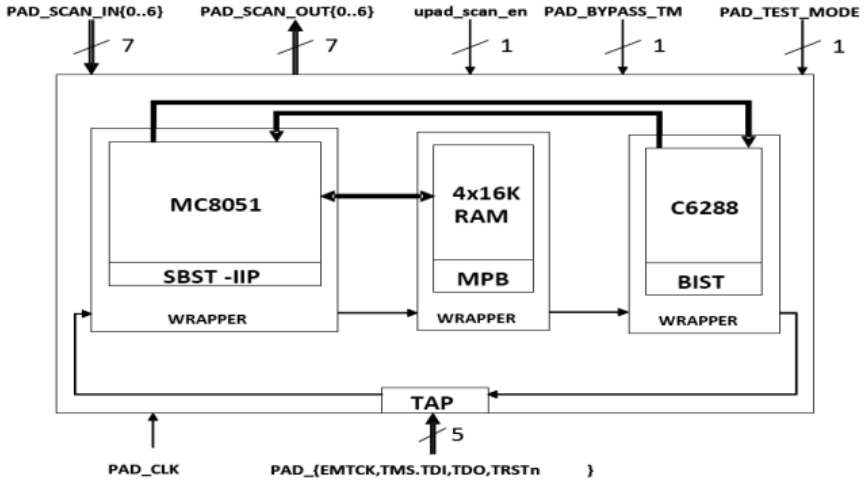


Figure 3.10 : Architecture de système sur puce

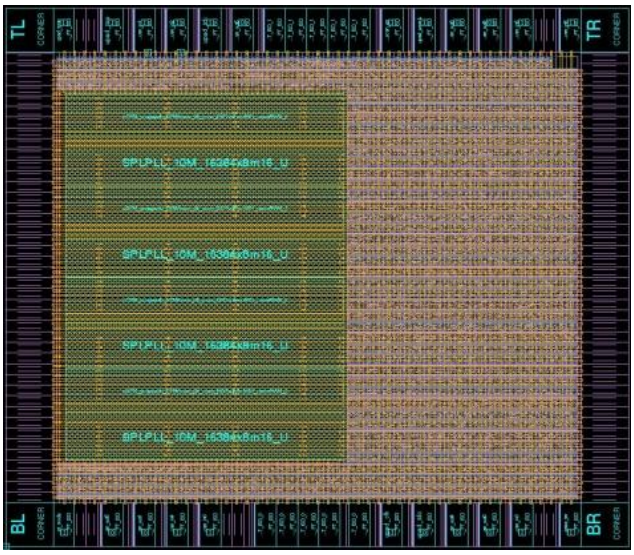


Figure 3.11 : L'image de layout de système sur puce

Les caractéristiques du circuit sont données dans le Tableau 3.16. Nous donnons respectivement le nombre d'entrées primaires et de sorties primaires du circuit (colonne 1 et 2), puis le nombre des portes logiques (colonne 3), de chaines de scan (colonne 4), de bascules (colonne 5), de bascules scanables (colonne 6) et de bascules non-scan (colonne 7). Enfin, nous indiquons respectivement dans les deux dernières colonnes la composition du circuit en terme de nombre de mémoires (#Mem) et de nombre de blocs logiques (#Logic Blocks). Notons qu'il s'agit ici d'une IP logiques de type « partial scan ».

#PIs	*#POs*	*#Logic Gates*	*#Scan Chains*	*#FFs*	*#Scan FFs*	*#Non-scan FFs*	*#Mem*	*#Logic Blocks*
15	8	43608	7	2859	2853	6	1	2

Tableau 3.16 : Caractéristiques du SoC

Les vecteurs de test ont été fournis par les ingénieurs de la société STMicroelectronics. La génération du dictionnaire de fautes a été réalisée par le biais de l'outil commercial TetraMax ([Tmax07], version 2007.12) en utilisant le modèle de faute de collage. 1448 vecteurs de test ont été générés pour tester les 177634 fautes de ce système avec un taux de couverture de 97.71%. La taille mémoire du dictionnaire de fautes obtenu est de 5Go et le temps nécessaire à la génération de ce dictionnaire a été de 7 jours (en utilisant 4 machines de type SUN).

Nous avons réalisé deux types d'expérimentations. Les premières expérimentations ont visé à valider l'adaptation de notre technique (décrite dans le paragraphe 3.2) sur un système sur puce industriel réel en traitant uniquement les IPs logiques séquentielles de type « partial scan ». Le bloc mémoire a été remplacé par une boîte noire. Pour ces expérimentations, nous avons injecté à chaque fois et de façon aléatoire une faute dans la netlist des IPs logiques (entre 10 à 20 injections par modèle de faute), et nous avons simulé le circuit fautif pour obtenir les datalogs. L'objectif a été de vérifier que la faute injectée se trouvait bien à chaque fois parmi les lignes suspectes fournies par notre technique de diagnostic.

Les expérimentations suivantes ont eu pour but d'évaluer la technique de diagnostic par rapport à un outil de référence dans un environnement industriel. Dans ce contexte, les datalogs n'ont pas été générés après une simulation de fautes du circuit fautif, mais ont été fournis directement à partir des résultats du testeur. Dans ce cas, nous n'avons donc aucune idée, ni sur le nombre de fautes, ni sur le type de fautes qui ont causé la défaillance du circuit.

Circuit	**Scenario**	**#FTV**	**#FPO**	**Reference Tool**			**Our Approach**		
				Res	**Succ**	**CPU(mn)**	**Res**	**Succ**	**CPU(mn)**
SoC STM	Stuck-at	720.77	917	5	Y	9	4.38	Y	54
	Transition	345.88	346	4.62	Y	10	4.25	Y	72
	Bridging	718.33	718	9.5	P	4	5.16	Y	41

Tableau 3.17 : Résultats de diagnostic des premières expériences (fautes injectées)

Les résultats de diagnostic des premières expériences sur le circuit « SoC STM » sont résumés dans le Tableau 3.17. Pour l'outil de référence, nous indiquons dans la sous-colonne FM (Fault Model) si l'outil fournit, parmi les suspects, le bon modèle de faute injecté ('G' pour Good) ou si, au contraire, il fournit un modèle de faute incorrect ('W' pour Wrong).

Le premier commentaire relatif à ces résultats est lié à la résolution du diagnostic obtenue dans le cas où une faute de collage « stuck-at » a été injectée dans la netlist du système. Cette résolution est légèrement meilleure que celle obtenue avec l'outil de référence.

Le deuxième commentaire concerne la résolution du diagnostic obtenue dans le cas où une faute de délai « transition » a été injectée dans la netlist du système. Comme précédemment, nous pouvons observer que cette résolution est légèrement meilleure que celle obtenue avec l'outil de référence. Cependant, il apparaît que parmi les suspects fournis par l'outil de référence, seules des fautes de collage ont été identifiées (ce qui est normal puisque cet outil ne traite que ce modèle de faute), alors la faute injectée est une faute de transition. Ceci montre clairement que, même si le site de la faute a été identifié correctement avec l'outil de référence, le modèle de faute associé est incorrect ('W'). En revanche, les résultats obtenus avec notre méthode montrent clairement que le site de la faute est déterminé à chaque fois avec exactitude ('Y').

Le troisième commentaire concerne la résolution du diagnostic obtenue dans le cas où une faute de court-circuit « bridging » a été injectée dans la netlist du système. Nous pouvons observer que dans ce cas là, la faute injectée ne figure pas dans la liste de suspects fournis par l'outil de référence (d'où l'indication de succès partiel 'P'). En revanche, notre méthode fournit un succès complet, puisque la faute injectée fait bien partie de la liste de suspects obtenue. Nous remarquons aussi que l'outil de référence fournit un modèle de faute erroné ('W'). Suite à la limitation du paragraphe 2.2.3, nous n'avons pas affecté les modèles de fautes pour les lignes suspectes. Notons que le temps d'exécution (CPU) total de notre méthode reste relativement raisonnable (une cinquantaine de minutes).

Considérons maintenant le deuxième type d'expérimentations réalisées sur le circuit « SoC STM ». Nous comparons à nouveau les résultats de diagnostic fournis par la technique proposée avec ceux obtenus par l'outil de référence. Nous rappelons que dans ce cas, nous avons récupéré les données fournies par le testeur suite au test réel de ce système sur puce

défectueux, et n'avons donc aucune idée ni sur la localisation des fautes présentes dans le système, ni sur leur modèle de faute. Les datalogs fournis par les ingénieurs de test proviennent de trois lots (G1, G2, G3); chaque lot contient plusieurs wafers et dans chaque wafer, nous avons plusieurs datalogs (#Datalogs). Nous soulevons une limitation de notre méthode de diagnostic pour traiter les défauts affectant les bascules de chaines de scan, puisque la liste des suspects générée est vide. En effet, nous n'avons retenu que les datalogs qui ne reportent pas ces types de défauts.

Circuit	**#Data logs**	**#FTV**	**#FPO**	**Reference Tool**		**Our Approach**	
				Res	**CPU (mn)**	**Res**	**CPU (mn)**
SoC STM	58	276.67	341.31	8	55	10.96	88
	91	338.52	726.12	32.5	192	34.43	536
	116	263	324.63	14.65	82	15.32	213

Tableau 3.18 : Résultats de diagnostic dans un environnement industriel

Le Tableau 3.18 résume les résultats obtenus. La première colonne indique le nom du circuit. Les colonnes suivantes indiquent : le nom du lot (#Lots), le nombre de wafers (#Wafer), le nombre total de datalogs (#Datalogs), le nombre moyen de vecteurs de test fautifs (#FTV) et le nombre moyen de sorties primaires fautives (#FPO). L'avant dernière colonne (Reference Tool) donne la résolution moyenne de diagnostic obtenue par l'outil de référence. La dernière colonne (Our approach) donne la résolution obtenue par la technique proposée (sous-colonne Res) et le temps nécessaire au diagnostic (sous-colonne CPU).

A partir de ces résultats, nous observons que le nombre de suspects fourni par notre méthode de diagnostic est légèrement supérieur à celui fourni par l'outil de référence. Nous pouvons donner les explications suivantes à cela.

La première explication est liée à la différence de résolution dans le cas du lot G1. Afin de comprendre cette différence, nous avons analysé en détail les résultats obtenus avec un outil de STMicroelectronics capable de traiter les fautes multiples en utilisant l'outil de référence. Nous avons constaté plusieurs bascules de chaines de scan parmi les suspects, ces derniers ont un comportement des fautes multiples. Comme notre méthode ne les traite pas encore, il paraît normal de voir un nombre plus important de suspects identifiés.

La deuxième explication concernant ces résultats est liée au cas des lots G2 et G3. Dans ce cas, la résolution obtenue avec l'outil de référence est légèrement meilleure à celle obtenue avec notre outil de diagnostic. Comme dans le cas des circuits ITC'99, cette différence s'explique par le fait que l'outil de référence considère uniquement le modèle de faute de collage dans son analyse. Il ne fournit donc que des suspects pouvant être liés à des fautes de collage. Comme notre technique de diagnostic considère un ensemble plus complet de modèles de fautes, elle ne peut éliminer certaines lignes suspectes qui pourraient être à l'origine des erreurs observées. Elle propose donc un ensemble de suspects plus important.

En ce qui concerne le temps d'exécution (sous-colonne CPU), nous observons que notre outil de diagnostic reste moins rapide que l'outil de référence. Des améliorations algorithmiques et une implémentation optimisée pourraient très certainement remédier à ce problème.

3.4 Conclusion

Les expérimentations menées ont permis de valider toutes les étapes de développement de notre méthode de diagnostic basée sur l'approche « Cause-à-Effet ». La validation a été consolidée en deux étapes. Dans la première étape, nous avons validé le processus de diagnostic sur les circuits benchmark de la famille ITC'99. Dans la deuxième étape, nous avons appliqué la méthode de diagnostic sur les blocs logiques séquentiels d'un système sur puce de la société STMicroelectronics.

Dans un premier temps, nous avons pu constater que la résolution et la précision (en particulier dans le cas des erreurs générés par l'injection de fautes) de notre méthode de diagnostic logique est équivalente pour les fautes de collage et meilleure pour les fautes de transition et court-circuit que celles obtenues avec l'outil de référence. Dans un deuxième temps, l'utilisation de datalogs venant directement du testeur a montré une légère diminution de performances de notre résolution de diagnostic.

En résumé, la méthode de diagnostic développée permet de traiter un ensemble de fautes (St: collage, Tr: transition, Br: court-circuit) potentielles pouvant affecter un circuit industriel « full scan » ou (partiellement) séquentiel (Figure 3.12), et cela en utilisant uniquement le modèle de faute de collage pour générer le dictionnaire de fautes. Les avantages et inconvénients de cette méthode de diagnostic sont représentés ci-dessous :

Circuit	Benchmark						SoC STM		
Type	Combinational			Sequential			Sequential		
Tools/Fault Models	St	Tr	Br	St	Tr	Br	St	Tr	Br
Reference Tool (2007.12)	☺			☺			☺		
Our Approach (Cause-Effect)	☺	☺	☺	☺	☺	☺	☺	☺	☺

Figure 3.12 : Comparaison entre l'outil de référence et notre approche de diagnostic

Avantages :

- Traitement des circuits industriels « full scan » ou (partiellement) séquentiel

- Utilisation d'un seul modèle de faute (collage) pour la génération du dictionnaire
- Résolution meilleure par rapport à celle obtenue avec l'outil de référence (version 2007.12) dans le cas où une faute simple de type délai ou court-circuit affecte le circuit.
- L'affectation d'un modèle de fautes associé à un symbole permet de cibler plusieurs fautes.

Inconvénients :

- La taille du dictionnaire de fautes nécessaire à notre méthode est due au format de stockage choisi (format texte)
- Le temps d'exécution (CPU) est plus élevé que celui obtenu avec l'outil de référence (max = x 11, min = x 2.5).
- L'absence de traitement des fautes multiples et des fautes affectant les bascules des chaines de scan.
- L'affectation des modèles de fautes n'a pas été possible pour les circuits séquentiels.

Conclusion générale

Aujourd'hui les technologies 65nm et 45nm sont disponibles et d'ici 2012 la technologie 32nm sera développée pour permettre la mise sur le marché de circuits intégrés encore plus performants. Ces nouvelles technologies imposent donc de nouveaux défis, notamment via l'apparition de nouveaux mécanismes de défaillances difficilement caractérisables ou assimilables à un modèle simple de collage, principalement en raison de l'augmentation du nombre de portes par puce, la diminution de la taille des transistors, et la complexité des circuits actuels tels que les systèmes sur puce.

Parallèlement au test de fabrication, le diagnostic de pannes est un processus important qui vise à localiser les défaillances mises en évidence lors du test. Dans ce contexte, l'objectif de ma thèse a été de développer de nouvelles approches de diagnostic permettant de réduire l'espace de recherche tout en garantissant une localisation fiable et précise permettant de faciliter et accélérer le processus d'analyse de défaillances.

Le premier chapitre s'est focalisé sur l'ensemble des modèles de fautes considérés qui ont été pris en compte et manipulés par les techniques de diagnostic proposées tout au long de cette thèse. Nous avons fait une analyse complète de ceux-ci afin de déterminer les conditions de sensibilisation de chacun d'entre eux. Nous avons vu que certains modèles de fautes ne provoquent qu'une seule erreur dans le circuit, et n'affectent qu'une seule ligne (cas d'une panne simple), alors que d'autres peuvent soit provoquer sur une même ligne une erreur pouvant avoir plusieurs polarités en fonction des conditions de sensibilisation, soit affecter plusieurs lignes du circuit (cas des erreurs multiples). Les approches de diagnostic que nous avons développées dans ce manuscrit tiennent compte de toutes ces conditions de sensibilisation.

Le deuxième chapitre est consacré à la description d'une méthode de diagnostic de pannes dédiée au traitement des systèmes sur puce pouvant contenir des IPs diverses. Dans la première partie, nous avons présenté une synthèse de la méthode de diagnostic développée durant la Thèse d'A. Rousset intitulée « Diagnostic de pannes dans les circuits logiques » et qui a servie de point de départ à notre étude. Cette méthode est scindée en deux étapes. La première étape nommée *localisation de fautes* permet de déterminer un ensemble de lignes suspectes qui peuvent être à l'origine des erreurs observées lors de la phase du test. Chaque ligne obtenue possède deux paramètres associés, un symbole porté ainsi que le nombre de fois où chaque ligne a été tracée. Cette étape est basée sur une approche « Effet-à-Cause », et elle

comprend trois phases : la *simulation multi-valuée* , le *traçage de chemins critiques* et à partir de là, l'*intersection des listes de suspects*. La deuxième étape nommée *affectation des modèles de fautes* affecte un ensemble des modèles de fautes en fonction du symbole porté pour chaque ligne suspecte déterminée lors de la première étape.

La méthode de diagnostic développée sous l'acronyme eDERRIC traite uniquement les circuits combinatoires. Par ailleurs, l'outil eDERRIC a été développé pour traiter une IP logique à la fois, alors qu'un système sur puce possède à priori plusieurs IPs de natures diverses. La deuxième partie est consacrée aux modifications apportées à l'outil eDERRIC pour permettre de prendre en compte les systèmes sur puce.

Les modifications ont été apportées à deux niveaux. D'abord, l'adaptation du format des fichiers d'entrée manipulés par l'outil eDERRIC. Elle consiste à modifier la description du système sur puce, les vecteurs de test, et les réponses du testeur. Ensuite, l'adaptation au niveau du processus de diagnostic. Il s'agit ici de modifier *le codage des signaux*, ainsi que *le processus de traçage de chemins critiques*. Enfin, nous avons étendu les types de symboles dans la phase d'affectation des modèles de fautes afin de tenir compte de toutes ces modifications.

Les résultats expérimentaux présentés dans la troisième partie ont permis la validation de notre méthode de diagnostic sur des systèmes sur puce de taille et d'architecture différentes. La comparaison avec un outil de référence a montré l'efficacité et la fiabilité de notre méthode de diagnostic en termes de résolution et de précision.

L'inconvénient majeur de cette approche est qu'elle ne traite que les circuits combinatoires. Afin d'y remédier, nous avons développé dans le troisième chapitre une autre méthode de diagnostic basée sur une approche « *Cause-à-Effet* » et qui comprend deux phases : une phase de *localisation des sites de pannes potentiels* qui s'appuie sur le paradigme SLAT (Single Location At Time), et une phase *d'affectation des modèles de fautes* aux sites de pannes potentiels qui fait appel aux conditions de sensibilisation de pannes détaillées dans le premier chapitre. L'avantage principal de cette méthode est l'utilisation d'un seul dictionnaire de fautes de collage capable de traiter différents modèles de fautes.

La première partie de ce troisième chapitre, nous a permis de présenter les deux phases de diagnostic. Tout d'abord, nous avons décrit une étape préliminaire nommée *pré-traitement*,

qui fournit les informations (dictionnaire de fautes et 'CUT Dump') requises par le processus de diagnostic. La première phase réalise ensuite l'analyse des données stockées dans le dictionnaire de fautes par rapport aux réponses du testeur. Elle extrait une ou plusieurs listes des lignes suspectes qui expliquent complètement ou partiellement les erreurs observées lors du test. La deuxième phase permet d'affecter un symbole pour chaque ligne suspecte obtenue auparavant. Ce dernier permet de remonter à l'ensemble de défauts susceptibles d'être à l'origine de défaillances.

Dans la deuxième partie, nous avons développé un flot permettant l'adaptation de la méthode sur des circuits séquentiels. Ce flot consiste à prendre en compte la complexité réelle de ces circuits. Il comporte des améliorations au niveau de la génération de dictionnaire de fautes et de la localisation de fautes.

La validation de la méthode de diagnostic est consolidée par des résultats expérimentaux. Elle a été menée en deux étapes. Dans la première étape, nous avons validé le processus de diagnostic sur les circuits benchmark de la famille ITC'99. Ensuite, nous l'avons appliqué sur les blocs logiques séquentiels d'un système sur puce de la société STMicroelectronics. Les résultats expérimentaux ont montré une bonne résolution de diagnostic en terme de nombre de suspects. De plus, la comparaison avec un outil de référence a démontré la fiabilité de notre technique de diagnostic en terme de localisation de fautes.

En résumé, la première méthode de diagnostic basée sur l'approche « Effet-à-Cause » détaillée au second chapitre traite les circuits combinatoires et les systèmes sur puce modifiés et assimilables à des circuits combinatoires. La deuxième méthode de diagnostic basée sur l'approche « Cause-à-Effet » permet de diagnostiquer les circuits combinatoires, séquentiels et les blocs logiques séquentiels des systèmes sur puce. Les deux méthodes développées durant ma thèse prennent en compte différents types de fautes (collage, délai, court-circuit), alors que l'outil de référence n'utilise que les fautes de collages dans son dictionnaire de fautes.

Les perspectives que nous pouvons envisager pour la suite de notre travail concerne la deuxième méthode de diagnostic. Nous avons vu que cette méthode est capable de traiter les circuits séquentiels, mais il reste à finaliser l'adaptation de la phase d'affectation des modèles de fautes dans le cas du diagnostic de circuits séquentiels.

Par la suite, il faudra redéfinir un format de stockage de données permettant de réduire l'espace mémoire occupé par le dictionnaire de fautes et ainsi d'accélérer l'analyse d'information et donc diminuer le temps d'exécution du diagnostic.

Nous pourrions réévaluer notre technique de diagnostic en la comparant avec une version récente de l'outil de référence (2009.06). De même, analyser les fautes qui n'ont pas été injectées au sein du circuit (par exemple : faute de circuit ouvert) pourrait être intéressant.

Aujourd'hui, notre technique de diagnostic utilise un fournisseur d'outils CAO précis pour la génération d'un dictionnaire de fautes, or elle ne sera apte à traiter que les circuits contenant des modules de compression/décompression de vecteurs de test générés par ce même fournisseur. Afin de pouvoir fonctionner avec tous les fournisseurs du marché, il faudrait compléter notre technique de diagnostic pour générer le dictionnaire de fautes par l'insertion de ces modules en fonction du fournisseur.

Enfin, une autre évolution à envisager concerne la prise en compte des pannes de court-circuit. Pour ces pannes, notre méthode de diagnostic fournit une liste de couples composés d'une ligne agresseur et d'une ligne victime pouvant affecter le circuit. Cependant, si nous faisons une corrélation entre les résultats obtenus et les informations venant du layout, nous pourrions affiner et réduire la liste de couples en se basant sur la position et la distance entre les lignes au sein du circuit.

Références personnelles

Publications dans les Conférences Internationales

[ATS09] **Y. BENABBOUD, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, O. Riewer,**

"Delay Fault Diagnosis in Sequential Circuits"

18th IEEE Asian Test Symposium, Taichung, Taiwan, Novembre 2009, pp 355-360.

[DDECS09] **Y. BENABBOUD, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**

"Comprehensive Bridging Fault Diagnosis based on the SLAT Paradigm"

12th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, Liberec, République Tchèque, Avril 2009, pp 264-269.

[DTIS09] **Y. BENABBOUD, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**

"A Fault-Simulation Based Approach for Logic Diagnosis"

4rd IEEE International Conference on Design & Test of Integrated Systems in Nanoscale Technology, Caire, Avril 2009, pp 216-221.

[ETS09] **Y. BENABBOUD, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**

"A Logic Diagnosis Approach for Sequential Circuits"

14th IEEE European Test Symposium, Séville, Espagne, Mai 2009.

[ISQED09] **Y. BENABBOUD, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**

"Case Study on Logic Diagnosis"

10th IEEE International Symposium on Quality Electronic Design, San Jose, CA, Mars 2009, pp 253-259.

[PRIME09] **Y. BENABBOUD, A. Bosio, O. Riewer,**

"A Diagnosis Method For System-On-Chip"

5th Conference on Ph. D Research in Microelectronics and Electronics, Cork, Irlande, Juillet 2009, pp 276-279.

Publications dans les Conférences Nationales

[GDR08] **Y. BENABBOUD, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**
"Logic Diagnosis of System-On-Chip"
Proc. GDR SOC-SIP : Groupement De Recherche SOC-SIP, Paris, France, Juin 2008.

Publications dans les Séminaires Internationaux

[SETS08] **Y. BENABBOUD, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**
"Case Study on Logic Diagnosis"
South European Test Symposium (SETS), Obergurgl, Autriche, Mars 2008.

[SETS09] **Y. BENABBOUD, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute,**
"A fault-Simulation-Based Approach for Logic Diagnosis"
South European Test Symposium (SETS), Valmorel, France, Mars 2009.

Références bibliographiques

[Abra80] M. Abramovici et al, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis", IEEE Transactions on Computer, vol. c-29, n°6, p. 451-460, 1980.

[Abr83] M. Abramovici, P.R. Menon, D.T. Miller, "Critical Path Tracing – an Alternative to Fault Simulation", *Proceedings of the 20th conference on Design automation*, p. 214-220, 1983.

[Abra90] M. Abramovici et al., "Digital System Testing and Testable Design", IEEE *Press*, 1990.

[Amy06] M.E. Amyeen et al., "Improving Precision Using Mixed-level Fault Diagnosis", *IEEE Int.Test Conf.*, p. 1-10, 2006.

[App06] D. Appello et al., "On the Automation of the Test Flow of Complex SoCs", IEEE VLSI Test Symposium, p. 166-171, 2006.

[App08] D. Appello[1], M. Rotigni[1], V. Tancorre[1], P. Bernardi[2], M. Grosso[2], M. Sonza Reorda[2], " A case study on SoC low-cost Silicon Debug and Diagnosis ", A case study on SoC low-cost Silicon Debug and Diagnosis" ([1]STMicroelectronics, [2]Politecnico di Torino), IEEE International Workshop on Silicon debug and Diagnosis, 2008.

[Aza04] F. Azais, S. Bernard, Y. Bertrand, M.L. Flottes, P. Girard, C. Landrault, L. Latorre, S. Pravossoudovitch, M. Renovell, et B. Rouzeyre, "Test de Circuits et de Systèmes Intégrés", ISBN 2-7462-0864-4, Editions Hermes Science, 2004.

[Bar83] Z. Barzilai, B.K. Rosen, "Comparison of AC Self-Testing Procedures", *Proceedings of the International Test Conference*, p. 89-94, 1983.

[Ber85] L. Berghier, "Analyse de défaillances de circuits VLSI par microscope à balayage", *Thèse de Doctorat, Grenoble, INP*, 1985.

[Bop96] V. Boppana, I. Hartanto, W.K. Fuchs, "Full fault dictionary storage based on labeled tree encoding", IEEE VLSI Test Symposium, p 174-179, 1996.

[Bos08] A. Bosio, et al, "SoC Symbolic Simulation: a case study on delay fault testing", IEEE DDECS, p. 320-325, 2008.

[Bos09] A. Bosio, P. Girard, S. Pravossoudovitch et A. Virazel, “A Comprehensive Framework for Logic Diagnosis of Arbitrary Defects”, IEEE Transactions on Computers, Vol. PP, issue 99, p. 1-13, 2009.

[Car87] J.L. Carter, V.S. Iyengar, B.K. Rosen, "Efficient Test Coverage Determination for Delay Faults", *Proceedings of the International Test Conference*, p. 418-427, 1987.

[Che06] Y.Y. Chen, J.J. Liou, "Enhancing Diagnosis Resolution For Delay Faults By Path Extension Method", *Proceedings of the 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, p. 428-438, 2006.

[DC07] www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/DesignCompilerGraphical.aspx

[Des06] R. Desineni et al, "A Logic Diagnosis Methodology for Improved Localization and Extraction of Accurate Defect Behavior" Proc. IEEE Int. Test Conf., p. 1-10, 2006.

[Eng03] P. Engelke, I. Polian, M Renovell, B Becker, "Simulating Resistive Bridging and Stuck-At Faults", *Proceedings of the International Test Conference*, p. 1051-1059, 2003.

[Gho00] J. Ghosh-Dastidar, N. A. Touba, "Diagnosing Resistive Bridges Using Adaptive Techniques", *IEEE Custom Integrated Circuits Conference*, p. 79-82, 2000.

[Gir92a] P. Girard, "Diagnostic de Pannes Temporelles dans les Circuits Digitaux" *Thèse de Doctorat*, LIRMM / Université Montpellier II, 8 Avril 1992.

[Gir92b] P. Girard, C. Landrault, S. Pravossoudovitch, "Delay-Fault Diagnosis by Critical-Path Tracing", *IEEE Design & Test of Computers*, vol. 9, issue 4, p. 27-32, 1992.

[Gir96] P. Girard, C. Landrault, S. Pravossoudovitch et B. Rodriguez, “A Diagnostic ATPG for Delay Faults Based on Genetic Algorithms”, IEEE International Test Conference, pp. 286-293, Washington, USA, 22-24 Octobre 1996.

[Han03] Shi Yu Hang, "A Symbolic Inject-and-Evaluate Paradigm for Byzantine Fault Diagnosis", *Journal of Electronic Testing: Theory and Applications*, vol. 19, issue 2, p. 161-172, 2003.

[Hol07] S. Holst, H.-J. Wunderlich, "Adaptive Debug and Diagnosis without Fault Dictionaries", *IEEE European Test Symposium*, p. 7-12, 2007.

[Hsi77] E.P. Hsieh, R. Rasmussen, L. Vidunas, W. Davis, "Delay Test Generation", *Proceedings of the Design Automation Conference*, p. 486-492, 1977.

[Hui04] L. M. Huisman, “Diagnosing Arbitrary Defects in Logic Designs Using the Single Location At a Time (SLAT)”, *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 23; No 1, 2004, pp.91-101.

[Kis86] K. Kishida, F. Shirotori, Y. Ikemoto, S. Ishiyama, T. Hayashi, "A Delay Test System for High Speed Logic LSI’s", *Proceedings of the Design Automation Conference*, p. 786-790, 1986.

[Kit91] Y. kitamura, "Sequential Circuit fault simulation by fault information tracing algorithm:FIT", *proceedings of the 28th ACM/IEEE Design Automation Conference*, p. 151-154, 1991.[Lav98] D.B. Lavo et al. , “Diagnosing Realistic Bridging Faults with Single Stuck-At Information”, *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol.17, No 3, p. 255-2671998.

[Lev86] Y. Levendel, P.R. Menon, "Transition Faults in Combinational Circuit: Input Transition Test Generation and Fault Simulation", *Proceedings of the Fault-Tolerant Computing Symposium*, p. 278-283, 1986.

[Lin07] Yung-Chieh Lin, Feng Lu, Kwang-Ting Cheng, “Multiple-Fault Diagnosis Based On Adaptive Diagnostic Test Pattern Generation”, *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 26; No 5, p. 932-942 2007.

[Liu05] J.B. Liu, A. Veneris, "Incremental Fault Diagnosis", IEEE *Transaction on Computer Aided Design of Integrated Circuits and Systems*, vol. 24, n°2, p. 240-251, 2005.

[Lo00] C.-K. Lo, P. C. H. Chan., "An Efficient Structural Approach to Board Interconnect Diagnosis", *International journal of electronics*, vol. 87, no10, pp. 1239-1255, 2000.

[Meno93] S.M. Menon, et al., "Testable Design of BiCMOS Circuits for Stuck-Open Fault Detection using Single Patterns", *Proceedings of the VLSI Test Symposium*, p. 296-302, 1993.

[Pom97] I. Pomeraz et al., "On Dictionnary-Based Fault Location in Digital Logic Circuits", IEEE *Transaction on Computers*, vol. 46, n° 1, p. 48-59, 1997.

[Pra88] A.K. Pramanick, S.M. Reddy, "On the Detection of Delay Faults", *Proceedings of the International Test Conference*, p. 845-856, 1988.

[Rou07$_1$] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, "DERRIC a Tool for Unified Logic Diagnosis", *IEEE European Test Symposium*, p. 13-20, 2007.

[Rou07$_2$] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, "Fast Bridging Fault Diagnosis using Logic Information", *16th IEEE Asian Test Symposium* ", p. 33-38, 2007.

[Rou08] A. Rousset, "Diagnostic de pannes dans les circuits logiques : Développement d'une méthode ciblant un ensemble élargi de modèles de fautes", *Thèse de Doctorat*, LIRMM / Université Montpellier II, 01 Avril 2008.

[Sch87] M.H. Schulz, F. Brglez, "Accelerated Transition Fault Simulation", *Proceedings of the Design Automation Conference*, p. 237-243, 1987.

[Shi00] K. Shigeta, T. Ishiyama, "An improved fault diagnosis algorithm based on path tracing with dynamic circuit extraction", Proc. IEEE Int. Test Conf., p. 235-244, 2000.[Smi85] G.L. Smith, "Model for Delay Faults Based upon Paths", *Proceedings of the International test Conference*, p. 342-349, 1985.

[STIL07] http://grouper.ieee.org/groups/1450/index.html

[Tcl07] www.tcl.tk; http//fr.wikipedia.org/wiki/Tool_Command_Language

[Tmax07] www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx

[Ven97] S. Venkataraman, W. K. Fuchs, "A Deductive Technique for Diagnosis of Bridging Faults", *Proceedings of the IEEE/ACM international conference on Computer-aided design*, p. 562-567, 1997.

[Ven01] S. Venkataraman, S.B. Drummonds, "Poirot: applications of a logic fault diagnosis tool", *IEEE Design & Test of Computer*, vol. 18, No 1, pp. 19-30, 2001.

[VSIA] VSIA. VSI Alliance Web Site. http://www.vsi.org.

[Wai86] J.A. Waicukausky, E. Lindbloom, B. Rosen, V. Iyendar, "Transition Fault Simulation by Parallel Pattern Single Fault Propagation", *Proceedings of the International Test Conference*, p. 542-549, 1986.

[Wai89] J.A. Waicukausky et E. Lindbloom, "Failure Diagnosis of Structured VLSI", *IEEE Design & Test of Computer*, vol. 6, No 4, pp. 49-60, 1989.

[Wan06] L.T. Wang, C.W. Wu et X. Wen, "VLSI Test Principles and Architectures", *Elsevier - Morgan Kaufmann Publishers*, 2006.

[Wood87] B. W. Woodhall, et al, "Empirical Results on Undetected CMOS Stuck-Open Failures", *Proceedings of the International Test Conference*, p. 166-170, 1987.

Liste des figures

Figure 1 : Rendement vs Technologie 10
Figure 1.1 : Exemple de collage à 1 (Ca1) d'une sortie de porte ET 17
Figure 1.2 : Modèle générale du court-circuit 19
Figure 1.3 : Court-circuit de type Dominant Or 19
Figure 1.4 : Court-circuit de type Dominant And 20
Figure 1.5 : Court-circuit de type Strong Driver Wired 21
Figure 1.6 : Court-circuit de type Wired Or 21
Figure 1.7 : Court-circuit de type Wired And 22
Figure 1.8 : Court-circuit de type Byzantine 23
Figure 1.9: Cas des circuits résistifs 24
Figure 1.10 : Exemple d'un circuit ouvert à 1 25
Figure 1.11 : Exemple d'un circuit ouvert résistif entraînant une transition « lent-à-monter » 25
Figure 1.12 : Exemple d'un délai « lent-à-monter » (Slow-to-Rise) sur une porte ET 26
Figure 1.13 : Exemple d'un « délai lent-à-descendre » sur une porte ET 26
Figure 1.14 : Transistor T1 stuck-open 27
Figure 2.1 : Synoptique du processus de diagnostic développé 34
Figure 2.3 : Défauts susceptibles d être à l'origine d'une erreur 35
Figure 2.4 : Simulation du vecteur (E1, E2, E3, E4, E5) = (C0, C1, F0, C0, R1) 38
Figure 2.6 : Traçage de Chemins Critiques pour le vecteur V21 42
Figure 2.7 : Traçage de Chemins Critiques pour le vecteur V65 43
Figure 2.11 : Exemple de propagation de la valeur 'X' 53
Figure 2.12 : Remplacement de flip flop par des buffers 54
Figure 2.13 : IP contenant des éléments séquentiels (bascules) 55
Figure 2.14 : IP combinatoire (suppression des bascules) 55
Figure 2.15 : Exemple de vecteurs de test de type 'Stil' 56
Figure 2.16 : Les 12 nouveaux symboles de l'algèbre à 18 valeurs 60
Figure 2.17 : Tables de transfert des portes élémentaires avec l'algèbre à 18 valeurs 61
Figure 2.18 : Algorithme de CPT en tenant compte des valeurs X et Z 62
Figure 2.19 : Exemple de l'application de CPT 63
Figure 2.20 : Architecture de système sur puce 69

Figure 2.21 : Comparaison entre l'outil de référence et notre approche de diagnostic 71
Figure 3.1 : Flot d'adaptation de la méthode de diagnostic 75
Figure 3.2 : Synoptique du processus de diagnostic développé 76
Figure 3.3 : les différents types de défauts (statiques et dynamiques) 77
Figure 3.4 : Un exemple de circuit 77
Figure 3.5 : Un exemple de séquence de test en format STIL 78
Figure 3.6 : Algorithme de localisation de fautes 80
Figure 3.7 : l'algorithme d'affectation des modèles de fautes 85
Figure 3.8 : Représentation d'un circuit séquentiel dans les intervalles de temps 89
Figure 3.9 : Un exmple de propagation de la faute F sur le nœud A vers une sortie primaire 90
Figure 3.10 : Architecture de système sur puce 98
Figure 3.11 : L'image de layout de système sur puce 98
Figure 3.12 : Comparaison entre l'outil de référence et notre approche de diagnostic 103

Liste des tableaux

Tableau 1.1 : Différents modèles de courts-circuits 18
Tableau 1.2 : Comportement logique en présence d'un Stuck-open 27
Tableau 2.1 : Tables de transfert des portes élémentaires 38
Tableau 2.2 : Liste des suspects pour le vecteur V21 42
Tableau 2.3 : Vecteurs et réponses associés 43
Tableau 2.4 : Liste des suspects pour le vecteur V65 et actualisation du compteur CNT .. 43
Tableau 2.5 : Opération d'intersection entre signaux 45
Tableau 2.6 : Classement et résultat d'intersection entre les deux listes des suspects 46
Tableau 2.7 : Modèles de fautes associés aux lignes critiques en fonction du symbole 47
Tableau 2.8 : Réponses du testeur de l'exemple (Figure 2.15) 58
Tableau 2.9 : La liste des suspects 63
Tableau 2.10 : Opération d'intersection entre signaux avec l'algèbre à 18 valeurs 64
Tableau 2.11 : Modèles de fautes associés en tenant compte des symboles 'X' et 'Z' 65
Tableau 2.12 : Rapport de diagnostic 65
Tableau 2.14 : Résultats expérimentaux en utilisant des blocs mémoires fonctionnels 67
Tableau 2.15 : Résultats expérimentaux en utilisant une boîte noire pour la mémoire 69
Tableau 2.16 : Caractéristiques du système sur puce 70
Tableau 2.17 : Résultats expérimentaux 70
Tableau 3.1 : Le dictionnaire de fautes 79
Tableau 3.2: Le fichier 'CUT Dump' 79
Tableau 3.3 : Datalog de la faute de transition « StF » sur la ligne « c » 82
Tableau 3.4 : Liste des suspects du vecteur de test TV3 82
Tableau 3.5 : Liste des suspects des vecteurs de test TV3 et TV8 83
Tableau 3.6 : Datalog de l'exemple Wired Or 84
Tableau 3.7 : Première liste de suspects 85
Tableau 3.8 : Deuxième liste de suspects 85
Tableau 3.9 : Opération d'intersection entre signaux avec l'algèbre à 18 valeurs 86
Tableau 3.10 : Intersection de symboles entre les vecteurs de test fautifs (exemple 1) 86
Tableau 3.11 : Intersection de symboles entre les vecteurs de test fautifs (exemple 2) 87
Tableau 3.12 : Modèles de fautes associés en tenant compte des symboles 'X' et 'Z' 88

Tableau 3.13 : Datalog du circuit b02 92
Tableau 3.14 : ‘CUT Dump’ du circuit b02 93
Tableau 3.15 : Résultats expérimentaux sur des circuits benchmark de la famille ITC’99 96
Tableau 3.16 : Caractéristiques du SoC..... 99
Tableau 3.17 : Résultats de diagnostic des premières expériences (fautes injectées) 99
Tableau 3.18 : Résultats de diagnostic dans un environnement industriel..... 101

Annexe

A- Phase de localisation de fautes

La phase de localisation de fautes est basée sur le paradigme SLAT, elle consiste à identifier les lignes suspectes qui ont causé la défaillance du circuit. Nous tenons en compte des informations provenant de la phase de test (stockées dans le fichier Datalog), des vecteurs de test fautifs FTVi (vecteurs de test fautifs) et des sorties primaires fautives (FPOs).

1: for each failing pattern **FTVi** do
2: for each failing primary output **FPOi** do
3: open fault dictionary **FD_TVi**
4: If (#Pi_{FD} == #$FTVi_{D}$**)** then
5: If ((#POi_{FD} == #$FPOi_{D}$ **)||(**#$DFFi_{FD}$ == #$DFFi_{D}$ **))**then
6: **$Susps_{FTVi}$ << Fi**
7: Endif
8: Endif
9: close fault dictionary **FD_TVi**
10: Done
11: Done
12: Curr_susps = ∅
13:for each set of suspects **$Susps_{FTVi}$** do
14: for each set of suspects **$Susps_{FTVi+1}$** do
15: If (#**$Susps_{FTVi}$** ∩ **$Susps_{FTVi+1}$= Fi)** then
16: **Curr_susps** << **Fi**
17 else (#**$Susps_{FTVi}$** ∩ **$Susps_{FTVi+1}$**== ∅**)** then
18 **$New_Curr_suspens_{i+1}$** << **Fi**
19: Endif
20: Done
21:Done
22:Print **Curr_susps, $New_Curr_suspens_{i+1}$**

Figure I : L'algorithme de diagnostic

La Figure I représente le pseudo-code de l'algorithme adopté pour la localisation de fautes en utilisant un dictionnaire de fautes généré par l'outil TetraMax [Tmax09]. Pour chaque vecteur fautif 'FTVi' (ligne 1 de la Figure i) et chaque sortie primaire fautive 'FPOi' (ligne 2 de la Figure I), nous ouvrirons le fichier de dictionnaire de fautes correspond au vecteur TVi 'FD_TVi' (ligne 3 de la Figure I). Puis, nous recherchons dans ce fichier d'abord toutes les fautes ayant le vecteur TVi (#TVi_{FD}) égale au vecteur fautif FTVi de datalog (#$FTVi_{D}$), ensuite, parmi ces fautes obtenues, nous récupérons seulement celles ayant la sortie primaire POi (#POi_{FD}) égale au sortie primaire fautive de datalog FPOi (#$FPOi_{D}$) dans le cas ou la faute a été propagée directement en sorties primaires. Par contre, si la faute a été propagée en

sorties primaires via une bascule dans la chaine de scan, nous récupérons seulement les fautes ayant le même indicateur de la bascule scan (#DFFi$_{FD}$ de dictionnaire de fautes égale à celui de datalog #DFFi$_{D}$). Nous écrivons ensuite pour chaque vecteur fautif FTVi, les fautes trouvées dans la liste des suspects FTVi Susps$_{FTVi}$ (ligne 4, 5 et 6 de la Figure I).

Pour avoir la liste de suspects de chaque pattern fautif la plus courte possible, nous ne rapportons pas deux fois la même faute. La liste initiale de lignes suspectes est vide (ligne 12 de la Figure I). Nous faisons chaque fois l'intersection entre deux listes de lignes suspectes de deux vecteurs fautifs successifs (ligne 13 et 14 de la Figure I), afin de trouver les suspects communs entre les deux listes (ligne 15 de la Figure I). Cependant, si l'intersection entre deux listes de lignes correspond à un néant, cela signifie que nous aurons deux voir plus de listes à définir (ligne 17 de la Figure I). Nous déterminons ainsi de suite la ou les listes finales de suspects (lignes 16 et 18 de la Figure I).

Le résultat final de la phase de localisation de faute est une ou plusieurs listes de suspects (ligne 22 de la Figure I) ayant expliqués complètement ou partiellement les réponses du testeur observées durant la phase de test.

En analysant la ou les listes de suspects obtenues, nous pouvons trouver avec certitude la faute affectant le circuit.

Delay Fault Diagnosis in Sequential Circuits

Y. Benabboud[12], A. Bosio[1], L. Dilillo[1], P. Girard[1], S. Pravossoudovitch[1], A. Virazel[1]
[1]LIRMM– Université Montpellier II / CNRS
161 rue Ada, 34392, Montpellier Cedex 05, France
<lastname>@lirmm.fr

O. Riewer[2]
[2]STMICROELECTRONICS
850 rue Jean Monnet 38920 Crolles, France
{olivia.riewer, youssef.benabboud}@st.com

***Abstract*—The importance of delay faults proportionally increases when entering in the nano-technology era, and logic diagnosis must localize delay faults as precisely as possible to speed-up yield ramp-up. This paper presents a logic diagnosis approach targeting delay faults. The proposed approach is based on the Single-Location-at-A-Time (SLAT) paradigm used to determine a set of suspects. It addresses the case of sequential circuits tested at-speed. The main advantages of this approach are that it can manage a comprehensive set of delay faults, and that it is independent on the size of the delay (induced by the fault). Experimental results show the effectiveness of the proposed approach in terms of absolute number of suspects.**

Keywords- Diagnosis, fault models, delay faults

I. INTRODUCTION

With the technology entering the nano-dimension era, manufacturing processes are less and less reliable leading to failures that can no longer be modeled by the classical stuck-at fault model. Numerous actual failures exhibit timing or parametric behaviors that are not represented by stuck-at faults. Such failures have to be taken into account during the test process in order to reach acceptable quality figures [1]. Defects leading to timing failures are usually modeled by means of delay fault models. The authors of [2] show that a certain number of delay faults can only be detected if the test vectors are applied at-speed.

To speed-up yield ramp-up, logic diagnosis must be able to correctly locate the root cause of observed failures as precisely as possible. Information provided by the diagnosis process is then used to guide the circuit physical observation during the next step of the failure analysis. The classical types of logic diagnosis approaches are namely *Cause-Effect* and *Effect-Cause[3]*. The Cause-Effect approach is based on fault simulation. Using fault simulation, a fault dictionary is created, on which all the succeeding steps of the logic diagnosis process will depend. The main advantage of such approach is to be able to handle both combinational and sequential circuits in the same way. The main drawback of the Cause-Effect approach is the need to have an accurate description of the fault models used during fault simulation.

The Effect-Cause approach starts from the set of faulty responses. It uses an error backtracking process originating from the Circuit Under Test (CUT) outputs, such as the Critical Path Tracing (CPT) process [4]. It does not need to explicitly consider each fault model and the required amount of data is negligible compared to the one generated by a Cause-Effect approach. The main drawback of the Effect-Cause approach is that it difficultly copes with sequential circuits.

A more recent approach works under the Single-Location-at-A-Time (SLAT) paradigm [8] that compares the failing test vectors, coming from the test phase with those stored in a unique pre-computed stuck-at fault dictionary or obtained by performing a stuck-at fault simulation. The diagnosis process considers the stuck-at faults that fully explain the failing test vectors. In this case, the diagnosis report consists of a set of single faults (suspects) that behave like a stuck-at fault model.

Several techniques have been developed to target the problem of delay fault logic diagnosis. Techniques based on the Cause-Effect approach can be further classified depending on the accuracy of the adopted fault models. In [15], authors resort to static timing analysis [16] to extract information on the most likely delay size. Then, they check the extracted delay size by performing fault simulation in order to minimize the error between simulation data and observed failures.

Delay fault diagnosis techniques based on the Effect-Cause approach use a deductive strategy based on the CPT process. In [5] [9] [10] and [11], authors address the problem of delay fault diagnosis only for combinational (full-scan) circuits. An extension of this principle to the case of sequential circuits is presented in [6]. However, it is limited to the use of variable clock testing schemes, also referred to as slow-fast-slow testing, in which a test sequence is not fully applied at-speed. In [13], the CPT is adapted to the case of sequential circuits where the test

sequence is applied at-speed. This approach has a high computational time cost and it is applied only on a small set of ISCAS'89 benchmarks.

A modified version of the SLAT approach is proposed in [14] where instead of using a stuck-at fault dictionary, authors resort to transition fault dictionary to achieve delay fault diagnosis. Only combinational (full-scan) circuits are considered.

In this paper, we present a diagnosis approach targeting delay fault in sequential circuits tested at-speed. The proposed approach is performed in two phases, (i) an off-line preprocessing phase provides the stuck-at fault dictionary and a set of critical paths by means of fault simulation and static timing analysis respectively. Then, the (ii) diagnosis phase, based on the comparison between the fault dictionary and observed failures, determines a set of suspects that is finally matched with the set of critical paths.

The main advantages of the proposed approach are that it can manage all types of delay faults, and that it is independent of the size of the delay (induced by the fault). Moreover, since the computational cost is limited to the comparison performed during the second phase, the proposed approach suits well the constraints of high volume diagnosis. Experiments on ITC'99 benchmark circuits show the efficiency of the proposed solution in terms of diagnosis resolution (i.e., absolute number of suspects).

This paper is organized as follows. Section II presents the taxonomy of delay fault models. Section III introduces the proposed approach. Section IV details the step performed during the diagnosis. Section V discusses the experimental results, while Section VI concludes the paper.

II. DELAY FAULT MODEL TAXONOMY

Before introducing the diagnosis methodology, we first summarize the taxonomy of delay fault models. This taxonomy is deeply analyzed in [12] and here reported for the sake of readability.

The **Transition Fault Model** (TFM) assumes that a gate input or output has a *slow-to-rise* transition fault or a *slow-to-fall* transition fault. The size of the delay caused by TFM is considered large enough to be propagated in any path through the circuit. An example is given in Fig. 1.a where the TFM sensitized at the output of $gate_i$ is propagated through any path, from p_1 to p_4.

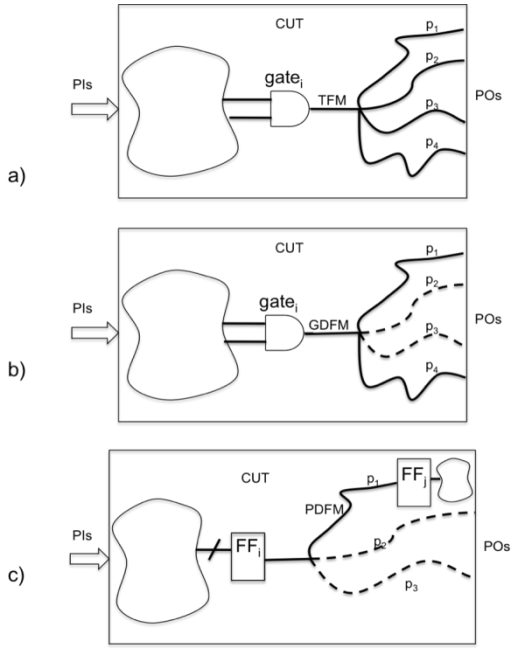


Figure 1. Examples of delay faults: a) TFM, b) GDFM, c) PDFM

The **Gate-Delay Fault Model** (GDFM) assumes that a small delay can affect a gate input or output. Thus, the fault will be propagated only through long paths (i.e., subset of all the possible paths). An example is given in Fig. 1.b where the GDFM sensitized at the output of $gate_i$ is propagated only through path p_1 and p_4. Path p_2 and p_3 are too short to correctly propagate the GDFM.

The **Path-Delay Fault Model** (PDFM) assumes that the delay is distributed on a combinational path. A combinational path is composed of a sequence of gates between primary inputs or clocked flip-flops (startpoints) and primary outputs or clocked flip-flops (endpoints) of the circuit. An example is given in Fig. 1.c where the PDFM is distributed through the path p_1 between FF_i and FF_j.

The PDFM is more general than GDFM and TFM since a test targeting PFDM also detect the GDFM and TFM.

III. PROPOSED APPROACH

In this section we describe the proposed diagnosis approach. Fig. 2 sketches the implemented diagnosis flow divided in two phases.

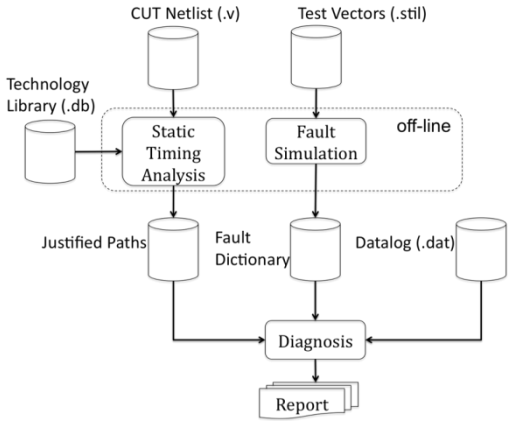


Figure 2. Diagnosis Flow

The first phase is applied off-line, in the sense that it is performed only one time for each couple of circuit and test vectors. By using a static timing analysis tool a set of justified paths is extracted from the CUT. The set of justified paths contains the complete set of functionally testable path-delay faults [20]. This set of justified paths will be further use during the diagnosis process. The required inputs are the CUT netlist described at gate-level (.v), and the target technology library (.db) used to synthesize the circuit. In parallel to the justified path extraction, a stuck-at fault simulation is performed in order to generate the full dictionary. The required inputs are the CUT netlist and the applied test vectors (.stil). The cost of this phase is the amount of memory required to store the stuck-at fault dictionary and the justified paths.

The second phase performs the diagnosis by considering failures observed during the test, stored in the datalog file (.dat). The diagnosis consists in the comparison between the pre-computed single stuck-at fault dictionary and the datalog. The result is a ranked set of suspected nets in the CUT. Suspects are the nets on which a given failure can explain (or partially explain) the observed errors. The set of suspects is finally cross-checked with the pre-computed set of justified paths in order to exclude those faults that do not belong to a justified path (i.e., they cannot be a realistic delay fault). It also determines the set of possible paths that can be the root cause of the observed errors.

Next sections will provide details about preprocessing and diagnosis phases.

IV. DIAGNOSIS PROCESS

The proposed diagnosis approach relies on the concepts defined in [7] and [8]. The basic idea of [8] is the independence of any fault model during the diagnosis since it resorts to the concept of conditional stuck-at fault. A conditional stuck-at fault can be defined as a particular stuck-at fault with a specific set of sensitization conditions. For instance, a delay fault can be viewed as a conditional stuck-at fault sensitized by a transition. Once the conditional stuck-at fault is sensitized, the propagation of the induced logic error is equivalent to the propagation of the classical stuck-at. Even if this assumption is true for the large part of fault models, as for the transition fault model, it is not correct when addressing gate and path-delay fault models. This problem has been pointed out in [14], where authors resort to a transition fault dictionary instead of the standard stuck-at fault dictionary to address delay fault diagnosis. However, the authors only consider information coming from failing vectors and outputs. Thus, the list of suspects is not reduced.

Here, we adapt the idea of conditional stuck-at fault model to perform gate, path-delay as well as transition fault diagnosis. Moreover, we take into account information coming from both failing and passing primary outputs during the application of the failing test vectors.

To describe our approach we consider the conditional stuck-at fault when targeting each type of delay fault model (i.e., TFM, GDFM and PDFM).

Fig. 3 shows the case of TFM. As already defined in section II, a TFM is sensitized by applying a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ (Fig. 3.a), but once the TFM is sensitized the fault effect is propagated exactly like a stuck-at fault under the application of the same test vector (Fig. 3.b).

We can formalize the above consideration as shown in the following. Given a set of test vectors V, we can define the sub-sets FV $\subseteq$ V and PV $\subseteq$ V where:

- $fv \in$ FV, is a test vector that detect a stuck-at fault;
- $pv \in$ PV, is a test vector that does not detect a stuck-at fault.

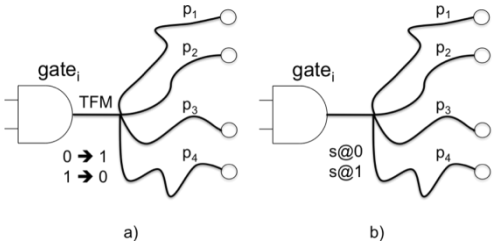


Figure 3. a) TFM, b) conditional stuck-at

We can also define the failing primary output set FPO(fv, s@), function of fv and s@, as the primary outputs on which the detected stuck-at fault s@ is observed. Note that FPO(pv, s@) = $\varnothing$.

- In the same way it is possible to define $FV_{TFM} \subseteq$ V and $PV_{TFM} \subseteq$ V where:
- $fv_{TFM} \in FV_{TFM}$, is a test vector that detect a transition fault;

$pv_{TFM} \in PV_{TFM}$, is a test vector that does not detect a transition fault.

Finally, the relation between TFM and stuck-at fault is formalized as follows:

$$\begin{cases} FV_{TFM} \subseteq FV \subseteq V \\ FPO\left(fv_{TFM}, s@\right) \geq FPO\left(fv_{TFM}, TFM\right) \end{cases} \quad (1)$$

Fig. 4 shows the case of GDFM. As already defined in section II, a GDFM is sensitized by applying a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ (Fig. 4.a), and once the GDFM is sensitized the fault effect is propagated only through long paths. In the example of Fig. 4.a, the GDFM is only propagated through long paths p_1 and p_4. On the other hand, the corresponding stuck-at fault is propagated through all paths from p_1 to p_4 under the application of the same test vector (Fig. 4.b).

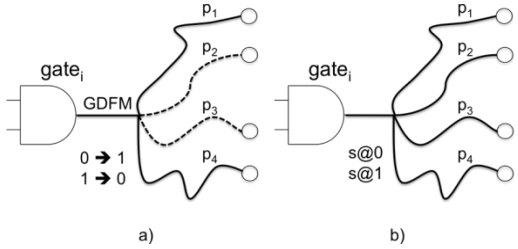


Figure 4. a) GDFM, b) conditional stuck-at

We can formalize the above consideration as shown in the following. We define $FV_{GDFM} \subseteq FV$ and $PV_{GDFM} \subseteq PV$ where:

- $fv_{GDFM} \in FV_{GDFM}$, is a test vector that detect a gate delay fault;
- $pv_{GDFM} \in PV_{GDFM}$, is a test vector that does not detect a gate delay fault.

Finally, the relation between GDFM and conditional stuck-at fault is formalized as follows:

$$\begin{cases} FV_{GDFM} \subseteq FV \subseteq V \\ FPO\left(fv_{GDFM}, s@\right) \geq FPO\left(fv_{GDFM}, GDFM\right) \end{cases} \quad (2)$$

Finally, the case of PDFM is shown in Fig. 5.

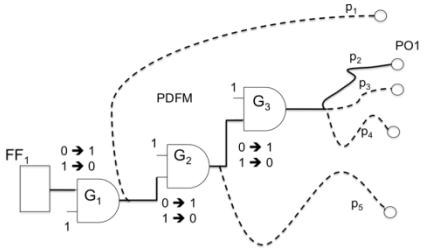


Figure 5. Case of PDFM

In this case, the path starts from FF_1 and is composed of 3 gates (G_1, G_2 and G_3). The ending point is the primary output PO1. The PDFM is sensitized by a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ applied on the starting point and then propagated through G_1, G_2 and G_3. PDFM can be viewed as a set of conditional stuck-at faults, one for each gate composing the path. The PDFM of Fig. 5 is therefore composed of 3 conditional stuck-at faults (S@-G_1, S@-G_2 and S@-G_3). As for the case of TFM and GDFM, we define $FV_{PDFM} \subseteq FV$, $PV_{PDFM} \subseteq PV$ and SG where:

- $fv_{PDFM} \in FV_{PDFM}$, is a test vector that detect a path delay fault;
- $pv_{PDFM} \in PV_{PDFM}$, is a test vector that does not detect a path delay fault;
- $SG = \{S@G_1, S@G_2, \ldots S@G_N\}$ is the set of stuck-at faults associated to each gate G_i belonging to the path. $1 \leq i \leq N$ and N is the number of combinational gate composing the path.

Finally, the relation between PDFM and conditional stuck-at fault is formalized as follows:

$$\begin{cases} FV_{PDFM} \subseteq FV \subseteq V \\ FPO\left(fv_{PDFM}, s@G_i\right) \geq FPO\left(fv_{PDFM}, PDFM\right) \\ 1 \leq i \leq n, s@G_i \in SG \end{cases} \quad (3)$$

Thanks to equations (1), (2) and (3) it is possible to define the likelihood between stuck-at faults and observed errors in order to find the best match. The Likelihood Function (*LF*) is define by considering the following sets:

- $FD = \{f_1, f_2, \ldots, f_M\}$ is the pre-computed stuck-at fault dictionary, each f_i is a stuck-at fault and M is the total number of stuck-at;
- $Pfpo = \{\#P_{f1}, \#P_{f2}, \ldots, \#P_{fm}\}$ where $\#P_{fi}$ is the number of primary outputs that are failing during both the stuck-at fault simulation and the test of the circuit. We denote it as the number of **predicted** failing primary outputs by stuck-at $f_i \in FD$, during the application of each failing test vector $fv \in V$ coming from the datalog;
- $Mfpo = \{\#MP_{f1}, \#MP_{f2}, \ldots, \#MP_{fm}\}$where $\#MP_{fi}$ is the number of primary outputs that are failing only during the stuck-at fault simulation (i.e., declared pass during the test of the circuit). We denote it as the number of **mispredicted** failing primary outputs by stuck-at $f_i \in FD$, during the application of each failing test vector $fv \in V$ coming from the datalog.

By considering the above sets, the likelihood function is defined as follows:

$$LF\left(Pfpo, Mfpo\right) = \begin{cases} \max(\#\ Pfpo_{fi}) \mid 1 \leq i \leq M,\ fi \in FD \\ \min(\#\ Mfpo_{fi}) \mid 1 \leq i \leq M, fi \in FD \end{cases} \quad (4)$$

The result of LF is therefore a set of faults that maximizes the number of predictions and minimizes the number of mispredictions. LF is used to rank faults in the

fault dictionary with respect to their capability to well fit with the datalog (i.e., the observed failures). The list of suspects corresponds to the results of the LF.

It is possible to refine the list of suspects by means of the sets Mfpo, Pfpo and equations (1), (2) and (3) in order to distinguish between TFM, GDFM and PDFM. This leads to the following cases:

1. $\#MP_{fi} > 0$ where f_i belongs to the list of suspects: in this case thanks to (1) we can exclude the transition fault model to be the root cause of suspected errors.
2. $\#P_{fi} = 0$ where f_i belongs to the list of suspects: in this case we exclude the fault f_i to be the root cause of the observed error since any failing primary output is correctly matched.

Finally, to provide meaningful diagnosis we check the obtained list of suspects with the pre-computed set of justified paths. This last step allows to remove those faults that do not belong to any critical path for the case 1) and to associate the justified path to the matching suspects. Then we provide the information on which paths can be the root cause of the observed errors due to both GDFM and PDFM for the case 1) and 2).

Let us consider an example to illustrate the proposed diagnosis approach. For the sake of simplicity we consider general gates (i.e., without specifying the gate function) and assume that the considered circuit is purely combinational, even if the proposed approach has been evaluated on sequential circuits tested at-speed during experiments.

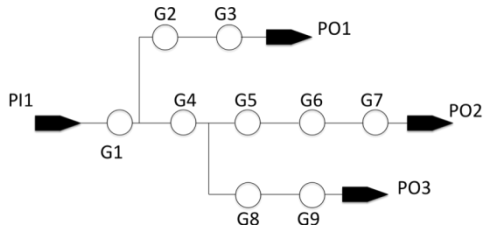


Figure 6. Example

Fig. 6 shows a schematic view of a circuit composed of 9 gates (from G1 to G9), one primary input (PI1) and 3 primary outputs (PO1, PO2 and PO3). For this circuit we have a set of test vectors V = {v_1, v_2}.

TABLE I. JUSTIFIED PATHS

ID Path	Nets
1	PI1 – G1 –G2 – G3 - PO1
2	PI1 – G1 –G4 – G5 – G6 – G7 – PO2
3	PI1 – G1 – G4 - G8 – G9 – PO3

TABLE II. FAULT DICTIONARY

Fault	v_1	v_2
S@G1	000	111
S@G2, S@G3	100	100
S@G4	000	011
S@G5, S@G6, S@G7	010	010
S@G8, S@G9	000	001

Table I and Table II give the set of justified paths and the stuck-at fault dictionary respectively obtained during the off-line phase. Table I reports for each justified path (ID Path) the nets belonging to the path (Nets). Table II reports, for each applied test vector v_i, the information on whether the fault has been detected. If the kth stored bit is equal to 0 it means that the fault has not been detected on the kth PO, otherwise the fault has been detected. The dictionary allows to define each FPO set, for instance: FPO(v_2, S@G4) = {PO2, PO3}. Finally, the notation S@Gi means either a Stuck-at 0 or Stuck-at 1 on the output of gate Gi.

The datalog received after the test corresponds to an error observed at PO2 during the application of v_2. From the datalog we determine the value of FPO(v_1, *fx*) = ∅ and FPO(v_2, *fx*) ={PO2} where *fx* is the unknown fault. We compute *Pfpo* and *Mfpo* sets for each fault of the fault dictionary and then we rank these by applying the LF function of (4). Table III gives the resulting ranked list of suspects. For each fault (Fault), is reported the number of predictions ($\#P_f$) and the number of mispredictions ($\#MP_f$). For instance, faults S@4 has FPO(v_2, S@G4) = {PO2, PO3} thus the number of primary outputs that are failing during both fault simulation and test is equal to 1. On the other hand, the number of primary outputs declared fail only during fault simulation is equal to 1 (i.e., misprediction).

TABLE III. RANKED LIST OF SUSPECTS

Fault	$\#P_f$	$\#MP_f$
S@G5, S@G6, S@G7	1	0
S@G4	1	1
S@G1	1	2
S@G8, S@G9	0	2
S@G2, S@G3	0	2

The final diagnosis report extracted from the ranked list of suspects and the crosschecking with the set of justified paths is shown in Table IV. Only G5, G6 and G7 have zero mispredictions ($\#MP_{f1} = 0$) there are the only suspects that can be affected by a TFM. The others suspects can be affected by GDFM and PDFM only. In this case, since the number of mispredictions is greater than 0 ($\#MP_{f1} > 0$), we can conclude that the fault has not been propagated through all the possible paths as the case of the related conditional stuck-at fault. Therefore is possible to exclude the TFM to be the root cause of the observed error. Fault G8, G9, G2

and G3 are excluded from the diagnosis report because they do not predict any failing primary outputs ($\#P_{f1} = 0$).

Finally, the only possible path associated to the PDFM is the number 2 (reported in Table I), since the suspects in the diagnosis report only belong to this path.

TABLE IV. DIAGNOSIS REPORT

Suspect	Faults
G5, G6, G7	TFM, GDFM, PDFM of path 2
G4	GDFM, PDFM of path 2
G1	GDFM, PDFM of path 2

V. EXPERIMENTAL RESULTS

The aim of the experiments is to validate the proposed approach, in terms of diagnosis resolution, on the ITC99 benchmark circuits b13 and b20, and on the microprocessor 8080 [17]. The adopted circuits have been synthesized by using the technology library AMS 0.35 μm.

TABLE V. CIRCUITS DETAILS

Circuit	#PIs	#POs	#PIOs	#FFs	#Gates
8080	4	21	8	243	5367
b13	12	10	-	51	346
b20	34	22	-	430	13463

Table V details the characteristics of each circuit: the number of primary inputs (#PIs), primary outputs (#POs) and primary input/outputs (#PIOs). Then, the number of Flip-Flops (#FFs) and gates (#Gates) is given.

Let us first define how we instrument the experimental environment. Each test vector sequence is generated by using the commercial ATPG tool TetraMAX® [18]. To simulate the behavior of a faulty circuit we randomly inject one delay fault at a time in the structural description of the circuit. For each circuit we injected the set of delay faults described in section II.

The off-line phase is done by using commercial tools. To determine the set of justified paths (i.e., static timing analysis) we use PrimeTime® [19], and to build the stuck-at fault dictionary we use TetraMAX® [18].

TABLE VI. OFF-LINE PHASE

Circuits	#F	#JP	FD (MB)	JP (MB)
8080	20687	8462	59	32
b13	1482	1045	2	3
b20	50844	427	9	1

Table VI reports the results of the off-line phase for each target circuit. The given information is the number of faults (#F) and justified paths (#JP). Then we give the size of the fault dictionary (FD) and the set of justified pattern (JP) in MegaBytes. Note that the total amount on memory is less than 100 MBs.

TABLE VII. OFF-LINE PHASE

Circuit	#FV	#FPO	Pos	Path	CPU(s)
8080	121.5	1013.67	9.17	0.38%	65.5
b13	19.8	25.8	8	1.97%	0.42
b20	3.75	16.5	7.5	4.25%	36.2

Table VII summarizes the average results obtained on the benchmark circuits. The first column (Circuit) provides the circuit name. The second (#FV) and third (#FPO) columns give the average number of failing test vectors and the number of failing primary outputs respectively. The fourth column (Pos) provides the average diagnosis resolution (i.e., the average number of suspects) achieved by using the proposed diagnosis approach. The fifth column (Path) reports the percentage of path identified during diagnosis. Finally, last column (CPU) shows the average CPU time in second required to perform the diagnosis.

The first comment about the results is that the proposed approach always includes in the list of suspects the fault actually injected in the circuit. Thus proving the reliability of the diagnosis.

A second comment is about to the relation between the achieved diagnosis resolution and the test sequence, expressed in terms of number of failing test vectors and failing primary outputs. Higher are the numbers of failing test vectors and failing primary outputs better is the diagnosis resolution. For example, in the case of the 8080 the path resolution is 0.38% while for the b20 is about 4.25%. Although the resolution may vary with the test set, we can consider that our approach is a very efficient diagnosis tool that can provide reliable and fruitful information to be used during the next phase of the failure analysis process.

A third comment concerns a comparison between our approach and the one of [13] which is the only diagnosis approach targeting path-delay faults in sequential circuit tested at-speed. In [13], the average percentage of path-delay achieved by diagnosis is between 18% and 62% w.r.t. the total number of paths, while our approach reports up to 4.25% of the number of paths. This highlights the improvement of the proposed approach w.r.t state of the art.

Finally, the reported CPU time is between 0.4s and 66s showing the feasibility of the proposed approach.

Conclusions

The diagnosis approach proposed in this paper addresses the problem of delay fault diagnosis in sequential circuits tested at-speed. It allows handling a comprehensive set of delay fault models in a unified manner. Experiments have shown the efficiency of the proposed approach in terms of diagnosis resolution.

REFERENCES

[1] Semiconductor Industry Association (SIA), "International Technology Roadmap for Semiconductors (ITRS)", http://www.itrs.net, 2007.

[2] J. Rearick, R. Rodgers, "Calibrating clock stretch during AC scan testing", IEEE International Test Conference, pp. 266–273, 2005.

[3] M. Abramovici et al., "Digital System Testing and Testable Design", IEEE Press, 1990.

[4] M. Abramovici et al., "Critical Path Tracing – An Alternative to Fault Simulation", IEEE Design & Test of Computers, vol 1, n°1, pp 83-92, 1984.

[5] P. Girard et al., "Delay-fault diagnosis by critical-path tracing", IEEE Design & Test of Computers, vol. 9, n°4, pp. 27-32, 1992.

[6] P. Girard et al., "Diagnostic of path and gate delay faults in non-scan sequential circuits", Proc. IEEE VLSI Test Symposium, pp. 380-386, 1995.

[7] D. B. Lavo et al., "Multiplets, Models, and the Search for Meaning: improving Per-Test Fault Diagnosis", IEEE International Test Conference, pp. 250-259, 2002.

[8] L. M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using the Single Location At a Time (SLAT)", IEEE Transaction on CAD, vol. 23; No 1, pp.91-101, 2004.

[9] A. Rousset et al, "DERRIC: a Tool for Unified Logic Diagnosis", IEEE European Test Symposium, pp. 13-20, 2007.

[10] Kai Yang, Kwang-Ting Cheng, "Timing-reasoning-based delay fault diagnosis", IEEE Design Automation and Test in Europe, pp. 1-6, 2006 .

[11] V.J Mehta et al., "Delay fault diagnosis for nonrobust test", International Symposium on Quality Electronic Design, pp. 466– 472, 2006.

[12] L-T Wang, et al., "System on chip test architectures", Morgan Kaufmann Publisher, 2008.

[13] P. Pant, A. Chatterjee, "Path-delay fault diagnosis in non-scan sequential circuits with at-speed test application", IEEE International Test Conference, pp. 245-252, 2000.

[14] Wu-Tung Cheng, et al., "Enhancing Transition Fault Model for Delay Defect Diagnosis", IEEE Asian Test Symposium, pp. 179-184, 2008.

[15] A. Krstic, et al., "Diagnosis of Delay Defects Using Statistical Timing Models", IEEE VLSI Test Symposium, pp. 339 – 344, 2003.

[16] J.-J. Liou, et al., "On Theoretical and Practical Considerations of Path Selection For Delay Fault Testing", IEEE/ACM International Conference on Computer Aided Design, pp. 94-100, 2002.

[17] http://www.opencores.org

[18] http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx

[19] http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx

[20] P. Bernardi, et al., "Hardware-accelerated Path-Delay Fault Grading of Functional Test Programs for Processor-based Systems", ACM/IEEE Great Lakes Symposium, pp. 411-416, 2007.

TITRE : Diagnostic de Pannes Electriques dans les Systèmes Logiques

RESUME : Les technologies avancées (finesse de gravure inférieure à 90nm) permettent la production de circuits de plus en plus complexes avec des performances très élevées. Ces nouvelles technologies imposent donc de nouveaux défis pour la conception de circuits, mais également pour les méthodologies de test de fabrication et de diagnostic. De ce point de vue, les défaillances observées dans ces technologies ne peuvent pas être modélisées par des fautes classiques de collage. Les fautes de délai, de court-circuit, de circuit ouvert, etc. doivent également être prises en compte. Dans ce contexte, l'objectif de cette thèse a été de développer une méthode de diagnostic logique capable à la fois de traiter un ensemble complet de modèles de fautes et de fournir une localisation fiable et précise des défaillances dans un système sur puce. Ce manuscrit est organisé comme suit. Dans la première partie, les modèles de faute existants sont analysés afin de montrer les conditions de sensibilisation de chacun d'eux. La deuxième partie présente une méthode de diagnostic logique basée sur une approche « Effet-à-Cause». La dernière partie propose une nouvelle technique de diagnostic basée sur une approche « Cause-à-Effet » et permettant de traiter les circuits séquentiels. Les deux approches de diagnostic proposées exploitent les conditions de sensibilisations afin de cibler un ensemble élargi de modèles de fautes durant le processus de diagnostic. La première technique a été complètement validée, la deuxième partiellement validée sur un ensemble important de circuits benchmark et sur des systèmes sur puce fournis par la société STMicroelectronics.

MOTS CLES : Diagnostic, modèles de fautes, simulation de faute, Cause-à-Effet, Effet-à-Cause, Système sur puce

DISCIPLINE : Microélectronique

TITLE: Diagnosis of Electrical Failures in Logic Systems

ABSTRACT: Advanced technologies (technology nodes below 90nm) allow the production of more and more complex and vey high performance circuits. These technologies lead to face with new challenges related to design, test and diagnosis. From this perspective, failures observed in these recent technologies can no longer be modeled by the classical stuck-at fault model. Delay faults, short-circuits, opens, etc. have also to be considered. In this context, the purpose of this thesis has been to develop a logic diagnosis approach able to deal with many types of faults as well as providing an reliable and accurate localization of failures in a system on chip. This manuscript is organized as follows. In the first part, existing fault models are analyzed in order to show the sensitization conditions related to each of them. The second part presents a logic diagnosis method based on the "Effect-Cause" paradigm. The last part proposes another diagnosis technique based on the "Cause-Effect" paradigm to deal with sequential circuits. The two proposed diagnosis approaches exploit the sensitization conditions in order to be able to consider a large set of fault models during the diagnosis process. First technique has been completely validated, second technique partially validated on a large set of benchmark circuits and on System-On-Chips provided by STMicroelectronics.

KEYWORDS : Diagnosis, fault models, fault simulation, Cause-Effect, Effect-Cause, System-On-Chip

Université de Montpellier II : Sciences et Techniques du Languedoc

LIRMM : Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier
161 Rue Ada – 34392 Montpellier Cedex 5.