ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN TECHNOLOGIE DES SYSTEMES
M.Ing.

PAR
YASSINE HARIRI

AMÉLIORATION DE LA MÉTHODE DE DIAGNOSTIC BASÉE SUR LES SIGNATURES PROBABILISTES DE ΔIDDQ

MONTRÉAL, LE 25 JUILLET 2002

© droits réservés de Yassine Hariri

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Claude Thibeault, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Naim Batani, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Jean Belzile, jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 27 JUIN 2002

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

AMÉLIORATION DE LA MÉTHODE DE DIAGNOSTIC BASÉE SUR LES SIGNATURES PROBABILISTES DE ΔIDDQ

Yassine Hariri

SOMMAIRE

Ce projet de maîtrise porte globalement sur l'amélioration d'une méthode existante de diagnostic des circuits intégrés, méthode basée sur l'utilisation des signatures probabilistes du courant. Les améliorations visées se situaient à différents niveaux. D'une part, il s'agissait d'accélérer la méthode en réduisant le nombre de sites potentiels de court-circuit. D'autre part, nous avions comme objectif de palier à certaines limitations de l'outil logiciel qui avait été développé pour permettre la validation de la méthode, comme le fait qu'il ne simule que les circuits purement combinatoires. La technique de réduction des sites physiques potentiels utilise les capacités parasites de routage extraites du dessin des masques du circuit. Les résultats d'expérimentation montrent que le pourcentage de réduction des sites physiques potentiels est toujours supérieur à 94%, et qu'il peut atteindre 99.99% dans le cas des circuits plus complexes. L'intégration de cette technique de réduction, a permis d'accélérer le processus de diagnostic. En effet, avec un nombre ainsi réduit de sites physiques, le programme converge plus rapidement vers la solution tout en utilisant moins de ressources matérielles. Du côté de l'outil logiciel d'émulation de la méthode, nous avons développé l'infrastructure nécessaire permettant de l'interfacer avec l'outil de simulation Verilog-xl de Cadence. Nous sommes donc maintenant en mesure d'utiliser la méthode de diagnostic sur des circuits combinatoires et séquentiels.

IMPROVEMENT OF A DIAGNOSIS METHOD BASED ON CURRENT PROBABILISTIC SIGNATURES

Yassine Hariri

ABSTRACT

This project relates overall on the improvement of an existing diagnosis method for integrated circuits, this method is based on probabilistic signatures of leakage currents. The improvements concerned were at various levels. First, the method is accelerated by reducing the number of potential sites of short-circuits. Second, we had as goal to deal with certain software limitations. For example, the software simulates only purely combinatorial circuits. The technique of reduction of the potential physical sites uses the parasitic capacities of routing extracted from the circuit layout. Experimentation results show that the reduction ratio of the potential physical sites is always higher than 94%, and can reach 99.99% in the case of more complex circuits. The integration of this reduction technique, allows the acceleration of the diagnostic process. Indeed, with a reduced number of physical sites, the program converges quickly towards the solution using less material resources. On the software aspects of the method, we developed the necessary infrastructure allowing the interface with the Verilog-xl simulation tool. We are now able to use the diagnostic tool to process combinatorial and sequential circuits.

AVANT-PROPOS

Cette recherche, qui entre dans le cadre du programme de maîtrise, a comme mission d'améliorer une méthode de diagnostic basée sur des signatures de courant.

Le but de ce rapport est de présenter les différentes phases d'amélioration et de faire connaître les démarches et les obstacles encourus lors de cette recherche. Ce travail aboutira à la conception d'un outil logiciel permettant d'émuler la méthode de diagnostic.

J'aimerais remercier le professeur Claude Thibeault qui est le concepteur de la méthode de diagnostic basée sur des signatures du courant, et qui fut d'une aide indispensable lors de la rédaction de mon mémoire et lors de l'analyse des résultats.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

INTRODUCTION

Avec l'augmentation sans cesse croissante du nombre de transistors sur une même puce, le diagnostic des défectuosités (imperfections au niveau physique) menant à des pannes (au niveau logique) devient une tâche de plus en plus difficile à réaliser, mais dont l'importance ne cesse de croître. C'est entre autres grâce au diagnostic que les manufacturiers de semi-conducteur parviennent à améliorer le rendement de leurs procédés de fabrication. En effet, le diagnostic permet l'identification et la localisation des défectuosités menant à des pannes et affectant les circuits intégrés. Ces étapes permettent de faire ressortir les points faibles d'un procédé menant à des mécanismes répétitifs de défectuosités, et à modifier ce procédé en conséquence. Par exemple, si les lignes de métal sont trop près l'une de l'autre, des court-circuits vont se produire de manière répétée. En identifiant cette cause, le diagnostic permet aux ingénieurs de procédés de redéfinir la distance minimale entre deux lignes de métal.

Le processus de diagnostic d'une défectuosité ou d'une panne se termine habituellement par un examen visuel du circuit intégré à l'aide d'un microscope électronique. Cette opération étant particulièrement longue, on essaie toujours de limiter au maximum les surfaces à examiner. Pour y arriver, on utilise diverses informations, à commencer par les résultats des différents tests effectués sur un testeur, ce qui inclut les niveaux logiques en sortie et le courant de consommation en régime permanent (I_{DDQ}).

Des travaux passés ont mené à l'élaboration d'une méthode de diagnostic basée sur le concept des signatures probabilistes du courant différentiel ΔI_{DDQ}. L'expression « courant différentiel ΔI_{DDQ} » signifie que l'on s'intéresse à la différence qui existe entre deux mesures consécutives de courant. Le terme « signature » indique que l'on tire profit du fait que certains types de pannes menant à des formes particulières de la distribution du courant différentiel, ce qui facilite leur identification. Le terme «probabiliste» s'applique à la manière de comparer la signature d'un circuit sous test avec celles anticipées des différents types de pannes. Cette méthode cible particulièrement les court-circuits.

Un programme a été conçu en langage C pour émuler et valider cette méthode de diagnostic. Ce dernier exécute deux phases principales. La première phase consiste à identifier les types de pannes les plus probables, grâce à la comparaison probabiliste des signatures. La deuxième phase tente de localiser la panne, en commençant par le type de panne le plus probable. Localiser une panne signifie de trouver un emplacement pour une défectuosité de type court-circuit pouvant causer une consommation de courant différentiel s'apparentant à celle mesurée sur le circuit sous test. Pour un type de panne donné, le programme doit considérer toutes les combinaisons possibles de court-circuits possibles afin de trouver le couple de nœuds en court-circuit.

Ce projet vise l'amélioration de cette méthode de diagnostic existante, et ce à différents niveaux. Notre premier objectif est de réduire le nombre de sites potentiels d'emplacements des court-circuits, afin de diminuer le temps requis pour effectuer le diagnostic. Pour effectuer cette première réduction, nous allons utiliser les capacités parasites de routage extraites à partir du dessin des masques du circuit. Une autre réduction est possible, et elle se base sur les résultats de l'émulation du testeur. Cette deuxième technique permet de réduire la liste des nœuds candidats, en utilisant les résultats erronés des sorties du circuit. Nous allons présenter dans ce mémoire les deux techniques de réduction, avec des résultats de tests effectués sur des circuits différents. Comme on peut le constater, ces réductions sont faites en greffant aux mesures de courant des informations jusqu'à présent non exploitées par la méthode de diagnostic.

Notre deuxième amélioration vise le programme d'émulation. Le simulateur intégré du programme d'émulation de la méthode de diagnostic présente certaines limitations, dont celle de ne supporter que les circuits logiques purement combinatoires. Or dans la plupart des applications, on rencontre souvent les circuits séquentiels. Pour palier à ce problème, nous allons utiliser un simulateur commercial à l'intérieur du programme d'émulation.

Ce mémoire est structuré comme suit. Le chapitre 1 présente dans un premier temps une description sommaire de la méthode de diagnostic avec des résultats pratiques, pris à partir d'un circuit intégré avec des défauts contrôlables, et introduit dans un deuxième temps le programme de simulation qui a été conçu pour émuler la méthode de diagnostic. Le chapitre 2 explique la procédure d'extraction des capacités parasites à partir du dessin des masques du circuit, et donne les résultats de réduction en se basant sur ce paramètre physique. Le chapitre 3 explique la procédure générale de modélisation des pannes, en se basant sur les résultats de simulation *Hspice*. Le chapitre 4 présente la procédure d'insertion du modèle de panne dans le circuit, et les techniques de réduction du graphe représentant le circuit en se basant sur les résultats erronés du testeur. Le chapitre 5 présente la procédure d'utilisation des résultats de simulation pris sur le simulateur Verilog-xl dans le programme d'émulation. Le chapitre 6 explique la procédure d'intégration finale au niveau du programme d'émulation avec des résultats de tests effectués sur différents circuits. Le mémoire se termine par une conclusion et quelques recommandations.

CHAPITRE 1

MÉTHODE DE DIAGNOSTIC BASÉE SUR DES SIGNATURES PROBABILISTES

Pour permettre une bonne compréhension du travail présenté dans ce document, il est opportun d'introduire la méthode de diagnostic basée sur des signatures probabilistes du courant différentiel ΔI_{DDQ}, et de rappeler certains concepts liés au domaine du test et du diagnostic. Ce chapitre présente dans un premier temps une description sommaire de la méthode de diagnostic avec des résultats pratiques, pris à partir d'un circuit intégré avec des défauts contrôlables, et introduit dans un deuxième temps le programme de simulation qui a été conçu pour émuler la méthode de diagnostic. À la fin de ce chapitre nous allons définir les aspects à améliorer. Pour chaque aspect, nous allons choisir une solution appropriée, et poser finalement un plan d'action qui sera suivi pour la mise en place des solutions choisies.

1.1 Description de la méthode de diagnostic basée sur le courant différentiel ΔI_{ddq}

La méthode de diagnostic basée sur des signatures probabilistes du courant différentiel $\Delta\ I_{ddq}$ permet, à partir des mesures de courant prises sur un circuit, d'identifier dans un premier temps les types de pannes les plus probables, et de localiser les pannes dans un deuxième temps. La méthode exploite le fait que lorsque plusieurs vecteurs de test I_{DDQ} sont appliqués sur un circuit, ce dernier génère une signature de courant constituée de différents niveaux relatifs, qui sont fonction de la panne dans le circuit sous test. De plus, comme il sera expliqué en détails plus loin, ces signatures sont construites de manière probabiliste et différentielle, ce qui donne à cette méthode une caractéristique unique qui la distingue des méthodes précédentes [5][6].

1.1.1 Courant différentiel ΔI_{DDQ}

On définit une mesure différentielle du courant comme étant la différence entre la valeur lue du courant I_{DDQ} à un vecteur de test donné et la valeur lue au vecteur de test précédent. La figure 1 montre graphiquement la relation entre I_{DDQ} et ΔI_{DDQ}. Les deux niveaux I_{DDQ} apparents se transforment en trois niveaux ΔI_{DDQ}, les niveaux ΔI_{DDQ} sont centrés à 0.

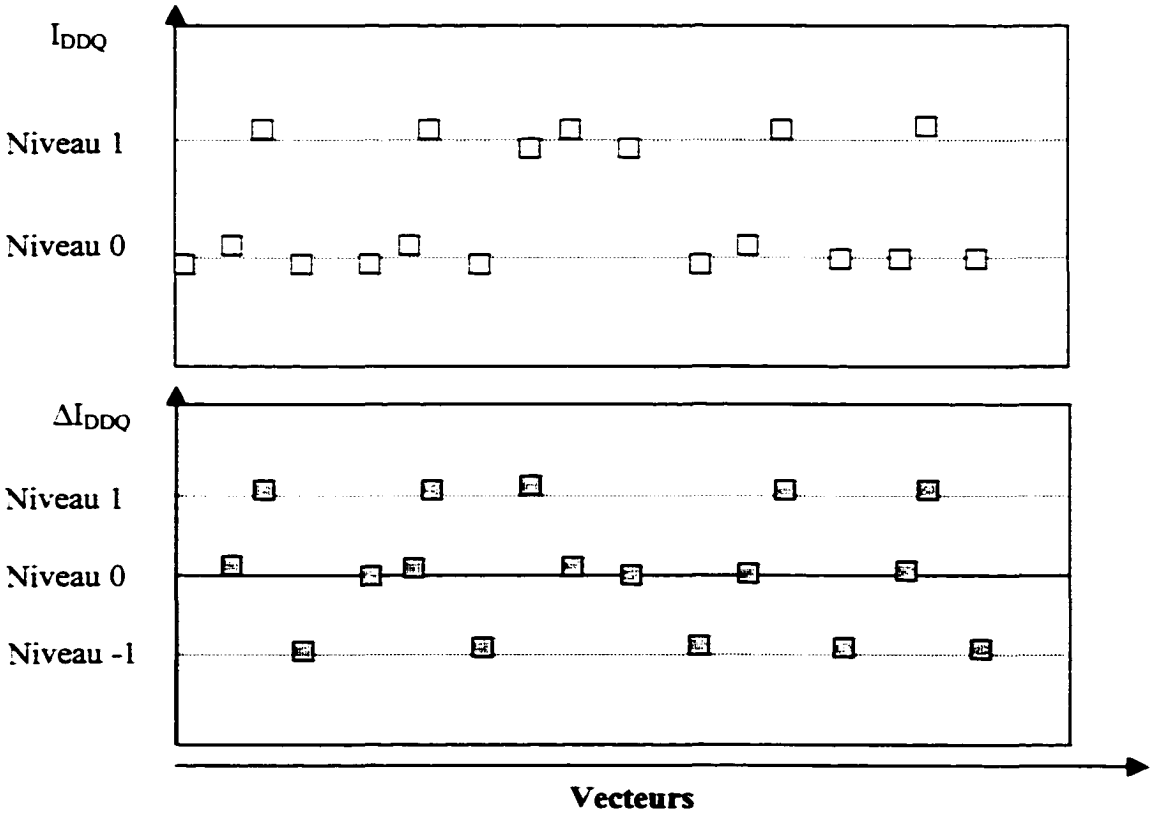


Figure 1 Mesures de I_{DDQ} pour chaque vecteur et passage à ΔI_{DDQ}

Comme indiqué par l'équation 1-1, le courant I_{DDQ} lu à chaque vecteur de test peut être représenté par un courant engendré par une panne dans le circuit I_{br} (le courant circulant dans le court-circuit parasite), et la somme de tous les courants de fuite I_{fuite} dans le circuit.

$$I_{DDQ} = I_{br} + \sum I_{fuite} \qquad (1\text{-}1)$$

Le courant de fuite total peut varier d'un circuit à l'autre ainsi que d'un vecteur de test à l'autre. Il n'est pas réaliste de supposer que le courant de fuite qui n'est pas causé par des défectuosités reste constant pour tous les vecteurs de test appliqués, ceci pouvant être justifié par le nombre de transistors à l'état fermé qui vari d'un vecteur de test à l'autre, et par le fait que ces transistors n'ont pas nécessairement la même géométrie.

Pour une raison de simplification de la comparaison des signatures de la méthode, nous supposons que le courant de fuite suit une distribution normale d'un vecteur de test à l'autre avec une moyenne de zéro et une certaine variance. Cette supposition est appuyée par les résultats d'expérimentation, qui suggèrent que la consommation de courant de fuite dans un circuit de grande dimension peut être modélisée par une telle distribution [1].

La figure 2 représente de manière symbolique la distribution du courant de fuite de circuits sans défectuosités et celle de circuits comportant une panne consommant un courant constant. On remarque que les pannes dans un circuit déplacent latéralement la distribution normale du courant.

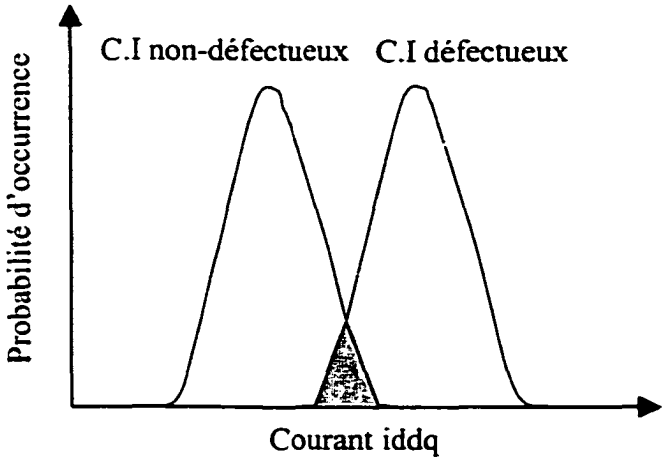


Figure 2 Distribution du courant

Lorsque modélisé de manière adéquate, le recoupement entre les deux distributions constaté sur la figure 2 correspond à la probabilité d'une mauvaise décision de test.

Il est d'intérêt de mentionner qu'une comparaison détaillée entre le courant I_{DDQ} et le courant différentiel ΔI_{DDQ} a été faite en [1], montrant que le courant différentiel ΔI_{DDQ} peut diminuer par un ou deux ordres de grandeur la probabilité d'une mauvaise décision de test basée sur le courant, en diminuant la variance des distributions du courant, autrement dit, en diminuant la région de recoupement des deux distributions. Cette diminution de la variance va également être bénéfique pour le diagnostic. Outre la diminution de la variance, l'utilisation du courant différentiel offre l'avantage de permettre le diagnostic de court-circuits sur des signaux actifs malgré la présence de défectuosités causant un niveau I_{DDQ} constant.

1.1.2 Signatures de courant

Les signatures de courant telles que proposées en [1] permettent la reconnaissance des pannes à partir des niveaux différents du courant différentiel ΔI_{DDQ}. La nature probabiliste de la méthode, définie plus loin, rend celle-ci plus robuste face aux diverses sources de variation.

La signature du courant peut être définie comme étant la valeur excessive prévue du courant pour un type de panne donné. Les pannes considérées sont associées à des problèmes de court-circuit entre les nœuds de sortie des portes logiques. Considérons par exemple une panne engendrée par le court-circuit de la sortie d'un inverseur et la masse, i.e. une panne de type collé à 0 (stuck at 0), telle que représentée à la figure 3.

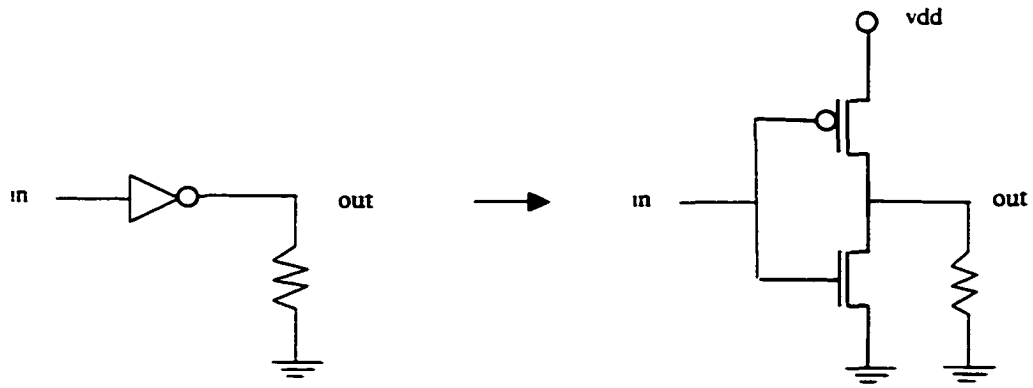


Figure 3 Panne de type collé à 0 de la sortie d'un inverseur

Le tableau I présente la table de vérité et donne les états prévus de la sortie de l'inverseur, dans deux cas possibles. Le premier cas présente le fonctionnement normal sans panne, et le deuxième la présence d'une panne de type collé à 0. En utilisant les principes de contrôlabilité, on trouve que pour détecter le type de panne collé à 0, on doit mettre l'entrée « in » à la valeur logique « 0 ».

Tableau I

Table de vérité pour les deux type de blocage

in	out sans panne	out collé à 0
0	1	0
1	0	0

Le tableau II présente les valeurs de courant lors de l'application d'un vecteur de test k-1 et lors du vecteur de test suivant k. Le résultat ΔI_{DDQ} est la soustraction de la valeur de courant au vecteur k-1 à celle au vecteur k.

Soit IL la valeur de courant lors d'un conflit. (i.e. lorsque in = 0). Tout comme ce fut le cas pour le grapne de la figure 1, les résultats présentés au tableau II montrent que les deux niveaux de courant I_{DDQ} (0 et IL) se transforment en trois valeurs de ΔI_{DDQ} (0,+IL,-IL). À ce stade, on peut introduire la nature probabiliste de la méthode, basée sur le fait que chacune des trois valeurs correspond à la moyenne μ_j d'un évènement Δ_J. Une variance σ^2_j et une certaine probabilité P(j) sont également associées à cet événement.

Tableau II

Delta I_{DDQ} pour l'inverseur branché à la masse

vecteur k-1		vecteur k		ΔI_{DDQ}	probabilité
entrée	I_{DDQ}	entrée	I_{DDQ}		
0	IL	0	IL	0	0.25
0	IL	1	0	-IL	0.25
1	0	0	IL	+IL	0.25
1	0	1	0	0	0.25

Le calcul de P(j) suppose une équiprobabilité de chacun des vecteurs. Par exemple, dans le tableau précédant, les probabilités d'avoir un ΔI_{DDQ} de +IL ou de -IL sont de 0.25 chacune, et la probabilité d'avoir un ΔI_{DDQ} de zéro est de 0.5.

La figure 4 représente la signature basée sur les résultats de l'exemple précédent.

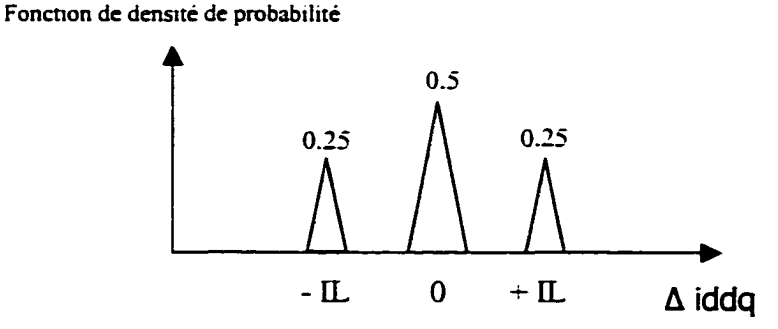


Figure 4 : Signature d'une panne de type collé à 0 de la sortie de l'inverseur

Pour faire le diagnostic d'un CI, on doit construire un dictionnaire de signatures pour tous les types possibles de pannes pouvant êtres présents à partir de la liste des différentes portes utilisées.

Pour notre preuve de concept, les portes logiques considérées jusqu'à présent sont : l'inverseur, le tampon, la porte Non-Et à deux entrées, la porte Non-Et à trois entrées, la porte Non-Ou à deux entrées, et la porte Non-Ou à trois entrées. Elles représentent jusqu'à 94% des types de portes utilisées dans les circuits (pourcentage calculé à partir du banc d'essai ISCAS (Brglez et Fjiwara[2])). Les signatures peuvent êtres construites à partir de résultats de simulation ou d'expérimentations.

1.1.3 Phase d'identification des types de pannes les plus probables

Comme indiqué précédemment, la méthode comporte deux phases. La première phase joue un rôle de pré-traitement. Elle consiste à classer les types de pannes suivant leurs probabilités d'apparition. La figure 5 montre comment ce processus se déroule.

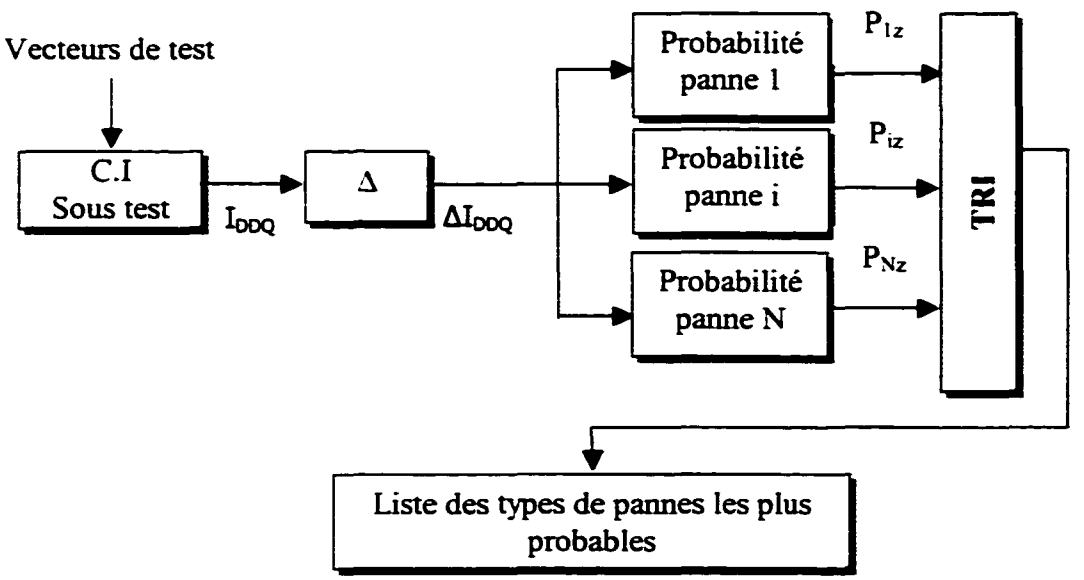


Figure 5 Identification des types de pannes les plus probables

Dans un premier temps, le circuit est stimulé par des vecteurs d'entrées. Après la stabilisation du réseau logique du circuit suite à l'application d'un vecteur de test, on mesure le courant d'alimentation du circuit. Tel que mentionné précédemment, la valeur ΔI_{DDQ} pour un vecteur est la différence entre la valeur du courant lue à ce vecteur et la lecture précédente du courant.

On commence par calculer pour chaque type de panne la probabilité P_{iz} que le ΔI_{DDQ} mesuré soit dû à ce type de panne. Le calcul des probabilités P_{iz} utilise la technique de l'estimation à maximum de vraisemblance (« Maximum Likelihood Estimation », MLE) utilisé notamment en télécommunication [3]. Notons également que l'algorithme de Viterbi peut être adapté [4] pour évaluer cette probabilité. Pour plus de détails sur le calcule des probabilités, voir l'annexe 1.

Après le calcul de toutes les probabilités, on procède à un tri suivant un ordre décroissant des P_{iz} pour créer la liste des pannes les plus probables.
Le minimum d'information nécessaire pour effectuer ces calculs est la liste des portes utilisées dans le circuit. Une connaissance du netlist ou du dessin des masques du circuit n'est donc pas nécessaire pour cette phase. On pourrait cependant exploiter ces informations afin de raffiner le calcul des probabilités. Jusqu'à présent, ces informations additionnelles n'ont pas été utilisées. Leur exploitation fait partie des travaux futurs.

1.1.4 Phase de localisation

La phase de localisation a comme objectif la création d'une liste de nœuds ayant la plus grande probabilité d'être impliqués dans un type de pannes donné. La figure 6 représente le processus de localisation.

La procédure de localisation utilise la liste de résultats générée lors de la phase d'identification. On commence par analyser le type de pannes le plus probable. Une liste de paire de nœuds pouvant engendrer ce type de pannes est préparée au départ. En se basant sur les vecteurs d'entrée, le simulateur calcule le courant I_{DDQ} qui serait généré par une panne située sur chacune de ces combinaisons de nœuds.

En utilisant le MLE pour comparer les ΔI_{DDQ} simulés et mesurés, les localisations possibles pour le type de panne en cours de traitement sont placées dans une liste ordonnée. Le calcul de ces probabilités se fait de la même façon que pour la phase d'identification. La plus élevée des PL_{iz} (probabilité pour qu'une faute de type donné affecte la localisation i avec un ensemble z de ΔI_{DDQ}) correspond à la localisation la plus probable dans la liste ainsi définie.

On utilise un seuil de décision (voir l'annexe 1) pour éliminer les localisations les plus improbables. Si aucune localisation de nœud ne satisfait le critère pour un type de panne donné, on considère automatiquement que ce type de panne n'est pas présent dans le circuit, et sera donc rejeté, et on passe au type de panne suivant dans la liste.

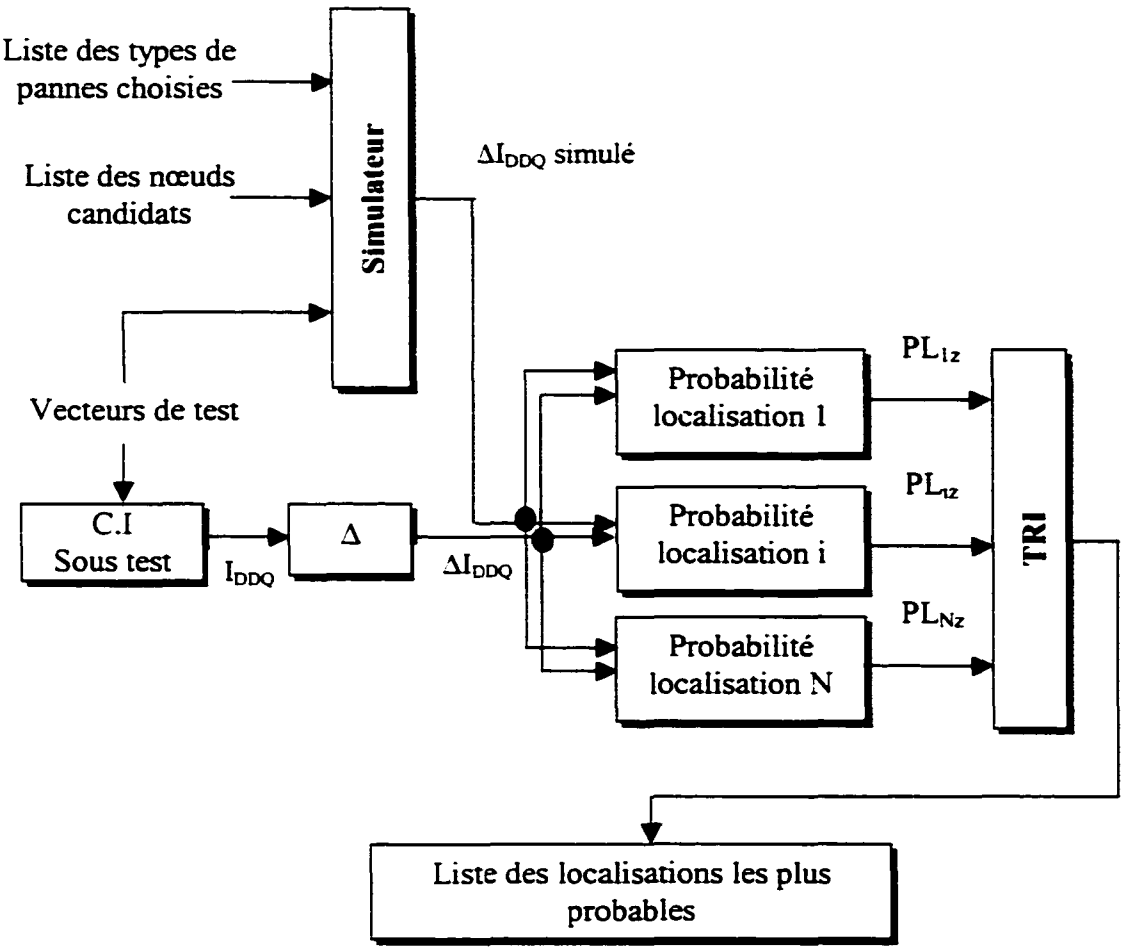


Figure 6 Localisation des pannes.

Si une (ou plusieurs) combinaison de nœuds demeure dans la liste après l'application du seuil, alors cette (ou ces) paire(s) de nœuds est (sont) considérées comme la localisation de la panne, et le diagnostic est considéré comme ayant été accompli.

Avant le début des travaux liés à ce mémoire, un programme a été conçu pour émuler la méthode de diagnostic. Ce programme est décrit en détail à l'annexe 2. De plus, l'annexe 3 présente une démonstration du programme d'émulation sur un petit circuit.

1.1.5 Autres méthodes utilisant des signatures de courant

Le diagnostic des circuits intégrés a déjà vu l'utilisation des signatures de courant. Une méthode a été développée par Soden, Hawkins et Miller [5], qui consiste à appliquer une rampe de tension sur l'alimentation et sur les entrées du circuit ayant la valeur logique un. La courbe du courant lu peut être interprétée pour identifier le type de défectuosité présente.

Une autre méthode proposée par Maly et Gattiker [6] consiste à trier en ordre croissant les mesures de courant, et le graphique résultant peut être analysé pour extraire les informations de test. Comme conclusion à leur travail, il semble qu'une signature de courant représentée par un niveau bas sans marches importantes dans la signature indique une fuite de courant qui probablement ne donne pas d'erreurs fonctionnelles, tandis qu'une signature avec marches importantes semble indiquer une défectuosité potentiellement critique. Les signatures de courant peuvent fournir une autre information, à savoir la nature des défectuosités basée sur l'amplitude des courants présents. La nature probabiliste et différentielle des signatures de courant utilisées par la méthode décrite dans ce chapitre est l'élément qui la distingue des autres méthodes existantes et la rend plus robuste.

1.2 Aspects à améliorer et solutions proposées

L'objectif de cette section est de définir plus en détails le contexte du mémoire, qui entre dans le cadre d'un effort de recherche permettant l'amélioration de la méthode de diagnostic ci-avant présentée. Nous allons décrire les aspects à améliorer, proposer des solutions permettant d'atteindre nos objectifs et poser un plan d'action qui va être suivi afin de mettre en place chacune des solutions choisies.

1.2.1 Premier aspect : nombre de sites potentiels pour les court-circuits

Les technologies microélectroniques ainsi que les chaînes de CAO actuelles permettent la conception de plus en plus rapide de circuits intégrés très complexes. Le diagnostic demande des ressources matérielles et logicielles qui augmentent significativement en fonction du nombre de portes dans le circuit à diagnostiquer. Tel que décrit précédemment, la méthode de diagnostic passe par deux phases, dont la phase de localisation des pannes qui demande présentement le traitement de **toutes** les paires de nœuds possibles pour le type de panne cible.

Tableau III

Statistiques sur le nombre de sites physiques pour des circuits différents

Circuit Informations	Multiplicateur 4bits	FIR 4 étages	FIR 7 étages	FIR 15 étages	FIR 25 étages	FIR 50 étages
Nombre de portes	94	901	1 641	3 521	6 468	13 127
Nombre de nœuds	117	986	1 772	3 759	7 262	13 855
Nombre de sites physiques	990	72 771	271 598	938 904	3 128 751	8 889 436

NB : FIR = Filtre à réponse impulsionnelle finie (FIR : Finite Impulse Response).

Le tableau III présente le nombre de sites physiques à considérer pour des circuits de taille différente. On définit le nombre de sites physiques comme étant le nombre total de toutes les paires de nœuds possibles pour chaque type de pannes.
Les types de pannes considérés dans le cadre de ce projet sont présentés au tableau IV. On rappelle que le modèle de panne utilisé est celui d'un court-circuit entre les sorties de deux portes logiques.

Tableau IV

Types de pannes considérées

Pannes	Description	Portes impliquées
TF0	INV - INV	inverseur - inverseur
TF1	INV-NAND2	inverseur - porte non-et à 2 entrées
TF2	INV-NAND3	inverseur - porte non-et à 3 entrées
TF3	INV-NAND4	inverseur - porte non-et à 4 entrées
TF4	NAND2-NAND2	porte non-et à 2 entrées - porte non-et à 2 entrées
TF5	NAND2-NAND3	porte non-et à 2 entrées - porte non-et à 3 entrées
TF6	NAND2-NAND4	porte non-et à 2 entrées - porte non-et à 4 entrées
TF7	NAND3-NAND3	porte non-et à 3 entrées - porte non-et à 3 entrées
TF8	NAND3-NAND4	porte non-et à 3 entrées - porte non-et à 4 entrées
TF9	NAND4-NAND4	porte non-et à 4 entrées - porte non-et à 4 entrées

On constate une augmentation exponentielle du nombre de sites physiques en fonction du nombre de portes logiques dans le circuit. Cette augmentation demande d'une part un temps de calcul très important, et d'autre part une grande quantité d'espace mémoire afin d'effectuer le diagnostic.

Notre premier objectif est donc de réduire le nombre de sites potentiels pour le diagnostic, en exploitant au départ certaines caractéristiques physiques du circuit. Nous n'avons pas besoin de traiter tous les sites potentiels du circuit. En effet, ces sites potentiels représentent toutes les paires de nœuds qu'il est possible de former à partir de la liste de nœuds cibles pour un type de pannes donné. Or, il faut que les nœuds soient à proximité pour qu'il puisse y avoir un court-circuit. La présence d'une capacité parasite entre deux nœuds est un indicateur assez fiable de la probabilité de court-circuit entre ces deux nœuds. Nous allons donc utiliser cet indicateur pour établir la liste des paires de nœuds pouvant réellement devenir le site d'un court-circuit, et par le fait même

réduire le nombre des sites potentiels. Le prochain chapitre explique la procédure d'extraction des capacités parasites à partir du dessin des masques du circuit, et donne les résultats de réduction en se basant sur ce paramètre physique.

Nous allons également investiguer l'utilisation des valeurs logiques erronées aux sorties du circuit sous test comme moyen de réduire le nombre de sites potentiels. Dans ce cas-ci, les résultats erronés des sorties du circuit suite à l'application des vecteurs de test servent à identifier les nœuds pouvant être à l'origine des erreurs. Pour émuler le testeur, nous avons besoin dans un premier temps des modèles de pannes. Le chapitre 3 explique la procédure générale de modélisation, en se basant sur les résultats de simulation par le logiciel Hspice. Le chapitre 4 présente la procédure d'insertion du modèle de panne dans le circuit, et des techniques de réduction du graphe représentant le circuit en se basant sur les résultats erronés du testeur.

1.2.2 Deuxième aspect : extension vers les circuits séquentiels

Le programme développé pour émuler la méthode de diagnostic ne supporte que les circuits purement combinatoires, alors que dans la plupart des applications rencontrées, on trouve des circuits séquentiels. L'extension du programme d'émulation du diagnostic vers les circuits séquentiels représente donc un deuxième objectif. Deux alternatives sont possibles :

La première consiste à améliorer le simulateur logique existant dans le programme d'émulation pour qu'il supporte les circuits séquentiels, et la deuxième vise l'intégration d'un simulateur commercial au programme. Au chapitre 5, nous discutons les critères de choix et la mise en place de la stratégie choisie. Le chapitre 6 présente la procédure d'intégration finale au niveau du programme d'émulation avec des résultats de test effectué sur des circuits différents.

1.3 Conclusion

Nous avons présenté la méthode de diagnostic qui constitue le point de départ de ce projet. Nous avons également présenté les aspects que nous comptions améliorer, aspects à partir desquels découlent les objectifs de ce projet.

CHAPITRE 2

RÉDUCTION DU NOMBRE DE SITES POTENTIELS EN CONSULTANT LE DESSIN DES MASQUES

Ce chapitre présente la procédure d'extraction des capacités parasites de routage à partir du dessin des masques d'un circuit. Nous allons d'abord étudier les différentes étapes préparant cette extraction. Puis nous présentons le processus d'extraction des capacités ainsi que son automatisation. À la fin de ce chapitre, nous allons donner quelques résultats de l'extraction à partir de différents dessins des masques, et le degré de réduction des sites physiques obtenus.

2.1 Généralités sur l'extraction des capacités parasites

Les capacites parasites de routages sont generalement utilisées dans le cadre des simulations logiques tenant compte des delais sur les lignes d'interconnections, l'utilisation de ces capacités pour identifier les pannes de type pont est proposée en [9][10].

Les étapes préparatoires à l'extraction sont celles menant à l'obtention du dessin des masques d'un circuit intégré. Après avoir décrit et simulé le circuit à l'aide d'un langage approprié (VHDL, Verilog ou autres...), les étapes suivantes sont habituellement la synthèse et la création du dessin des masques du circuit, cette dernière étape est réalisée en utilisant un outil comme « Cadence », qui permet le placement et le routage automatisés des cellules normalisées. Le tout se termine par diverses étapes de vérification.

Dans une puce typique de circuits numériques à très haute échelle d'intégration, les capacités parasites d'interconnexion peuvent apparaître de différentes façons. Chaque

ligne d'interconnexion (fil) est une structure tridimensionnelle de métal et/ou de polysilicium, avec des variations significatives de forme, d'épaisseur, et de distance. En outre, chaque ligne d'interconnexion est typiquement entourée par un certain nombre d'autres lignes, au même niveau ou à différents niveaux.

La figure 7 montre une situation possible et réaliste où des lignes d'interconnexions forment trois niveaux.

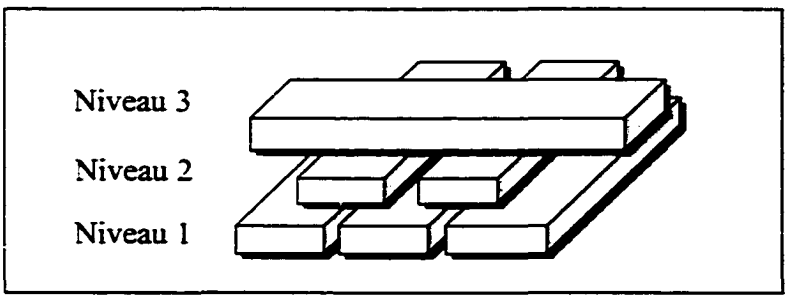


Figure 7 Exemple de six lignes d'interconnexion en trois niveaux.

La figure 8 montre la vue en coupe d'une structure CMOS à double métal, où les différentes capacités parasites entre les couches sont également indiquées. La section transversale ne montre pas un transistor MOSFET, mais juste une partie d'une région de diffusion au-dessus de laquelle quelques lignes en métal peuvent passer. Les capacités de couches intercalaires entre le metal-2 et le metal-1, metal-1 et polusilicium, et metal-2 et polusilicium sont étiquetées comme C_{m2m1}, C_{m1p} et C_{m2p}, respectivement.

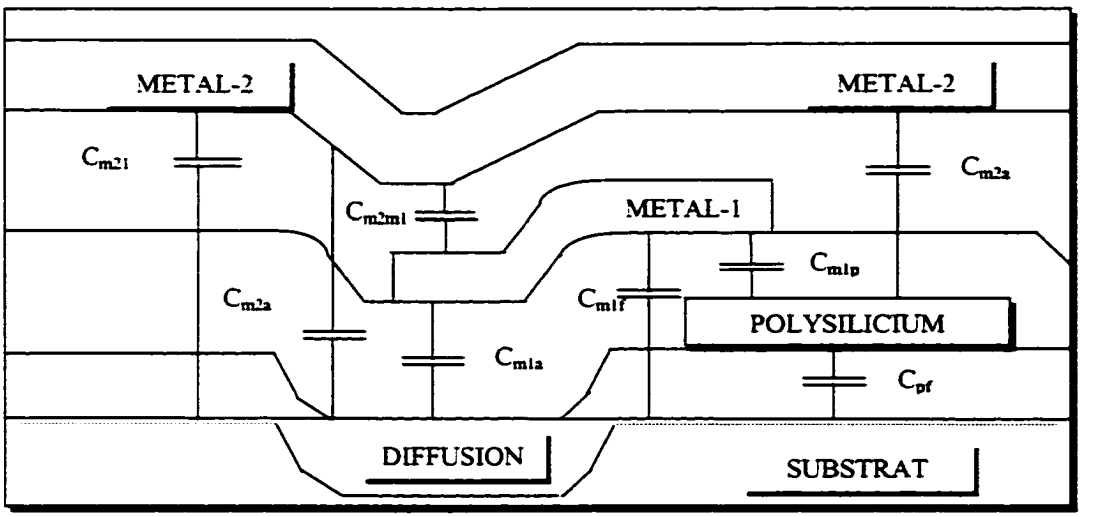


Figure 8 Vue en coupe d'une structure CMOS à double-métal, avec les capacités parasites entres les couches

Pour une évaluation précise des capacités d'interconnexion dans une structure tridimensionnelle, la géométrie exacte doit être prise en considération pour chaque partie du fil. Cependant, ceci exige une quantité jugée inacceptable de calculs dans un grand circuit, même si des formules simples sont appliquées pour le calcul des capacités. Des hypothèses simplificatrices sont donc utilisées pour réduire les efforts de calculs. Habituellement, les fabricants de puces fournissent la capacité de zone et les chiffres de capacités de périmètre pour chaque couche. Ces données peuvent être employées pour extraire les capacités parasites à partir de la disposition des masques, comme c'est le cas pour l'outil « Analog Artist » de Cadence, que nous avons choisi pour effectuer cette tâche. Cet outil utilise une base de données, contenant des informations utiles pour l'extraction. Ces informations dépendent évidemment de la technologie utilisée, cmosp35 (0.35µm) dans le cadre de notre étude.

2.2 L'extraction des informations utiles du dessin des masques

L'objectif de l'extraction est donc d'obtenir l'ensemble des capacités parasites de routage dans un dessin des masques. Tel que mentionné, nous utilisons l'extracteur inclus dans l'outil « Analog Artist » de Cadence, qui génère la liste des capacités sous forme d'un fichier. On trouve dans le même fichier les capacités parasites de routage, le « netlist » extrait du dessin des masques du circuit qui sera utilisé pour repérer les nœuds et connaître les références de chaque capacité parasite de routage, et la définition des interfaces pour chaque type de porte utilisée dans le circuit. Après l'obtention de ce fichier, il sera filtré afin d'en extraire les informations pertinentes. Pour ce faire, un programme en langage C a été conçu afin d'automatiser le processus. Il a comme fichier d'entrée, le fichier généré par l'extracteur, et il génère à son tour plusieurs fichiers contenant des données qui seront directement utilisables ultérieurement.

La figure 9 donne l'exemple d'un fichier représentant les informations sur le dessin des masques d'un circuit. On peut clairement identifier trois sections dans ce fichier. La première section donne toutes les capacités parasites du circuit. Cette section sera filtrée pour ne laisser que les capacités parasites de routage, i.e. les capacités qui se trouvent entre les nœuds du circuit, excluant les nœuds d'alimentation (Vdd , Gnd). La deuxième section donne le « netlist » extrait du dessin des masques. Mentionnons ici qu'il y a une différence entre le « netlist » du schéma provenant de l'outil de synthèse et le netlist extrait du dessin des masques du circuit. Cette différence se situe sur le fait que le dessin des masques prend en considération le routage des cellules (portes logiques). La notation des nœuds est par conséquent différente de celle du « netlist » du schéma. L'extracteur dispose d'une fonction (layout versus schématique) qui permet de comparer ces deux netlists pour valider la procédure de placement et routage. La section trois du fichier présente les interfaces et les capacités parasites internes des portes utilisées dans le circuit. Nous allons utiliser les interfaces afin d'identifier les nœuds du dessin des masques.

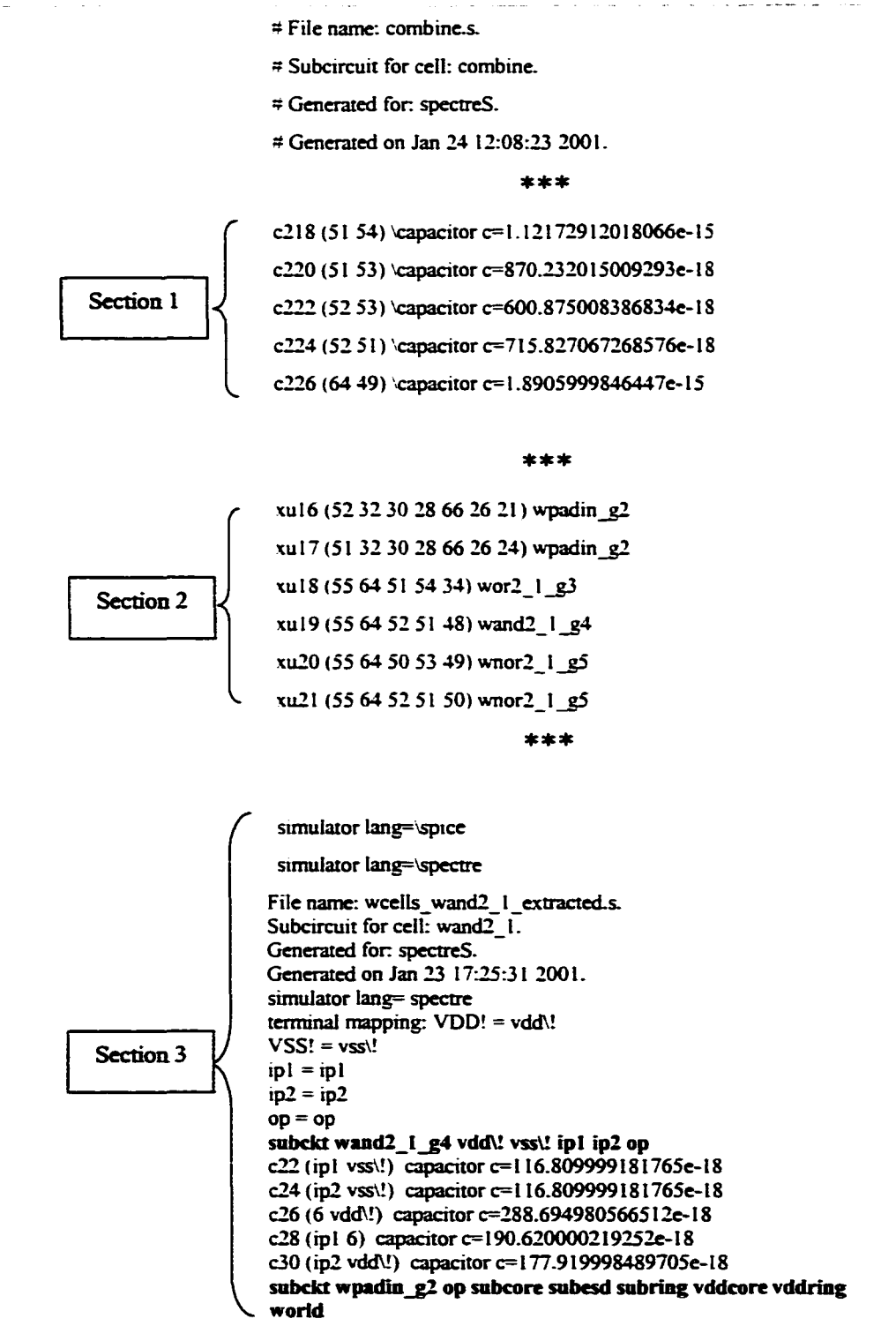


Figure 9 Structure du fichier contenant les informations sur le dessin des masques

Pour une explication plus détaillée des trois sections, voir l'annexe 4. Nous allons présenter maintenant le processus d'automatisation de l'extraction.

2.2.1 L'automatisation du processus d'extraction

L'automatisation du processus de l'extraction des capacités parasites est réalisée par un programme écrit en langage C. Ce programme lit le fichier généré par l'extracteur (figure 9) et génère à son tour plusieurs fichiers, chacun représentant des informations utiles dans notre étude.

Pour bien expliquer la procédure d'automatisation de l'extraction des capacités parasites de routage, nous allons tout d'abord présenter l'organisation structurelle des données utilisées dans le programme, pour ensuite présenter l'algorithme haut niveau du programme.

2.2.1.1. Organisation structurelle des données

La création d'un ensemble de structures de données permet de sauvegarder les différentes informations extraites à partir du fichier source, et de les traiter par la suite. On rappelle que le fichier source contient les capacités parasites de routage, le « netlist » du dessin des masques du circuit et les interfaces de la librairie cible.

➢ Structure 1

On associe une structure de données pour chaque capacité parasite. La définition de la structure est donnée comme suit :

Nom : capacité.

```
structure
{
  char nom[25];
  int ref1,ref2;
  float valeur;
```

Figure 10 Structure de données pour une capacité parasite

L'exemple suivant montre le lien entre une ligne texte de la section 1 du fichier (figure 9) représentant une capacité parasite, et la structure de données définie ci-avant :

➢ Exemple :

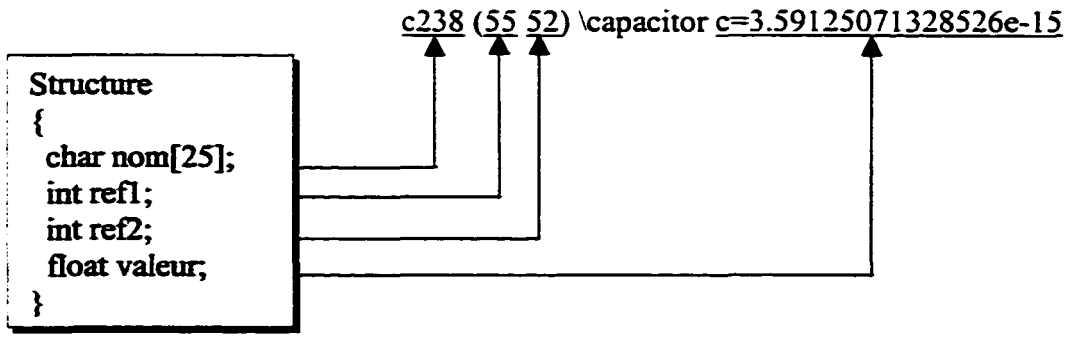


Figure 11 ligne texte versus structure de données

Dans le programme d'automatisation, nous allons créer un vecteur de structures, qui sera alloué dynamiquement pour sauvegarder temporairement les données concernant chaque capacité parasite de routage.

- Structure 2 :

  Nom : nœud.

```
structure
{
char nom[5]
char origine[15], destination[25]
}
```

Figure 12 Structure de données pour un noeud

On définit une structure de données pour les nœuds du circuit figure 12. Cette structure contient trois champs. Le premier champs emmagasine le numéro du nœud, le deuxième emmagasine la ou les cellule(s) d'origine(s), et le troisième champs, la ou les cellule(s) de destination. La figure 13 présente un exemple.

➢ Exemple :

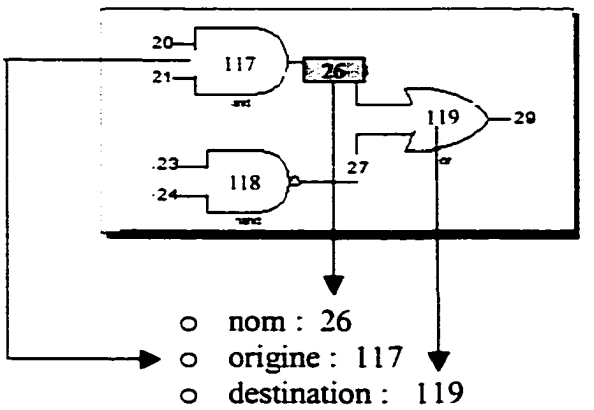


Figure 13 Exemple de structure de données pour un noeud

➢ Structure 3 :

Comme le montre la figure 14, cette structure permet de sauvegarder les informations des portes logiques utilisées dans le circuit. L'extraction du dessin des masques par l'outil « analog artist » de Cadence donne pour chaque porte logique les informations suivantes :

Nom : cellule.

```
Structure
{
    char name[20];
    char xname[20];
    char wname[20];
```

Figure 14 Structure de données pour une porte logique

Voici un exemple :

- xu18 (54 56 117 118 210) **wor2_1_g3** (provient de la section 2 du fichier d'extraction).

    - name ---------- **wor2_1**
    - xname ---------- xu18
    - wname ---------- **wor2_1_g3**
    - interface_cell ----- (54 56 117 118 210)

2.2.1.2. Description algorithmique

L'étape algorithmique consiste à créer le corps du programme, généralement un ensemble de fonctions coordonnées par la fonction principale (main), dont l'objectif est d'exploiter les informations du fichier d'entrée généré par « analog artist » afin d'extraire les données utiles pour notre étude. Ces données seront emmagasinées dans d'autres fichiers sous un format approprié.

La figure 15 représente le schéma bloc global du programme d'automatisation. Le fichier d'entrée donne les trois sections à traiter. Après le traitement de chaque section, le programme passe à l'étape de traitement final qui consiste à générer les fichiers de sorties contenant les données sous un format approprié. Pour plus d'informations sur les fichiers de sortie, nous présentons en détails à l'annexe 5 les différents fichiers de sortie d'un circuit simple.

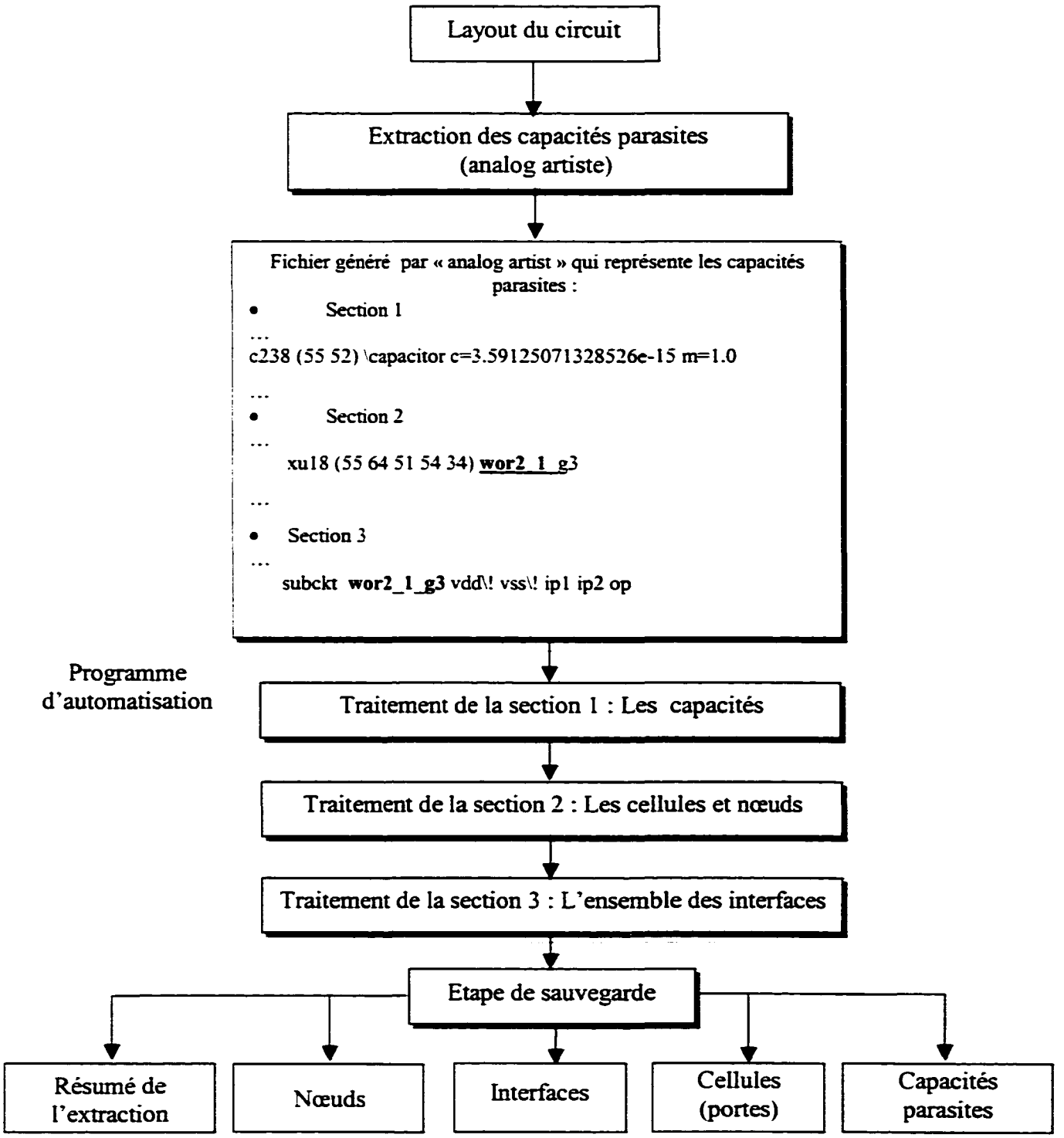


Figure 15 Plan du programme d'automatisation du processus d'extraction des capacités parasites de routage

2.2.2 Résultats et statistiques de l'extraction

La figure 16 montre le dessin des masques d'un multiplicateur 4x4. Les petits rectangles blancs sur le dessin des masques représentent les capacités parasites de routage.

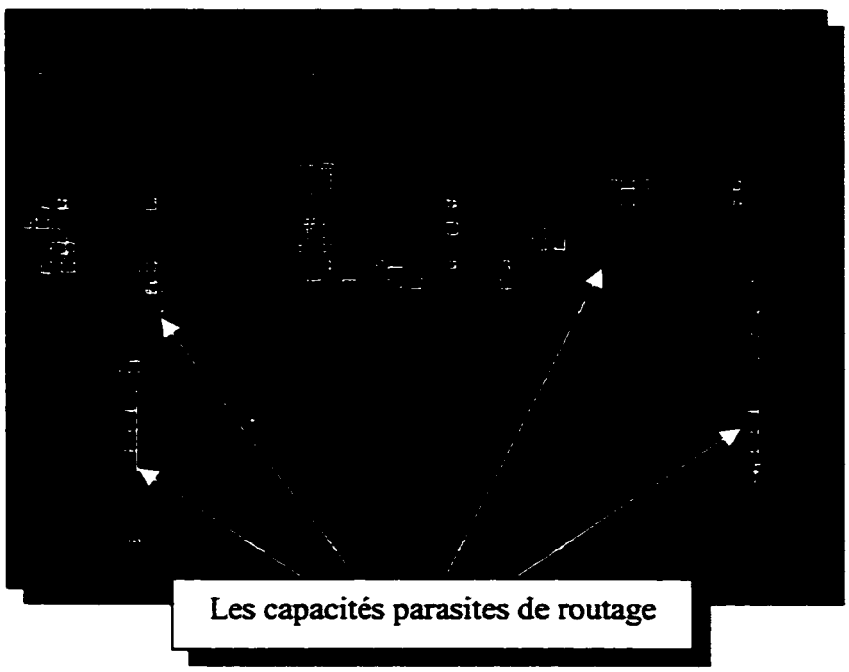


Figure 16 Capacités parasites de routage

Le tableau V présente les résultats de l'extraction des informations à partir des dessins des masques de différents circuits : multiplicateur 4x4, des filtres FIR à 4, 25 et 50 étages. La figure 17 donne un graphique représentant les capacités parasites de routage en fonction du nombre de nœuds.

Tableau V

Résultats de l'extraction des capacités parasites

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Nombre de capacités	Surface du circuit (um^2)
Multiplicateur 4bits	94	117	8	8	600	$2x10^5$
FIR 4 étages	901	986	42	8	9 027	$14x10^5$
FIR 7 étages	1 641	1 772	66	8	14 599	$24x10^5$
FIR 15 étages	3 521	3 759	130	8	46 269	$40x10^5$
FIR 25 étages	6 468	7 962	210	8	102 098	$51x10^5$
FIR 50 étages	13 127	13 855	410	8	293 415	$89x10^5$

NB : Pour plus d'informations sur les résultats de l'extraction, voir l'annexe 4.

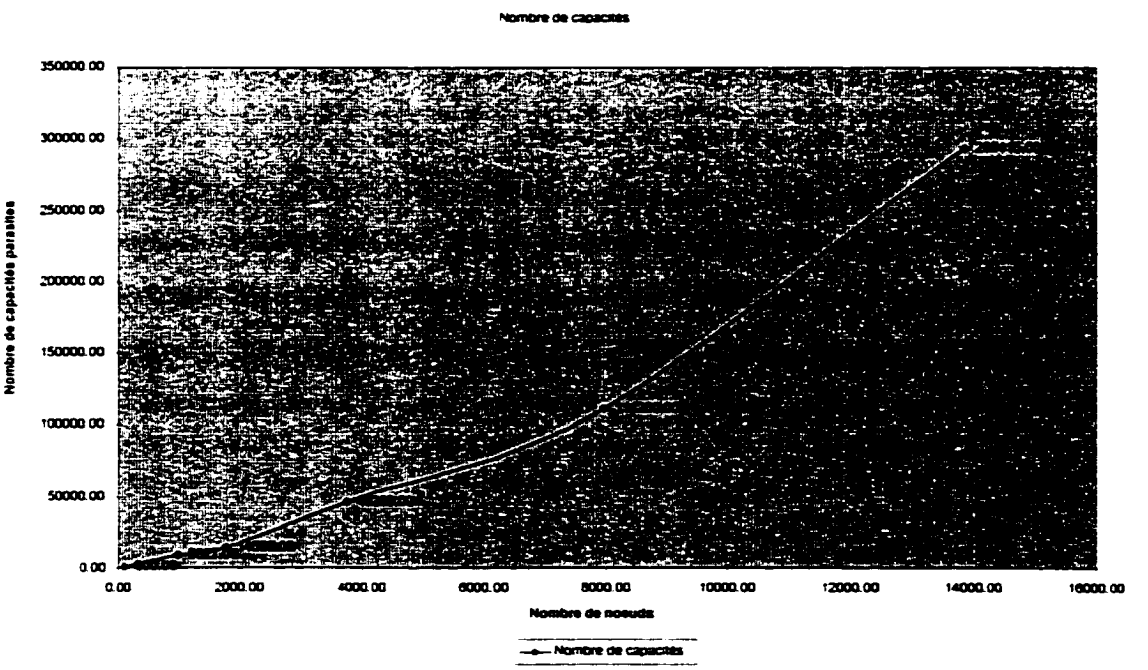


Figure 17 Capacités parasites de routage en fonction du nombre de nœuds

2.3 Processus de réduction des sites physiques potentiels en consultant les capacités parasites

Rappelons que notre objectif actuel est d'exploiter les capacités parasites de routage pour réduire le nombre de sites physiques pour le diagnostic. Donc, au lieu de considérer toutes les combinaisons possibles entre les nœuds du circuit, nous allons considérer seulement les combinaisons des nœuds susceptibles d'être court-circuités.

Pour un type de pannes donné, nous allons définir la liste des nœuds candidats. Cette liste représente tous les nœuds du circuit pouvant provoquer ce type de pannes. Par exemple, pour un court-circuit entre les sorties de deux portes Non-Et, nous allons définir une liste qui contient toutes les combinaisons possibles entre les sorties de toutes les portes Non-Et du circuit. À partir de cette liste, nous allons conserver juste les nœuds qui forment des capacités parasites de routage. Cette décision sera prise à la suite d'une consultation du fichier des capacités parasites.

La figure 18 représente le plan de réduction des sites physiques du circuit en consultant le fichier des capacités parasites de routage. On rappelle que ce fichier a été généré par le programme d'automatisation de l'extraction des capacités parasites de routage (figure 15).

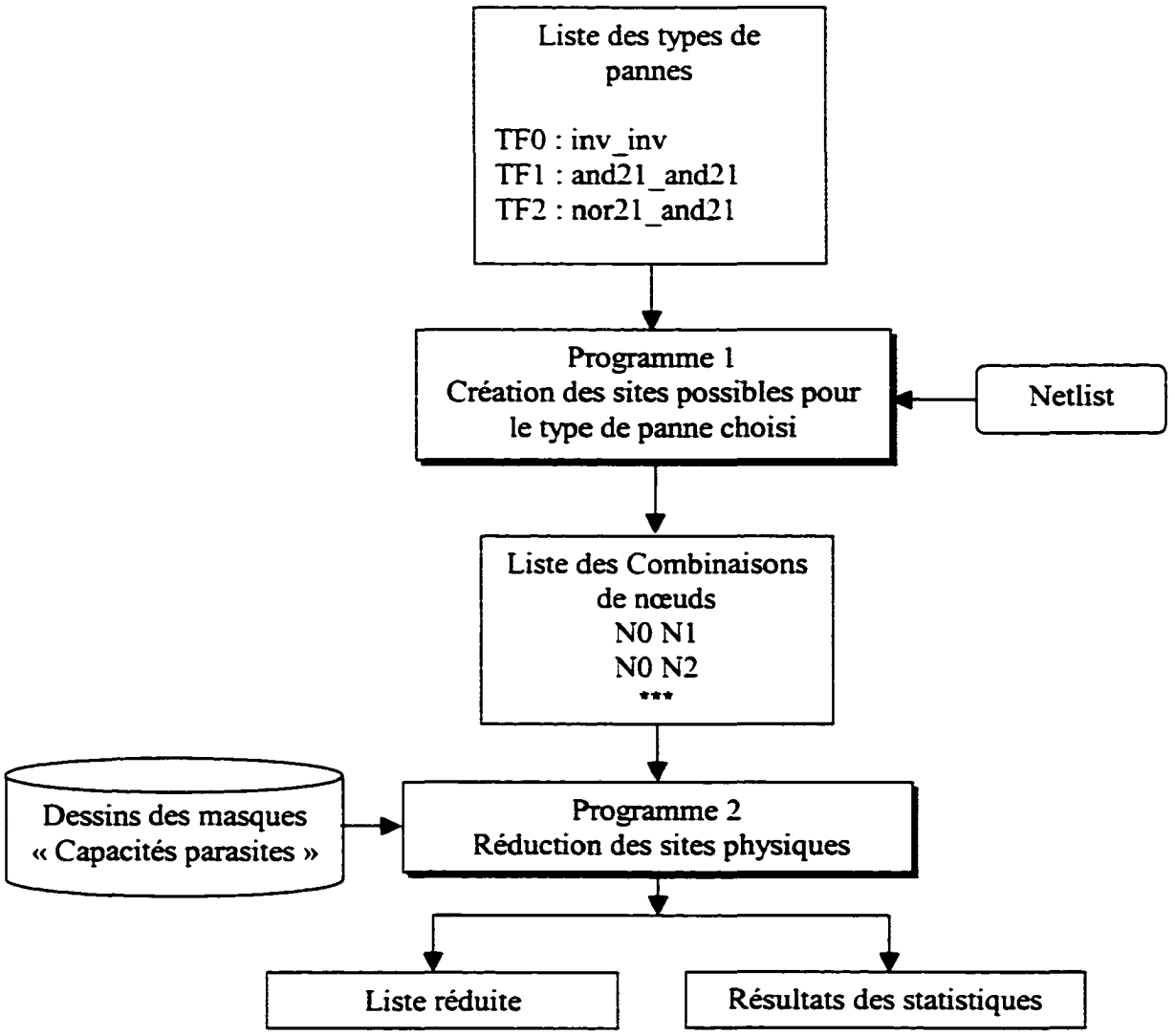


Figure 18 Plan de réduction des sites physiques en consultant les capacités parasites de routage

Le processus de réduction commence par la création des sites possibles pour un type de panne donné. Ensuite, cette liste sera passée à un autre programme qui va la réduire en consultant le fichier des capacités parasites de routage.

Comme le montre la figure 19, le programme extrait du netlist, toutes les combinaisons possibles de nœuds correspondant à un type de panne fixé au départ.

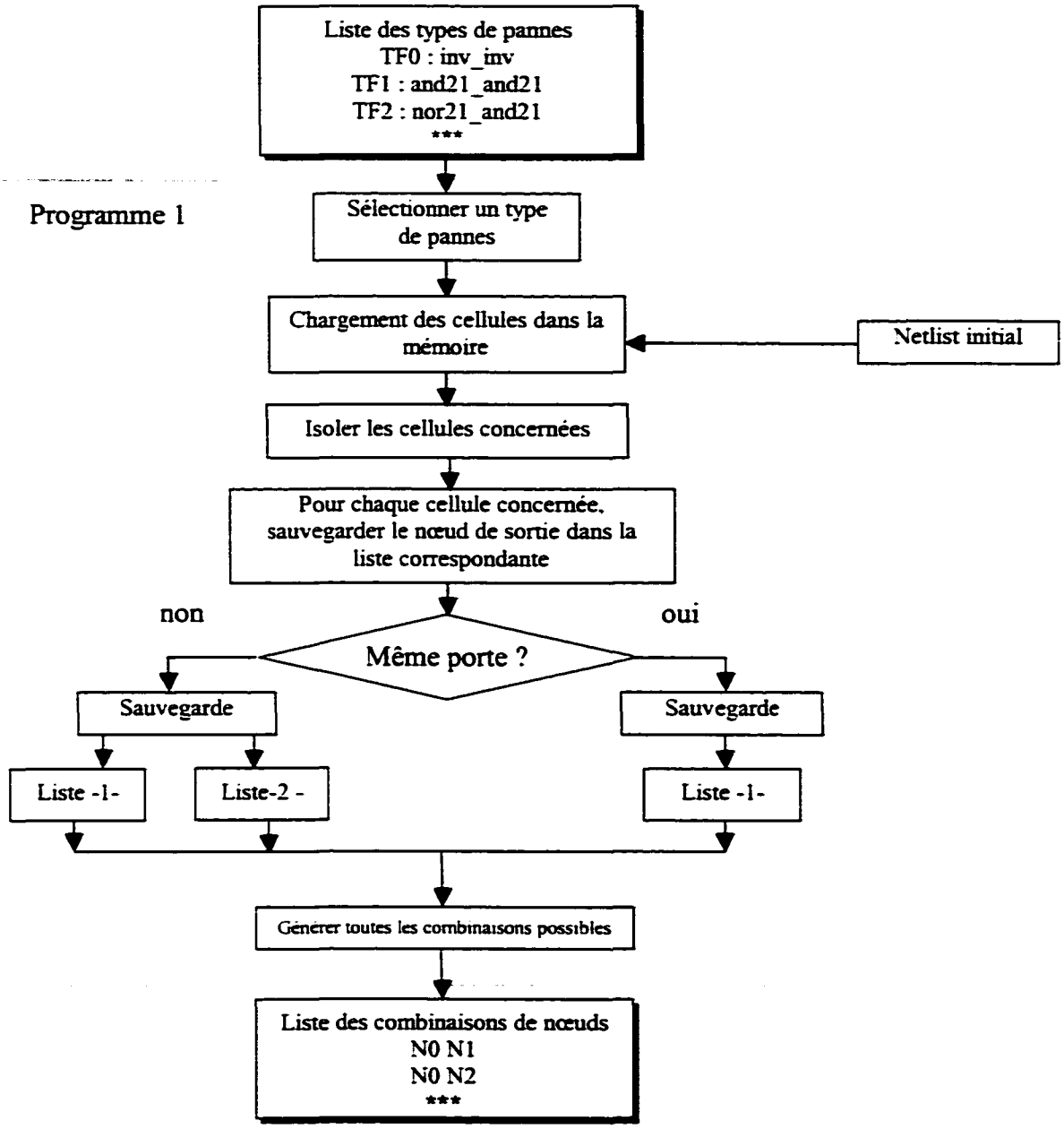


Figure 19 Organigramme du programme 1 : Génération de la liste des sites candidats

Tel que mentionné, la liste des combinaisons de nœuds candidats pour le type de panne fixé au départ est par la suite passée au programme 2 effectuant la réduction de cette liste en se basant sur les capacités parasites de routage. La figure 20 montre comment le processus de réduction se déroule.

L'algorithme de réduction est simple. Tout d'abord, le programme choisit une paire de nœuds à partir de la liste des sites candidats pour un type de panne. L'étape suivante consiste à consulter le fichier des capacités afin de vérifier l'existence d'une capacité parasite de routage entre les nœuds choisis. Si une capacité parasite existe entre les deux nœuds en cours, on ajoute cette paire à la liste des sites physiques potentiels, qu'on appelle la liste réduite. Dans le cas où il n'existe pas de capacité parasite de routage entre les nœuds du couple choisi, on rejette cette combinaison, et on passe à la paire suivante.

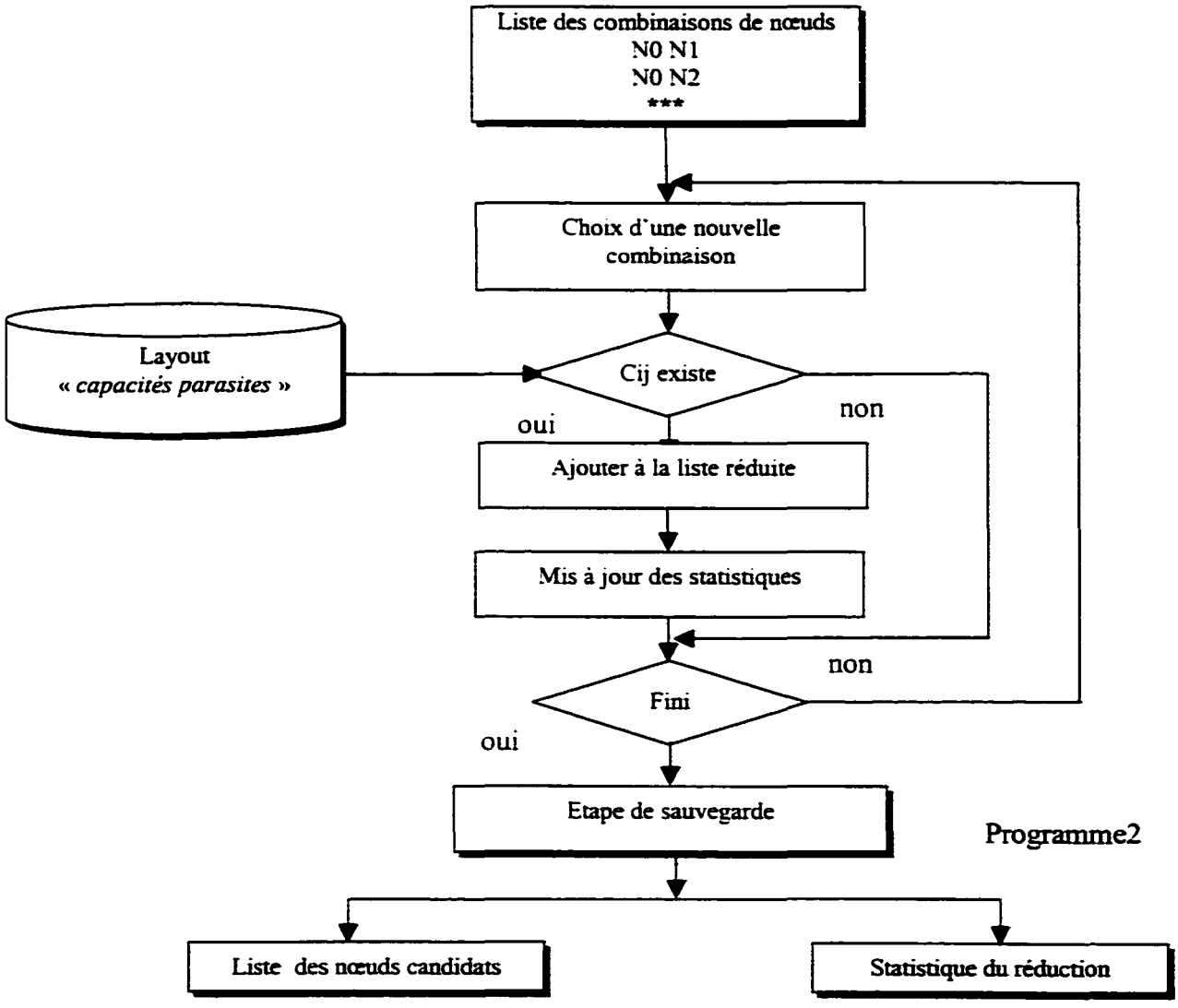


Figure 20 Organigramme du programme 2 : Réduction de la liste des sites candidats en consultant le fichier des capacités parasites

2.4 Résultats de la réduction

Avant de présenter les résultats de réduction des sites physiques potentiels, mentionnons qu'il y a deux notations pour les nœuds, puisque deux outils différents sont utilisés (*Cadence et Synopsys*). Le chapitre 6 donne plus de détails sur cette différence.

La figure 21 présente les résultats de réduction des couples candidats à une panne de type pont entre les sorties de deux portes Non-Et à deux entrées pour un circuit simple.

Le fichier statistic.yas généré par le programme de réduction

——— la phase réduction de la liste des combinaisons ———

Type de panne : cc entre wnand2_1 <—> wnand2_1

——— Couple de noeuds Capacités ———

..n101.._..62.. <-> ..n44.._..50.. * capa = 2.732037e-16
..n51.._..65.. <-> ..n65.._..25.. * capa = 6.853140e-16
..n51.._..65.. <-> ..n66.._..29.. * capa = 2.534561e-16
..n55.._..97.. <-> ..n66.._..29.. * capa = 6.052765e-16

——— Résumé ———

Nombre total de sites : 60
Nombre réduit de sites : 4

60 combinaisons
(-*-*- Algorithme de réduction en consultant le « layout » -*-*-)
4 combinaisons retenues

6.67 % des combinaisons sont retenues après la consultation des dessins des masques
Le pourcentage de reduction est de 93.33%

Figure 21 Résultats de la réduction des sites candidats

Avant réduction, il y a 60 sites candidats pour le type de panne (court-circuit entre les sorties de deux portes Non-Et). Après la procédure de réduction seulement 6.67%, i.e. 4 sites sont conservés. Le programme de réduction génère un fichier où figure la liste des sites physiques potentiels.

Les résultats de la réduction des sites physiques pour d'autres circuits sont présentés au tableau VI.

Tableau VI

Résultats de l'extraction et de la réduction des sites physiques

Circuit informations	Multiplicateur 4bits	FIR 4 étages	FIR 7 étages	FIR 15 étages	FIR 25 étages	FIR 50 étages
C : nombre de cellules	94	901	1641	3521	6468	13127
N : nombre de nœuds	117	986	1772	3759	7262	13855
T : nombre total de sites	990	72 771	271 598	938 904	3 128 751	8 889 436
R : nombre réduit de sites	58	649	723	2101	2815	4167
% de réduction des sites ((T-R)/T)*100%	94.14	99.10	99.73	99.77	99.91	99.95
% des sites retenus (R/T)*100%	5.86	0.90	0.27	0.22	0.09	0.05
R/C =	0.61	0.72	0.44	0.52	0.43	0.31

Ces résultats montrent que le pourcentage de réduction des sites dans le cas des exemples choisis est toujours supérieur à 94%, et qu'il peut atteindre 99.99% dans le cas des circuits plus complexes. On remarque également que le nombre de nœuds est légèrement supérieur au nombre de cellules. Ceci est dû au fait que certaines cellules

ont été décomposées en portes simples. L'annexe 8 présente des résultats plus détaillés de la réduction des couples de nœuds candidats après la consultation des capacités parasites de routage.

La figure 22 montre la comparaison graphique entre le nombre total et réduit de sites en fonction du nombre de cellules. L'échelle utilisée dans le graphique est une échelle logarithmique. On constate que le degré de réduction est très important, ce degré augmente en fonction de la taille du circuit

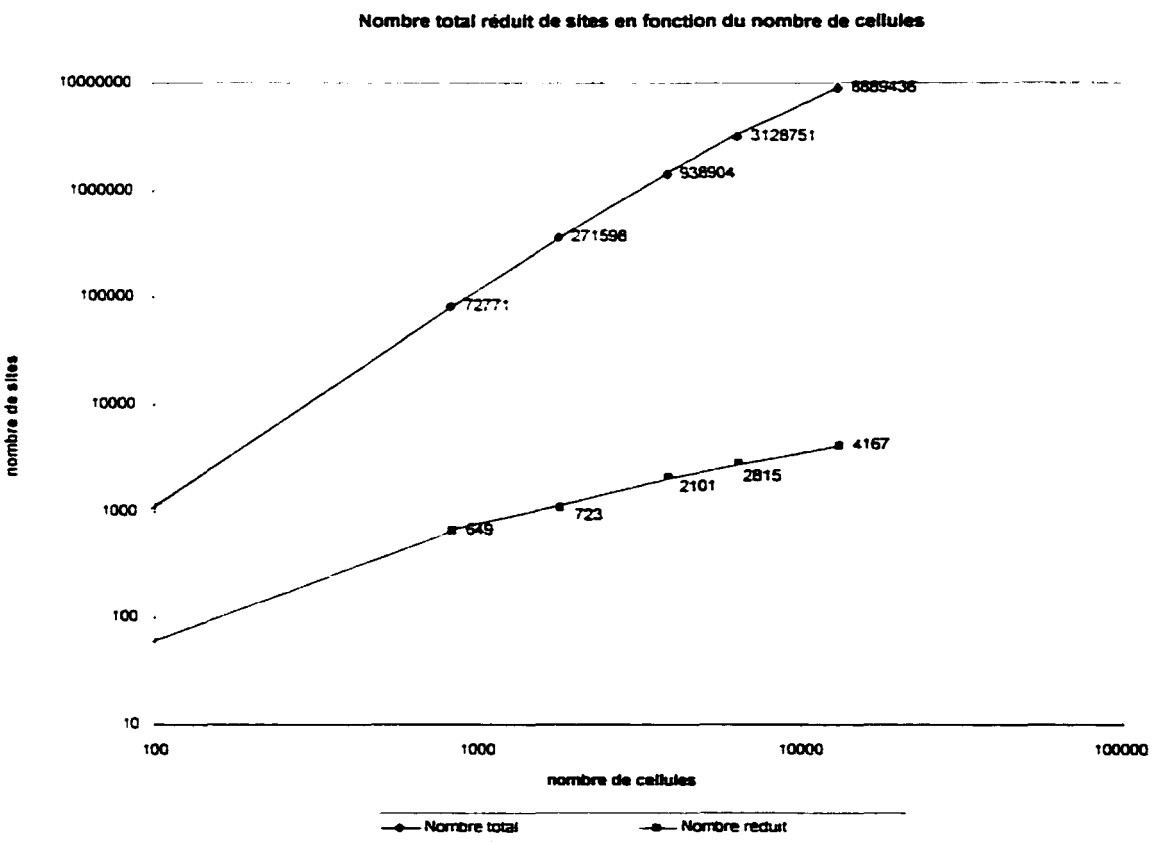


Figure 22 Nombre total et réduit de sites en fonction du nombre de cellules

En appliquant une régression linéaire sur chacune des deux courbes (en utilisant le log du nombre de cellules et le log du nombre de sites), nous obtenons les relations suivantes :

$$R = 0.63*C^{0.69} \quad (2\text{-}1)$$

$$T = 0.43*C^{1.79} \quad (2\text{-}2)$$

Où R est le nombre réduit de sites, T le nombre total des sites et C le nombre de cellules. Ces relations montrent que R augmente moins rapidement que C, i.e. un peu plus rapidement que $C^{1/2}$ et que T augmente presqu'avec le carré de C.

2.5. Conclusion

L'objectif principal de la réduction de la liste des sites physiques est d'accélérer le processus de diagnostic. Rappelons que la phase de localisation des pannes nécessite la simulation de chaque site possible dans le circuit. L'utilisation des capacités parasites de routage permet donc de réduire de manière significative le nombre de sites à considérer lors de cette phase.

Tel que mentionné précédemment, nous désirons analyser une autre technique de réduction du nombre de sites, à savoir l'utilisation des sorties (logiques) erronées lors du test du circuit. Pour atteindre cet objectif nous allons tout d'abord explorer au chapitre suivant la simulation des pannes nécessaires pour émuler l'appareil de test. Au chapitre 4, nous allons présenter comment ces sorties erronées seront utilisées pour la réduction désirée.

CHAPITRE 3

MODELISATION ET SIMULATION DE PANNES

Nous avons présenté dans le chapitre précédant les résultats de réduction des sites physiques, en exploitant les capacités parasites de routage. Nous allons maintenant explorer la possibilité de réduire le nombre de sites en utilisant le résultat des tests logiques. Cette exploration sera effectuée en deux temps. Tout d'abord nous allons nous intéresser à la modélisation et à la simulation de pannes ce qui est l'objet de ce chapitre. C'est une étape préparatoire pour le prochain chapitre dans lequel une méthode de réduction du « netlist » partant du modèle de faute sera présentée avec quelques résultats.

3.1 Introduction

Avant de parler de modèles de pannes, il est opportun de rappeler la distinction entre une défectuosité et une panne. Selon la classification d'Avizienis [7], il existe plusieurs niveaux d'abstraction : physique, logique, informationnel et externe. Une défectuosité est associée au niveau physique, alors qu'une panne est associée au niveau logique. Une défectuosité ne mène pas nécessairement à une panne, et une panne n'est pas nécessairement causée par une défectuosité.

Dans le cadre de ce projet, les défectuosités qui nous intéressent sont celles causant des court-circuits entre deux ou plusieurs lignes. Nous ciblons ce type de défectuosités car elles sont dominantes dans les technologies CMOS [8]. De telles défectuosités modifient le comportement d'un réseau logique. Pour prédire le comportement d'un circuit dans la présence des défectuosités, des simulations au niveau circuit (transistor) seront effectuées en tenant en compte des différents types de pannes possibles. Des modèles de pannes (niveau logique) seront extraits des résultats de ces simulations,

effectuées à l'aide du logiciel Hspice et basées sur le modèle au niveau transistor des portes contenues dans une librairie de cellules d'une technologie CMOS 0.35 um.

3.2 Simulations de panne à l'aide de Hspice

3.2.1 Pannes ciblées

Le tableau VII rappelle les types de pannes ciblées dans le cadre de ce projet (voir tableau IV). Rappelons également que les types de pannes sont des court-circuits entre une paire de nœuds constituant les sorties de deux portes logiques de même type ou de type différent. Par exemple, le type **TF5 NAND2-NAND3** est un court-circuit entre la sortie d'une porte Non-Et à 2 entrées et une porte Non-Et à 3 entrées.

Le logiciel Hspice sera utilisé pour simuler toutes les combinaisons possibles aux entrées des portes impliquées dans chaque type de pannes, afin de connaître la tension et le niveau logique résultant pour chacune des combinaisons.

Tableau VII

Types de court-circuits considérés

Fault	Description
TF0	INV-INV
TF1	INV-NAND2
TF2	INV-NAND3
TF3	INV-NAND4
TF4	NAND2-NAND2
TF5	NAND2-NAND3
TF6	NAND2-NAND4
TF7	NAND3-NAND3
TF8	NAND3-NAND4
TF9	NAND4-NAND4

3.2.2 Analyse et modélisation des fautes

Considérons l'exemple du type de pannes TF0. Cette panne représente un court-circuit entre les sorties de deux inverseurs. Comme le montre la figure 23, ce court-circuit est modélisé par une résistance Rbr dont la valeur a été fixée à 10 Ω. Les inverseurs en aval de Rbr servent à faciliter l'établissement de la valeur logique correspondante.

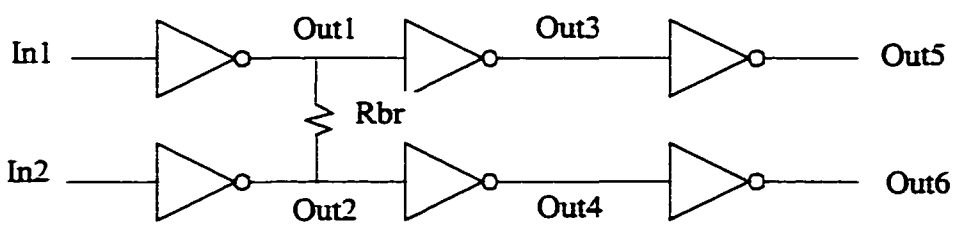


Figure 23 Schéma logique représentant un court-circuit entre deux inverseurs

La figure 24 montre les résultats de la simulation du circuit de la figure 24.

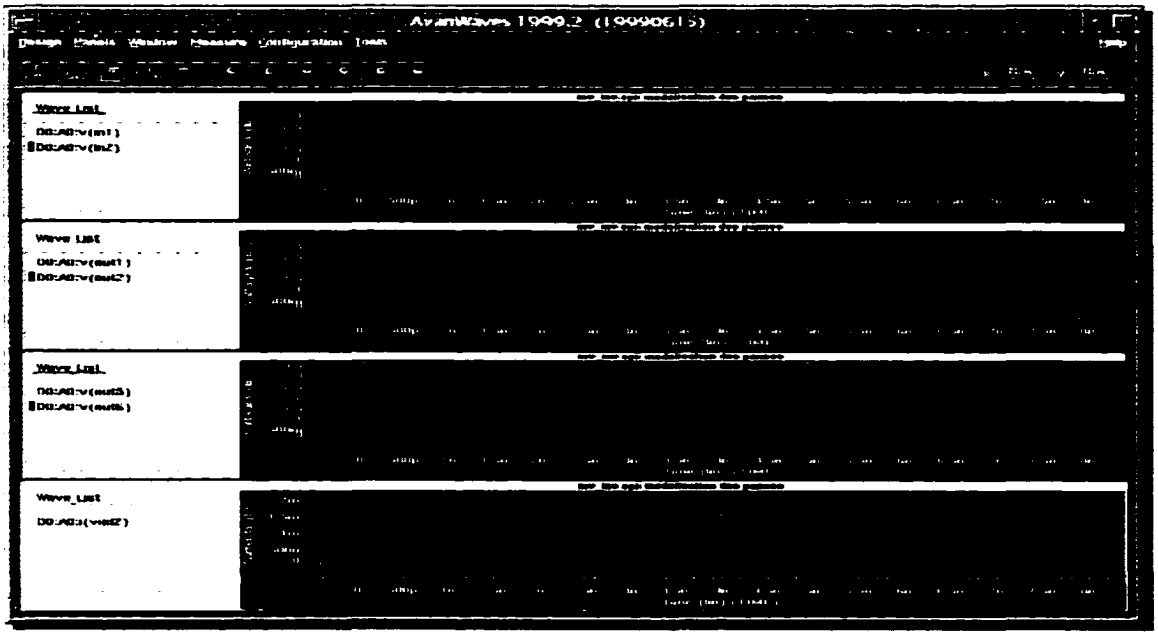


Figure 24 Résultats de la simulation d'un court-circuit entre les sorties de deux inverseur

Ces résultats permettent de construire une table de vérité qui représente les valeurs logiques de sorties en fonction des valeurs logique d'entrées.

Le tableau VIII représente cette table de vérité. On constate que les sorties out5 et out6 (voir figure 24) sont égales (avec Rbr). La valeur finale des nœuds court-circuités dépend de la force relative des transistors activés par les signaux d'entrées. Ainsi, dans cet exemple les transistors de type N de l'inverseur dont l'entrée est à '1' emporte sur le transistor de type P de l'inverseur dont l'entrée est à '0'.

Tableau VIII

Table de vérité pour le type de faute TF0

Entrées		Sorties sans Rbr		Sorties avec Rbr	
In2	In1	Out5 (out1)	Out6 (out2)	Out5 (out1)	Out6 (out2)
0	0	1	1	1	1
0	1	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0

De plus, on remarque que le court-circuit se comporte ici comme une porte ET dont les deux entrées sont respectivement les nœuds out1 et out2 et dont la sortie remplace les 2 mêmes nœuds comme entrées des inverseurs en aval de Rbr. Le tout est illustré à la figure 25.

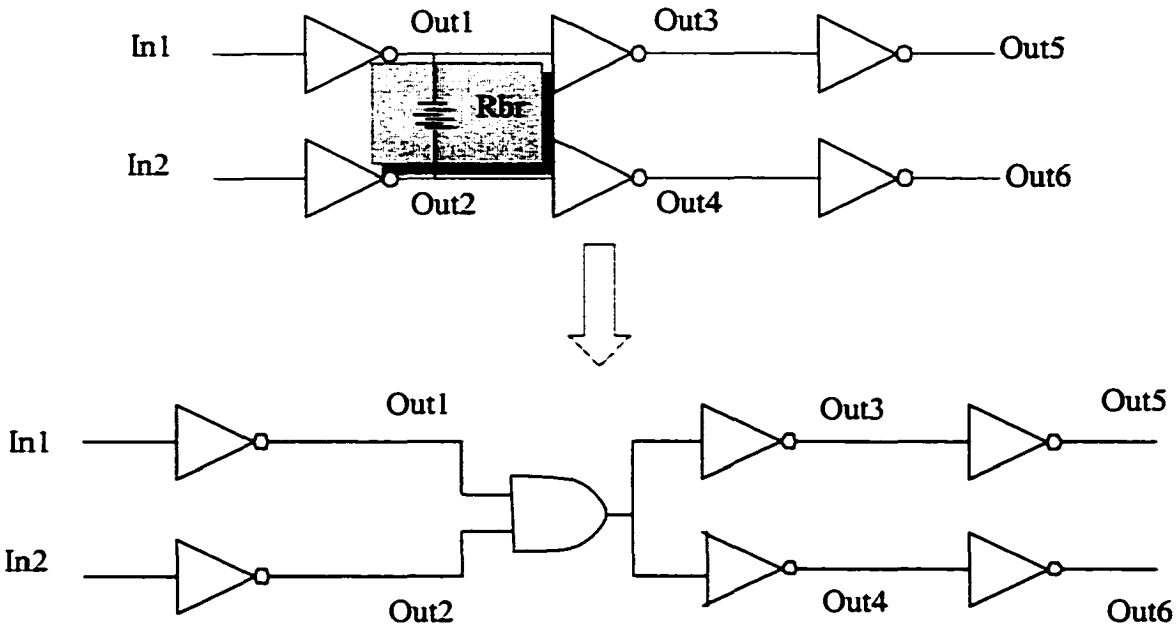


Figure 25 Insertion du modèle dans le circuit

L'annexe 9 présente les résultats de simulation pour les autres types de pannes considérées.

3.3 Utilisation des modèles

Le remplacement de la résistance par une porte logique constitue le résultat de notre modélisation de pannes. Tel que mentionné précédemment, cette modélisation va permettre la simulation de pannes et permet de fournir la valeur des sorties telles qu'on les retrouvait après un test. Nous appelons ce processus l'émulation d'un circuit sous test. Après le remplacement de la défectuosité (Rbr) par son modèle de pannes équivalent (dans notre exemple, une porte Et), la simulation logique est effectuée afin d'identifier les sorties erronées du circuit

Le tableau IX donne la table de vérité suite à une simulation logique sur Verilog-xl. On remarque que les résultats logiques coïncident parfaitement avec les résultats sur Hspice

Tableau IX

Simulation logique après insertion du modèle de pannes

Décimal	In2	In1	Out5	Out6	out
0	0	0	1	1	1
1	0	1	0	0	0
2	1	0	0	0	0
3	1	1	0	0	0

3.4 Conclusion

Ce chapitre portait sur le processus de modélisation de pannes, comme étape préparatoire à la simulation menant à l'identification des sorties erronées. Au chapitre suivant, ces sorties seront exploitées afin de réduire d'avantage la quantité d'information requise pour le diagnostic.

CHAPITRE 4

RÉDUCTION DES SITES PHYSIQUES POTENTIELS EN UTILISANT LES RÉSULTATS DE L'ÉMULATION DU CIRCUIT SOUS TEST

Toujours dans le but de réduire le nombre des sites physiques potentiels, nous allons présenter dans ce chapitre la procédure permettant d'exploiter les résultats logiques erronés des sorties du circuit, issus de son émulation.

4.1 Introduction

Le processus d'émulation du circuit sous test passe par trois phases distinctes. La première est une phase de prétraitement du « netlist ». Elle consiste à insérer le modèle de panne dans le « netlist » et de le réduire par la suite afin d'accélérer la simulation logique. La deuxième phase consiste à simuler le « netlist » réduit contenant la panne, et de déterminer les sorties erronées pour chaque vecteur de test. La troisième et dernière phase a le rôle de réduire la liste des sites physiques potentiels. Pour se faire, nous allons présenter un algorithme dont le rôle est de déterminer l'ensemble des nœuds menant à la sortie erronée. À la fin de ce chapitre, nous allons présenter les résultats de cette réduction.

4.2 Phase de prétraitement du « netlist »

Cette phase consiste à préparer le « netlist » pour la simulation logique en deux étapes. Comme l'indique la figure 26, la première étape a comme objectif l'insertion du modèle de panne dans le « netlist », effectué par le programme dit d'insertion. La deuxième étape de cette phase consiste à réduire le « netlist » contenant la panne. Les deux sections qui suivent présentent les deux étapes en détails avec des exemples.

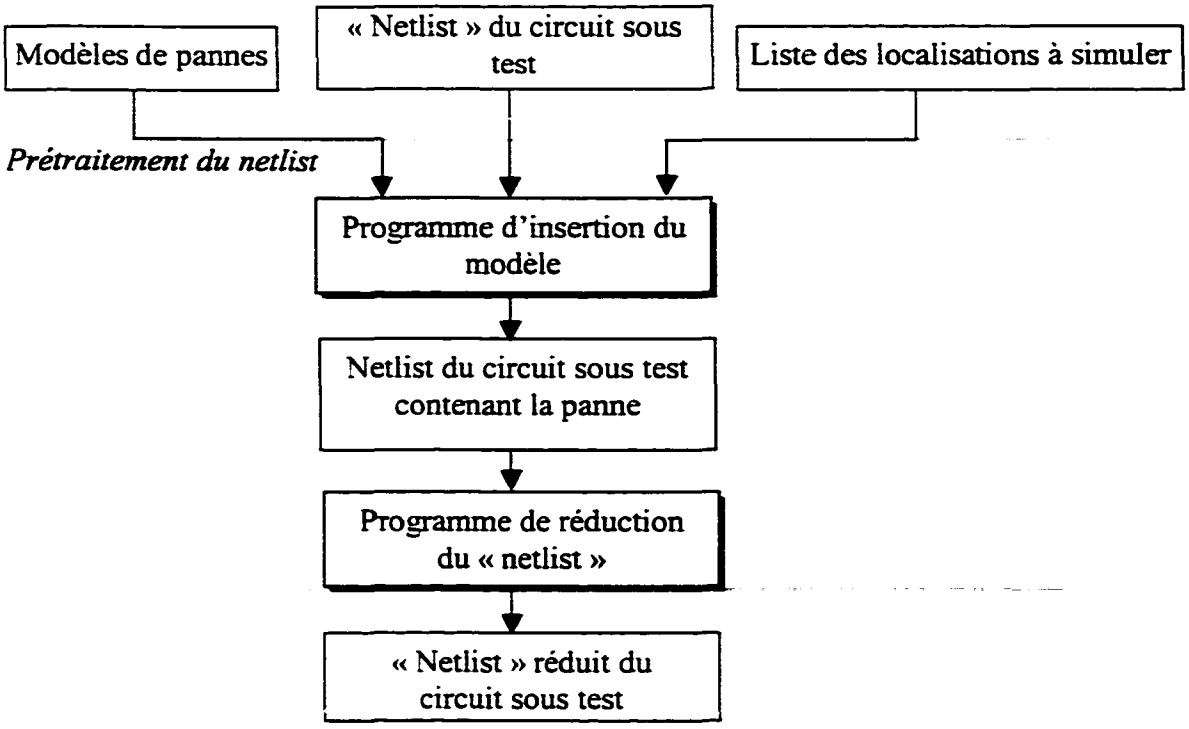


Figure 26 Phase de prétraitement du « netlist »

4.2.1 Insertion du modèle de panne dans le « netlist »

Tel que mentionné à la figure 26, l'insertion du modèle de panne se fait par l'intermédiaire d'un programme, écrit en langage C. Ce programme lit d'abord le fichier contenant le « netlist », et insère par la suite le modèle de panne correspondant à la localisation choisie au hasard. La figure 27 présente un exemple de schéma avant l'insertion du modèle de panne. Les deux lignes X et Y sont en court-circuit. Le modèle de panne correspondant à ce court-circuit va dépendre des types de porte U1 et U2 (voir chapitre 3). La figure 28 représente le nouveau schéma après l'insertion du modèle de panne. On constate ici que la sortie du modèle remplace les nœuds X et Y.

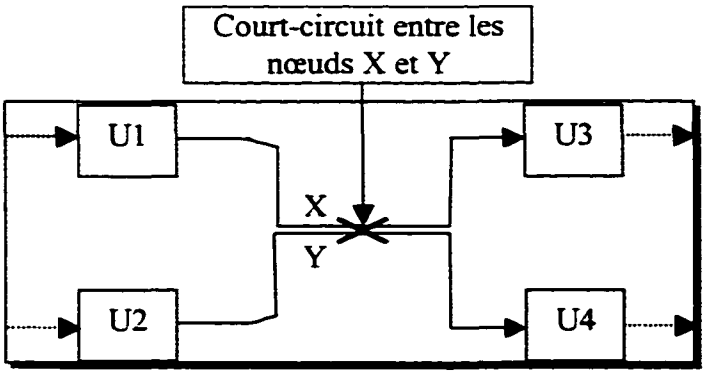


Figure 27 Exemple d'un schéma avant l'insertion du modèle

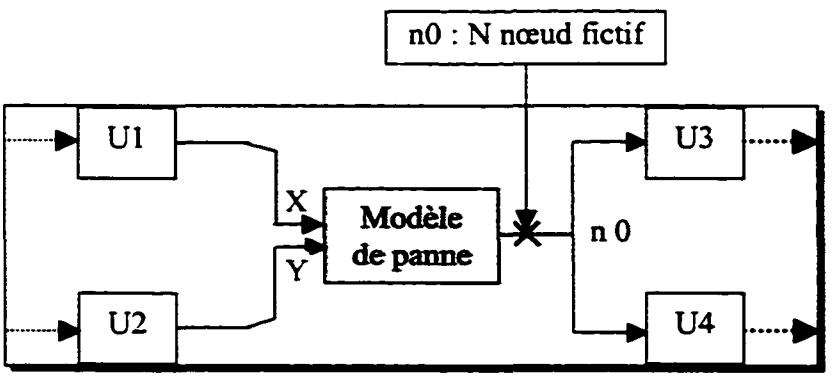


Figure 28 Schéma après l'insertion du modèle de panne

4.2.2 Réduction du netlist

La réduction du netlist a comme objectif la diminution du temps de simulation. Il s'agit ici de modifier le « netlist » afin de ne conserver que les portes logiques nécessaires à l'obtention du résultat désiré. Comme indiqué sur la figure 28, un programme conçu, en langage C, permet de réduire le « netlist » en partant du modèle de panne inséré. Le programme lit le « netlist » de base et effectue une réduction à partir du modèle de panne inséré à l'étape précédente.

Nous avons développé deux modes de réduction, qui seront utilisés dans divers contextes. Donc au départ, le programme affichera un menu permettant de choisir une option de réduction. La figure 29 montre clairement les deux options du menu de la réduction du « netlist ».

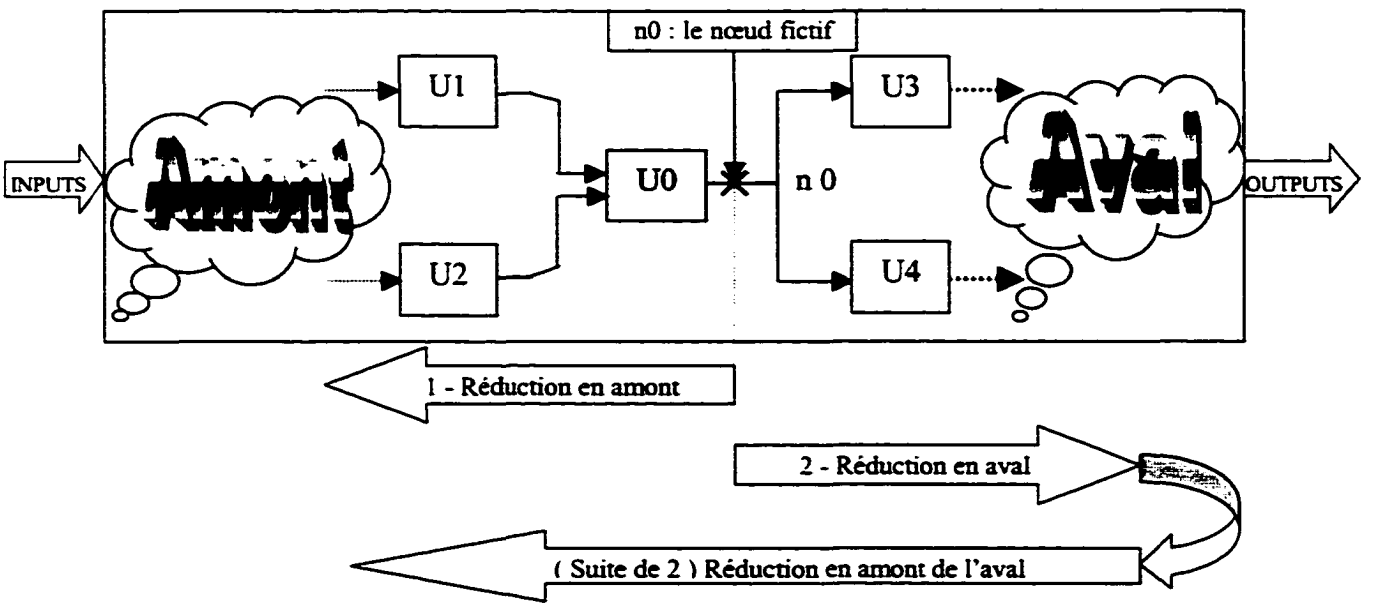


Figure 29 Sens des 2 types de réduction

➢ Réduction en amont :

La première option du menu sera utilisée afin de déterminer l'ensemble des nœuds reliés avec une sortie erronée, nous allons revenir à ce point plus loin dans ce chapitre.

➢ Réduction en amont et en aval et en amont de l'aval :

La deuxième option consiste à réduire le « netlist » et ne garder que les portes logiques liées directement ou indirectement au modèle de panne. C'est le type de réduction qui sera utilisé pour l'identification des sorties erronées.

4.2.3 Résultats de la réduction du « netlist »

Nous allons présenter maintenant quelques résultats de la réduction du « netlist ». Ces résultats sont obtenus en exécutant la deuxième option du programme de réduction. Le tableau X donne les informations sur les « netlists » que nous allons tester.

Tableau X

Information sur les netlists sous test

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (μm^2)
Multiplicateur 4bits	94	117	8	8	$2x10^5$
FIR 4 étages	901	986	42	8	$14x10^5$
FIR 7 étages	1641	1772	66	8	$24x10^5$

Le tableau XI présente le nombre de portes retenues après la réduction des trois « netlists » partant de trois localisations différentes choisies au hasard.

Tableau XI

Résultats de la réduction des netlists pour trois localisations choisies au hasard

		Localisation1 Nand2_Nand2	Localisation2 Inv_Inv	**Localisation3 Inv_Nand2**
Nombre de portes après réduction / Nombre de porte total	Multiplicateur 4bits	80 / 94	88 / 94	**90 / 94**
	Fir 4 étages	725 / 901	702 / 901	**786 / 901**
	Fir 7 étages	**1420 / 1641**	**1434 / 1641**	**1461 / 1641**

À première vue, ces résultats peuvent sembler modestes. Cependant, il s'agit ici de circuits arithmétiques dans lesquels on retrouve une interdépendance importante entre les cellules, notamment en raison de la présence des signaux de retenue.

4.3 Phase de simulation logique sur Verilog-xl

Cette phase consiste à simuler sur l'outil Verilog-xl le « netlist » réduit contenant la panne. La figure 30 montre les entrées/sorties du simulateur Verilog-xl. On doit disposer d'un fichier test contenant les vecteurs de simulation et d'un fichier contenant la librairie « cmosp35 ». Le simulateur génère à son tour un fichier binaire contenant les résultats formatés de la simulation. Pour chaque vecteur de test, il est possible de déterminer les sorties erronées.

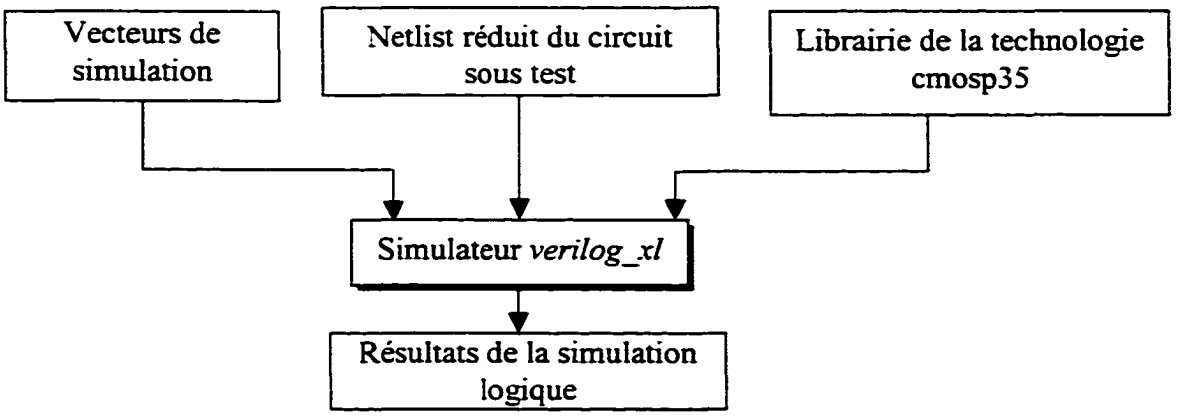


Figure 30 Entrées/sorties du simulateur logique

L'exemple que nous proposons est la simulation du multiplicateur 4 bits contenant une panne de type pont (court-circuit entre les sorties de deux portes Non_Et à deux entrées). Le fichier test contient 5 vecteurs de simulation. Les résultats obtenus sont résumés dans le tableau XII. Les cases marquées par (X) indiquent des résultats erronés à la sortie, celles marquées (-) indiquent que le résultat logique à la sortie est correcte.

Tableau XII

Résultats de la simulation du multiplicateur contenant une faute de type pont entre les sorties de deux portes Non_Et à deux entrées

		Les sorties du multiplicateur 4 Bits							
		1	2	3	4	5	6	7	8
Les vecteurs de test	1	-	-	-	-	-	-	-	-
	2	-	X	-	-	X	-	X	-
	3	X	-	X	-	-	-	-	X
	4	-	-	-	-	-	-	X	-
	5	X	-	X	-	X	-	-	-

4.4 Phase de réduction des sites physiques basée sur les résultats des tests logiques

Cette phase consiste à réduire le nombre des sites physiques en se basant sur les résultats des tests logiques (ou simulation de panne). Dans ce cas-ci, le type et la localisation de la panne sont encore inconnus. La réduction se fera en éliminant tout les paires de nœuds pour lesquelles aucun des nœuds ne se trouve dans la zone potentielle identifiée à partir des sorties erronées.

Un premier exemple est présenté à la figure 31, dans ce cas, une seule sortie est erronée. Un deuxième exemple est montré à la figure 32, où deux sorties sont erronées. Dans tous les cas le programme de réduction fait une union entre les différents ensembles de nœuds trouvés en amont des sorties erronées.

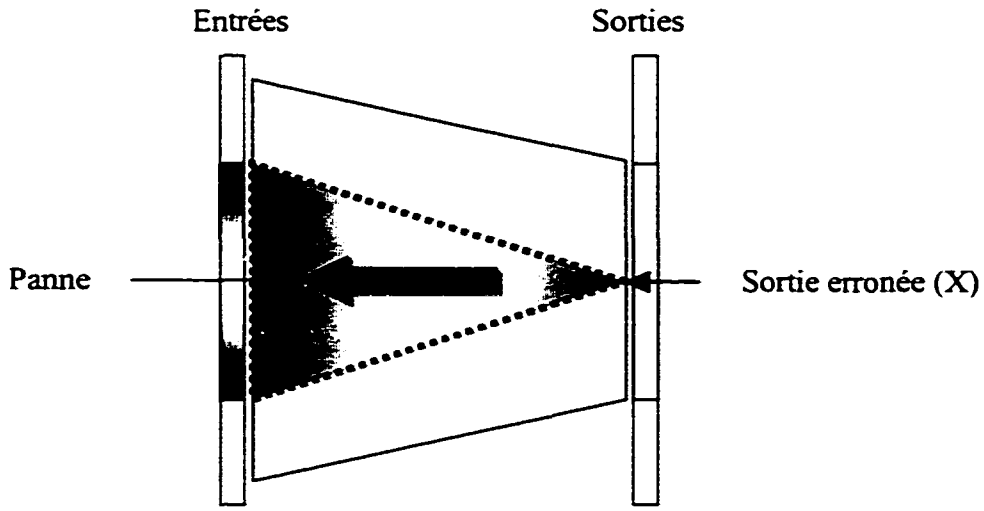


Figure 31 Recherche des nœuds candidats dans le cas d'une seule sortie erronée

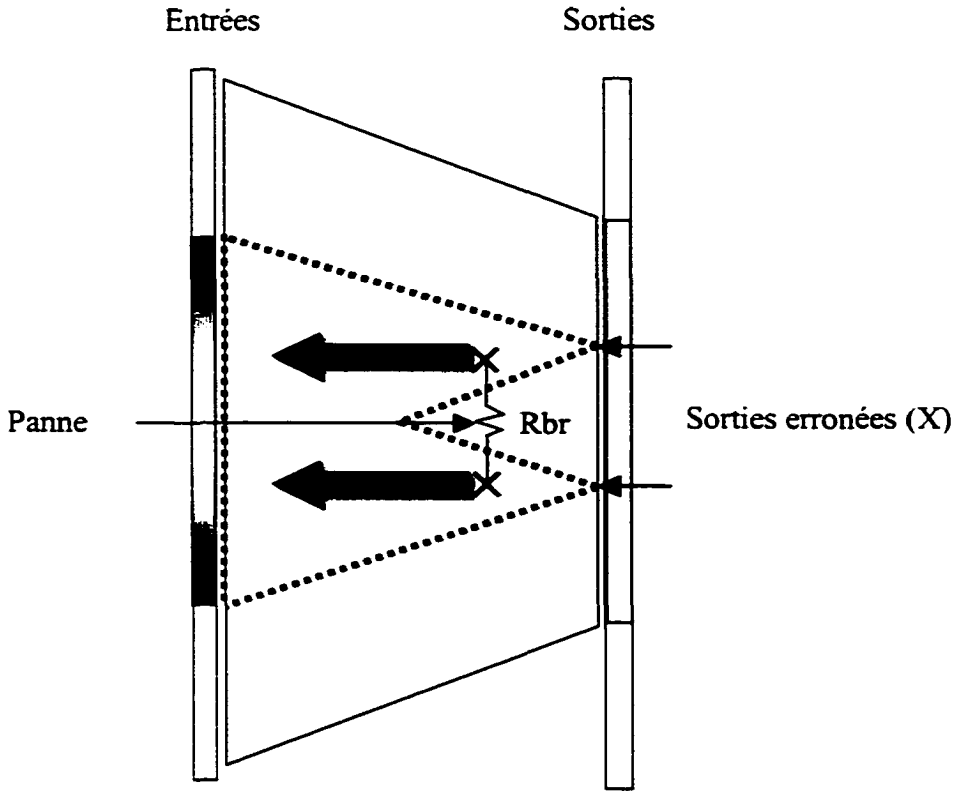


Figure 32 Recherche des nœuds candidats dans le cas deux sorties erronées

Cette technique de réduction peut s'ajouter aux autres déjà mentionnées. En résumé, si toutes ces techniques sont appliquées, une paire de nœuds fera partie de la liste des sites potentiels si :

- les nœuds sont des sorties de portes ciblées par le type de pannes considéré (réduction introduite par le modèle de panne liée au type de portes impliquées),

- une capacité parasite a été détectée entre les deux nœuds (réduction liée à l'utilisation des capacités parasites).

- au moins un des deux nœuds se retrouve dans l'ensemble des nœuds situés en amont des sorties erronées.

Tel que mentionné précédemment, cette liste de sites potentiels sert de base à la phase de localisation de la panne.

4.5 Résultats de la réduction des sites physiques basée sur les sorties erronées

Nous nous intéressons ici à la réduction due aux sorties erronées. Le tableau XIII présente quelques résultats d'expérience de la procédure de réduction sur le multiplicateur 4 bits.

Tableau XIII

Résultats de la réduction des netlists pour trois localisations choisies au hasard

Multiplicateur 4bits	Localisation1 Nand2_Nand2	Localisation2 Inv_Inv	Localisation3 Inv_Nand2
Nombre total des sites	465	55	341
Nombre réduit des sites	355	32	295
Pourcentage des combinaisons retenues	76.34 %	58.18 %	86.51%

On note une réduction intéressante du nombre de sites, en dépit du fait que le bloc utilisé contienne plusieurs « fan out », en raison de sa nature arithmétique. En théorie, si N est le nombre total de nœuds et M le nombre de nœuds en amont des sorties erronées, le pourcentage de réduction est donné par la formule suivante :

$$\text{Pourcentage de réduction} = 100 * \left[1 - \frac{(N * M) - 0.5 * M(M+1)}{0.5 * N(N-1)} \right] \quad (2\text{-}2)$$

À ce point, il serait intéressant d'analyser en profondeur les statistiques de réduction des sites potentiels basée sur les capacités parasites et l'émulation du circuit sous test.

4.6 Résultats de réduction combinée basée sur les capacités parasites et l'émulation du circuit sous test

Le tableau XIV présente les résultats de la réduction des sites physiques potentiels pour un multiplicateur 4 bits. Pour chaque type de panne, on construit un ensemble contenant toutes les paires de nœuds sorties de portes ciblées par le type de pannes considéré (réduction introduite par le modèle de panne liée au type de portes impliquées). L'étape suivante consiste à réduire cet ensemble en ne laissant que les paires de nœuds contenant une capacité parasite. La dernière étape consiste à utiliser les résultats de l'émulation du circuit sous test, donc, l'ensemble final va contenir un ensemble de paires de nœuds, au moins un des deux nœuds de chaque paire de l'ensemble final, doit se retrouver dans l'ensemble des nœuds situés en amont des sorties erronées.

Tableau XIV

Résultats de la réduction des netlists pour trois localisations

Type de faute	Description et nombre de portes		Nombre de couple	Nombre de couples réduits		Réduction finale intersection
				Réduction basée sur les capacités	Réduction basée sur les sorties erronées	
TF0	INV	INV	55	0	-	-
	11	11				
TF1	INV	NAND2	341	21	295	12
	11	31				
TF2	INV	NAND3	33	2	32	2
	11	3				
TF4	NAND2	NAND2	465	25	355	22
	31	31				
TF5	NAND2	NAND3	93	10	88	3
	31	3				
TF7	NAND3	NAND3	3	0	-	-
	3	3				
Total			990	58	822	39

Les résultats du tableau XIV montrent que le nombre des sites potentiels final, obtenu après l'application combinée des deux types de réduction (celle basée sur les capacités et celle basée sur les sorties erronées) est toujours inférieur ou égal au nombre de sites potentiels obtenu en appliquant juste la réduction basée sur les capacités parasites. Dans cet exemple, le pourcentage de réduction est passé de 94.1% à 95.8%.

4.7 Conclusion

Nous avons présenté dans ce chapitre une méthode de réduction des sites physiques en se basant sur les résultats des sorties erronées d'un circuit obtenues à l'aide son émulation. Après les tests effectués sur des circuits différents (le multiplicateur 4 bits, et les filtres FIR à 4 étages et à 7 étages), on constate que cette méthode de réduction peut être utilisée afin de réduire la liste des nœuds candidats à un type de pannes donné. Cependant, par manque de temps, cette technique n'a pu être totalement intégrée dans la dernière version de l'outil. Cette intégration fait partie des travaux à venir. Par conséquent, l'intégration finale qui est l'objet du chapitre 6 ne va tenir en compte que de la méthode de réduction basée sur l'extraction des capacités parasites de routage.

Le prochain chapitre présente des solutions permettant de palier aux limites du simulateur logique utilisé avant le début de ce projet. Une de ces solutions sera mise en application afin que l'outil logiciel puisse traiter des circuits séquentiels.

CHAPITRE 5

REMPLACEMENT DU SIMULATEUR LOGIQUE INTÉGRÉ PAR LE SIMULATEUR VERILOG-XL

Au début de ce projet, le simulateur intégré de l'outil logiciel de diagnostic (diag_total) présentait certaines limitations, dont celle de ne supporter que les circuits logiques purement combinatoires. Or, dans la plupart des applications, les circuits sont séquentiels. Pour palier à ce problème, nous proposons deux solutions, soit améliorer le simulateur intégré de l'outil logiciel pour qu'il supporte les circuits séquentiels, soit utiliser l'outil commercial de simulation tel Verilog-xl (Cadence) qui supporte les deux types de circuits, combinatoires et séquentiels.

Le présent chapitre présente dans un premier temps le simulateur logique intégré, ainsi que l'émulateur (I_{DDQ}) qui l'accompagne. Après cette présentation, nous allons discuter plus en détail des limitations du simulateur. Ensuite, nous allons proposer des solutions pour ce type de limitations. À la fin de ce chapitre, nous allons mettre en place et expliquer la procédure d'utilisation de la solution choisie.

5.1 Présentation du simulateur logique intégré et de l'émulateur du courant I_{DDQ}

Ces deux éléments permettent d'estimer le courant de consommation (I_{DDQ}) causé par une panne de type court-circuit entre deux nœuds donnés. Cette estimation sera utilisée tout d'abord pour émuler les valeurs de courant fournies par un testeur (dans le cas où celui-ci ne serait pas disponible). Ensuite, cette estimation sera utilisée dans le cadre de la phase de localisation, pour comparer les valeurs du courant mesurées (ou émulées) avec celles anticipées pour chacun des sites de pannes potentiels.

Ainsi, après le choix d'une paire de nœuds constituant un site potentiel d'un court-circuit, nous démarrons la simulation logique (voir la figure 33) qui consiste à injecter

une série de vecteurs pseudo-aléatoires sur les nœuds d'entrées du circuit. La représentation du graphe du circuit, telle qu'expliquée à l'annexe 2, permet l'utilisation des algorithmes de récursivité et de parcours de graphe afin de mettre à jour la valeur logique de tous les nœuds du circuit représenté par son netlist. Chaque entité du netlist, donc du graphe, est caractérisée par son type de porte. À l'aide de ce type, le simulateur logique applique une simple table de vérité au(x) sortie(s) de l'entité.

Lorsque la mise à jour des valeurs logiques est complétée pour le vecteur de simulation en application, l'émulateur de courant est invoqué. Celui-ci vérifie tout d'abord s'il y a conflit entre les nœuds court-circuités. Si c'est le cas, il estime la quantité de courant anticipée en accédant à des tables précompilées. L'émulateur peut estimer également la consommation de courant qui est due aux portes situées en aval du court-circuit, toujours sur la base de tables prérecompilées. Ces valeurs permettent d'estimer la quantité totale de courant I_{DDQ} causée par le court-circuit. Après cette estimation, l'émulateur permet l'ajout de bruit gaussien pour émuler les variations paramétriques et l'imprécision des mesures. À la fin de la simulation, le niveau de bruit ajouté est vérifié par le calcul de la variance d' I_{DDQ}.

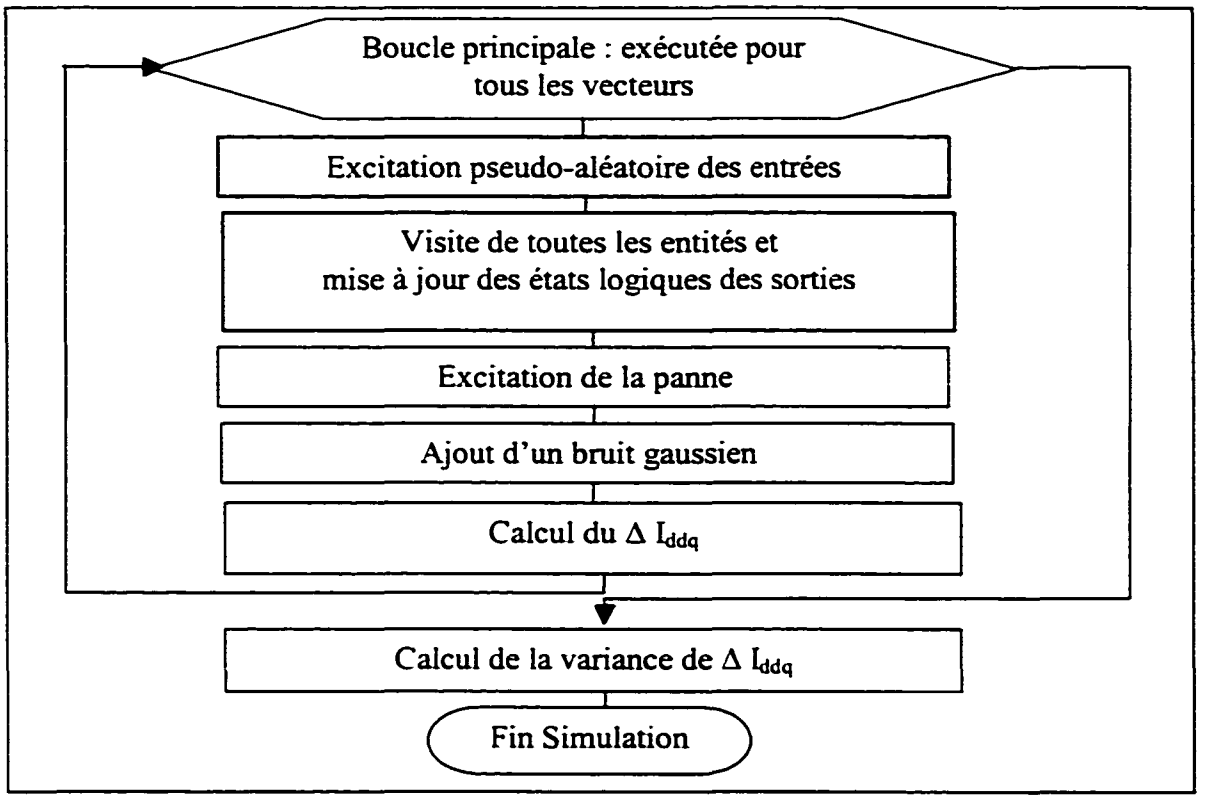


Figure 33 Simulation logique et émulation du courant

5.2 Limitations du simulateur logique du logiciel diag_total

Tel que mentionné précédemment, notre premier problème se situe au fait que le simulateur intégré du programme de diagnostic ne supporte que les circuits purement combinatoires. Il est donc impossible de simuler les circuits séquentiels.

Le deuxième problème découle de la nature algorithmique du bloc de simulation. Le parcours du graphe représentant le « netlist » se fait pour chaque vecteur de simulation. Le but de ce parcours est la mise à jour des valeurs logiques de tous les nœuds du circuit.

Après l'analyse de l'algorithme du parcours du graphe, on constate que la mise à jour des entités et les nœuds du netlist ne se fait pas d'une façon directe. En effet, la mise à jour de la sortie d'une porte logique par exemple, nécessite que les entrées de cette porte soient déjà mises à jour. Si ce n'est pas le cas, la mise à jour aura lieu quand tous les nœuds menant à ce nœud soient déjà mises à jour, donc, dans un cycle futur de parcours du graphe représentant le « netlist ». Alors le programme doit passer par plusieurs itérations afin de mettre à jour toutes les composantes du « netlist ».

5.3 Solutions possibles

Notre principal objectif est de pouvoir simuler les circuits séquentiels. Il est donc nécessaire de trouver une solution adéquate pour ce type de problème.

Deux solutions sont envisagées :

Solution 1 :

- L'utilisation d'un simulateur commercial puissant à l'intérieur du programme d'émulation de la méthode de diagnostic. Nous proposerons dans ce cas le simulateur logique Verilog-xl, qui se caractérise par sa popularité auprès des concepteurs des circuits intégrés, ainsi que par sa compatibilité avec la plupart des outils de conception.

Solution 2 :

- L'amélioration du simulateur déjà existant dans le programme « diag_total ». Dans ce cas, nous devons modifier le code source du bloc de simulation dans le programme d'émulation de la méthode de diagnostic. Cette modification aurait comme objectif l'ajout d'une fonction de mémorisation des états logiques des sorties pour toutes les bascules du circuit après chaque cycle de simulation, et les utiliser pour le cycle suivant.

Notre choix dépendra de plusieurs critères. Vu la complexité croissante des systèmes numériques et les outils qui les accompagnent, la solution choisie doit tenir compte d'une part du nombre croissant de portes logiques sur une même puce, et d'autre part de la compatibilité avec les autres outils de conception. Nous allons dans ce qui suit définir les avantages et les inconvénients de chaque solution.

5.4 Critères de choix et solution choisie

Le tableau XV donne les avantages et les inconvénients pour chacune des deux solutions.

Tableau XV

Avantages et inconvénients de chaque solution

<table>
<tr><td colspan="2">Simulateur Verilog-xl</td><td colspan="2">Simulateur intégré de diag_total</td></tr>
<tr><td>Avantages</td><td>Inconvénients</td><td>Avantages</td><td>Inconvénients</td></tr>
<tr><td>Commercial</td><td rowspan="4">Pas de code source</td><td rowspan="4">Code source disponible</td><td>Version universitaire</td></tr>
<tr><td>Rapide</td><td>Lent</td></tr>
<tr><td>Combinatoire et séquentiel</td><td>Combinatoire seulement</td></tr>
<tr><td>Compatible</td><td>Incompatible</td></tr>
<tr><td colspan="2">Mise en œuvre</td><td colspan="2">Mise en œuvre</td></tr>
<tr><td colspan="2">Faire sortir de manière automatisée les valeurs logiques des nœuds sous forme de matrice</td><td colspan="2">Améliorer le code de la fonction do_logic_sim() dans le logiciel de diagnostic diag_total</td></tr>
</table>

Le simulateur « Verilog-xl » offre plus d'avantages que le simulateur intégré du programme d'émulation. Il nous offre la possibilité de simuler des circuits séquentiels, ce qui n'est pas possible avec la version actuelle du simulateur intégré. En plus, il est compatible avec la plupart des outils de conception. Bref, notre choix s'arrête sur

l'utilisation du simulateur commercial « Verilog-xl » car il satisfait les critères fixés au départ. Dans ce qui suit, nous allons présenter la procédure permettant de sortir de manière automatisée les valeurs logiques des nœuds sous forme de matrice.

5.5 Procédure de simulation sur Verilog-xl

Avant de définir la procédure d'utilisation du simulateur « Verilog-xl », nous allons expliquer d'abord le principe de fonctionnement de ce dernier. Pour simuler le fonctionnement d'un circuit, on doit disposer d'une part de son « netlist » qui définit suivant le format Verilog l'interconnexion entre les portes logiques du circuit, et d'autre part du fichier de simulation qui contient les vecteurs de test et les consignes de simulation.

Rappelons que notre objectif est de remplacer le simulateur intégré du programme d'émulation par le simulateur « Verilog-xl ». Une simulation logique du netlist sur « Verilog-xl » doit se faire avant d'exécuter le programme d'émulation. Suite à cette simulation nous allons obtenir un fichier contenant une matrice sous la forme du tableau XVI.

Tableau XVI

Format de la matrice résultat de la simulation

	nœuds							
vecteurs		0	1	2	3	4	...	m
	0	1	0	1	1	0	...	1
	1	0	0	0	0	1	...	0
	2	1	0	1	0	0	...	1
	3	0	1	0	1	0	...	0
	4	1	0	1	0	1	...	1
	⋮	⋮	⋮	⋮	⋮	⋮	...	⋮
	n	1	0	1	0	1	...	0

Chaque élément de la matrice représente une valeur logique (0/1) correspondant à un couple (vecteur, nœud). Par exemple, la valeur logique du nœud 3 correspondant au vecteur 4 est 0.

La matrice de simulation constitue une entrée pour le programme d'émulation. La conception d'un algorithme permettant de lire et compresser la matrice résultat de la simulation est nécessaire pour remplacer le bloc de simulation déjà existant. Nous allons présenter dans le prochain chapitre l'étape d'intégration finale de la nouvelle technique de simulation basée sur la lecture directe des valeurs logiques des nœuds à partir d'une table de conversion (look up table).

Passons maintenant à la procédure de simulation sur l'outil « Verilog-xl ». Partant d'un netlist de format verilog et d'un fichier de test « test bench », nous pouvons obtenir pour chaque vecteur de simulation les valeurs logiques de tous les nœuds du circuit. Le fichier de test contrôle le déroulement de la simulation. Pour extraire la valeur logique d'un

nœud interne par exemple, on doit connaître le nom de ce nœud, ainsi que le nom du module contenant ce nœud. Un traitement du « netlist » à simuler est nécessaire afin de déterminer ces éléments.

La figure 34 présente la procédure à suivre pour extraire les valeurs logiques des nœuds pour chaque vecteur de simulation. Des fichiers de commandes « scripts » ont été conçus pour le prétraitement du « netlist » et la mise sous format matriciel des résultats de la simulation. Le premier fichier de commande s'appelle « filtre ». Il permet d'extraire à partir du netlist à simuler tous les noms des nœuds à simuler ainsi que le format de sauvegarde de ces nœuds. Il traite aussi le « netlist » de départ en ajoutant les nœuds d'alimentation et de masse à toutes les portes logiques utilisées et génère un autre « netlist » compatible avec l'outil de simulation.

Le deuxième fichier de commande, qui s'appelle « prepare testbench », permet de générer le fichier de test qui sera utilisé pour simuler le « netlist » sur Verilog-xl. Ce fichier de commande génère les vecteurs de test en utilisant un générateur pseudo-aléatoire et utilise comme paramètres de simulation les noms des nœuds et le format de sauvegarde généré par le premier fichier de commande « filtre ». À ce stade, nous pouvons démarrer le simulateur Verilog-xl, qui exécute automatiquement le fichier de test. Les valeurs logiques des nœuds internes sont sauvegardées après l'application de chaque vecteur de test. À la fin de la simulation, nous allons obtenir les résultats sous forme d'un fichier. Ce fichier contient la matrice logique et d'autres informations sur la simulation. Le troisième fichier de commande traite le contenu de ce fichier et génère à son tour un fichier contenant les résultats logiques des nœuds sous forme d'une matrice. Ce dernier fichier sera utilisé dans le programme d'émulation (voir l'annexe 10 pour plus d'information sur la procédure de simulation avec l'outil « Verilog-xl », les fichiers de simulation sont aussi présentés).

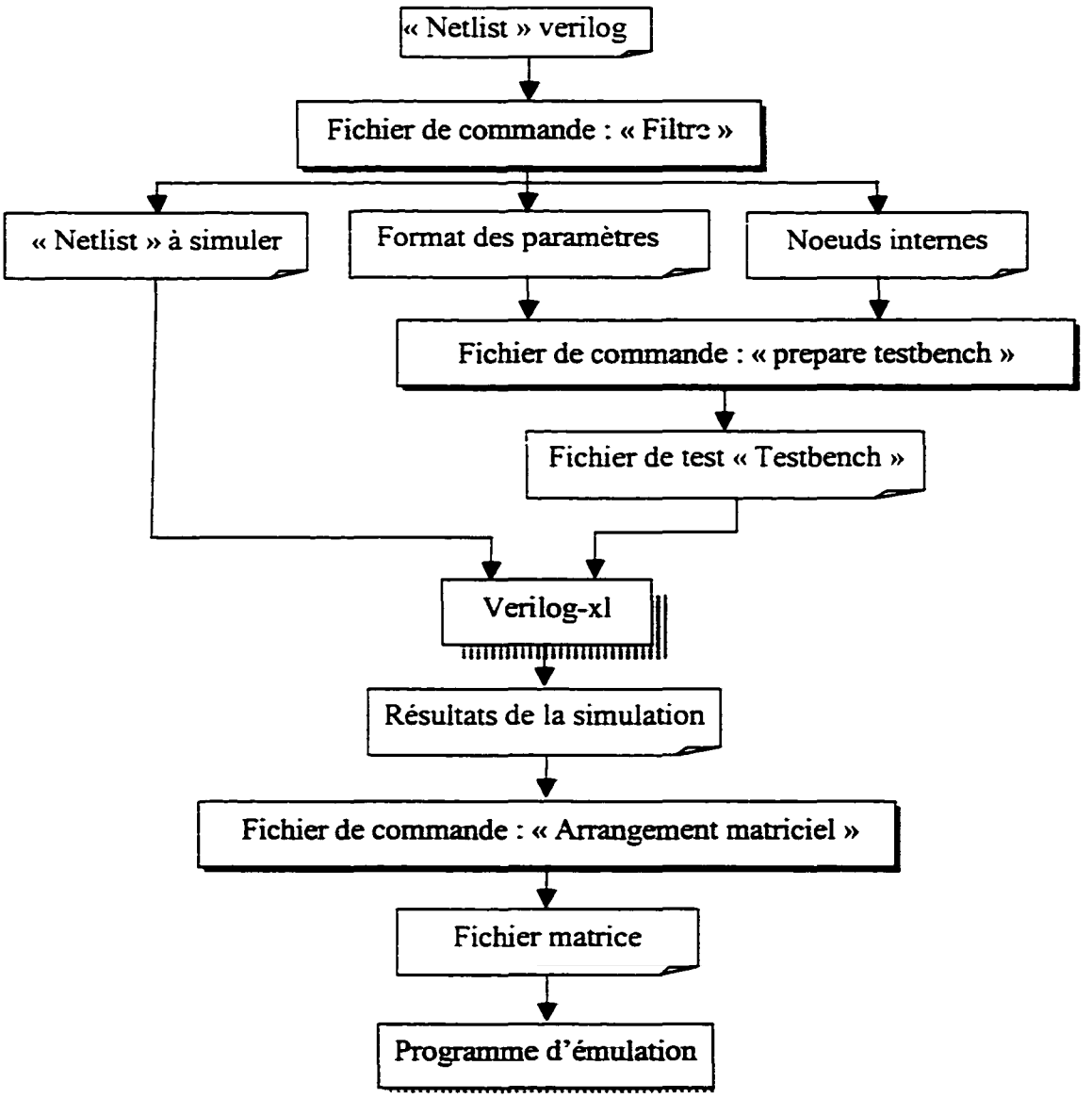


Figure 34 Schéma-bloc pour le génération des valeurs logiques par « Verilog-xl ».

Comme il sera expliqué au chapitre suivant, le programme d'émulation compresse la matrice de simulation pour réduire la taille mémoire requise pour la sauvegarde de la matrice. Pour des circuits de grande taille, la compression s'avère nécessaire afin de limiter l'espace mémoire consacré au programme d'émulation.

5.6 Conclusion

Nous avons présenté dans ce chapitre les limitations de l'outil logiciel au niveau de la simulation logique, telles qu'elles apparaissaient au début de ce projet. Nous avons identifié deux solutions possibles et après analyse, nous avons opté pour l'utilisation de l'outil logiciel « Verilog-xl ». Les résultats de la simulation sont traités et formatés sous forme de matrice, et sauvegardés dans un fichier qui constitue une entrée pour le programme d'émulation. Le prochain chapitre traite en détails l'étape d'intégration finale dans le programme d'émulation.

CHAPITRE 6

INTÉGRATION FINALE DANS L'OUTIL LOGICIEL DE DIAGNOSTIC

L'objectif de ce chapitre est de présenter l'étape d'intégration finale de tous les éléments constituant l'outil logiciel de diagnostic. La première étape d'intégration sert à établir la cohérence entre le netlist du schéma provenant de l'outil de synthèse et le netlist provenant de l'extraction du dessin des masques.

Nous avons présenté dans le chapitre 2 la procédure de réduction de la quantité d'information basée sur les capacités parasites de routage, extraites à partir du dessin des masques, et nous avons constaté qu'un pourcentage significatif de réduction pouvait être atteint. La deuxième étape d'intégration consiste à utiliser la liste des combinaisons réduites.

Nous avons vu au chapitre 5 la procédure de la simulation logique sur l'outil « Verilog-xl », et que les résultats logiques de cette simulation sont sauvegardés dans un fichier sous une forme matricielle. La troisième étape d'intégration consiste à utiliser ces résultats. Nous allons présenter à la fin de ce chapitre les résultats de l'intégration finale pour plusieurs circuits.

6.1 Cohérence entre le netlist du schéma et le netlist du dessin des masques

La première étape d'intégration consiste à établir le lien entre le netlist représentant le schéma du circuit et le netlist représentant le dessin des masques. On rappelle que le netlist du schéma donne les informations d'interconnexions entre les portes logiques du circuit, ce dernier étant généré par l'outil de synthèse « Synopsys ».

Le dessin des masques est représenté par un autre type de netlist. On trouve dans ce même netlist les capacités parasites, le netlist du dessin des masques et d'autres informations (voir Chapitre 2). Ce fichier est généré par l'extracteur du dessin des masques.

La figure 35 présente un exemple de netlist du dessin des masques et de netlist du schéma. On constate qu'il y a une différence de notation entre le netlist du schéma et le netlist du dessin des masques. Ceci est causé par le fait que les deux types de netlist proviennent de deux outils différents.

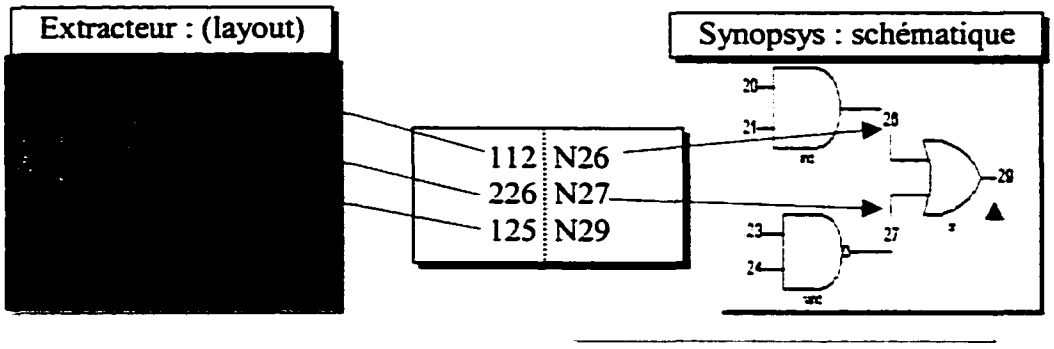


Figure 35 Les deux notations des nœuds : netlist du dessin des masques (à gauche) et netlist du schéma (adroite).

Pour éviter les problèmes dus à cette différence de notation, nous avons créé un programme de préparation qui permet de générer une liste contenant tous les nœuds du circuit, où apparaissent les deux types de représentation. Le programme permet aussi de vérifier la cohérence entre les deux netlists, et permet par la suite de générer un nouveau netlist en format compatible avec le programme d'émulation (voir la page 222).

La figure 36 présente les entrées et les sorties du programme de préparation. Les fichiers de sorties de ce programme seront les fichiers d'entrées pour le programme d'émulation de la méthode de diagnostic.

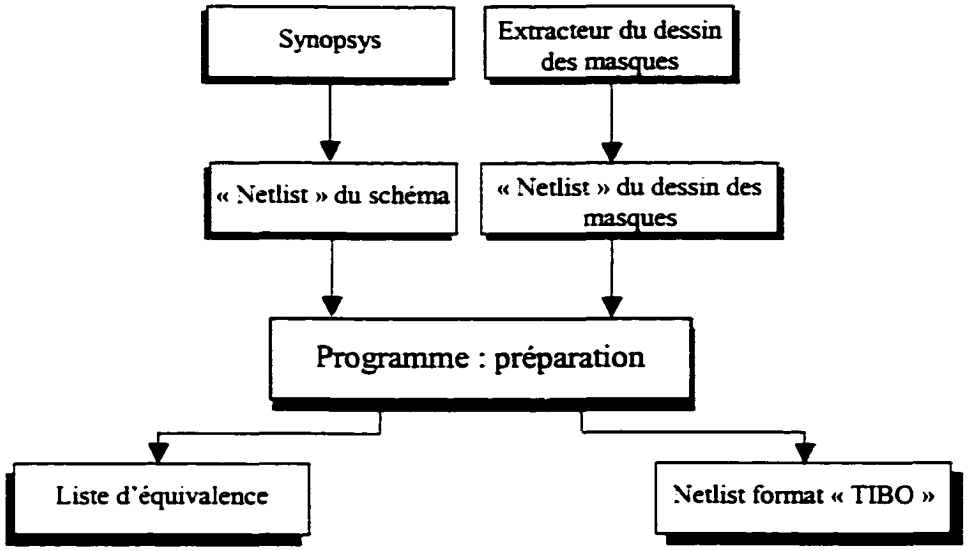


Figure 36 les entrées sorties du programme de préparation

La simplification des portes complexes comme les multiplexeurs est nécessaire pour générer un netlist en format « TIBO » compatible avec le programme d'émulation. On retrouve à l'annexe 1 une démonstration complète du programme de la figure 36 pour un circuit simple.

6.2 Utilisation de la liste réduite dans le programme d'émulation

Tel que mentionné à l'annexe 2, le programme d'émulation simule une liste contenant toutes les combinaisons possibles de nœuds candidats pour un type de faute fixé au départ. Or, comme nous avons vu au chapitre 2, cette liste peut être sujette à une

réduction en se basant sur les capacités parasites de routage. L'objectif de la deuxième phase d'intégration est l'utilisation de la liste réduite dans le programme d'émulation.

La figure 37 représente l'organigramme de la fonction qui génère la liste des combinaisons de nœuds à simuler. L'utilisateur peut choisir entre la génération d'une liste contenant toutes les combinaisons possibles de noeuds candidats à un type de panne « Avant l'intégration », ou l'utilisation de la liste réduite qui tient compte des capacités parasite de routage « Après l'intégration ».

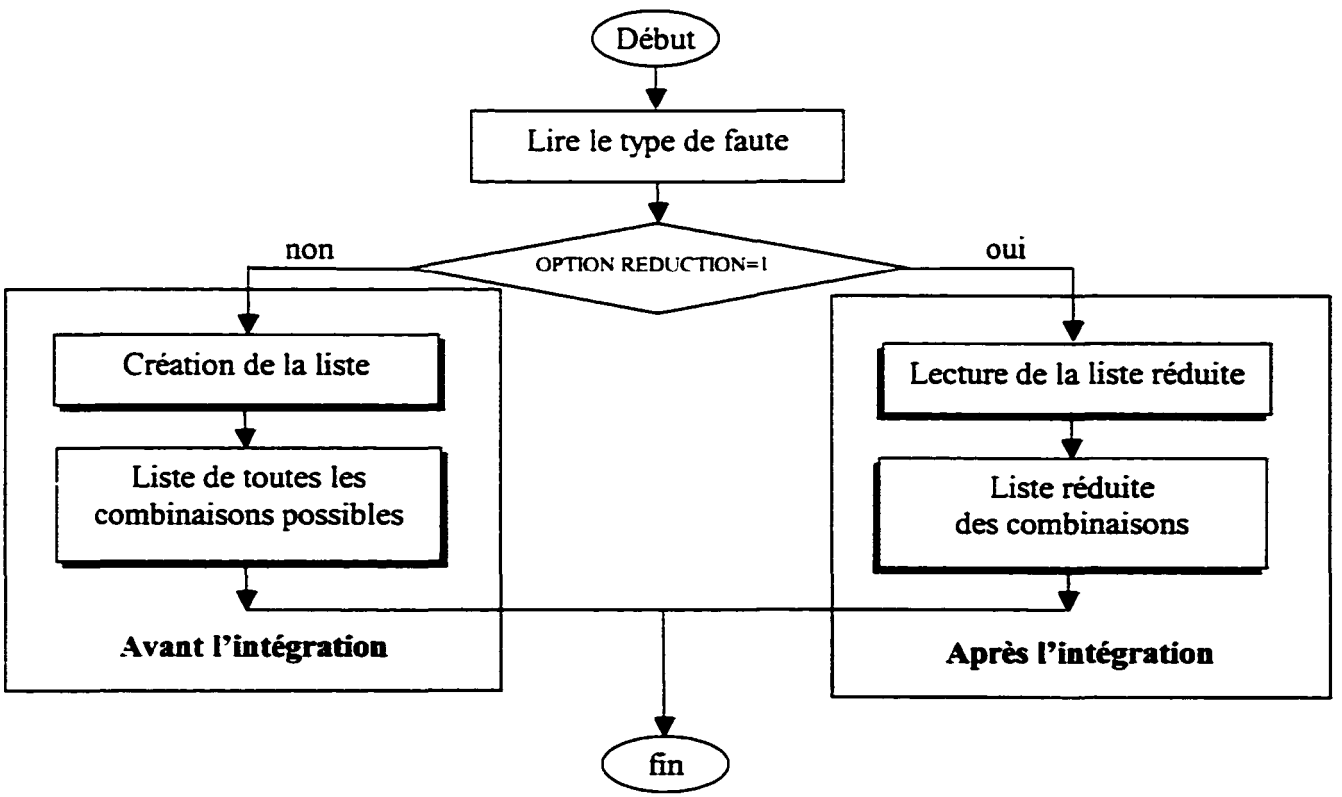


Figure 37 Organigramme de la fonction qui génère la liste des combinaisons de nœuds à simuler

Nous allons voir d'abord la lecture de la liste réduite dans le programme d'émulation. Ensuite, nous allons expliquer comment cette liste sera utilisée dans la phase de localisation des fautes.

6.2.1 Lecture de la liste réduite des combinaisons

La lecture de la liste réduite se fait à partir d'un fichier dans lequel est énuméré l'ensemble des combinaisons de nœuds qui correspondent aux sorties ciblées par le type de panne choisi et entre lesquels on retrouve une capacité parasite.

Un nœud est défini comme une structure de données à deux champs. Comme nous avons vu dans la section 6-1, nous devons garder une certaine cohérence entre le « netlist » du schéma et le « netlist » du dessin des masques. Donc pour chaque nœud de la liste réduite, nous allons utiliser deux notations. La figure 38 représente le vecteur qui contient l'ensemble des combinaisons. Chaque élément du vecteur représente un nœud avec ses deux notations, et deux nœuds successifs dans la liste réduite composent une combinaison (site potentiel). Le nombre d'éléments dans le vecteur est par conséquent paire.

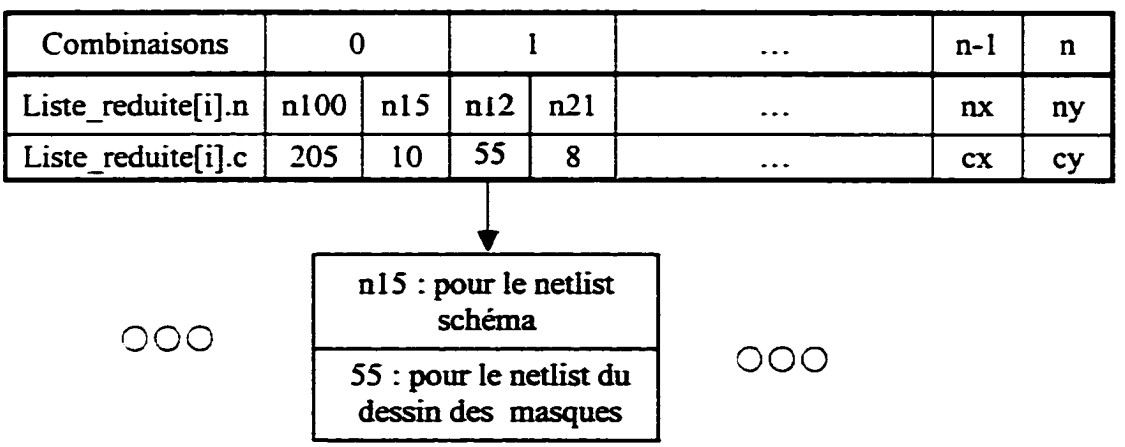


Figure 38 Le vecteur représentant la liste réduite des combinaisons

6.2.2 Lecture des mesures de courant ou émulation d'une panne

Tel que mentionné précédemment, le processus de diagnostic passe par l'acquisition de mesures de courant I_{DDQ}. Dans le cas où de telles mesures sont disponibles, l'outil de diagnostic en fait la lecture. Dans le cas contraire, nous remplaçons ces mesures réelles provenant d'un testeur par des mesures de courant fournies par l'émulateur I_{DDQ}. Pour générer ces valeurs, l'émulateur choisit au hasard un site de pannes, pour s'assurer que le site choisi corresponde à un site potentiel. Le choix se fait en consultant la liste réduite des sites potentiels.

Tel que décrit au chapitre 1, l'émulateur calcule la variation du courant de fuite I_{DDQ} en faisant la différence entre la valeur du courant au vecteur d'entrée courant et sa valeur au vecteur d'entrée précédent. Le programme passe par la suite à la phase d'identification des types de pannes les plus probables.

6.2.3 Phase d'identification des types de pannes les plus probables

Après la lecture ou l'émulation de valeurs de courant, le programme entre dans la phase d'identification des types de pannes les plus probables, qui compare les valeurs ΔI_{DDQ} aux signatures des différentes pannes possibles considérées.

On rappelle que cette comparaison est probabiliste. Cette phase permet d'obtenir la liste des pannes les plus probables détectées par la méthode. La phase suivante, la localisation des pannes, utilise la liste des types de pannes les plus probables afin de trouver dans le « netlist » le couple de nœuds responsable de la panne insérée dans ce dernier.

6.2.4 Phase de localisation faisant appel à la liste réduite des combinaisons

La procédure de localisation emploie la liste réduite pour comparer les valeurs réelles ou simulées du courant I_{DDQ} aux valeurs estimées du courant pour chaque site possible de la liste réduite. La figure 39 représente l'organigramme de la nouvelle fonction de localisation des pannes, en commençant par le type de panne le plus probable. Cette fonction sert en fait à confirmer le type de panne identifié dans la phase précédente.

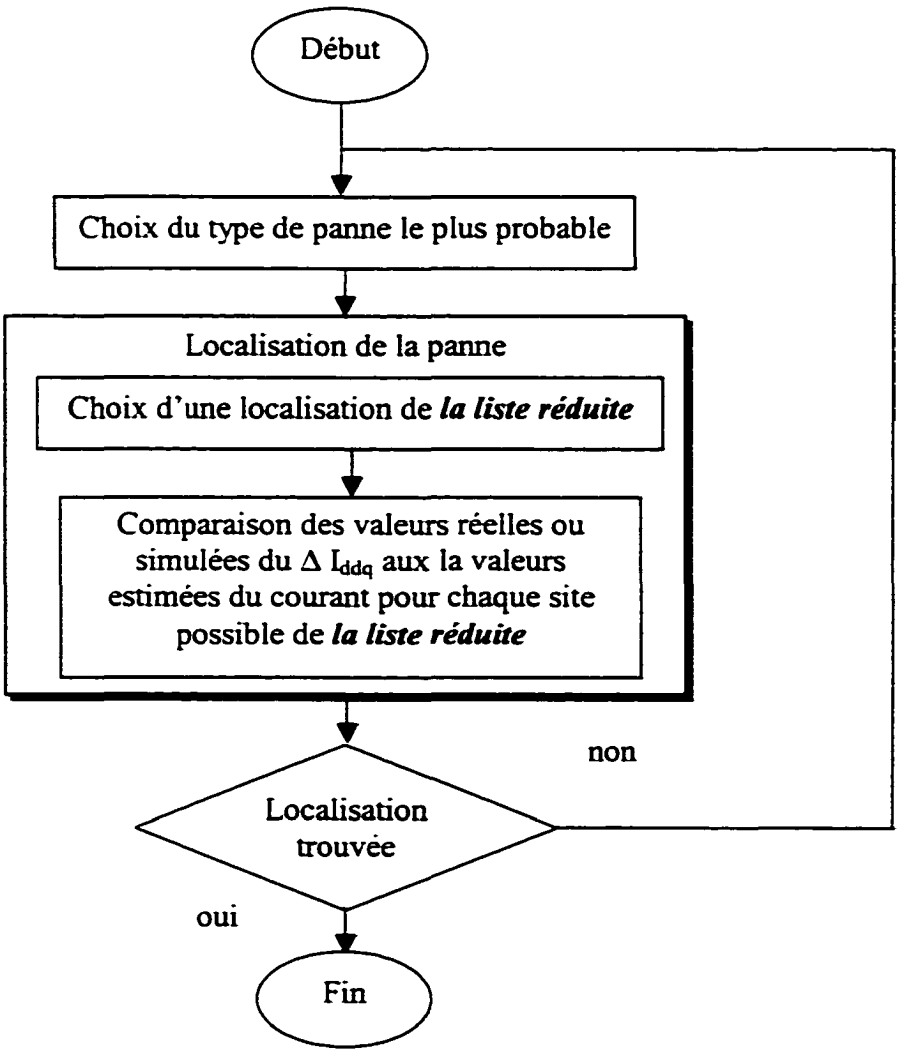


Figure 39 Localisation des pannes en utilisant la liste réduite

Tel qu'indiqué par l'organigramme, la localisation des pannes se fait en testant chaque site dans la liste réduite. Avant l'intégration, la phase de localisation se faisait en testant tous les sites potentiels du « netlist » correspondant au type de pannes en cours de traitement. Cette réduction du nombre de sites à tester se traduit par une réduction du temps machine nécessaire pour effectuer le diagnostic. Des estimations du gain de temps machine seront présentés à la section 6.5.

6.3 Intégration du bloc de simulation faisant appel au simulateur Verilog-xl

La troisième étape d'intégration consiste à utiliser les résultats de la simulation logique obtenus à l'aide de l'outil Verilog-xl dans le programme d'émulation de la méthode de diagnostic. Comme nous avons vu dans le chapitre 5, les résultats de la simulation sont sauvegardés dans un fichier sous une forme matricielle, et pour utiliser ces données, on doit d'abord transférer la matrice des résultats logique en mémoire.

Le problème qu'on rencontre ici est lorsque l'espace mémoire disponible ne suffit pas pour sauvegarder la matrice de simulation. Pour résoudre ce problème, nous allons passer par une étape de compression de la matrice.

6.3.1 Compression de la matrice de simulation

➢ *Présentation du problème :*

Le tableau XVII présente l'espace mémoire requis pour sauvegarder une matrice contenant les résultats de simulation logique en fonction du système d'exploitation utilisé. Nous avons utilisé dans cet exemple les paramètres suivants :

- Nombre de nœuds = 1 million.
- Nombre de vecteurs = 100 vecteurs.

Tableau XVII

Espace mémoire requis pour sauvegarder une matrice de simulation
de taille 100x1Millions éléments

Système d'exploitation	Taille de l'entier	Taille de la matrice 100x1Millions	
		En bits	En meg
Dos	2 bytes = 16 bits	1600 Millions bits	1525.8 mégabits
Unix	4 bytes = 32 bits	3200 Millions bits	3051.7 mégabits
Windows	4 bytes = 32 bits	3200 Millions bits	3051.7 mégabits

Le problème de la mémoire peut être résolu en utilisant une technique de compression. La solution que nous proposons consiste à exploiter la nature des éléments de la matrice. Comme nous traitons les valeurs binaires « 0/1 », nous pouvons adresser la mémoire en bit, en utilisant la conversion du binaire au décimal. Pour avoir la valeur logique d'un nœud à un vecteur donné, nous allons effectuer une opération logique sur le nombre représentant les valeurs logiques de simulation de ce nœud. Voici un simple exemple.

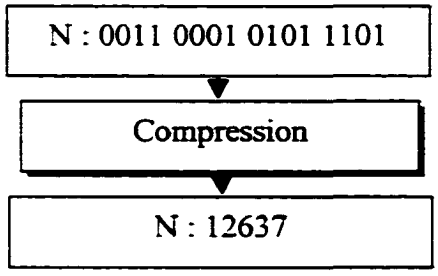


Figure 40 Exemple de compression

La figure 40 présente un exemple de compression effectuée sur 16 valeurs logiques pour un seul nœud. Le résultat de la compression peut être sauvegardé dans une variable entière. Pour obtenir la valeur logique du nœud à un vecteur de simulation donné, nous appliquons sur le chiffre compressé une opération logique pour en extraire la valeur du bit correspondant (voir l'annexe 12 pour un exemple de compression d'une matrice représentant les valeurs logiques de 117 nœuds correspondant à 20 vecteurs de simulation).

6.3.2 Simulation logique faisant appel à la matrice compressée

Pour se servir de la matrice compressée, nous avons ajouté dans le programme d'émulation une fonction qui permet de mettre à jour les valeurs logiques de tous les nœuds du graphe pour chaque cycle de simulation. Cette fonction remplace la fonction de simulation déjà existante (pour plus d'information sur l'ancienne fonction, voir l'annexe 2).

La figure 41 présente l'organigramme de la nouvelle fonction de simulation. Cette fonction diffère de l'ancienne fonction par le fait que cette dernière utilise les résultats de simulation importé de l'outil Verilog-xl. On constate ici que la mise à jour des valeurs logiques des nœuds du graphe se fait directement à partir de la matrice logique compressée. Avant l'intégration de cette fonction, la simulation se faisait en parcourant à plusieurs reprises le graphe afin de visiter toutes les portes et les nœuds.

Rappelons que l'avantage principal de l'intégration de cette technique de simulation dans le programme d'émulation est de permettre la simulation des deux types de circuits séquentiels et combinatoires.

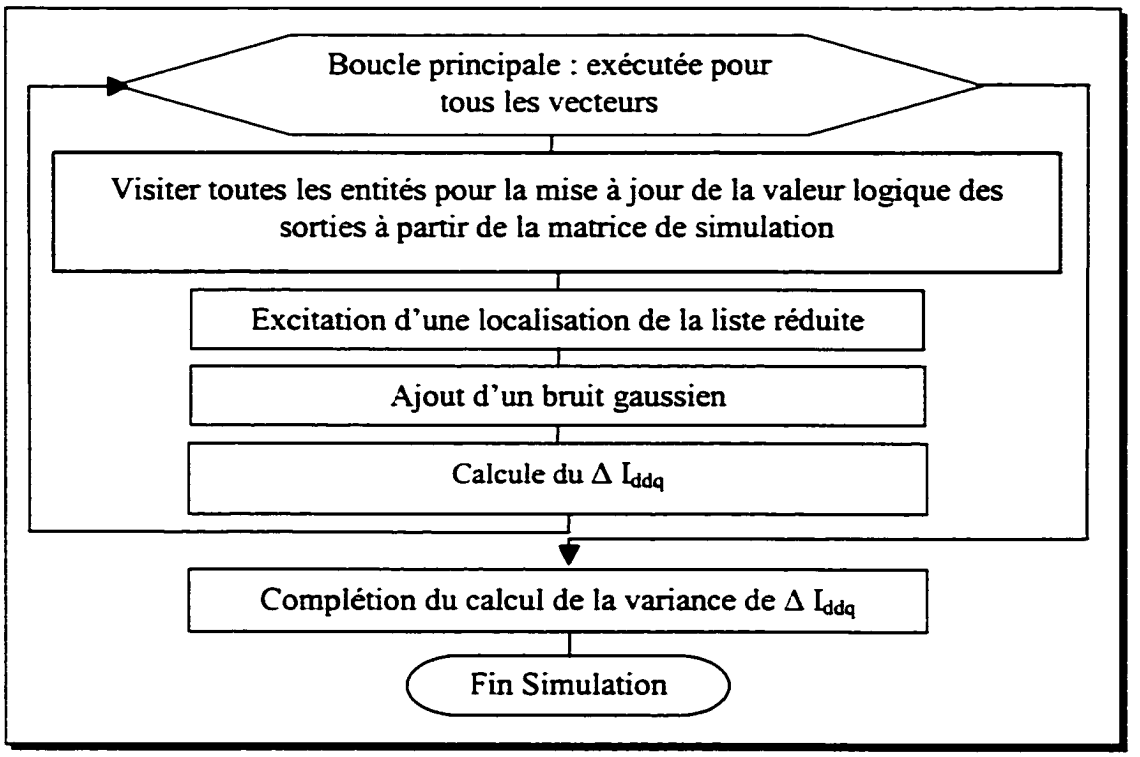


Figure 41 Simulation logique et émulation du courant

6.4 Entrées sorties du programme avant et après l'intégration finale

En guise de récapitulation, la figure 42 présente les entrées et les sorties du programme d'émulation avant et après la phase d'intégration finale. Les entrées avant l'intégration sont décrites à l'annexe 3. Ces entrées sont toujours présentes après l'intégration. Nous allons revenir dans ce qui suit sur les nouvelles entrées du programme :

- le fichier d'entrée « équivalence des nœuds » qui sert à établir la cohérence entre les nœuds du « netlist » fourni par l'outil de synthèse et ceux du « netlist » créé par l'outil d'extraction du dessin des masques,

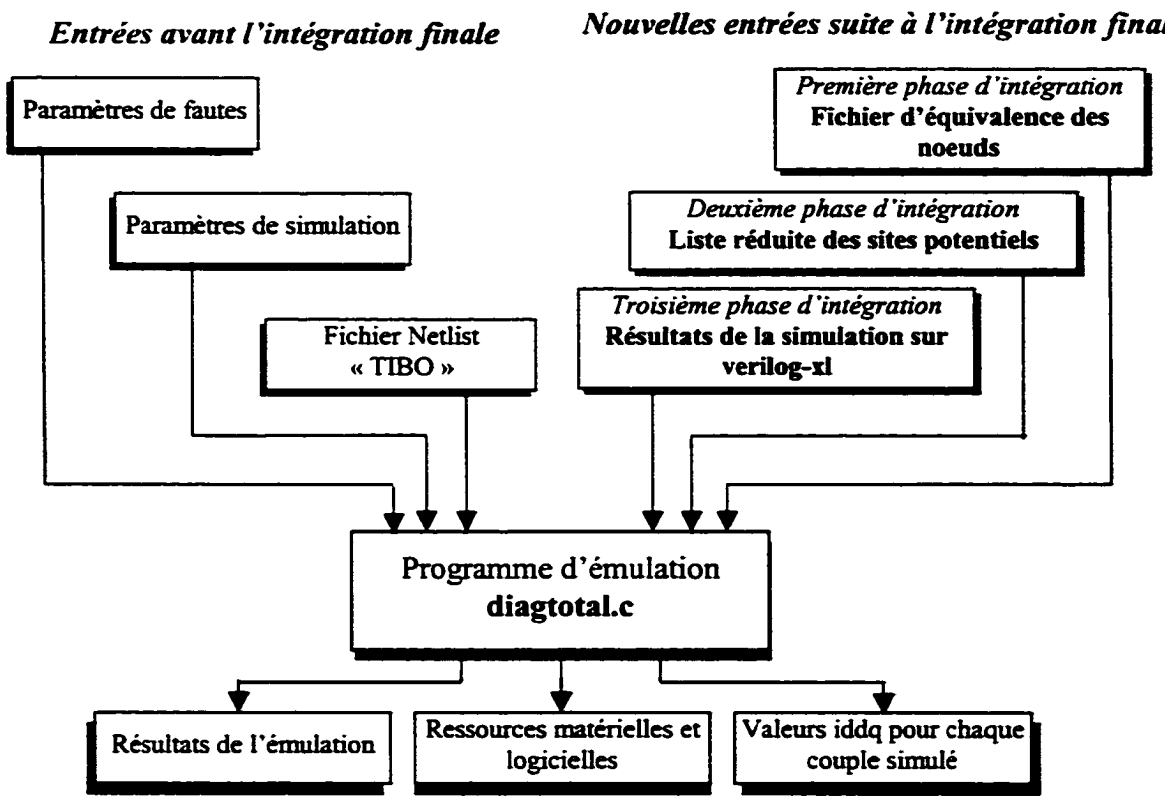


Figure 42 Nouvelles entrées/sorties

- la liste réduite des sites potentiels pour chaque type de pannes, permettant de réduire l'effort de calcul lié à la localisation de la panne, et

- les résultats de simulation « Verilog-xl » sauvegardés sous un format matriciel permettant le diagnostic de circuits séquentiels.

6.5 Résultats de l'outil logiciel de diagnostic

Nous avons testé l'outil logiciel de diagnostic après l'intégration sur des circuits de tailles différentes. Les mesures prises sont résumées au tableau XVIII.

Tableau XVIII

Résultats du programme d'émulation pour différents circuits

Circuit / Informations	**Multiplicateur 4bits**	Fir 4 étages	Fir 7 étages	Fir 15 étages	Fir 25 étages	Fir 50 étages
Nombre de cellules	94	901	1641	3521	6468	13127
Nombre de nœuds	117	986	1772	3759	7262	13855
Nombre de simulations	25	132	261	450	512	671
Espace mémoire dynamique (Kbytes)	54	466	875	1876	3000	6650
TMD en ms (sans émulation)	0.25	0.58	1.20	2.56	10.05	28.5
Temps cpu du programme en ms	5.57	50.52	205.87	832.87	2186.94	2761.92

Nb. significations :

- Nombre de simulations : représente le nombre réduit de sites physiques potentiels.
- Espace mémoire dynamique (Kbytes) : Représente l'espace requis pour sauvegarder les informations liées au graphe représentant le « netlist », la liste des sites physiques potentiels et la matrice résultat de la simulation logique.
- TMD : Temps moyen de diagnostic (sans émulation du courant I_{DDQ}), c'est-à-dire le temps moyen requis pour effectuer le diagnostic incluant le temps d'identification de la panne ainsi que le temps de localisation de la panne.

- Temps cpu du programme en ms : temps requis pour le programme afin d'effectuer la simulation de tous les sites physiques potentiels de la liste réduite (avec émulation du courant I_{DDQ}).

Les ressources matérielles déployées par le programme se classent en deux catégories. La première donne l'espace de mémoire allouée dynamiquement par le programme pour chaque circuit, c'est-à-dire l'espace mémoire utilisé pour sauvegarder les informations du graphe représentant le « netlist ». On remarque que cet espace augmente en fonction de la taille des circuits. La deuxième donne le temps *cpu* utilisé par le programme pour le traitement des circuits (pour avoir plus d'informations sur les mesures temporelles, voir l'annexe 3).

Le tableau XIX présente les résultats du programme d'émulation avant et après intégration pour le multiplicateur 4bit. On remarque une amélioration à deux niveaux, d'une part, de l'espace mémoire occupé par le programme, et d'autre part, du temps moyen et total de diagnostic. En effet, le programme converge plus rapidement vers la solution tout en utilisant moins de ressources matérielles, résultats de la réduction de la liste des sites physiques potentiels et de l'utilisation du simulateur Verilog-xl.

Tableau XIX

Résultats du programme d'émulation avant et après intégration pour le multiplicateur 4bit

Circuit Informations	**Avant intégration**	**Apres intégration**
Nombre de cellules	94	94
Nombre de nœuds	117	117
Nombre de simulations	465	25
Espace mémoire dynamique(Kbytes)	129	54
TMD en ms (sans émulation)	0.32	0.25
Temps cpu du programme en ms	6.30	5.57

6.6 Conclusion

Nous avons présenté dans ce chapitre l'intégration finale de tous les éléments constituant l'outil logiciel de diagnostic. La première étape d'intégration a permis d'établir la cohérence entre le netlist du schéma provenant de l'outil de synthèse et le netlist provenant de l'extraction du dessin des masques. Nous avons présenté également l'intégration de la procédure d'utilisation de la liste réduite des sites physiques potentiels basée sur les capacités parasites de routage extraites à partir du dessin des masques. Par la suite, nous avons présenté l'intégration de l'outil de simulation « Verilog-xl ».

L'intégration finale a donc permis de remédier à la limitation du simulateur existant qui ne supportait que les circuits logiques purement combinatoires. De plus, l'intégration de la méthode de réduction basée sur les capacités parasites de routage a permis d'accélérer le processus de diagnostic. En effet, le programme converge plus rapidement vers la solution (site physique causant la panne) tout en utilisant moins de ressources matérielles.

CONCLUSION

Ce projet de maîtrise portait globalement sur l'amélioration d'une méthode existante de diagnostic des circuits intégrés, méthode basée sur l'utilisation des signatures probabiliste du Δ_{IDDQ}. Les améliorations visées se situaient à différents niveaux. D'une part, il s'agissait d'accélérer la méthode en réduisant le nombre de sites potentiels de court-circuits. Nous avons donc présenté dans ce mémoire deux techniques de réduction des sites physiques potentiels. La première utilise les capacités parasites de routage extraites du dessin des masques du circuit. Les résultats d'expérimentation ont montré que le pourcentage de réduction des sites physiques potentiels est toujours supérieur à 94%, et qu'il peut atteindre 99.99% dans le cas des circuits plus complexes. La deuxième méthode de réduction, basée sur les résultats des sorties erronées du circuit sous test obtenues à l'aide son émulation, permet aussi de réduire le nombre de sites physiques.

D'autre part, nous avions comme objectif de palier à certaines limitations de l'outil logiciel qui avait été développé pour permettre la validation de la méthode. Du côté de l'outil logiciel, Nous avons développé donc l'infrastructure nécessaire permettant de l'interfacer avec l'outil de simulation Verilog-xl de Cadence. Ce développement a permis de remédier à la limitation du simulateur existant qui ne supporte que les circuits logiques purement combinatoires. Nous sommes donc maintenant en mesure d'utiliser la méthode de diagnostic sur des circuits combinatoires et séquentiels.

Les résultats obtenus démontrent que nos objectifs de départ ont été atteints.

RECOMMANDATIONS

Les deux méthodes de réduction des sites physiques potentiels, présentées dans ce mémoire, doivent être testées sur des circuits plus grands. Il serait également important de tester les deux méthodes de réduction sur un circuit moniteur contenant des défectuosités contrôlables.

Par manque de temps, la technique de réduction basée sur l'émulation du circuit sous test n'a pu être totalement intégrée dans la dernière version de l'outil de diagnostic. Cette intégration doit fait partie des travaux à venir.

ANNEXE 1

Calcul des probabilités pour la phase d'identification des types de pannes les plus probables

La procédure d'identification est détaillée comme suit. Prenons un ensemble z de V-1 mesures ΔI_{DDQ}, $z = \{z_1, z_2, \ldots, z_k, \ldots, z_{v-1}\}$, où z_k est égal à I_{DDQ} au vecteur d'entrée k moins I_{DDQ} au vecteur d'entrée k-1. En se basant sur le MLE, le type de pannes le plus probable est celui qui a la plus forte probabilité P_{iz} correspondant à la probabilité pour qu'une panne soit de type i étant donné un vecteur z de mesure ΔI_{DDQ}. On peut exprimer P_{iz} par :

$$Piz = \frac{P\{z \mid fti\}P\{fti\}}{P\{z\}}, \tag{A-1}$$

où $P\{ft_i\}$ est la probabilité pour qu'une panne soit de type i, $P\{z\}$ la probabilité d'observer l'ensemble z de mesures, et $P\{z|ft_i\}$ la probabilité d'observer l'ensemble z de mesures si la faute est de type i. On peut estimer le terme $P\{z|ft_i\}$ comme suit:

$$P\{z \mid fti\} = \prod_{k=1}^{V-1} P\{zk \mid fti\}. \tag{A-2}$$

où V est le nombre de vecteurs de test. Le terme P{zk|fti} est défini comme suit :

$$P\{z_k \mid ft_i\} = \sum_{j=1}^{M} P\{z_k \mid (ft_i, \Delta j)\}P\{\Delta j\} \tag{A-3}$$

où La probabilité d'observer la mesure z_k si le type de panne est i est représenté par le terme $P\{z_k|(fti.\Delta j)\}$. ceci pour une des M possibles barres Δj dans la signature de ce type de panne i. On peut modéliser ce terme par une normale de moyenne μ_j et de variance σ_j^2:

$$P\{z_k \mid (fti, \Delta j)\} = \frac{-1}{\sigma j\sqrt{2\pi}} e^{-0.5(\frac{zk-\mu g}{\sigma j})^2} . \qquad \text{(A-4)}$$

Le terme P{z} demeure le même pour chaque type de panne. si on suppose l'équiprobabilité des différents types de panne dans le circuit, P{fti} est donc le même pour chaque type de faute. L'identification des types de pannes les plus probables revient alors à trouver les plus grands des P{z|fti}.

On peut utiliser un critère de sélection définit dans la formule 2-6, pour éliminer de la liste les types de fautes les plus improbables.

$$\frac{1}{V-1}\sum_{k}(\frac{z_k - \mu_{near}}{\sigma_{near}})^2 < th1 , \qquad \text{(A-5)}$$

où σ_{near} et μ_{near} sont la moyenne et l'écart type de la barre Δ_j qui, selon la loi normale, est la plus près de z_k.

Le terme th1 représente une valeur de comparaison permettant d'éliminer les types de pannes les plus improbables, qui ne rencontrent pas ce critère de sélection. Il est très important de noter que la valeur th1 doit être judicieusement choisie par essais et erreurs, pour permettre le rejet des pannes les plus improbables sans y éliminer les pannes réelles.

ANNEXE 2

Le programme d'émulation de la méthode de diagnostic

1. Présentation du programme d'émulation diag_total

Le programme *diag_total* a comme objectif l'émulation de la méthode de diagnostic basée sur des signatures du courant ΔI_{DDQ}. Ce programme a été conçu en langage «C». Il a comme mission la validation de la méthode de diagnostic. Son exécution permet de faire l'identification et la localisation des pannes simulées d'un circuit donné.

Pour valider la méthode de diagnostic le programme passe par les étapes suivantes :

- Fixer un type de panne.
- Génération de la liste des paires de nœuds candidats (toutes les possibilités).
- Excitation d'une location (un pont entre les sorties de deux portes logiques).
- Émulation du courant I_{DDQ} pour cette location.
- *La phase d'identification* des types de pannes les plus probables.
- *La phase de localisation* des pannes.

Commençons par introduire le programme *diag_total.* Après nous allons faire une analyse structurelle de données, et pour bien comprendre le séquencement des tâches, nous allons poser un descriptif général de séquencement du programme, en détaillant chaque fonction.

1.1. Introduction du programme

Le programme diag_total est codé en langage « C », et se caractérise par son style procédural, donc pour analyser le code on va passer par deux étapes qui sont :

- **Analyse structurelle de données :**

Cette étape permet de connaître les données (variables, types prédéfinis, const..) du programme.

➢ **Descriptif général du séquencement :**

Cette étape permet d'analyser le séquencement des étapes du programme. À ce niveau, nous allons analyser les fonctions les plus importantes, et poser pour chaque fonction l'organigramme fonctionnel correspondant, ce qui va faciliter notre analyse et modification s'il y a lieu.

Caractéristiques du programme *diag_total* :

Les caractéristiques techniques du programme *diag_total* sont résumées dans le tableau suivant :

Tableau XX

Caractéristiques du programme diag_total

Paramètres	Nombre
Nombre de fonctions	27 fonctions plus la fonction main()
Nombre de ligne de code	3414
Nombre de variable de base	330
Nombre de constante	36
Nombre de structure	2

1.1.1. Entrées sorties du programme

Comme le montre la figure 43, le programme *diag_total* utilise trois fichiers d'entrée. Le premier fichier « *faut_param.dat* » donne les paramètres de pannes à simuler. Le deuxième fichier « *sim_param.dat* » donne les paramètres de simulation. Quant au troisième fichier, il représente le netlist du circuit à simuler.

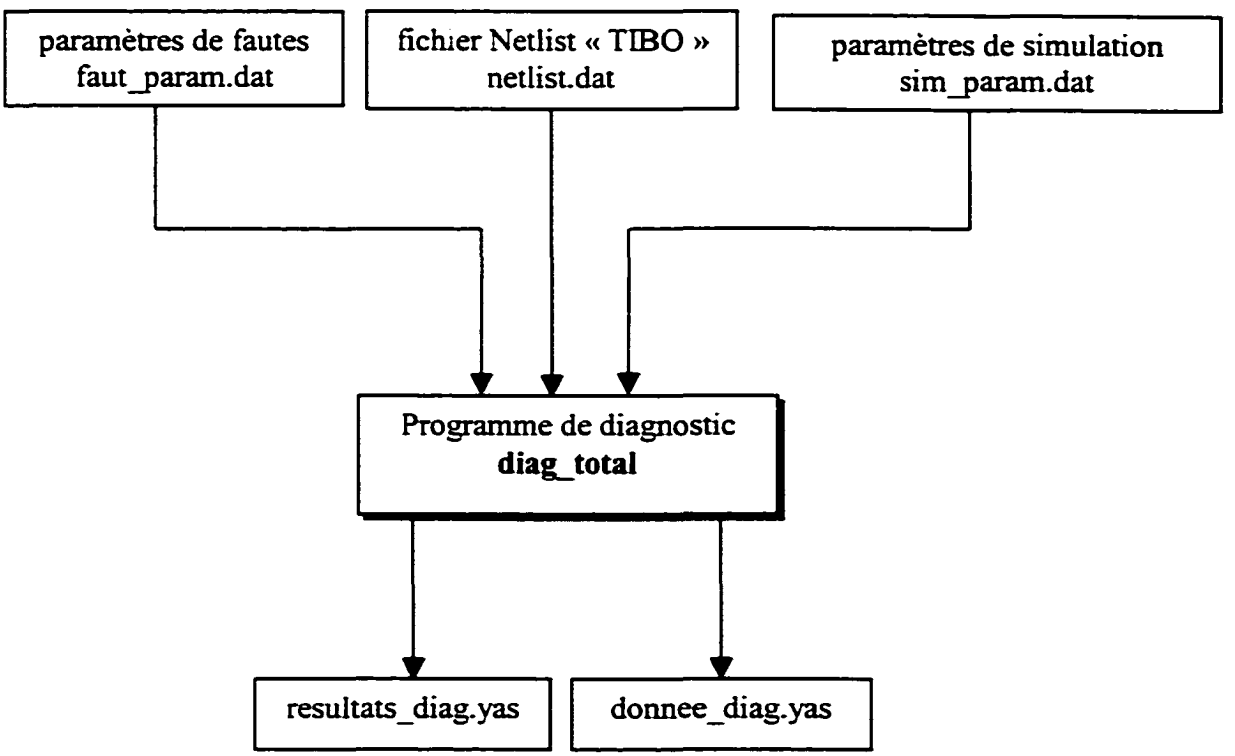


Figure 43 Entrées sorties du programme diag_total.

Les fichiers d'entrées et de sortie sont donnés à l'annexe 3.

1.1.2. Analyse structurelle de données

L'analyse structurelle de données consiste à étudier les différentes structures de donnée du programme. Cette analyse va nous permettre de comprendre comment le programme gère ses ressources matérielles et logicielles.

On constate la présence de deux structures dans le programme (voir figure 44). La première structure s'appelle « LISTE ». Elle contient un champ de données pour les entiers, et un champ pointeur vers le même type de structure. Ici on voit un élément d'une liste chaînée par des pointeurs. La deuxième structure s'appelle « ELEMENT ». Elle contient deux champs de donnée. Le premier est une chaîne de caractères et le deuxième est un entier, et deux champs pour les pointeurs vers la structure « LISTE ». Nous avons défini ici deux structures différentes utilisées par le programme. Nous allons voir par la suite comment ces deux structures sont utilisées.

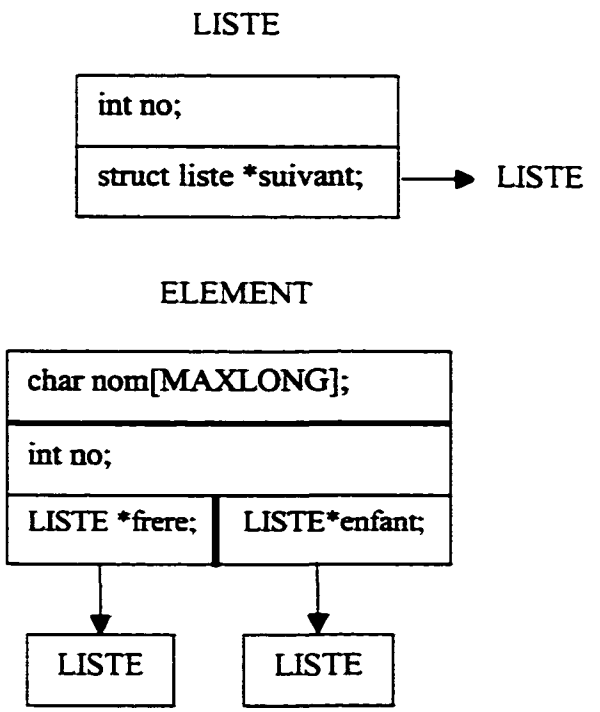


Figure 44 Les deux structures du programme

Le programme utilise les deux structures précédentes afin de construire le graphe qui va représenter le « netlist » du circuit à simuler. La figure 45 représente un vecteur de pointeurs vers le type de structure « ELEMENT ». La taille de ce vecteur est égale au nombre de portes logiques dans le « netlist » à simuler. Donc chaque élément du vecteur est un pointeur vers une entité « LISTE » qui représente une porte logique. Nous avons vue précédemment que le type de structure « LISTE » peut pointer vers une autre structure du même type. On peut donc utiliser cette fonction pour mémoriser les entrées et sorties des portes logiques. Pour construire le graphe, nous utilisons la technique d'allocation dynamique de la mémoire.

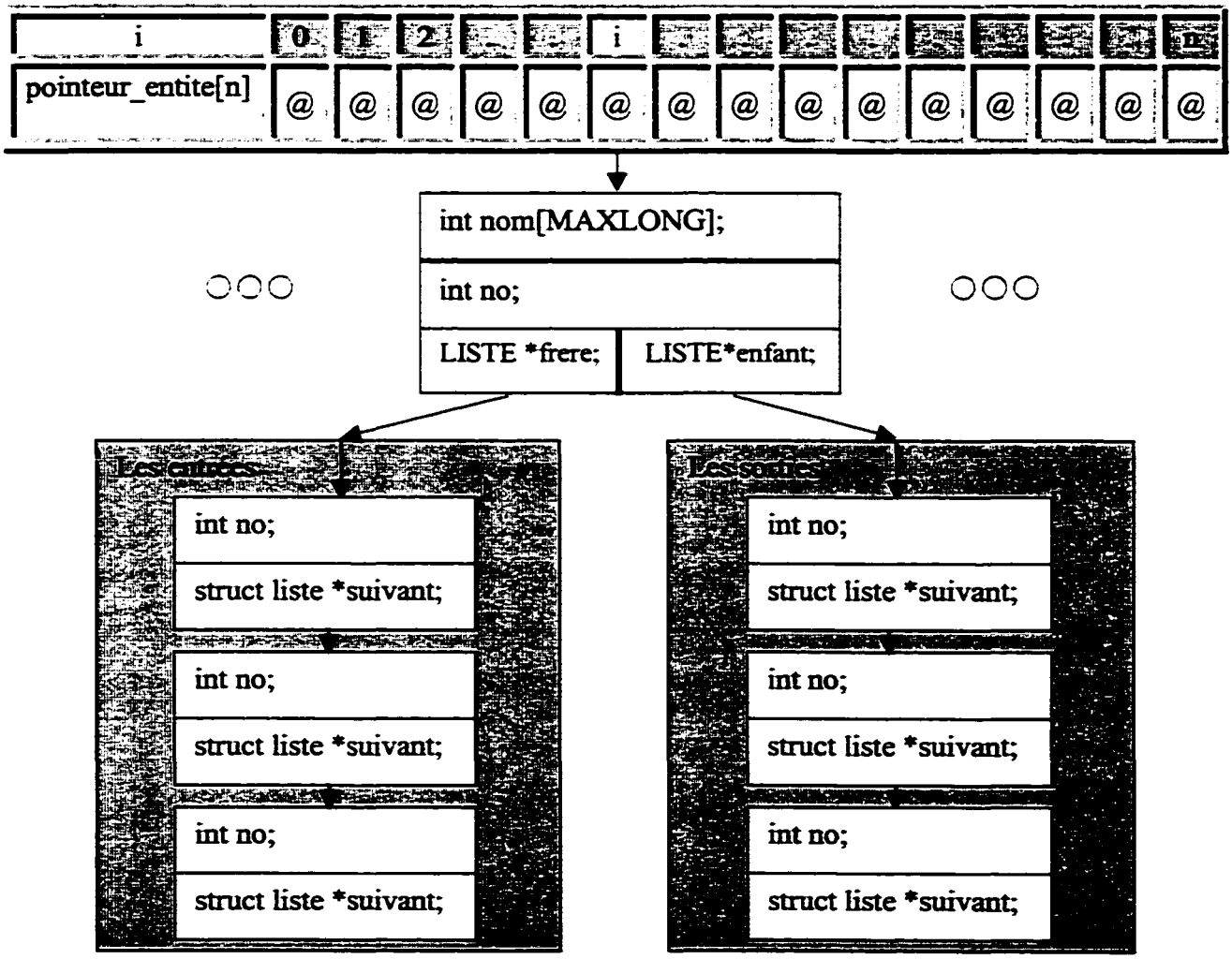
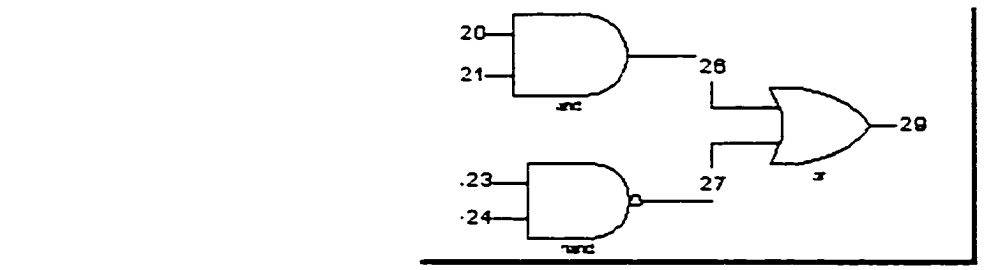


Figure 45 Structure du vecteur d'éléments

L'exemple suivant montre la construction d'un graphe à partir d'un simple netlist :



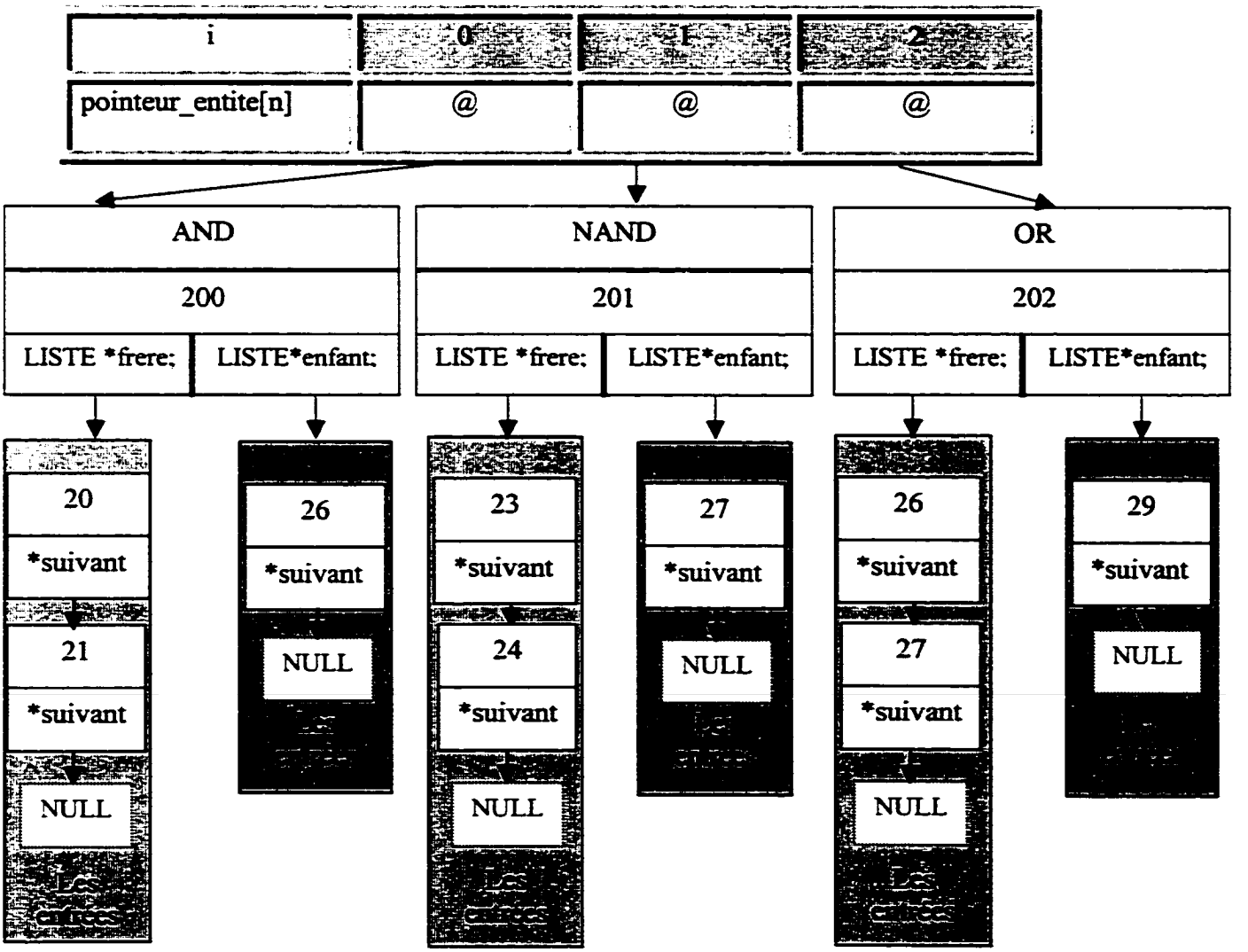


Figure 46 Exemple de transformation d'un netlist

1.1.3. Analyse fonctionnelle et descriptif général du séquencement

L'analyse fonctionnelle du programme consiste à étudier les fonctions les plus importantes. Cette étape est cruciale dans le cas où une modification du programme est probable. Donc l'objectif de cette section est la définition d'un descriptif général de séquencement, qui sera utilisé dans le cas d'intervention sur le code du programme d'émulation *diag_total*.

➢ Les fonctions du programme sont données au tableau de la page suivante :

Tableau XXI

Les fonctions du programme d'émulation

Prototype	Commentaire
1. void print_info_init(void)	impression des paramètres initiaux
2. void get_delta_iddq(void)	obtention des valeurs Delta I_{DDQ}
3. void fault_type_ident(void)	identification du type de panne(s)
4. void fault_location(void)	localisation de la (des) panne
5. void do_logic_sim(void)	faire simulation logique
6. void read_source_delta(void)	lire le fichier source_delta.dat
7. void read_sim_param(void)	lire le fichier sim_param.dat
8. void read_net_create_graph(void)	lire le fichier netlist.dat et créer le graphe
9. int gauss_noeud(void)	génération pseudo-aleatoire de la valeur des nœuds d'entrées
10. void create_iddq_tables(void)	créer le tableau des niveaux I_{DDQ}
11. void visite_entite(void)	mettre à jour la valeur des sorties des entités
12. void excite_fault(void)	excitation de la panne pour ajouter le niveau correspondant à I_{DDQ}
13. void add_noise_iddq(void)	ajouter bruit gaussien à I_{DDQ}
14. void get_signatures(void)	obtenir (créer ou lire) signatures
15. void create_signatures(void)	obtenir (créer ou lire) signatures
16. void create_loc_list(void)	créer la liste des sites probables
17. void estimate_loc_prob(void)	la prob. des sites probables
18. void estimate_loc_prob_ord(void)	la prob. des sites probables
19. void read_panne_param(void)	lecture du fichier panne_param.dat
20. void gen_panne_loc_sim(void)	génération des sites de pannes simulés par énumération
21. void estimate_loc_prob_remain_site (void)	prob. des sites probables avec nb de sites qui restent
22. void print_netlist_stat(void)	impression de stat. sur netlist
23. void gen_panne_loc_sim_rand(void)	génération des sites de pannes simulés de manière pseudo-aleatoire
24. void verify_indep(void)	vérifie si les pannes sont indépendantes
25. void verify_equiv(void)	vérifie si les pannes sont équivalentes
26. voiden_panne_loc_sim_rand_ft(void);	génération des sites de pannes simulés de manière pseudo-aleatoire pour un type de pannes donné
27. void recover_bad_diag(void)	tentative de récupération des mauvais diagnostics

1.1.3.1. La phase d'initialisation

La phase d'initialisation tel qu'indiquée par la figure 47 permet la lecture des paramètres de simulation déjà définis dans les fichiers d'entrées, la création des fichiers de sorties qui vont sauvegarder les résultats de simulation, et la construction du graphe à partir du fichier netlist.

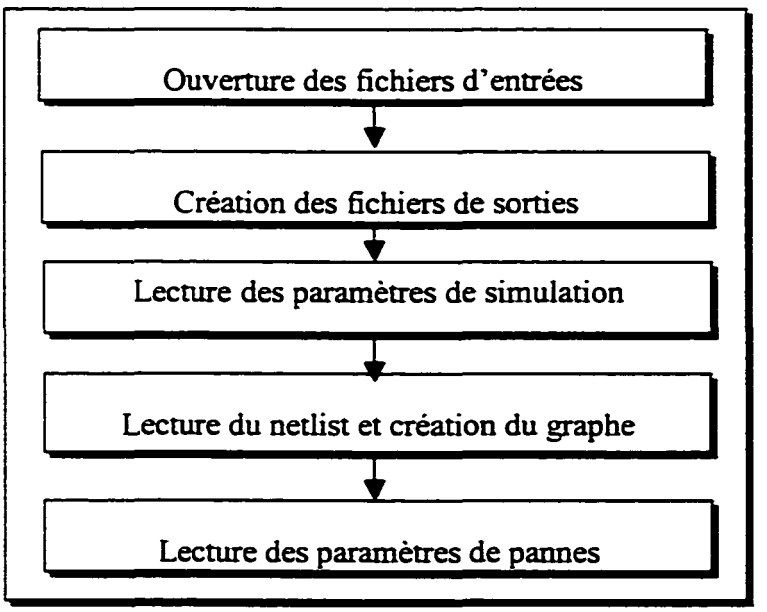


Figure 47 Lecture des paramètres de simulation

Dans l'exemple de la figure 46 on donne un exemple de construction d'un netlist simple de trois portes logiques.

1.1.3.2. La boucle principale du programme

Après la phase d'initialisation qui consiste à lire les paramètres de simulation et la création du graphe représentant le netlist, le programme entre dans la boucle principale qui permet la simulation de tous les couples de nœuds candidats au type de panne fixé au départ. l'organigramme de la figure 48 présente la boucle principale du programme.

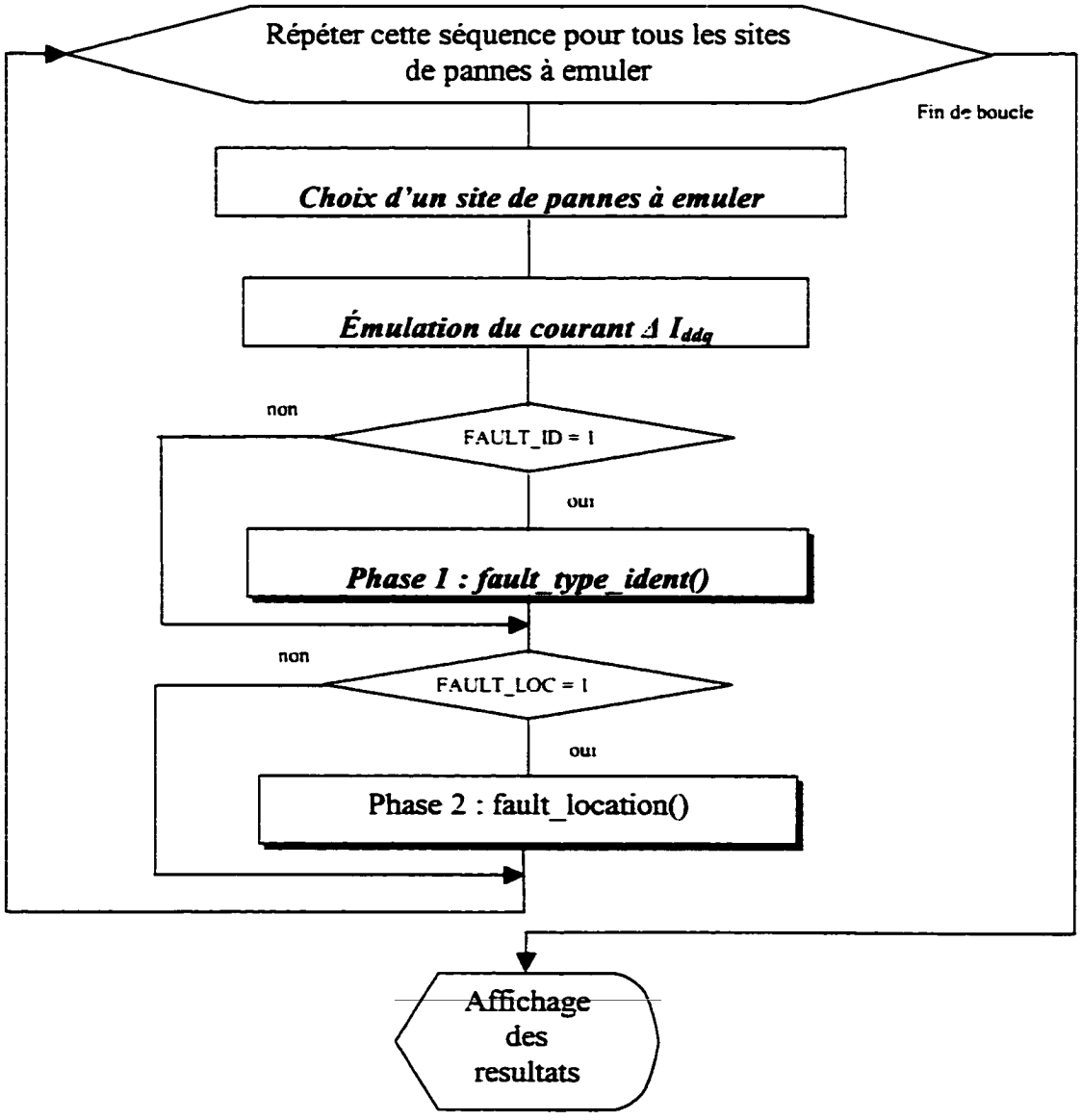


Figure 48 Organigramme de la boucle principale du programme

1.1.3.3. Choix d'un site de panne à émuler

Le choix d'un site de panne à émuler pour un type de panne donné consiste à trouver dans le graphe qui représente le netlist une paire de nœuds répondant au critère de la panne choisie.

Suivant la valeur de la variable optionnelle « GEN_FAULT_RANDOM », le programme offre trois possibilités de choix, la figure 49 donne l'organigramme de choix.

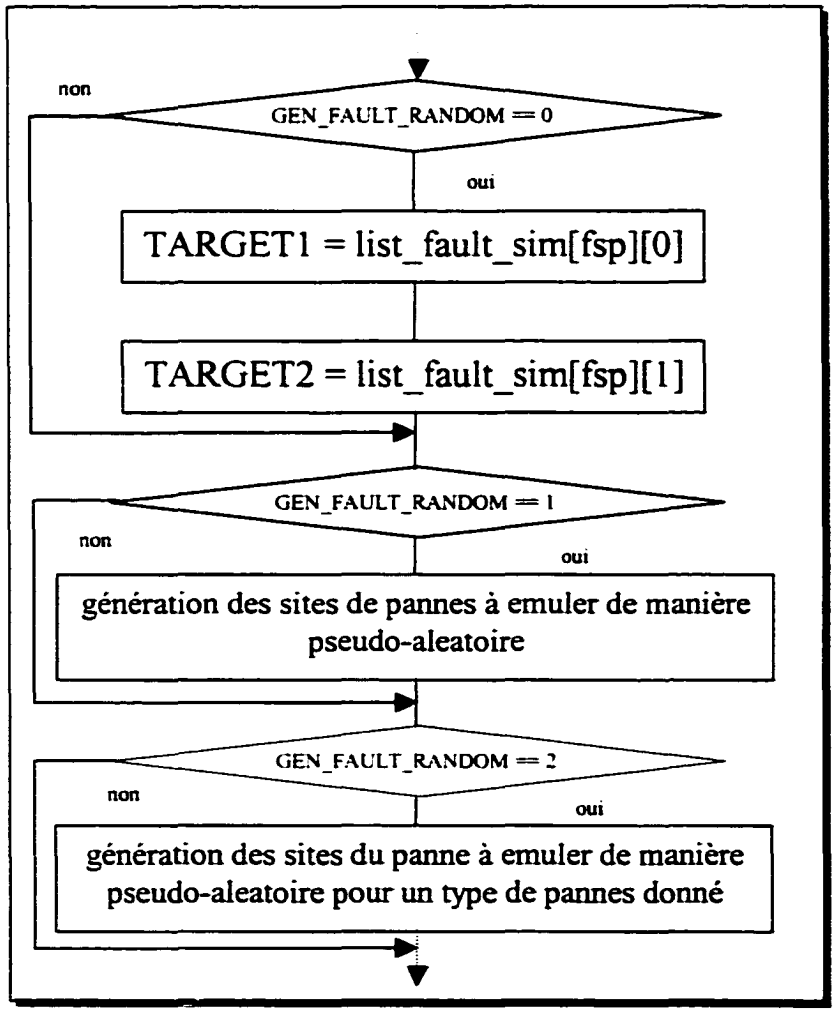


Figure 49 Organigramme du choix d'un site de panne à émuler

➢ **Cas 1** GEN_FAULT_RANDOM = 0 :

Les sites physiques potentiels sont choisies par l'utilisateur dans un vecteur « list_fault_sim ». Dans ce cas, le programme utilise directement cette liste.

➢ **Cas 2** GEN_FAULT_RANDOM = 1 :

La génération des sites de pannes à émuler se fait de manière *pseudo-aleatoire*. On peut emuler des pannes de types différents dans ce cas.

➢ **Cas 3** GEN_FAULT_RANDOM = 2 :

Dans ce cas, c'est l'utilisateur qui définit le type de panne à émuler, mais la génération des sites se fait de manière pseudo-aléatoire pour ce type de panne.

1.1.3.4. Simulation logique et émulation du courant I_{DDQ}

Le programme d'émulation de la méthode se divise en trois parties distinctes. La première partie est responsable de la simulation logique du circuit, ainsi que l'émulation du courant de consommation suite à une panne dans ce circuit. Ceci est fait en calculant le courant de source I_{DDQ} pour une série de vecteurs d'entrées. Le simulateur peut donc remplacer un vrai circuit en se basant sur son netlist.

La possibilité d'ajout d'un bruit gaussien ayant un écart type passé en paramètres, permet de donner plus de réalisme à la méthode. On sait que dans un environnement de test, il y a toujours des interférences qui peuvent affecter les mesures du courant prises par un testeur, et qui doivent être prises en considération.

La deuxième partie permet d'identifier la liste des pannes les plus probables suite à un court circuit entre deux sorties de portes logiques en conflit et calcule de ΔI_{DDQ} correspondant.

La troisième partie a le rôle de confirmer le type de pannes le plus probable identifié à la deuxième partie et après de le localiser. Dans le cas de rejet du type panne le plus probable, l'algorithme passe au type de panne suivant dans la liste.

Le programme permet de choisir la source des valeurs ΔI_{DDQ}. Tel qu'indiqué à la figure 50, la variable optionnelle « *SOURCE_DELTA* » offre le choix entre l'utilisation du simulateur intégré du programme *diag_total* ou bien l'utilisation des mesures de courant prises par un testeur et sauvegardées dans un fichier d'entrée pour le logiciel.

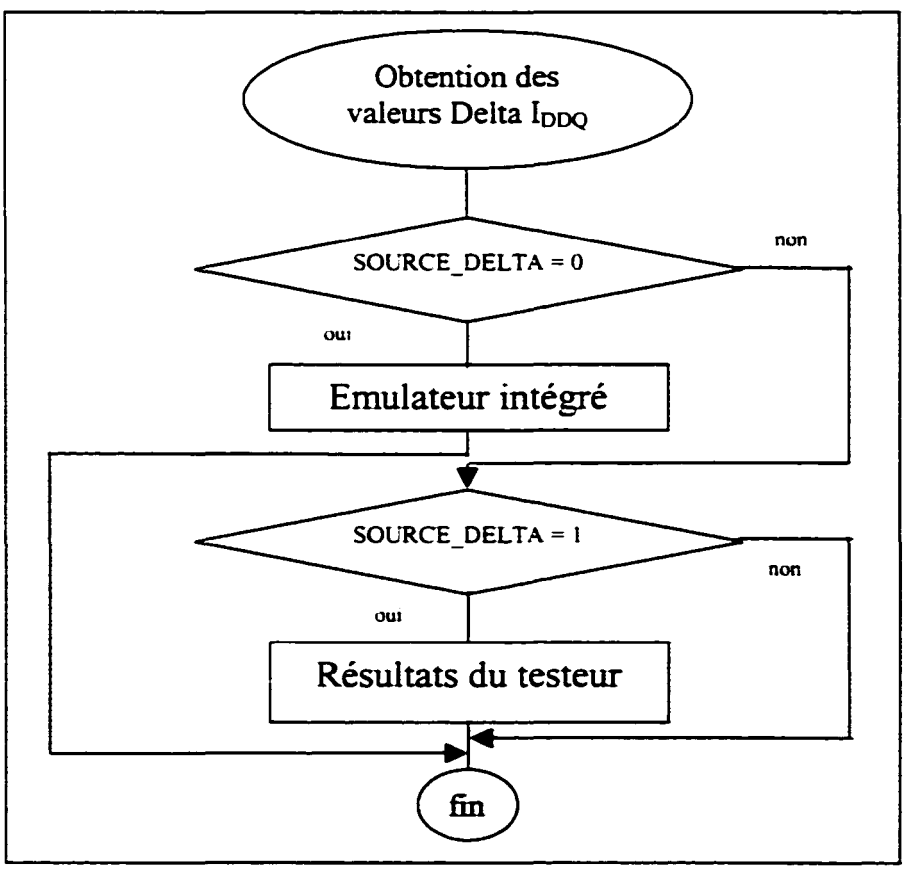


Figure 50 Choix de la source des valeurs ΔI_{DDQ}

Après le choix d'un couple de site à émuler, on passe à l'étape de simulation logique (voir la figure 51) qui consiste à injecter une série de vecteurs pseudo-aléatoires sur les nœuds d'entrées du circuit. La représentation du graphe du circuit tel qu'expliqué à la section 1.2.3 permet l'utilisation des algorithmes de récursivité et de parcours de graphe à fin de mettre à jour tous les nœuds du circuit représenté par son netlist. Chaque entité du netlist donc du graphe est caractérisé par son type de porte. À l'aide de ce type, le simulateur logique applique une simple table de vérité au(x) entrée(s) et au(x) sortie(s) de l'entité.

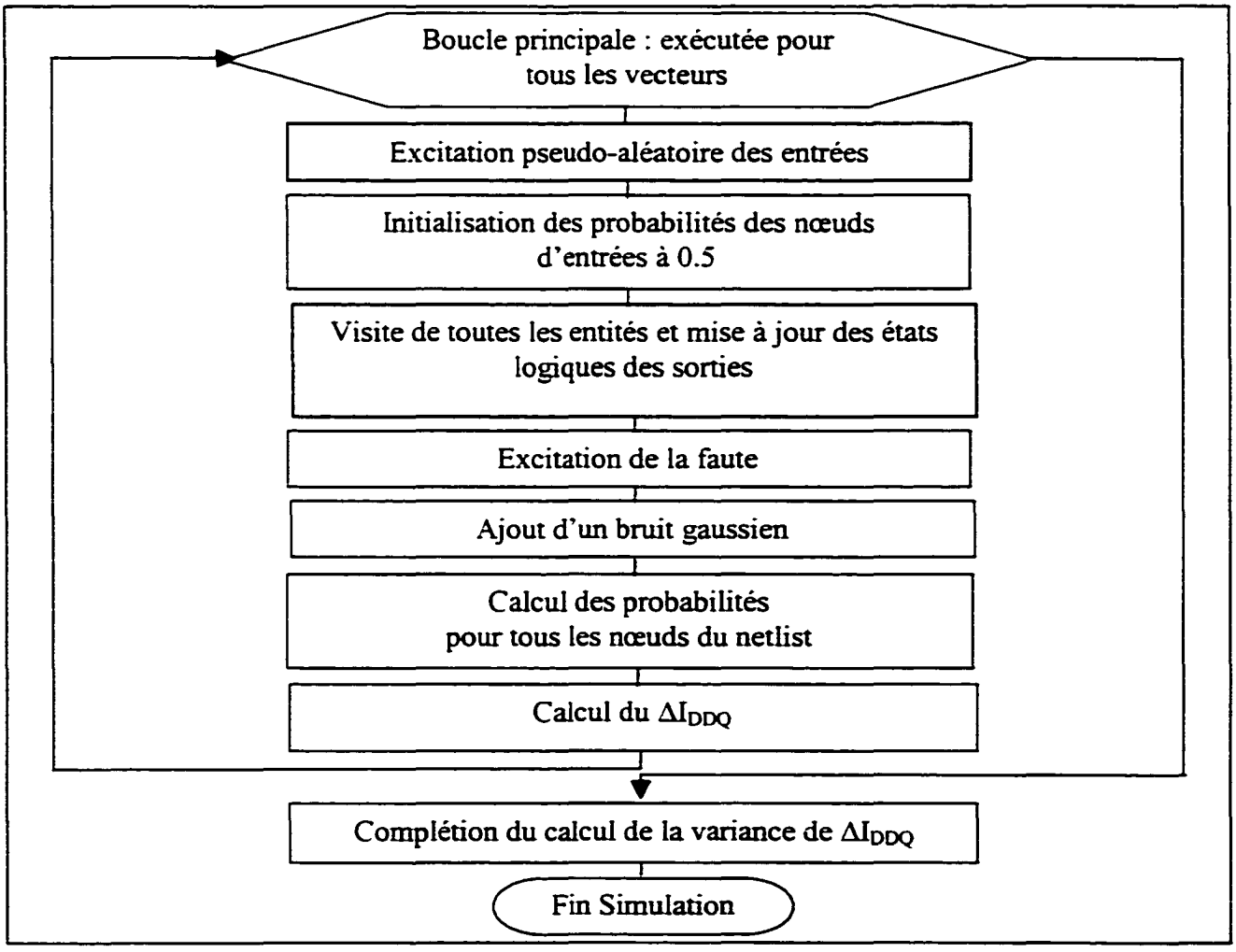


Figure 51 Simulation logique et émulation du courant

1.1.3.5. La phase d'identification des types de pannes les plus probables

Après l'émulation du courant I_{DDQ} pour une panne donnée par son type et sa localisation, on passe à l'étape d'identification des types de pannes les plus probables. Le courant inclut le courant de court-circuit calculé par la simulation logique, le courant du champ de pénétration, et le courant de bruit gaussien.

Le programme calcule la variation de ce courant en faisant la différence entre la valeur du courant au vecteur d'entrée courant et sa valeur au vecteur précédent, et compare le ΔI_{DDQ} résultant aux signatures des différentes pannes considérées.

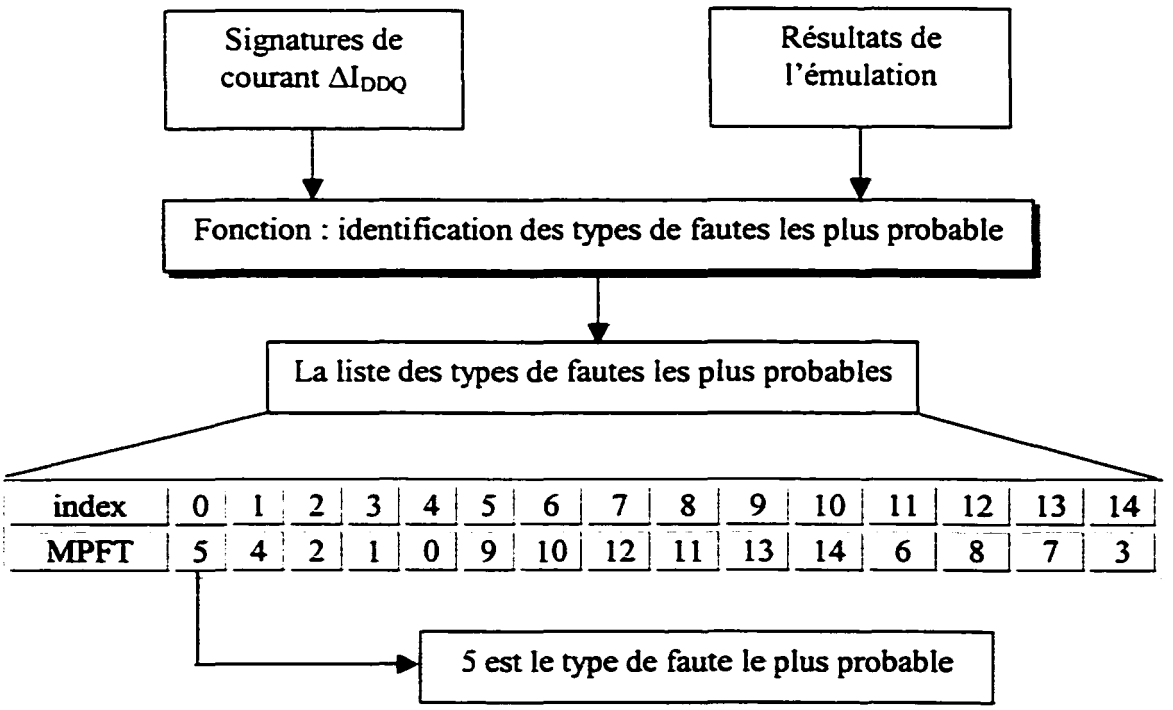


Figure 52 Identification des types de pannes les plus probables

La figure 52 montre les entrées du processus d'identification des types de pannes les plus probables. La fonction qui assure l'identification effectue une comparaison entre le ΔI_{DDQ} émulé et les signatures déjà préparées. Le principe du MLE est appliqué afin d'ordonner les types de pannes suivant leur probabilité d'être une des pannes actuelles, on appelle la liste résultante, la liste des pannes les plus probables qui sera passée par la suite à une autres fonction qui localise la panne simulée, en commençant par le type de panne le plus probable.

1.1.3.6. La phase de localisation

Le programme entre dans la dernière phase. Comme son nom l'indique, la phase de localisation (voir figure 53) consiste à confirmer le type de panne et de le localiser par la suite. Le principe de cette fonction est simple. Dans un premier temps, le programme définit une liste de localisations correspondant au type de panne le plus probable. Ceci est fait grâce à une fonction qui parcoure tout le netlist et crée une liste appelée : la liste des sites probables, qui n'est autre que l'ensemble des paires de nœuds candidats au type de panne le plus probable. Il ne reste alors qu'à comparer la valeur réelle ou simulée du ΔI_{DDQ} à la valeur estimée du courant pour chaque position possible de la panne dans le graphe donc le circuit.

Si aucune localisation dans la liste des sites physiques ne rencontre le critère de sélection pour un type de panne particulier, on rejette ce type de panne et on repart avec le suivant dans la liste.

La création des sites physiques probables :

Àpres l'étude approfondie de la fonction responsable de la création des sites physiques probables, on constate qu'elle constitue un goulot d'étranglement remarquable dans le programme ***diag_total***.

Cette fonction alimente la fonction ***qui localise les pannes*** par toutes les combinaisons possibles concernant un type de panne donné, dans notre cas le type de panne le plus probable.

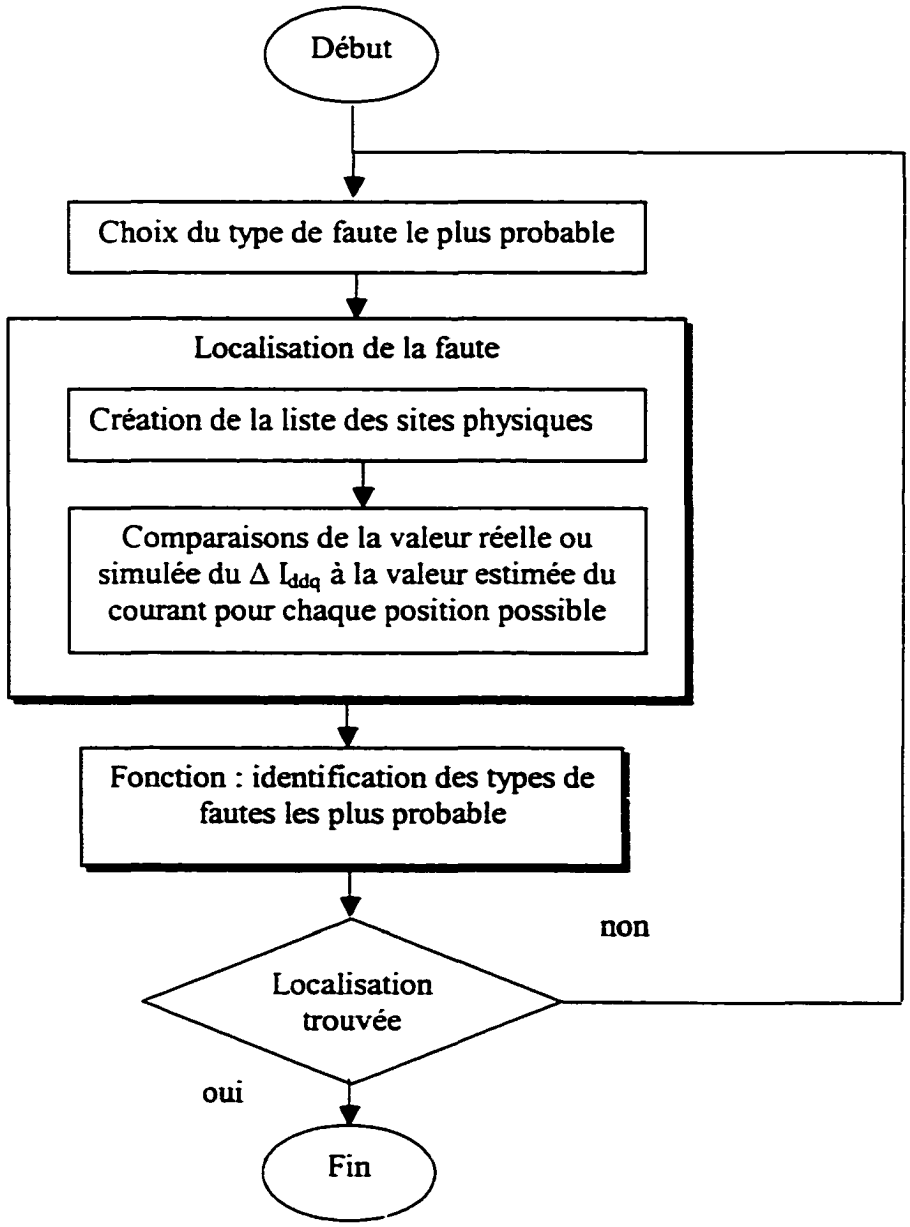


Figure 53 Localisation des pannes

1.1.3.7. Présentation de quelques résultats

Les résultats de simulation de l'annexe 3 montrent l'efficacité de la méthode de diagnostic basé sur des signatures probabilistique du courant I_{DDQ}. Àpres une étude approfondie de la méthode, on constate que cette dernière peut être sujette à amélioration.

ANNEXE 3

Les résultats d'émulation

Les fichiers d'entrée :

➧ Le fichier netlist.dat en format « TIBO » (nom donné par le concepteur):

```
nombre noeuds 117
nombre entites 109
nombre entrees primaires 8
entrees primaires 137 139 141 143 145 151 153 159
nombre sorties primaires 8
sorties primaires 111 113 117 121 123 125 131 130

entite AND
nb_d'entrees 2
entrees 145 137
nb_de_sorties 1
sortie 111

entite AND
nb_d'entrees 2
entrees 151 143
nb_de_sorties 1
sortie 12

entite AND
nb_d'entrees 2
entrees 153 143
nb_de_sorties 1
sortie 50

                                        ***


entite NOT
nb_d'entrees 1
entrees 132
nb_de_sorties 1
sortie 131

entite OR
nb_d'entrees 2
entrees 65  29
nb_de_sorties 1
sortie 0
```

- Le fichier sim_param.dat :

```
nb_vect 20              /* nombre de vecteur de simulation */
TECH 3                  /* technologie cible
                              - 2 = CMOS4S (0.8um)
                              - 3 = CMOS5is (0.5um) */
STD_DEV 10              /* écart type du bruit gaussien rajouté a IDDQ */
```

- Le fichier donnee_diag.yas :

```
FAULT_SET       2       /* voir page suivante */
fault_type_sim  5       /* voir page suivante */
nb_fault_sim    100     /* nombre de pannes simulées (bridge) */
nb_iter_min     1       /* nombre d'itérations minimum avant de pouvoir
                           rejeter une localisation potentielle de panne */
Th_loc          70      /* seuil de rejet des localisations potentielles de panne */
FL_SIMPLIFIED   0       /* clé permettant l'exploration de la simplification de la
                           localisation des pannes :
                           0 = pas de simplification
                           1 = simplification; consiste à modifier le calcul du delta
                           IDDQ, qui devient le niveau IDDQ du vecteur actuel moins
                           un offset correspondant à la moyenne du
                           niveau IDDQ; cette option reste à rafiner */

PRINT_NOEUDS 0          /* clé permettant l'impression la valeur des noeuds
                           impliqués dans le court-circuit (noeud1, noeud2) */
noeud1 65               /* noeud impliqué dans le court-circuit */
noeud2 29               /* noeud impliqué dans le court-circuit */
KEEP_SAME_VECT 1        /* clé permettant le réutilisation des mêmes
                           vecteurs pseudo-aléatoires dans la simulation
                           logique/IDDQ (do_logic_sim) pour fin de comparaison */
```

➢ Les types de pannes considérées

Fault	Description
TF0	INV-INV
TF1	INV-NAND2
TF2	INV-NAND3
TF3	INV-NAND4
TF4	NAND2-NAND2
TF5	NAND2-NAND3
TF6	NAND2-NAND4
TF7	NAND3-NAND3
TF8	NAND3-NAND4
TF9	NAND4-NAND4

NB : Dans le programme ***diag_total.c***, on dispose de deux variables, qui sont :

- FAULT_SET : catégorie de la panne simulée

- 0 = s@0
- 1 = s@1
- 2 = bridge (entre noeuds ni VDD ni GND)

- fault_type_sim : type de panne simulée (bridge)

- 0 = inv-inv
- 1 = inv-nand2
- 2 = inv-nand3
- 3 = inv-nand4
- 4 = nan2-nand2
- 5 = nand2-nand3
- 6 = nand2-nand4
- 7 = nand3-nand3
- 8 = nand3-nand4
- 9 = nand4nand4

Les fichiers de sortie

- Le fichier donnee_diag.yas :

```
*-* PROGRAMME DE DIAGNOSTIC *-*
  fsp / nb_fault_sim --> <0>/<100> /* numéro de simulation */
   * fsp          =    0
   * TARGET1      =   76
   * TARGET2      =   51
   * fault_type_sim =    5
   * fault_indep   =    1
   * p1[TARGET1]   = 0.7500
   * p1[TARGET2]   = 0.6500
   * moy_delta     = 0.1282
   * sqrt(var_delta)=  15.0

        i= 0 rank_sf[i]= 3 mpft_sf[i]= 5
        i= 1 rank_sf[i]= 4 mpft_sf[i]= 7
        i= 2 rank_sf[i]= 6 mpft_sf[i]= 6
        i= 3 rank_sf[i]= 5 mpft_sf[i]= 0
        i= 4 rank_sf[i]= 7 mpft_sf[i]= 1
        i= 5 rank_sf[i]= 0 mpft_sf[i]= 3
        i= 6 rank_sf[i]= 2 mpft_sf[i]= 2
        i= 7 rank_sf[i]= 1 mpft_sf[i]= 4
        i= 8 rank_sf[i]= 8 mpft_sf[i]= 8
        i= 9 rank_sf[i]= 9 mpft_sf[i]= 9
        i=10 rank_sf[i]=12 mpft_sf[i]=12
        i=11 rank_sf[i]=13 mpft_sf[i]=13
        i=12 rank_sf[i]=10 mpft_sf[i]=10
        i=13 rank_sf[i]=11 mpft_sf[i]=11
        i=14 rank_sf[i]=14 mpft_sf[i]=14
DIAGNOSTIC COMPLÉTÉ
i  = 76 j  = 51 sumddfp[i][j]/nb_vect      = 0.000000e+000 iter*Th_loc=   10

                                          ***

  fsp / nb_fault_sim --> <100>/<100>
   * fsp          =    0
   * TARGET1      =   70
   * TARGET2      =   50
   * fault_type_sim =    5
   * fault_indep   =    1
   * p1[TARGET1]   = 0.7500
   * p1[TARGET2]   = 0.6500
```

```
 * moy_delta      = 0.1282
 * sqrt(var_delta)=  15.0

      i= 0 rank_sf[i]= 3 mpft_sf[i]= 5
      i= 1 rank_sf[i]= 4 mpft_sf[i]= 7
      i= 2 rank_sf[i]= 6 mpft_sf[i]= 6
      i= 3 rank_sf[i]= 5 mpft_sf[i]= 0
      i= 4 rank_sf[i]= 7 mpft_sf[i]= 1
      i= 5 rank_sf[i]= 0 mpft_sf[i]= 3
      i= 6 rank_sf[i]= 2 mpft_sf[i]= 2
      i= 7 rank_sf[i]= 1 mpft_sf[i]= 4
      i= 8 rank_sf[i]= 8 mpft_sf[i]= 8
      i= 9 rank_sf[i]= 9 mpft_sf[i]= 9
      i=10 rank_sf[i]=12 mpft_sf[i]=12
      i=11 rank_sf[i]=13 mpft_sf[i]=13
      i=12 rank_sf[i]=10 mpft_sf[i]=10
      i=13 rank_sf[i]=11 mpft_sf[i]=11
      i=14 rank_sf[i]=14 mpft_sf[i]=14

DIAGNOSTIC COMPLÉTÉ
i  = 70 j  = 501 sumddfp[i][j]/nb_vect     = 0.000000e+000 iter*Th_loc=  140



End
 * nb. fault simulees =  100

 * nb_iter_min =    1


nb de sites concernes  =  31
nb de sites potentiels  =  465
 -> avrg fault type rank  (best = 0) =  0.11 /   15


nb moyen de sites non rejetés =  1.0800
nb_iter_total moyen par fault_sim =      732.4
sumddfp_other_min per iter=   77.1
sumddfp_other_moy per iter=  2256.2
sumddfp_max  per iter =   36.2
sumddfp_tg_moy per iter=  2.1181
nb_cas_moy_tg = 1881
sumka2/nb_cas_ka2   =  1.0546 nb_cas_ka2   = 2020
```

nb_vect_moy (pour rejet des autres noeuds) = 1.41
nb_cas_moy = 50586

no_vec	prob. site	prop. iter_max	prop. iter_max_ot
1	7.4950e-001	0.0000e+000	3.0769e-001
2	1.5975e-001	0.0000e+000	7.6923e-002
3	5.5094e-002	4.0000e-002	0.0000e+000
4	2.1646e-002	5.0000e-002	1.5385e-001
5	8.0655e-003	1.6000e-001	7.6923e-002
6	1.7594e-003	1.0000e-001	0.0000e+000
7	1.7396e-003	1.3000e-001	0.0000e+000
8	3.3606e-004	6.0000e-002	0.0000e+000
9	1.2256e-003	2.4000e-001	0.0000e+000
10	4.9421e-004	8.0000e-002	1.5385e-001
11	2.9652e-004	8.0000e-002	7.6923e-002
12	0.0000e+000	0.0000e+000	0.0000e+000
13	1.9768e-005	1.0000e-002	0.0000e+000
14	3.9537e-005	2.0000e-002	7.6923e-002
15	0.0000e+000	0.0000e+000	0.0000e+000
16	1.9768e-005	1.0000e-002	0.0000e+000
17	0.0000e+000	0.0000e+000	0.0000e+000
18	0.0000e+000	0.0000e+000	0.0000e+000
19	1.9768e-005	1.0000e-002	7.6923e-002

➢ **Le fichier resultats_diag.yas :**

```
        *-* PROGRAMME DE DIAGNOSTIC *-*


- le nom du programme : diag_total.c

1 - read_sim_param() < Fichier : sim_param.dat >
 * nb de vecteurs de la simulation      = 20
 * technologie cible  < CMOS5is (0.5um) > = 3
 * ecart type du bruit gaussien rajouté = 10

2 - read_net_create_graph() < Fichier : netlist.dat >
  * nb_noeuds      =  116
  * nb_entites     =  108
  * nb_entrees_prim =    8
  * nb_sorties_prim =    8

3 - read_panne_param() < Fichier : panne_param.dat >
 * FAULT_SET      = 2   < bridge >
 * fault_type_sim  = 5   < nand2-nand2 >
 * nb_fault_sim    = 100 < nombre de pannes simulées (bridge) >
 * nb_iter_min     = 1
 < nombre d'itérations minimum avant de pouvoir  rejeter une localisation potentielle de
panne >
 * Th_loc         = 70
 < seuil de rejet des localisations potentielles de panne >
 * FL_SIMPLIFIED   = 0
 *> clé permettant l'exploration de la simplification de la localisation des pannes
  -> 0 = pas de simplication
  -> 1 = simplification
 * PRINT_NOEUDS    = 0
 < clé permettant l'impression la valeur des  noeuds impliqués dans le court-circuit
(noeud1, noeud2) >
 * noeud1         = 22
 * noeud2         = 23
 * KEEP_SAME_VECT  = 1
 < clé permettant le réutilisation des mêmes  vecteurs pseudo-aléatoires dans la
simulation logique/I_DDQ  (do_logic_sim) pour fin de comparaison >

- Resultats du diagnostic
 * nb_iter_max moyen par fault_sim (si fault = faults-im)    =       6.3
```

ANNEXE 4

Les trois sections du fichier contenant les informations sur le dessin des masques

Section 1 : Les capacités parasites

La section-1 identifie toutes les capacités parasites extraites par l'outil « analog artist » de Cadence. Au niveau de chaque ligne, nous pouvons identifier les informations suivantes :

c238 (55 52) \capacitor c=3.59125071328526e-15

- nom ---------- c238
- (nœud 1 ref1) ---------- 55
- (nœud 2 ref2) ---------- 52
- valeur -------- 3.59125071328526e-15 farad.

NB : Il peut y avoir une redondance de données dans les capacités parasites. L'existence de deux ou plusieurs capacités qui ont des noms différents, mais qui sont branchées aux mêmes deux nœuds est possible. Dans ce cas, il faut garder une seule capacité en calculant la somme des valeurs de toutes les capacités branchées aux mêmes deux nœuds.

Section 2 : Le netlist du layout

La section 2 donne le netlist de notre circuit sous forme d'une liste de cellules. Voici un exemple de la représentation tel que fournie par l'outil :

- xu18 (55 64 51 54 34) **wor2_1** g3

Il s'agit ici d'une porte à deux entrées. Nous utilisons le nom de la cellule (wor2_1) qui nous permet de retrouver la définition de son interface (section 3 du même fichier)

Section 3 : Les interfaces

Cette section contient plusieurs informations, dont la définition des interfaces des cellules qui apparaissent aux lignes commençant par le mot clef **subckt,** voici celle de la cellule citée en exemple à la section 2 :

Donc on peu faire l'accord entre les chiffres et les nœuds :

- 55 ---------- vdd
- 64 ---------- vss
- 51 ---------- ip1
- 54 ---------- ip2
- 34 ---------- op

On retrouve également dans la même section les capacités parasites internes des portes logiques. Ces capacités parasites sont les mêmes pour chaque type de porte. Nous les négligeons dans notre étude car la probabilité d'une faute externe (causée par le routage) est beaucoup plus importante d'une panne interne à la porte logique.

Voici un exemple de capacités parasites internes pour le type de porte « Ou » à deux entrées et une sortie :

- c22 (ip1 vss\!) capacitor c=116.809999181765e-18
- c24 (ip2 vss\!) capacitor c=116.809999181765e-18
- c26 (op vdd\!) capacitor c=288.694980566512e-18
- c28 (ip1 ip2) capacitor c=190.620000219252e-18
- c30 (ip2 vdd\!) capacitor c=177.919998489705e-18

ANNEXE 5

Les fichiers de sorties du programme d'automatisation de l'extraction des capacités parasites

Les capacités parasites

Le fichier parasite.yas

```
capacite --<1>--
nom    : ..c188..
ref1   : ..55..
ref2   : ..64..
valeur : ..1.195151e-13..
  capacite --<2>--
nom    : ..c160..
ref1   : ..64..
ref2   : ..54..
valeur : ..3.489639e-15..
  capacite --<3>--
nom    : ..c164..
ref1   : ..64..
ref2   : ..53..
valeur : ..8.921005e-16..
---
 capacite --<22>--
nom    : ..c148..
ref1   : ..50..
ref2   : ..49..
valeur : ..2.112500e-16..
  capacite --<23>--
nom    : ..c126..
ref1   : ..48..
ref2   : ..34..
valeur : ..1.417200e-16..
```

Les cellules

Le fichier cell_out.yas

```
cellule --<1>--
nam     : ..wpadoutld..
xname   : ..xu12..
wname   : ..wpadoutld_g1..
interface: ..49 69 68 67 66 65 10 ..
cellule --<2>--
nam     : ..wpadoutld..
xname   : ..xu13..
wname   : ..wpadoutld_g1..
interface: ..34 20 19 18 66 17 11 ..
cellule --<3>--
nam     : ..wpadoutld..
xname   : ..xu14..
wname   : ..wpadoutld_g1..
interface: ..48 20 19 18 66 17 14 ..
---
cellule --<10>--
nam     : ..wnor2_1..
xname   : ..xu21..
wname   : ..wnor2_1_g5..
interface: ..55 64 52 51 50 ..
cellule --<11>--
nam     : ..winv_1..
xname   : ..xu22..
wname   : ..winv_1_g6..
interface: ..55 64 54 53 ..
```

Les interfaces

Le fichier inter_out.yas

```
wnor2_1_g5 vdd\! vss\! ip1 ip2 op
wand2_1_g4 vdd\! vss\! ip1 ip2 op
wor2_1_g3 vdd\! vss\! ip1 ip2 op
wpadin_g2 op subcore subesd subring vddcore vddring world
wpadoutld_g1 ip subcore subesd subring vddcore vddring world
winv_1_g6 vdd\! vss\! ip op
```

Les nœuds

Le fichier Nodes_out.yas

```
69
68
67
66
65
64
55
54 xu15 op xu18 ip2 xu22 ip
53 xu20 ip2 xu22 op
52 xu16 op xu19 ip1 xu21 ip1
51 xu17 op xu18 ip1 xu19 ip2 xu21 ip2
50 xu20 ip1 xu21 op
49 xu12 ip xu20 op
48 xu14 ip xu19 op
34 xu13 ip xu18 op
32
30
28
26
24
21
20
19
18
17
16
14
11
10
```

Résumé de l'extraction

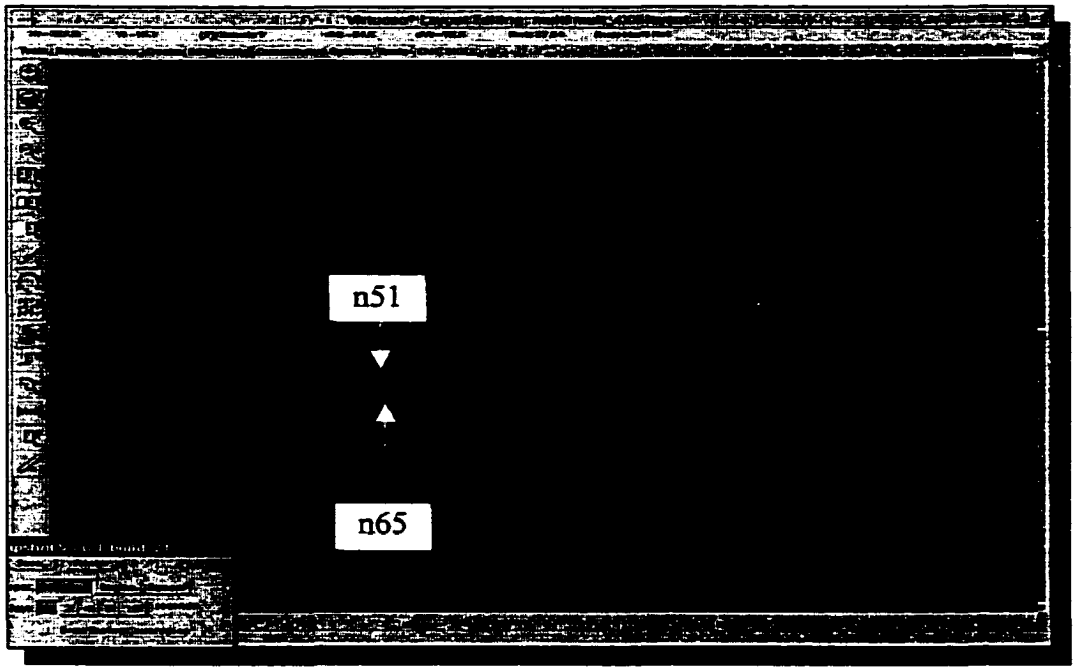
Le fichier resume_out.yas

.....Resume de l'extraction.....
Nombre Total de capacités de routages : 36
Nombre Total de nœuds : 29
Nombre Total de cellules : 11

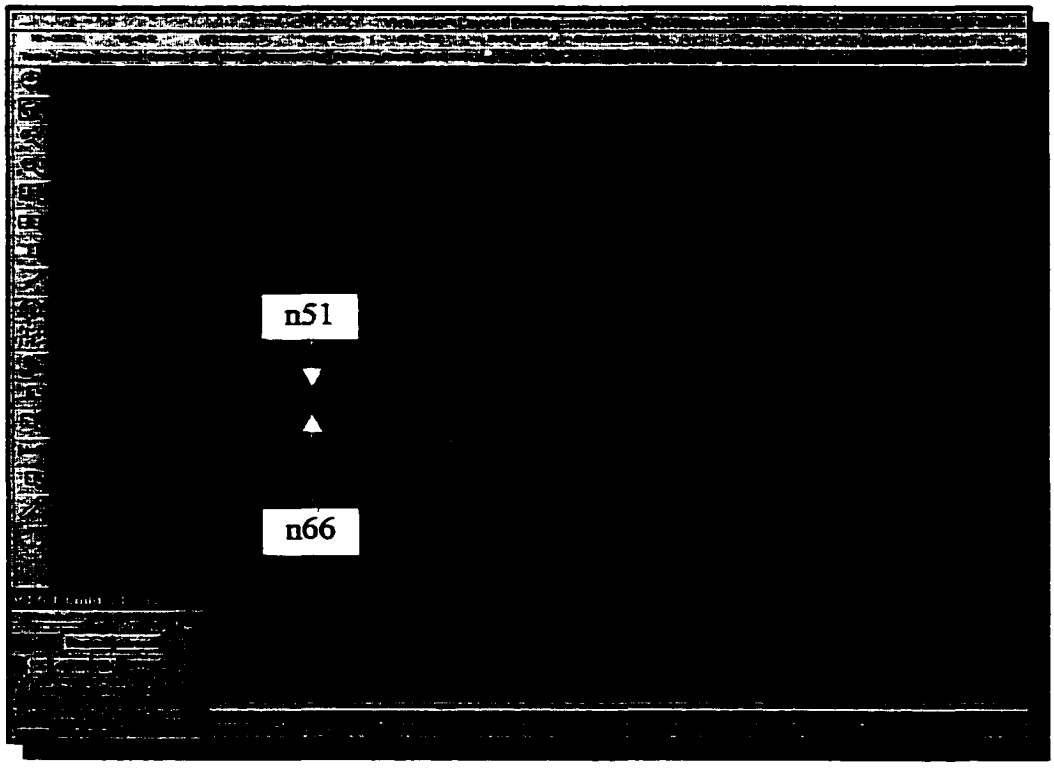
ANNEXE 6

Les sites candidats sur le « layout »

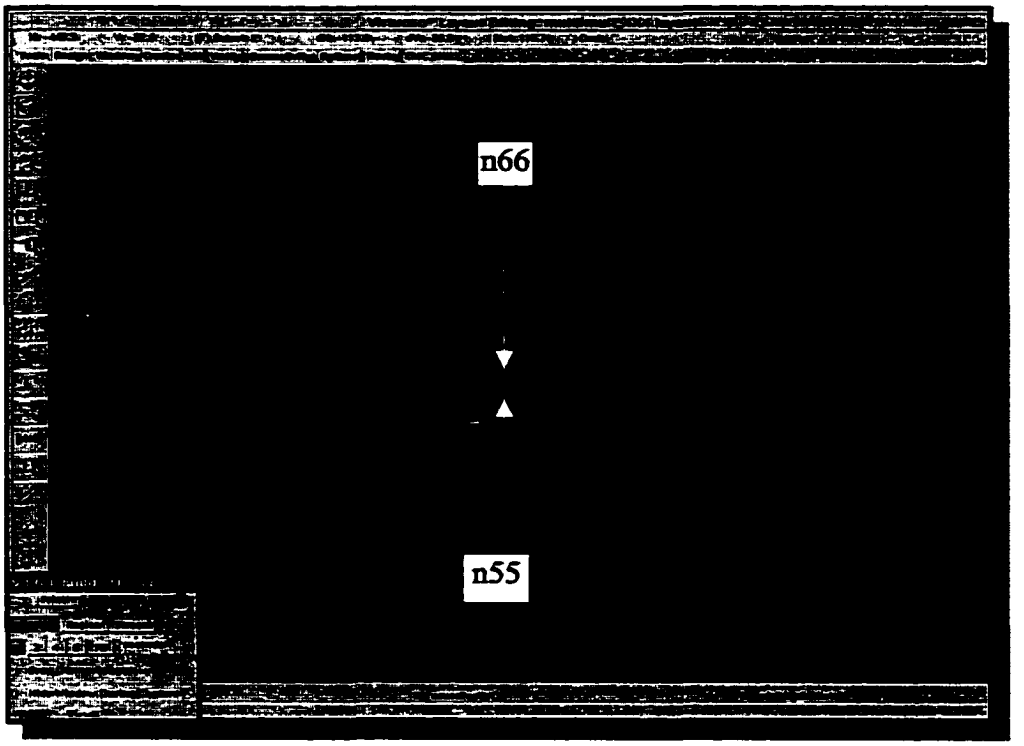
➢ ..n51.._..65.. <-> ..n65.._..25.. * capa = 6.853140e-16



➢ ..n51.._..65.. <-> ..n66.._..29.. * capa = 2.534561e-16



➢ ..n55.._..97.. <-> ..n66.._..29.. * capa = 6.052765e-16



ANNEXE 7

Informations sur les circuits sous test

Circuit 1 : Multiplicateur 4X4

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit en micron^2
Multiplicateur 4bits	94	117	8	8	$2x10^5$

- **Les types de portes utilisées :**
  - Nombre de cellules reconnues : 94
  - Nombre de cellules wpadin : 8
  - Nombre de cellules wpadout : 8
  - Nombre de cellules wand2_1 : 6
  - Nombre de cellules winv_1 : 10
  - Nombre de cellules winv_2 : 1
  - Nombre de cellules wmux2_2 : 10
  - Nombre de cellules wnand2_1 : 31
  - Nombre de cellules wnand3_1 : 3
  - Nombre de cellules wnor2_1 : 2
  - Nombre de cellules wor2_1 : 5
  - Nombre de cellules wxor2_2 : 10
- Nombre Total de capacités de routages : 600
- Le layout du circuit :

➢ ***Code VHDL du circuit :***

```
LES ARRAYS MULTIDIMENTIONNELLES NE SONT PAS SUPPORTEES EN SYNTHESE
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity mult_4X4 is
port (  ina : in std_logic_vector (3 downto 0);

inb : in std_logic_vector (3 downto 0);

prod : out std_logic_vector (7 downto 0));

end mult_4X4;

architecture behav of mult_4X4 is

(x,y) == (ligne, colonne)
signal ci0, ci1, ci2, ci3 :  std_logic_vector (3 downto 0); -- un vecteur par ligne, carry in
signal co0, co1, co2, co3 :  std_logic_vector (3 downto 0); -- un vecteur par ligne, carry out
signal si0, si1, si2, si3 :  std_logic_vector (3 downto 0); -- un vecteur par ligne, sum in
signal so0, so1, so2, so3 :  std_logic_vector (3 downto 0); -- un vecteur par ligne, sum in
signal pt0, pt1, pt2, pt3 :  std_logic_vector (3 downto 0); -- un vecteur par ligne, terme produit
begin
multiplication : process(ina, inb, ci0, ci1, ci2, ci3, co0, co1, co2, co3,
si0, si1, si2, si3, so0, so1, so2, so3, pt0, pt1, pt2, pt3)
variable i, j : integer;
begin
sum in de la  ligne  x=0
FOR j IN 0 to 3 LOOP -- boucle sur l'axe des y (operande B)
si0(j) <= '0' ;
END LOOP;
carry in de la  colonne y=0
ci0(0) <= '0';
ci1(0) <= '0';
ci2(0) <= '0';
ci3(0) <= '0';
FOR j IN 0 to 3 LOOP -- boucle sur l'axe des y (operande B)
pt0(j) <= inb(j) and ina(0) ;
pt1(j) <= inb(j) and ina(1) ;
pt2(j) <= inb(j) and ina(2) ;
```

```
pt3(j) <= inb(j) and ina(3) :
so0(j) <= ci0(j) xor si0(j) xor pt0(j);so1(j) <= ci1(j) xor si1(j) xor pt1(j);
so2(j) <= ci2(j) xor si2(j) xor pt2(j);
so3(j) <= ci3(j) xor si3(j) xor pt3(j);
co0(j) <= ((si0(j) xor pt0(j)) and ci0(j)) or (si0(j) and pt0(j));
co1(j) <= ((si1(j) xor pt1(j)) and ci1(j)) or (si1(j) and pt1(j));
co2(j) <= ((si2(j) xor pt2(j)) and ci2(j)) or (si2(j) and pt2(j));
co3(j) <= ((si3(j) xor pt3(j)) and ci3(j)) or (si3(j) and pt3(j));
END LOOP; FOR j IN 1 to 3 LOOP -- boucle sur l'axe des y (operande B)
ci0(j) <= co0(j-1);
ci1(j) <= co1(j-1);
ci2(j) <= co2(j-1);
ci3(j) <= co3(j-1);
END LOOP;
FOR j IN 0 to 2 LOOP -- boucle sur l'axe des y (operande B)
si1(j) <= so0(j+1);
si2(j) <= so1(j+1);
si3(j) <= so2(j+1);
END LOOP;
si1(3) <= co0(3);
si2(3) <= co1(3);
si3(3) <= co2(3);
prod(0) <= so0(0);
prod(1) <= so1(0);
prod(2) <= so2(0);
prod(3) <= so3(0);
FOR j IN 1 to 3 LOOP -- boucle sur l'axe des y (operande B)
prod(j+3) <= so3(j);
END LOOP;
prod(7) <= co3(3);
end process multiplication;
end behav;
```

Circuit 2 : Filtre Fir 4 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit en micron^2
Fir 4 étages	901	986	42	8	$14x10^5$

➢ **Les types de portes utilisées :**

- ❖ Nombre de cellules wpadin : 42
- ❖ Nombre de cellules wpadout : 8
- ❖ Nombre de cellules wand2_1 : 57
- ❖ Nombre de cellules wand3_1 : 2
- ❖ Nombre de cellules wbuf_1 : 3
- ❖ Nombre de cellules winv_1 : 74
- ❖ Nombre de cellules winv_2 : 1
- ❖ Nombre de cellules wmux2_2 : 27
- ❖ Nombre de cellules wnand2_1 : 268
- ❖ Nombre de cellules wnand3_1 : 35
- ❖ Nombre de cellules wnand4_1 : 4
- ❖ Nombre de cellules wnor2_1 : 105
- ❖ Nombre de cellules wnor3_1 : 7
- ❖ Nombre de cellules wor2_1 : 51
- ❖ Nombre de cellules wxor2_2 : 193
- ❖ Nombre de cellules wdrp_2 : 24

➢ Nombre Total de capacités de routages : 9027

➢ Le layout du circuit :

➢ ***Code VHDL du circuit :***

```
-- filtre 2 ieme strategie de synchronisation
-- fichier filfir4.vhd
-- auteur:  HARIRI YASSINE && ABDELMOUMEN MOAKI BENANI
-- date de conception : 15/10/2000

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity filfir4 is
    port (
        reset: in STD_LOGIC;
        x_ready: in STD_LOGIC;
        x: in signed (7 downto 0);
        h: in signed (31 downto 0);
        y: out signed (7 downto 0)

    );
end filfir4;

architecture filfir4_arch of filfir4 is

component reg_8bits
   port (
          clk : in std_logic;
          reset: in std_logic;
          din : in signed ( 7 downto 0);
          dout : out signed (7 downto 0));
end component;


component multc2

      port(

          ina : in signed (7 downto 0);
          inb : in signed (7 downto 0);
          pd : out signed (7 downto 0));
   end component;

component add8bits
   port (
          ina : in signed(7 downto 0);
          inb : in signed(7 downto 0);
          sum : out signed(7 downto 0)
          );
end
component;
```

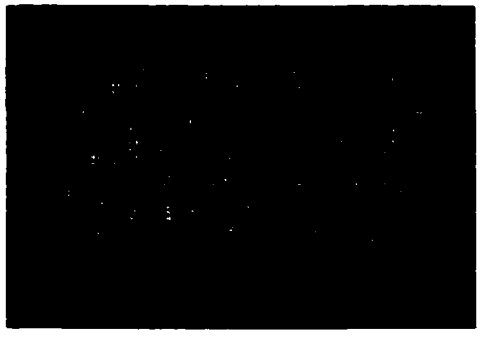
```
--definition des signaux interm.

SIGNAL h0,h1,h2,h3:signed(7 downto 0);
SIGNAL pd0,pd1,pd2,pd3:signed(7 downto 0);
SIGNAL t0,t1,t2,t3,t4:signed(7 downto 0);

begin

--initialisation des coeff

h0<=h(7  downto 0);
h1<=h(15 downto 8);
h2<=h(23 downto 16);
h3<=h(31 downto 24);

reg0:reg_8bits
   port map
   (clk=>x_ready,
    reset=>reset,
    din=>pd0,
    dout=>t0
   );

reg1:reg_8bits
   port map
   (clk=>x_ready,
    reset=>reset,
    din=>t1,
    dout=>t2
   );

reg2:reg_8bits
   port map
   (clk=>x_ready,
    reset=>reset,
    din=>t3,
    dout=>t4
   );

multi0:multc2
   port map
   ( ina=>x,
     inb=>h3,
     pd=>pd0
    );

multi1:multc2
   port map
   ( ina=>x,
     inb=>h2,
     pd=>pd1
    );
```

```
multi2:multc2
   port map
   ( ina=>x,
     inb=>h1,
     pd=>pd2
    );

multi3:multc2
   port map
   ( ina=>x,
     inb=>h0,
     pd=>pd3
    );

add1:add8bits
   port map
    (ina =>pd1,
     inb =>t0,
     sum =>t1
    );


add2:add8bits
  port map
    (ina =>pd2,
     inb =>t2,
     sum =>t3

     );

add3:add8bits
  port map
    (ina =>pd3,
     inb =>t4,
     sum =>y
     );

end filfir4_arch;
```

Circuit 3 : Filtre Fir 7 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit en micron^2
Fir 7 étages	1641	1772	66	8	$24x10^5$

- ***Les types de portes utilisées***
  - Nombre de cellules wpadin : 66
  - Nombre de cellules wpadout : 8
  - Nombre de cellules wand2_1 : 76
  - Nombre de cellules wand2_2 : 14
  - Nombre de cellules wand3_1 : 1
  - Nombre de cellules wbuf_1 : 1
  - Nombre de cellules winv_1 : 158
  - Nombre de cellules winv_2 : 35
  - Nombre de cellules wmux2_2 : 61
  - Nombre de cellules wnand2_1 : 541
  - Nombre de cellules wnand2_2 : 14
  - Nombre de cellules wnand3_1 : 13
  - Nombre de cellules wnor2_1 : 71
  - Nombre de cellules wnor2_2 : 42
  - Nombre de cellules wnor3_1 : 14
  - Nombre de cellules wor2_1 : 114
  - Nombre de cellules wor3_1 : 6
  - Nombre de cellules wxor2_2 : 358
  - Nombre de cellules wdrp_2 : 48
- Nombre Total de routages : 14599
- Le layout du circuit :



➢ ***Code VHDL du circuit :***

```
-- filtre 2 ieme strategie de synchronisation
-- fichier filfir7.vhd
-- auteur:  HARIRI YASSINE && ABDELMOUMEN MOAKI BENANI
-- date de conception : 15/10/2000

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity filfir7 is
    port (
        reset: in STD_LOGIC;
        x_ready: in STD_LOGIC;
        x: in signed (7 downto 0);
        h: in signed (62 downto 0);
        y: out signed (7 downto 0)

    );
end filfir7;

architecture filfir7_arch of filfir7 is

component reg_8bits
   port (
          clk : in std_logic;
          reset: in std_logic;
          din : in signed ( 7 downto 0);
          dout : out signed (7 downto 0));
end component;


component multc2

      port(

          ina : in signed (7 downto 0);
          inb : in signed (7 downto 0);
          pd : out signed (7 downto 0));
   end component;

component add8bits
   port (
          ina : in signed(7 downto 0);
          inb : in signed(7 downto 0);
          sum : out signed(7 downto 0)
          );
end
component;
```

```
--definition des signaux interm.

SIGNAL h0,h1,h2,h3:signed(7 downto 0);
SIGNAL pd0,pd1,pd2,pd3:signed(7 downto 0);
SIGNAL t0,t1,t2,t3,t4:signed(7 downto 0);

begin

--initialisation des coeff

h0<=h(7  downto 0);
h1<=h(15 downto 8);
h2<=h(23 downto 16);
h3<=h(31 downto 24);

reg0:reg_8bits
   port map
   (clk=>x_ready,
    reset=>reset,
    din=>pd0,
    dout=>t0
   );

reg1:reg_8bits
   port map
   (clk=>x_ready,
    reset=>reset,
    din=>t1,
    dout=>t2
   );

reg2:reg_8bits
   port map
   (clk=>x_ready,
    reset=>reset,
    din=>t3,
    dout=>t4
   );

multi0:multc2
   port map
   ( ina=>x,
     inb=>h3,
     pd=>pd0
    );

multi1:multc2
   port map
   ( ina=>x,
     inb=>h2,
     pd=>pd1
    );
```

```
multi2:multc2
   port map
   ( ina=>x,
     inb=>h1,
     pd=>pd2
    );

multi3:multc2
   port map
   ( ina=>x,
     inb=>h0,
     pd=>pd3
    );

add1:add8bits
   port map
    (ina =>pd1,
     inb =>t0,
     sum =>t1
    );


add2:add8bits
  port map
    (ina =>pd2,
     inb =>t2,
     sum =>t3

     );

add3:add8bits
  port map
    (ina =>pd3,
     inb =>t4,
     sum =>y
     );
---***---
end filfir7_arch;
```

Circuit 4 : Filtre Fir15 étages

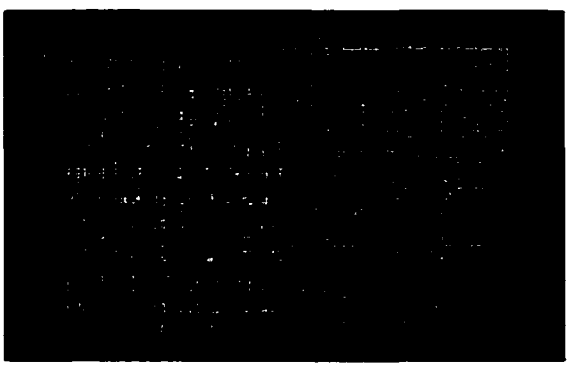
	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit en micron^2
Fir 15 étages	3521	3759	130	8	$40x10^5$

➢ *Les types de portes utilisées*

- Nombre de cellules reconnues : 3521
- Nombre de cellules wpadin : 130
- Nombre de cellules wpadout : 8
- Nombre de cellules wand2_1 : 270
- Nombre de cellules wand2_2 : 30
- Nombre de cellules wbuf_1 : 8
- Nombre de cellules winv_1 : 358
- Nombre de cellules winv_2 : 1
- Nombre de cellules wmux2_2 : 123
- Nombre de cellules wnand2_1 : 975
- Nombre de cellules wnand2_2 : 15
- Nombre de cellules wnand3_1 : 103
- Nombre de cellules wnor2_1 : 337
- Nombre de cellules wnor2_2 : 45
- Nombre de cellules wnor3_1 : 14
- Nombre de cellules wor2_1 : 215
- Nombre de cellules wor2_2 : 14
- Nombre de cellules wxor2_2 : 763
- Nombre de cellules wdrp_2 : 112

➢ Nombre Total de capacités de routages : 46269

➢ Le layout du circuit :



➢ Code VHDL du circuit :

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity filfir15 is
    port (
        reset: in STD_LOGIC;
        x_ready: in STD_LOGIC;
        x: in signed (7 downto 0);
        h: in signed (119 downto 0);
        y: out signed (7 downto 0)

    );
end filfir15;

architecture filfir15_arch of filfir15 is


component multc2
      port(
          ina : in signed (7 downto 0);
          inb : in signed (7 downto 0);
          pd : out signed (7 downto 0));
   end component;

    component etage
         port (
                reset   : in STD_LOGIC;
                clk     : in STD_LOGIC;
                data    : in signed  (7 downto 0);
                res_in : in signed  (7 downto 0);
                coeff   : in signed  (7 downto 0);
                res_out: out signed (7 downto 0));
    end component;

 --definition des signaux interm.

SIGNAL
pd0,pd1,pd2,pd3,pd4,pd5,pd6,pd7,pd8,pd9,pd10,pd11,pd12,pd13:signed(7
downto 0);
SIGNAL h0,h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11,h12,h13,h14:signed(7
downto 0);

begin

h0<=h(7  downto 0);
h1<=h(15 downto 8);
h2<=h(23 downto 16);
h3<=h(31 downto 24);
h4<=h(39 downto 32);
```

```
h5<=h(47 downto 40);
h6<=h(55 downto 48);

h7<=h(63  downto 56);
h8<=h(71 downto 64);
h9<=h(79 downto 72);
h10<=h(87 downto 80);
h11<=h(95 downto 88);
h12<=h(103 downto 96);
h13<=h(111 downto 104);
h14<=h(119  downto 112);
      multi0:multc2
         port map
         (    ina=>x,
            inb=>h14,
            pd =>pd0
          );

       etage_0: etage port map
                 (
                reset=>reset,
                clk=>x_ready,
                data=>x,
                res_in=> pd0,
                coeff=>  h13,
                res_out=>pd1
                );
       etage_1: etage port map
                 (
                reset=>reset,
                clk=>x_ready,
                data=>x,
                res_in=> pd1,
                coeff=>  h12,
                res_out=>pd2
                );
***

       etage_13: etage port map
                 (
                reset=>reset,
                clk=>x_ready,
                data=>x,
                res_in=> pd13,
                coeff=>  h0,
                res_out=>y
                );

end filfir15_arch;
```

Circuit 5 : Filtre Fir25 étages

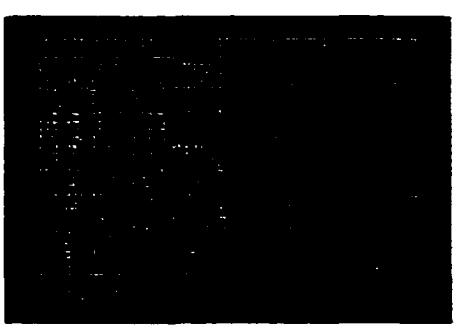
	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit en micron^2
Fir 25 étages	6468	7962	210	8	$51x10^5$

➢ ***Les types de portes utilisées***

- o Nombre de cellules reconnues : 6468
- o Nombre de cellules wpadin : 210
- o Nombre de cellules wpadout : 8
- o Nombre de cellules wand2_1 : 296
- o Nombre de cellules wand2_2 : 50
- o Nombre de cellules wand3_1 : 25
- o Nombre de cellules wbuf_1 : 10
- o Nombre de cellules winv_1 : 428
- o Nombre de cellules winv_2 : 5
- o Nombre de cellules wmux2_2 : 128
- o Nombre de cellules wnand2_1 : 1874
- o Nombre de cellules wnand2_2 : 26
- o Nombre de cellules wnand3_1 : 171
- o Nombre de cellules wnand4_1 : 24
- o Nombre de cellules wnor2_1 : 1308
- o Nombre de cellules wnor2_2 : 78
- o Nombre de cellules wnor3_1 : 120
- o Nombre de cellules wor2_1 : 276
- o Nombre de cellules wor2_2 : 24
- o Nombre de cellules wxor2_2 : 1215
- o Nombre de cellules wdrp_2 : 192

➢ Nombre Total de capacités de routages : 102098

➢ Le layout du circuit :



➢ Code VHDL du circuit :

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity filfir25 is
   port (
      reset: in STD_LOGIC;
      x_ready: in STD_LOGIC;
      x: in signed (7 downto 0);
      h: in signed (199 downto 0);
      y: out signed (7 downto 0)

   );
end filfir25;

architecture filfir25_arch of filfir25 is

component multc2
     port(
        ina : in signed (7 downto 0);
        inb : in signed (7 downto 0);
        pd : out signed (7 downto 0));
  end component;
   component etage
       port (
            reset  : in STD_LOGIC;
            clk    : in STD_LOGIC;
            data   : in signed  (7 downto 0);
            res_in : in signed  (7 downto 0);
            coeff  : in signed  (7 downto 0);
            res_out: out signed (7 downto 0));
   end component;
 --definition des signaux interm.


SIGNAL
pd0,pd1,pd2,pd3,pd4,pd5,pd6,pd7,pd8,pd9,pd10,pd11,pd12,pd13,pd14,pd15,pd16,pd17:
signed(7 downto 0);
SIGNAL pd18,pd19,pd20,pd21,pd22,pd23:signed(7 downto 0);
SIGNAL
h0,h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11,h12,h13,h14,h15,h16,h17,h18,h19,h20,h21,h22,
h23,h24:signed(7 downto 0);
```

```
begin

h0<=h(7  downto 0);
h1<=h(15 downto 8);
h2<=h(23 downto 16);
h3<=h(31 downto 24);
h4<=h(39 downto 32);
h5<=h(47 downto 40);
h6<=h(55 downto 48);

h7<=h(63  downto 56);
h8<=h(71 downto 64);
h9<=h(79 downto 72);
h10<=h(87 downto 80);
h11<=h(95 downto 88);
h12<=h(103 downto 96);
h13<=h(111 downto 104);

h14<=h(119  downto 112);
h15<=h(127 downto 120);
h16<=h(135 downto 128);
h17<=h(143 downto 136);
h18<=h(151 downto 144);
h19<=h(159 downto 152);
h20<=h(167 downto 160);

h21<=h(175  downto 168);
h22<=h(183 downto 176);
h23<=h(191 downto 184);
h24<=h(199 downto 192);

        multi0:multc2
          port map
          (    ina=>x,
                 inb=>h24,
                 pd =>pd0
           );

    etage_0: etage port map
           (
          reset=>reset,
          clk=>x_ready,
          data=>x, res_in=> pd0,
```

```
            coeff=>  h23,
            res_out=>pd1
            );
     etage_1: etage port map
            (
            reset=>reset,
            clk=>x_ready,
            data=>x,
            res_in=> pd1,
            coeff=>  h22,
            res_out=>pd2
            );
    etage_2: etage port map
            (
            reset=>reset,
            clk=>x_ready,
            data=>x,
            res_in=> pd2,
            coeff=>  h21,
            res_out=>pd3
            );
***
etage_22: etage port map
            (
            reset=>reset,
            clk=>x_ready,
            data=>x,
            res_in=> pd22,
            coeff=>  h1,
            res_out=>pd23
            );
    etage_23: etage port map
            (
            reset=>reset,
            clk=>x_ready,
            data=>x,
            res_in=> pd23,
            coeff=>  h0,
            res_out=>y
            );

end filfir25_arch;
```

Circuit 6 : Filtre Fir50 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit en micron^2
Fir 50 étages	13127	13855	410	8	$89x10^5$

- ***Les types de portes utilisées***
  - Nombre de cellules reconues : 13127
  - Nombre de cellules wpadin : 410
  - Nombre de cellules wpadout : 8
  - Nombre de cellules wand2_1 : 1448
  - Nombre de cellules wand3_1 : 154
  - Nombre de cellules wand4_1 : 49
  - Nombre de cellules wbuf_1 : 10
  - Nombre de cellules winv_1 : 1169
  - Nombre de cellules winv_2 : 12
  - Nombre de cellules wmux2_2 : 197
  - Nombre de cellules wnand2_1 : 2690
  - Nombre de cellules wnand3_1 : 344
  - Nombre de cellules wnand4_1 : 2
  - Nombre de cellules wnor2_1 : 1562
  - Nombre de cellules wnor3_1 : 52
  - Nombre de cellules wor2_1 : 1941
  - Nombre de cellules wor3_1 : 149
  - Nombre de cellules wor4_1 : 48
  - Nombre de cellules wxor2_2 : 2490
  - Nombre de cellules wdrp_2 : 392
- Nombre Total de capacités de routages : 293415
- Le layout du circuit :

➢ Code VHDL du circuit :

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity filfir50 is
    port (
        reset: in STD_LOGIC;
        x_ready: in STD_LOGIC;
        x: in signed (7 downto 0);
        h: in signed (399 downto 0);
        y: out signed (7 downto 0)

    );
end filfir50;

architecture filfir50_arch of filfir50 is
component multc2
      port(
          ina : in signed (7 downto 0);
          inb : in signed (7 downto 0);
          pd : out signed (7 downto 0));
   end component;
    component etage
         port (
                reset   : in STD_LOGIC;
                clk     : in STD_LOGIC;
                data    : in signed  (7 downto 0);
                res_in  : in signed  (7 downto 0);
                coeff   : in signed  (7 downto 0);
                res_out: out signed (7 downto 0));
    end component;
 --definition des signaux interm.

SIGNAL
pd0,pd1,pd2,pd3,pd4,pd5,pd6,pd7,pd8,pd9,pd10,pd11,pd12,pd13,pd14,pd15,p
d16,pd17:signed(7 downto 0);
SIGNAL
pd18,pd19,pd20,pd21,pd22,pd23,pd24,pd25,pd26,pd27,pd28,pd29,pd30,pd31,p
d32,pd33,pd34,pd35:signed(7 downto 0);
SIGNAL
pd36,pd37,pd38,pd39,pd40,pd41,pd42,pd43,pd44,pd45,pd46,pd47,pd48:signed
(7 downto 0);
SIGNAL
h0,h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11,h12,h13,h14,h15,h16,h17,h18,h19,h
20,h21,h22,h23,h24,h25,h26,h27:signed(7 downto 0);
SIGNAL
h28,h29,h30,h31,h32,h33,h34,h35,h36,h37,h38,h39,h40,h41,h42,h43,h44,h45
,h46,h47,h48,h49:signed(7 downto 0);
```

```
begin

n0<=h(7  downto 0);
h1<=h(15 downto 8);
h2<=h(23 downto 16);
h3<=h(31 downto 24);
h4<=h(39 downto 32);
h5<=h(47 downto 40);
h6<=h(55 downto 48);

h7<=h(63  downto 56);
h8<=h(71 downto 64);
h9<=h(79 downto 72);
h10<=h(87 downto 80);
h11<=h(95 downto 88);
h12<=h(103 downto 96);
h13<=h(111 downto 104);

h14<=h(119  downto 112);
h15<=h(127 downto 120);
h16<=h(135 downto 128);
h17<=h(143 downto 136);
h18<=h(151 downto 144);
h19<=h(159 downto 152);
h20<=h(167 downto 160);

h21<=h(175  downto 168);
h22<=h(183 downto 176);
h23<=h(191 downto 184);
h24<=h(199 downto 192);
h25<=h(207 downto 200);
h26<=h(215 downto 208);
h27<=h(223 downto 216);

h28<=h(231  downto 224);
h29<=h(239 downto 232);
h30<=h(247 downto 240);
h31<=h(255 downto 248);
h32<=h(263 downto 256);
h33<=h(271 downto 264);
h34<=h(279 downto 272);

h35<=h(287  downto 280);
h36<=h(295 downto 288);
h37<=h(303 downto 296);
h38<=h(311 downto 304);
h39<=h(319 downto 312);
h40<=h(327 downto 320);
h41<=h(335 downto 328);

h42<=h(343  downto 336);
```

```
h43<=h(351 downto 344);
h44<=h(359 downto 352);
h45<=h(367 downto 360);
h46<=h(375 downto 368);
h47<=h(383 downto 376);
h48<=h(391 downto 384);
h49<=h(399 downto 392);



      multi0:multc2
         port map
         (    ina=>x,
            inb=>h49,
            pd =>pd0
          );

       etage_0: etage port map
                 (
                reset=>reset,
                clk=>x_ready,
                data=>x,
                res_in=> pd0,
                coeff=>  h48,
                res_out=>pd1
                );
       etage_1: etage port map
                 (
                reset=>reset,
                clk=>x_ready,
                data=>x,
                res_in=> pd1,
                coeff=>  h47,
                res_out=>pd2
                );
***
  etage_48: etage port map
                 (
                reset=>reset,
                clk=>x_ready,
                data=>x,
                res_in=> pd48,
                coeff=>  h0,
                res_out=>y
                );

end filfir50_arch;
```

ANNEXE 8

Résultats de la réduction des sites potentiels

Types de pannes :

Le tableau suivant présente 9 types de pannes :

Fault	Description
TF0	INV-INV
TF1	INV-NAND2
TF2	INV-NAND3
TF3	INV-NAND4
TF4	NAND2-NAND2
TF5	NAND2-NAND3
TF6	NAND2-NAND4
TF7	NAND3-NAND3
TF8	NAND3-NAND4
TF9	NAND4-NAND4

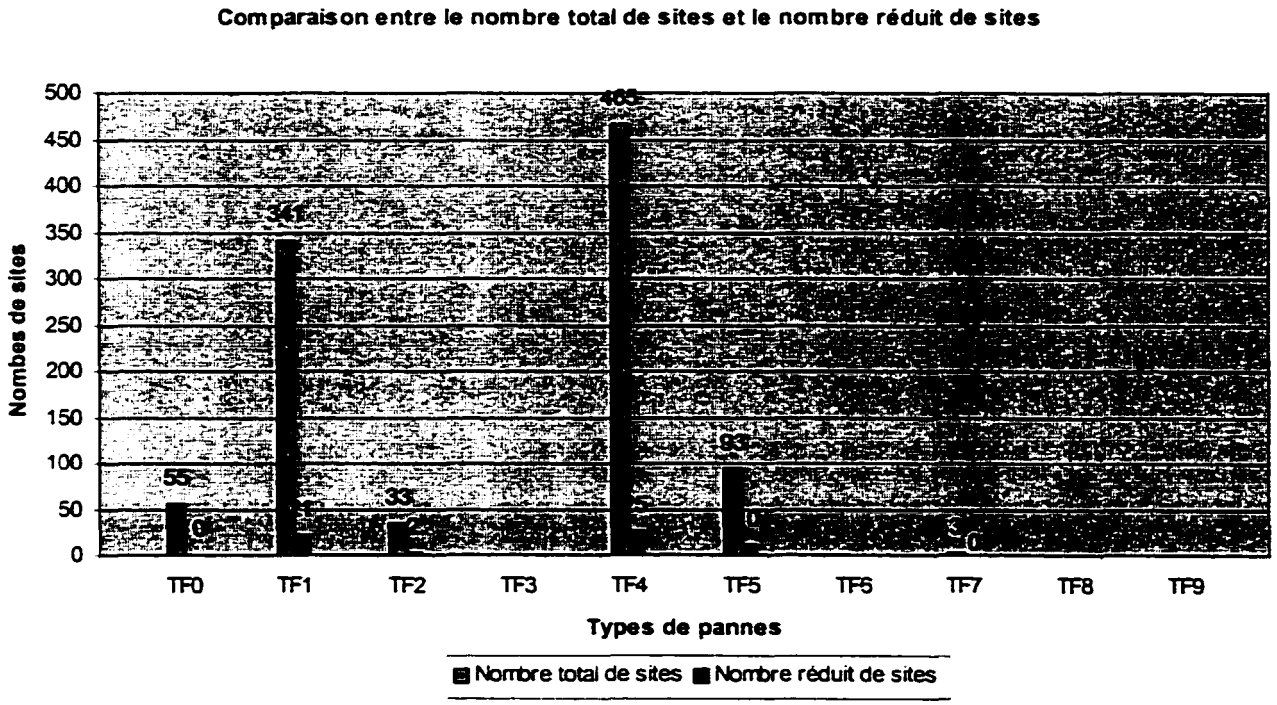
Remarque : les types de pannes sont des courts-circuits entre une paire de nœuds constituant les sorties de deux portes logiques de même type ou de type différent.

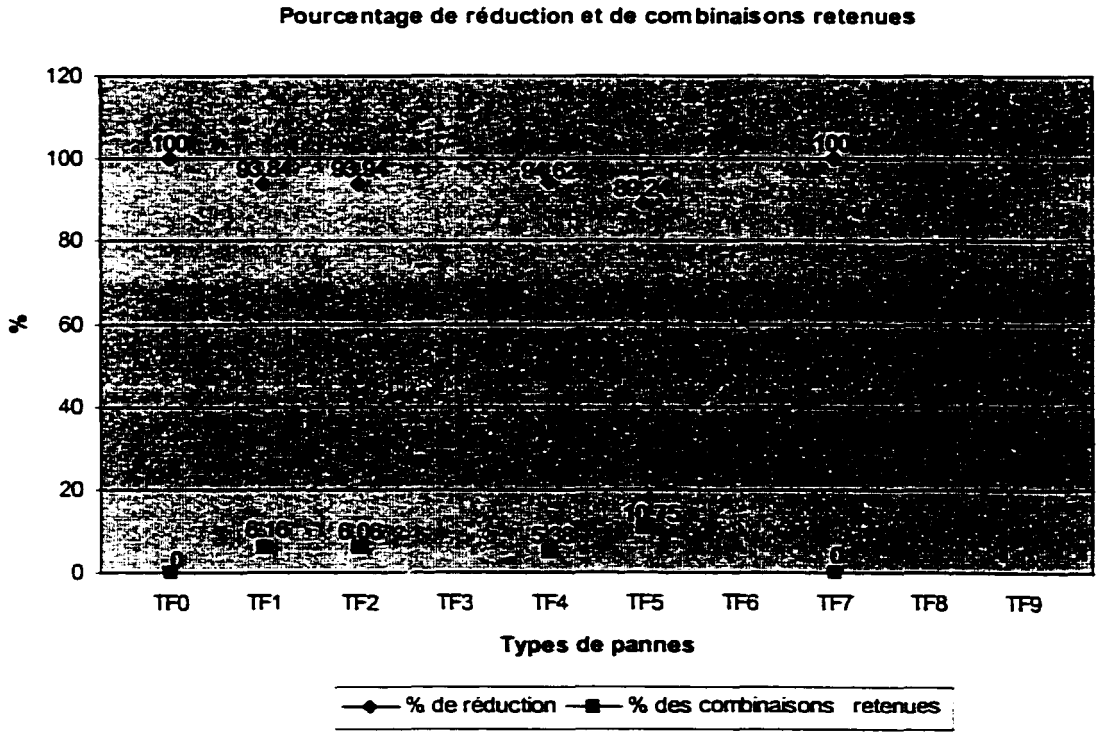
Exemple : ***TF8 NAND3-NAND4***

Ce type de panne est un court-circuit entre la sortie d'une porte nand3 et une porte nand4.

➢ Statistiques sur les types de pannes pour un Multiplicateur 4X4:

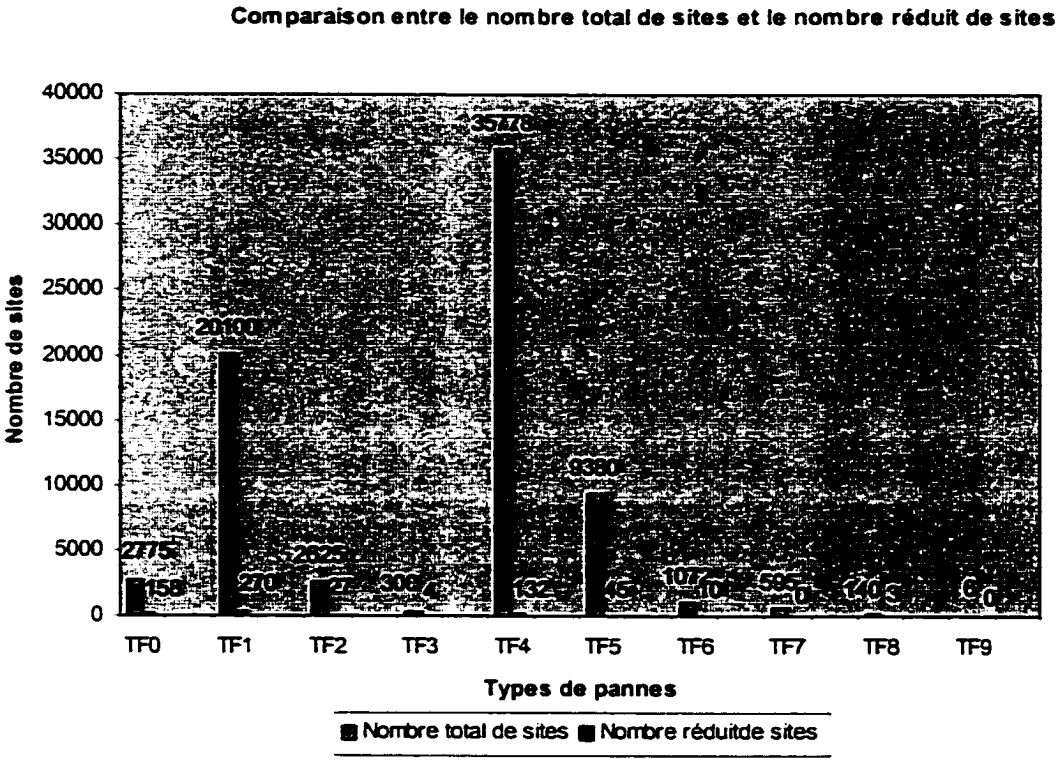
Type de pannes	Description et nombre		Nombre total de sites	Nombre réduit de sites	% de réduction	% des combinaisons retenues	sites/ cellule
TF0	INV	INV	55	0	100	0	0
	11	11					
TF1	INV	NAND2	341	21	93.84	6.16	0.22
	11	31					
TF2	INV	NAND3	33	2	93.94	6.06	0.021
	11	3					
TF3	INV	NAND4					
	11	0					
TF4	NAND2	NAND2	465	25	94.62	5.38	0.26
	31	31					
TF5	NAND2	NAND3	93	10	89.24	10.75	0.10
	31	3					
TF6	NAND2	NAND4					
	31	0					
TF7	NAND3	NAND3	3	0	100	0	0
	3	3					
TF8	NAND3	NAND4					
	3	0					
TF9	NAND4	NAND4					
	0	0					
	Total		***990***	***58***	***94.14***	***5.86***	***0.6***

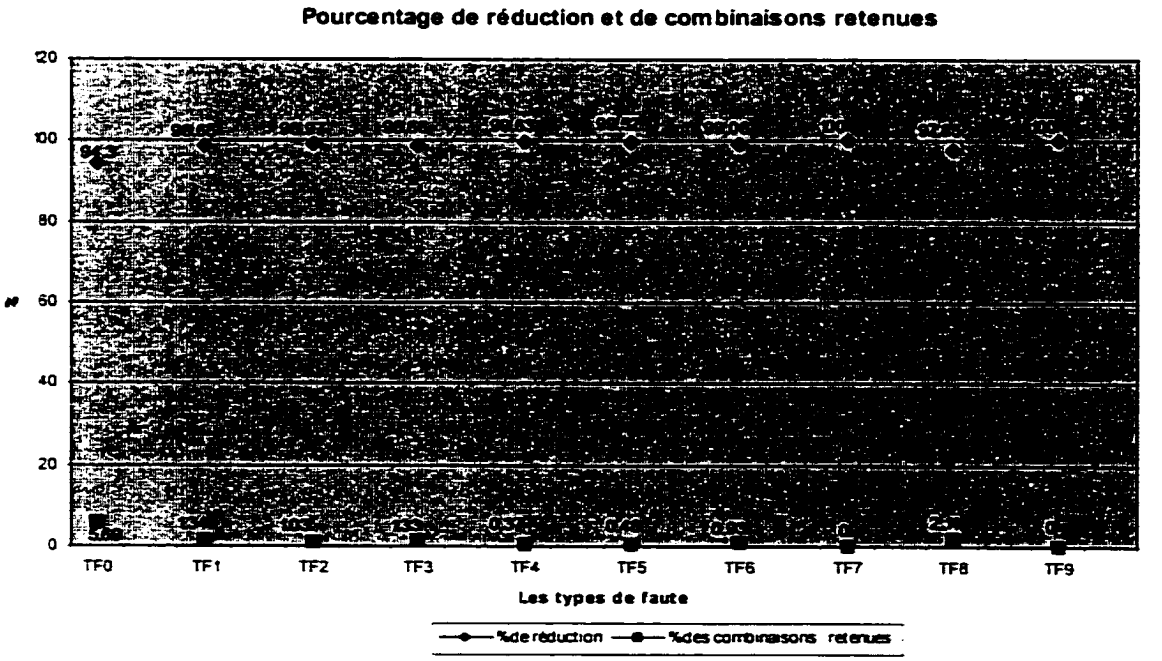

Comparaison entre le nombre total de sites et le nombre réduit de sites
Nombre de sites
500
450
400
350
300
250
200
150
100
50
0
55
0
341
33
465
93
0
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
Nombre total de sites
Nombre réduit de sites


Pourcentage de réduction et de combinaisons retenues
%
120
100
80
60
40
20
0
100
100
0
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
% de réduction
% des combinaisons retenues

➢ Statistiques sur les types de pannes pour un Filtre Fir 4 étages :

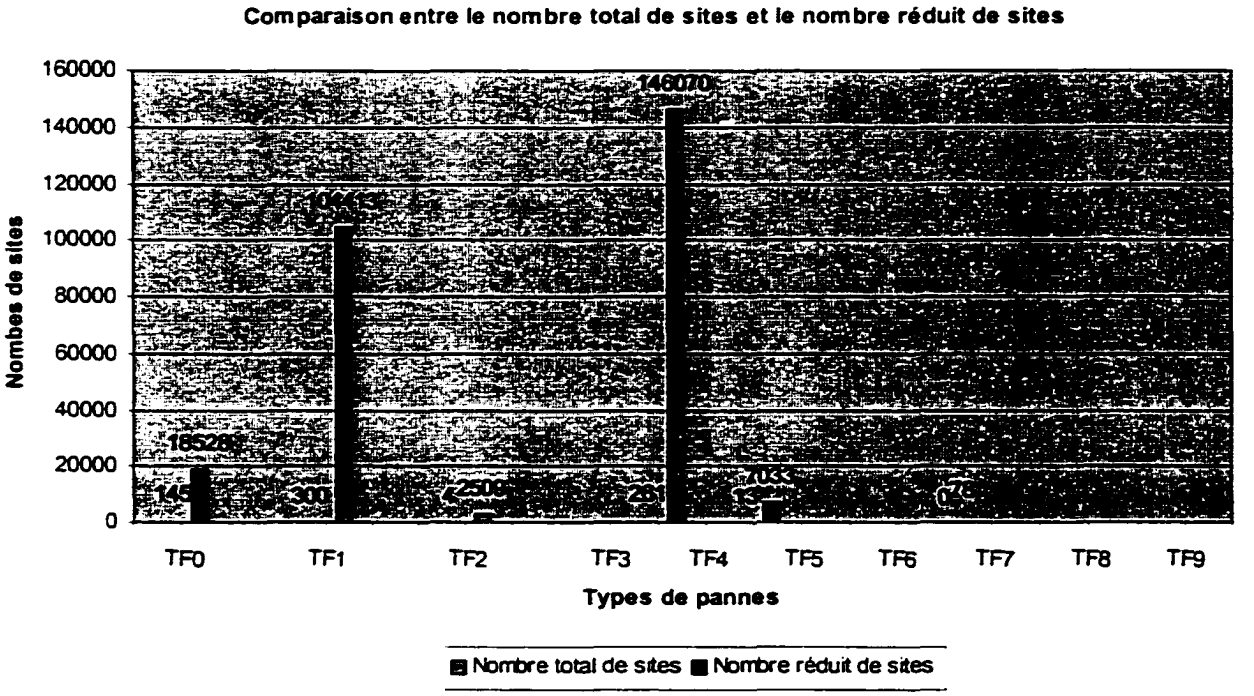
Type de panne	Description et nombre		Nombre total de sites	Nombre réduit de sites	% de réduction	% des combinaisons retenues	site/cellule
TF0	INV 75	INV 75	2775	158	94.30	5.69	0.17
TF1	INV 75	NAND2 268	20100	270	98.65	1.34	0.30
TF2	INV 75	NAND3 35	2625	27	98.97	1.03	0.03
TF3	INV 75	NAND4 4	300	4	98.66	1.33	0.0044
TF4	NAND2 268	NAND2 268	35778	132	99.63	0.37	0.14
TF5	NAND2 268	NAND3 35	9380	45	99.52	0.48	0.05
TF6	NAND2 268	NAND4 4	1072	10	99.06	0.93	0.011
TF7	NAND3 35	NAND3 35	595	0	100	0	0
TF8	NAND3 35	NAND4 4	140	3	97.85	2.14	0.0033
TF9	NAND4 4	NAND4 4	6	0	100	0	0
Total			***72771***	***649***	***99.10***	***0.89***	***0.72***

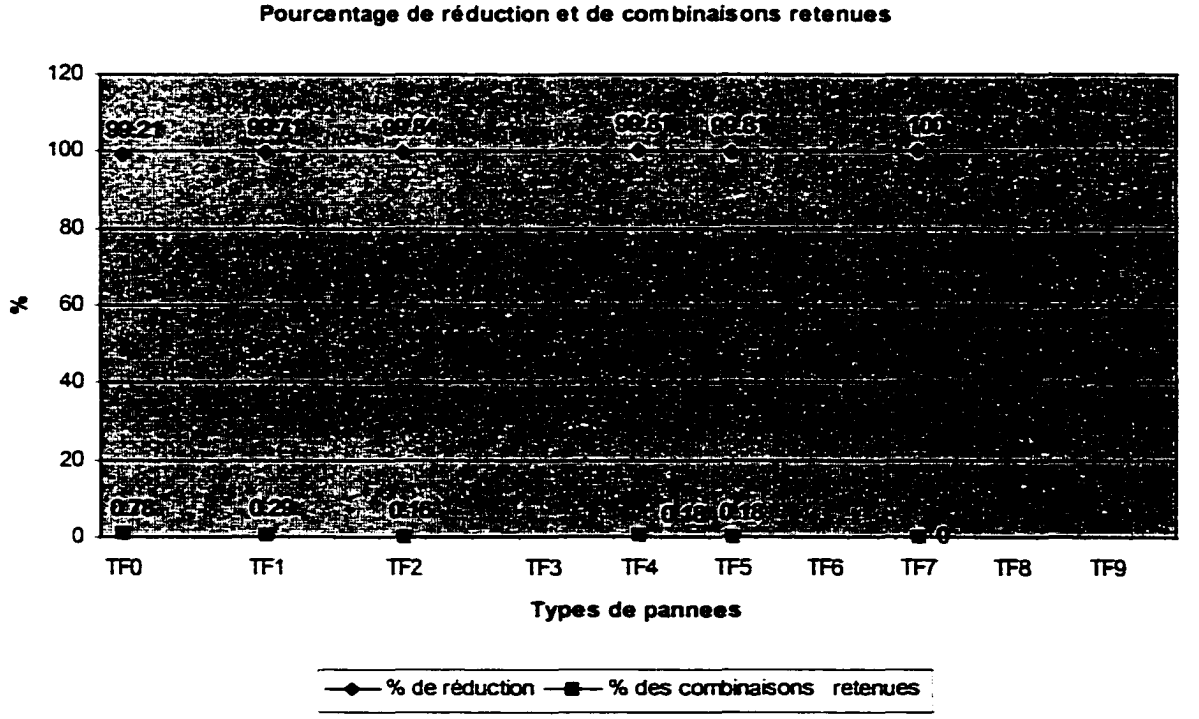

Comparaison entre le nombre total de sites et le nombre réduit de sites
Nombre de sites
40000
35000
30000
25000
20000
15000
10000
5000
0
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
Nombre total de sites
Nombre réduitde sites


Pourcentage de réduction et de combinaisons retenues
20
100
80
60
40
20
0
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Les types de faute
%de réduction
%des combinaisons retenues

➢ Statistiques sur les types de pannes pour un Filtre Fir 7 étages :

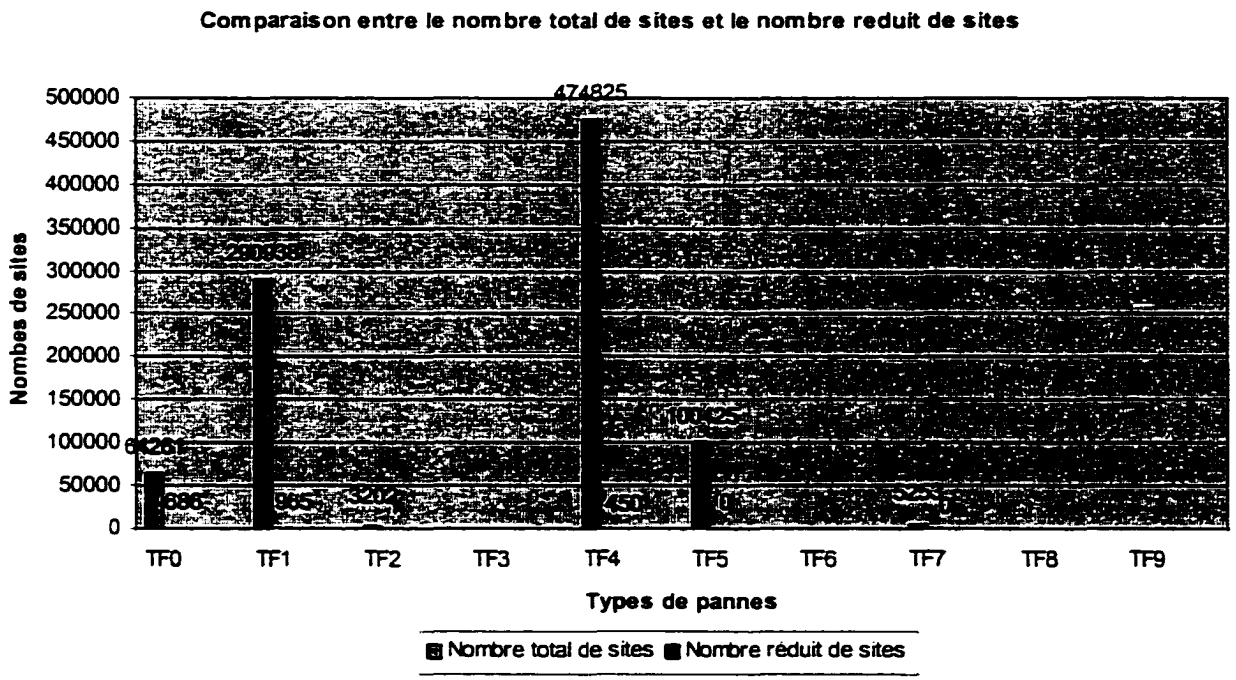
Type de panne	Description et nombre		Nombre total de sites	Nombre réduit de sites	% de réduction	% des combinaisons retenues	site/cellule
TF0	INV	INV	18528	145	99.21	0.78	0.0088
	193	193					
TF1	INV	NAND2	104413	300	99.71	0.29	0.18
	193	541					
TF2	INV	NAND3	2509	4	99.84	0.16	0.0024
	193	13					
TF3	INV	NAND4					
	193	0					
TF4	NAND2	NAND2	146070	261	99.81	0.18	0.159
	541	541					
TF5	NAND2	NAND3	7033	13	99.81	0.18	0.0079
	541	13					
TF6	NAND2	NAND4					
	541	0					
TF7	NAND3	NAND3	78	0	100	0	0
	13	13					
TF8	NAND3	NAND4					
	13	0					
TF9	NAND4	NAND4					
	0	0					
Total			271598	723	99.73	0.26	0.44

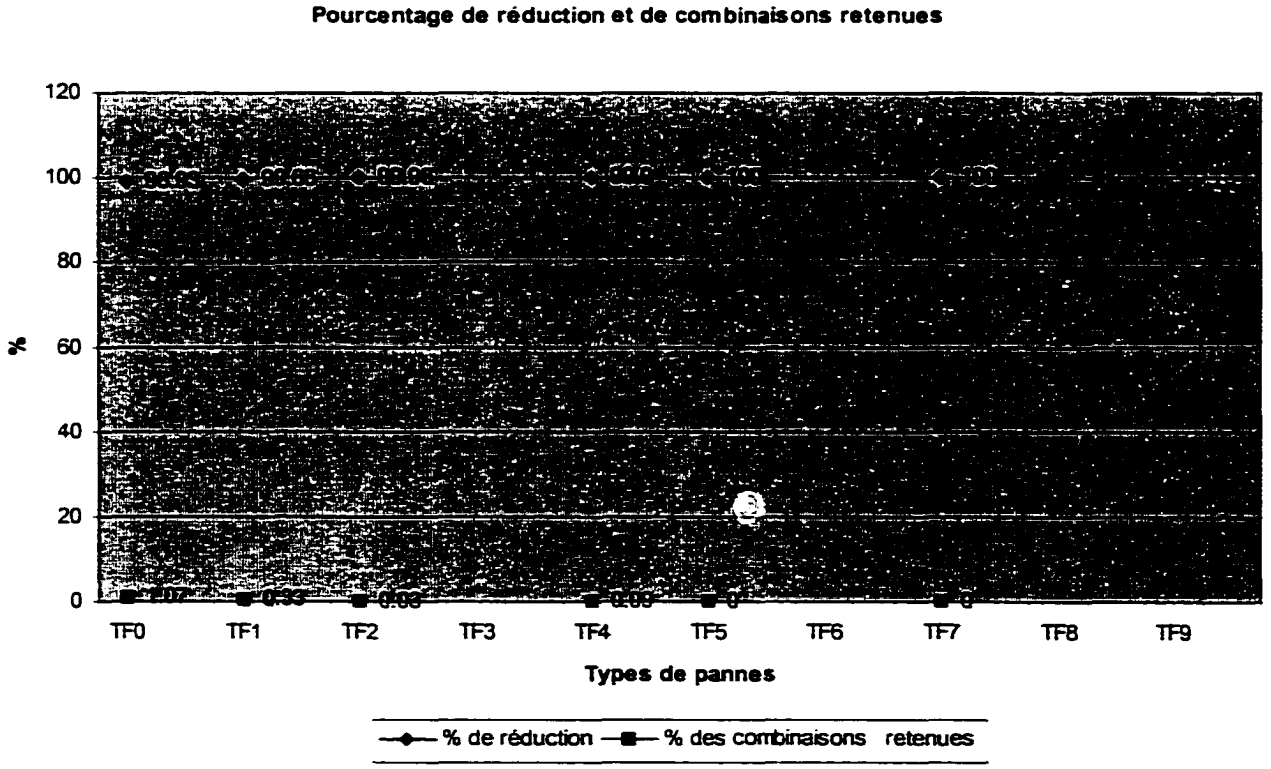

Comparaison entre le nombre total de sites et le nombre réduit de sites
160000
140000
120000
100000
80000
60000
40000
20000
0
Nombes de sites
146070
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
Nombre total de sites
Nombre réduit de sites


Pourcentage de réduction et de combinaisons retenues
120
100
80
60
40
20
0
%
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannees
% de réduction
% des combinaisons retenues

➢ Statistiques sur les types de pannes pour un Filtre Fir15 étages :

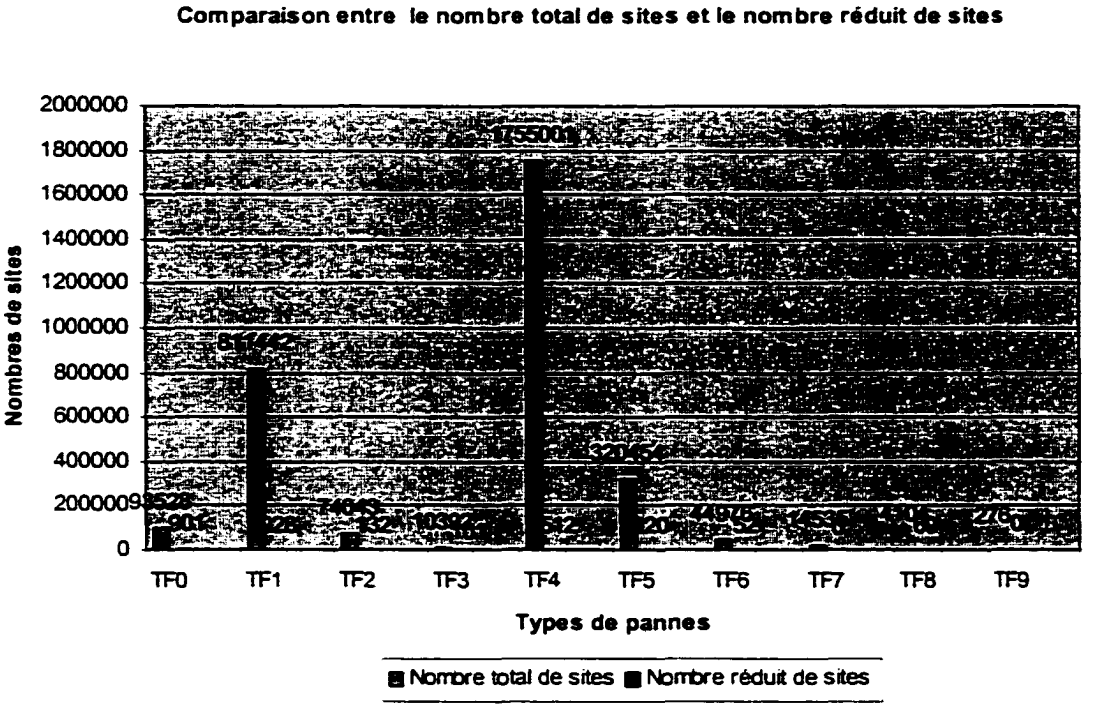
Type de panne	Description et nombre		Nombre total de sitee	Nombre réduit de sites	% de réduction	% des combinaisons retenues	site/cellule
TF0	INV	INV	64261	686	98.93	1.07	0.19
	359	359					
TF1	INV	NAND2	290938	965	99.66	0.33	0.27
	359	975					
TF2	INV	NAND3	3202	1	99.96	0.03	0.0003
	359	103					
TF3	INV	NAND4					
	359	0					
TF4	NAND2	NAND2	474825	450	99.90	0.09	0.13
	975	975					
TF5	NAND2	NAND3	100425	0	100	0	0
	975	103					
TF6	NAND2	NAND4					
	975	0					
TF7	NAND3	NAND3	5253	0	100	0	0
	103	103					
TF8	NAND3	NAND4					
	103	0					
TF9	NAND4	NAND4					
	0	0					
Total			938904	2102	99.77	0.22	0.59

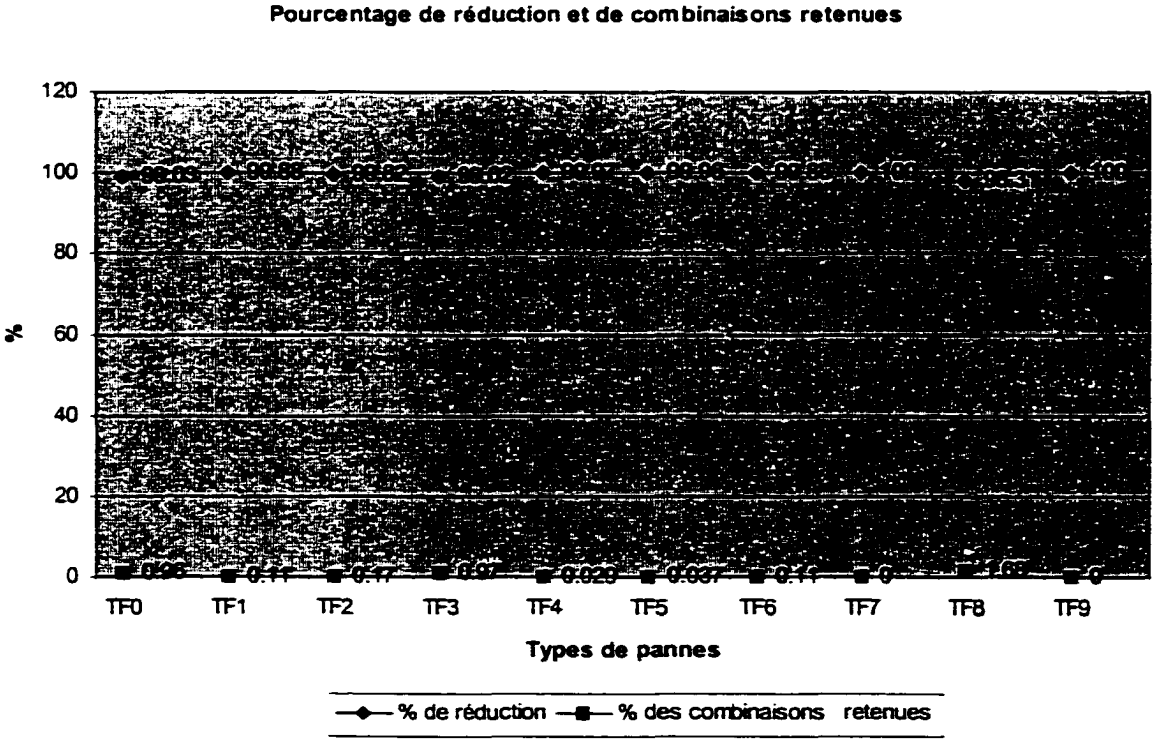

Comparaison entre le nombre total de sites et le nombre reduit de sites
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
Nombes de sites
474825
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
Nombre total de sites
Nombre réduit de sites


Pourcentage de réduction et de combinaisons retenues
120
100
80
60
40
20
0
%
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
% de réduction
% des combinaisons retenues

➢ Statistiques sur les types de pannes pour un Filtre Fir25 étages:

Type de panne	Description		Nombre totsl de test	Nombre réduit de couple	% de réduction	% des combinaisons retenues	site/cellule
TF0	INV 433	INV 433	93528	901	99.03	0.96	0.14
TF1	INV 433	NAND2 1874	811442	928	99.88	0.11	0.14
TF2	INV 433	NAND3 171	74043	132	99.82	0.17	0.02
TF3	INV 433	NAND4 24	10392	101	99.02	0.97	0.01
TF4	NAND2 1874	NAND2 1874	1755001	512	99.97	0.029	0.08
TF5	NAND2 1874	NAND3 171	320454	120	99.96	0.037	0.02
TF6	NAND2 1874	NAND4 24	44976	52	99.88	0.11	0.008
TF7	NAND3 171	NAND3 171	14535	0	100	0	0
TF8	NAND3 171	NAND4 24	4104	69	98.31	1.68	0.01
TF9	NAND4 24	NAND4 24	276	0	100	0	0
Total			3128751	2815	99.91	0.09	0.43

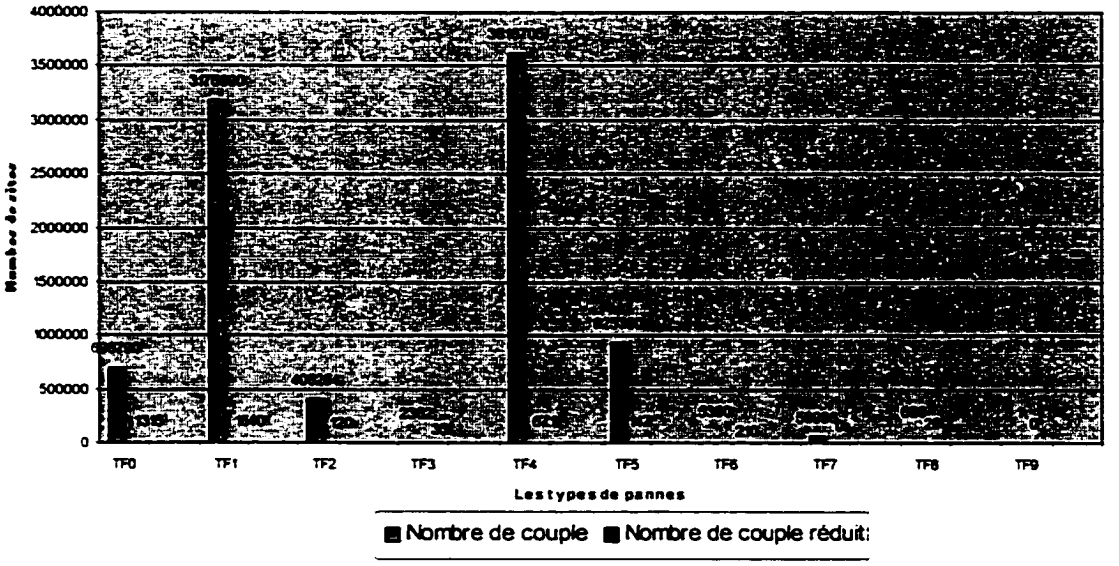

Comparaison entre le nombre total de sites et le nombre réduit de sites
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
0
Nombres de sites
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
Nombre total de sites
Nombre réduit de sites


Pourcentage de réduction et de combinaisons retenues
120
100
80
60
40
20
0
%
TF0
TF1
TF2
TF3
TF4
TF5
TF6
TF7
TF8
TF9
Types de pannes
% de réduction
% des combinaisons retenues

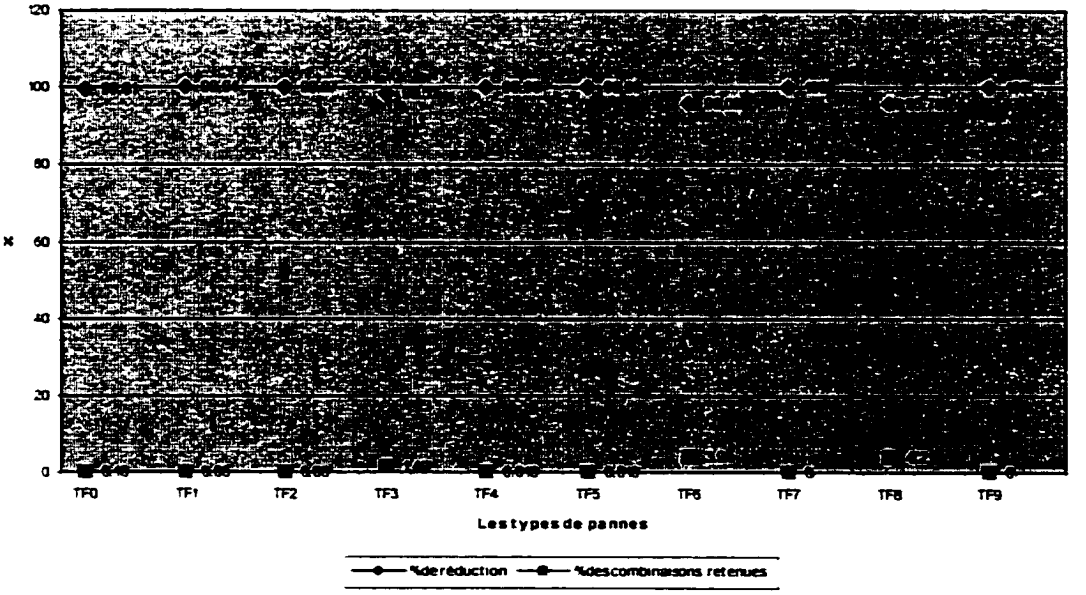
➢ Statistiques sur les types de pannes pour un Filtre Fir50 étages:

Type de panne	Description		Nombre total de sites	Nombre réduit de sites	% de réduction	% des combinaisons retenues	site/cellule
TF0	INV 1181	INV 1181	696790	1315	99.81	0.18	0.1
TF1	INV 1181	NAND2 2690	3176890	1640	99.94	0.05	0.125
TF2	INV 1181	NAND3 344	406264	120	99.97	0.03	0.009
TF3	INV 1181	NAND4 2	2362	35	98.51	1.49	0.002
TF4	NAND2 2690	NAND2 2690	3616705	671	99.98	0.018	0.05
TF5	NAND2 2690	NAND3 344	925360	147	99.98	0.018	0.01
TF6	NAND2 2690	NAND4 2	5380	210	96.09	3.9	0.016
TF7	NAND3 344	NAND3 344	58996	0	100	0	0
TF8	NAND3 344	NAND4 2	688	29	95.78	4.21	0.002
TF9	NAND4 2	NAND4 2	1	0	100	0	0
Total			8889436	4167	99.95	0.046	0.31

Comparaison entre le nombre total de sites et le nombre réduit de sites



Pourcentage de réduction et de combinaisons retenues



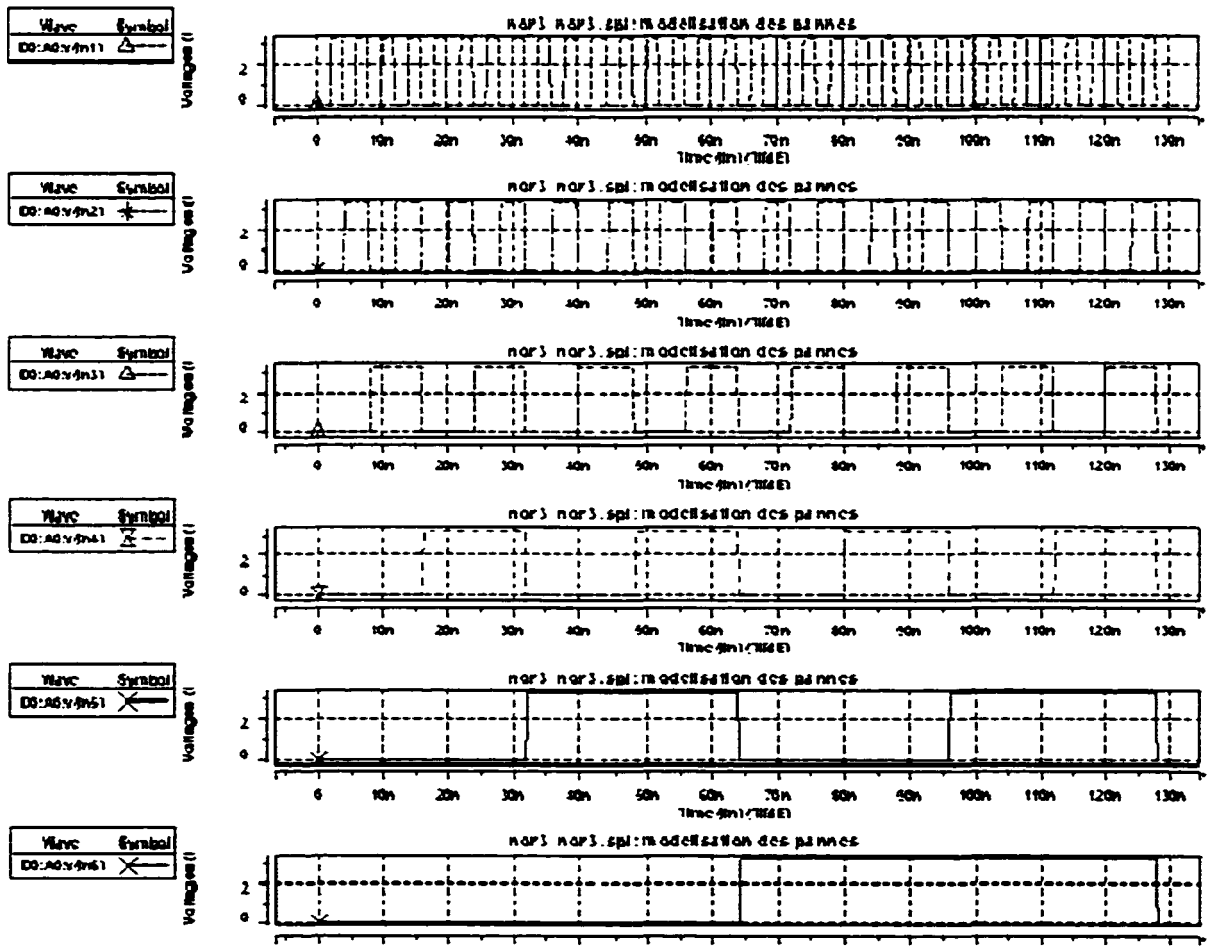
ANNEXE 9

Résultats de simulation sur Hspice

Signaux d'excitation :

Dépendamment du nombre d'entrées du circuit, on définit les signaux d'excitation comme suit. On couvre tous les cas possibles des valeurs logiques des entrées.

*signaux d'entrees	depart	fin	delai	rise	fall	high	periode
Vin1	in1 0	PULSE(0	3.3	2ns	0.15ns	0.15ns	1.7ns	4ns)
Vin2	in2 0	PULSE(0	3.3	4ns	0.15ns	0.15ns	3.7ns	8ns)
Vin3	in3 0	PULSE(0	3.3	8ns	0.15ns	0.15ns	7.7ns	16ns)
Vin4	in4 0	PULSE(0	3.3	16ns	0.15ns	0.15ns	15.7ns	32ns)
Vin5	in5 0	PULSE(0	3.3	32ns	0.15ns	0.15ns	31.7ns	64ns)
Vin6	in6 0	PULSE(0	3.3	64ns	0.15ns	0.15ns	63.7ns	128ns)

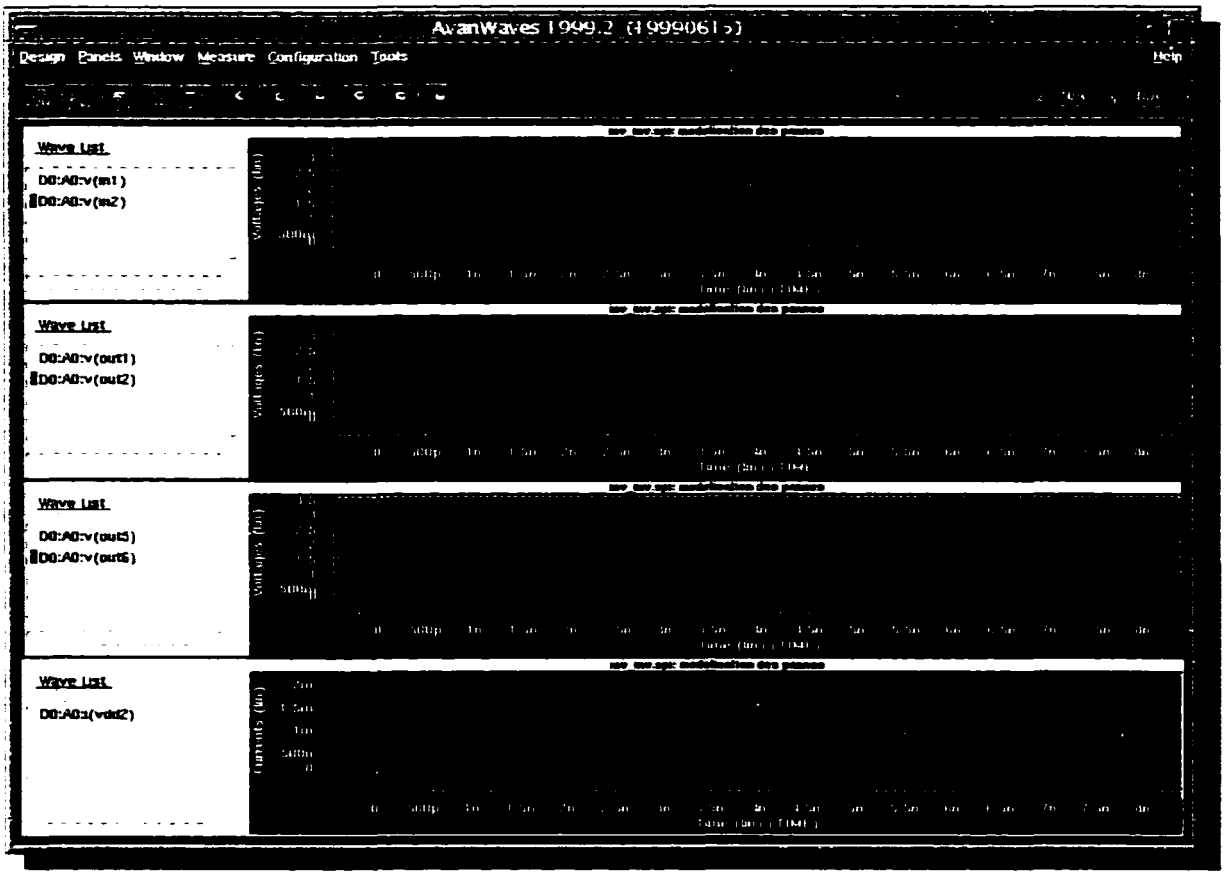


Faute -1- court circuit INV-INV :

❑ Code source :

```
inv_inv.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0       DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees      depart fin  delai1  rise    fall    high  periode
Vin1    in1  0  PULSE(  0   3.3   2ns   0.15ns  0.15ns  1.7ns  4ns)
Vin2    in2  0  PULSE(  0   3.3   4ns   0.15ns  0.15ns  3.7ns  8ns)
* resistance de court-circuit
Rbr out1 out2 br
.param  br=10
* portes impliquees dans le cc
X1 in1 out1 vdd2 0 inv
X2 in2 out2 vdd2 0 inv
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
-ABSTOL=1NA
-NOMOD
-POST
*-PROBE signaux a acceder
.TRAN   .1n 8n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param  br=15000  .END
```

- Résultats de la simulation :



- Modélisation :

Table de vérité

Décimal	In2	In1	Out1	Out2	out
0	**0**	**0**	**1**	**1**	**1**
1	0	1	0	1	0
2	1	0	1	0	0
3	1	1	0	0	0

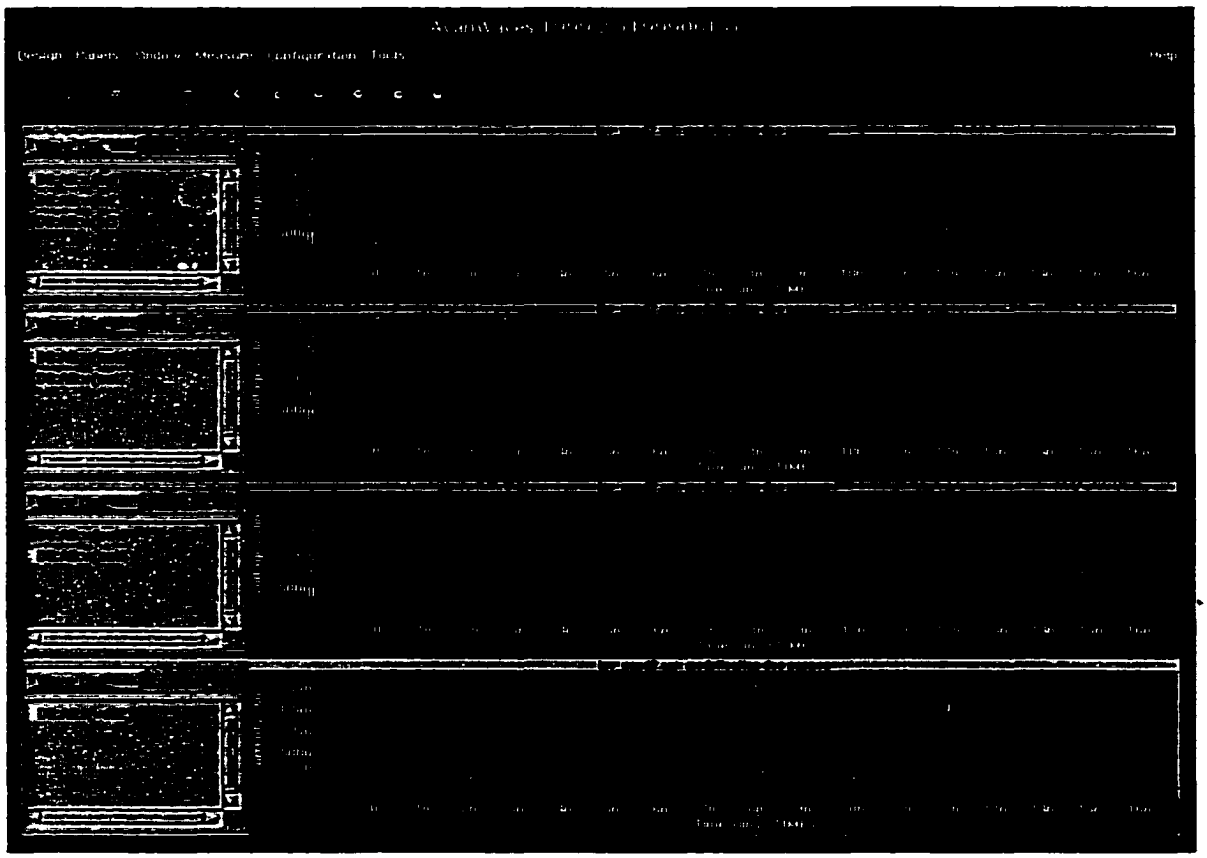
✓ Comportement d'une porte Et à 2 entrées (out1 et out2).

Faute - 2- court circuit INV-NAND2:

- Code source :

```
inv_nand2.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise    fall    high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 out1 vdd2 0 inv
X2 in2 in3 out2 vdd2 0 nand2
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
-ABSTOL=1NA
-NOMOD
-POST
*-PROBE signaux a acceder
.TRAN    .1n 16n
*.print i(vdd2) v(out1) v(out2)
*.alter *.param   br=15000 .END
```

- **Résultats de la simulation :**



- Modélisation :

Table de vérité

Décimal	In3	In2	In1	Out1	Out2	out
0	0	0	0	1	1	1
1	0	0	1	0	1	1
2	0	1	0	1	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	1
5	1	0	1	0	1	0
6	1	1	0	1	0	1
7	1	1	1	0	0	0

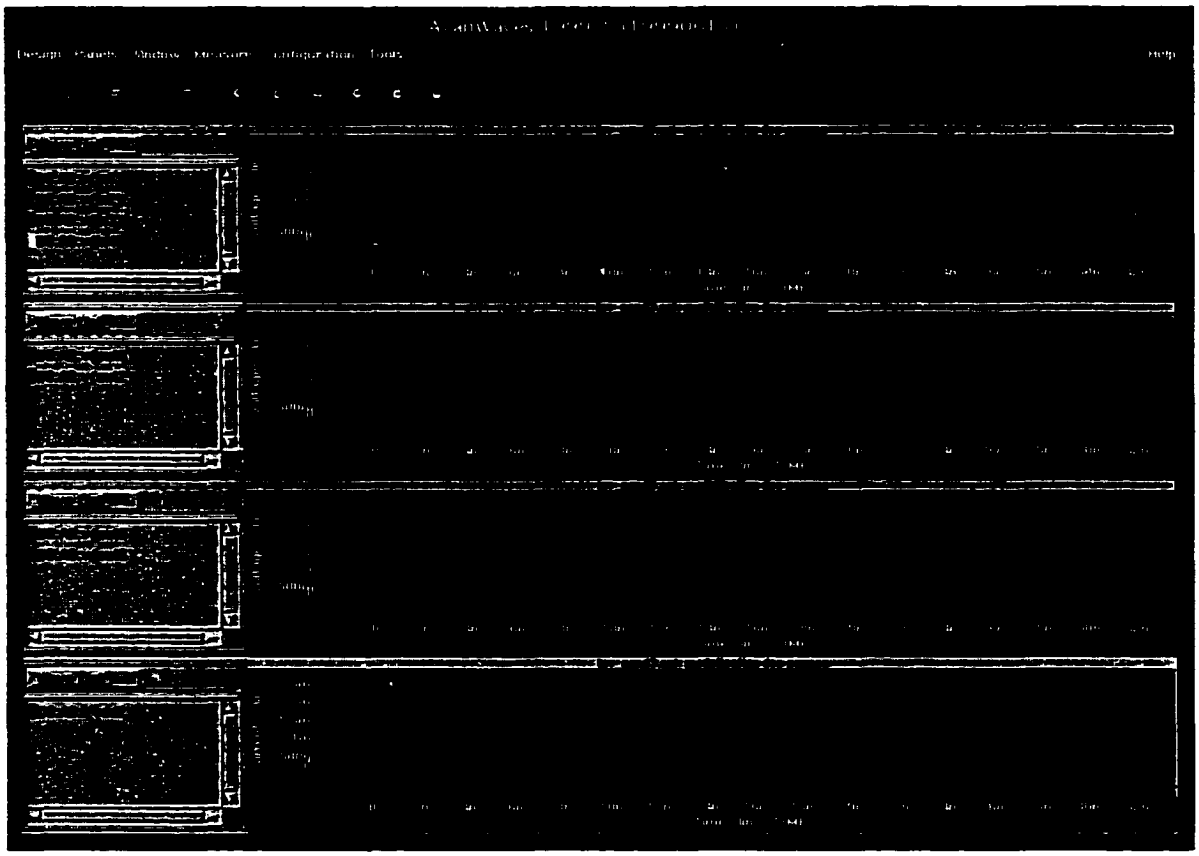
✓ Comportement d'une porte complexe :

Faute - 3- court circuit INV-NAND3:

- Code source :

```
inv_nand3.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise   fall   high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns  1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns  3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns  7.7ns   16ns)
Vin4    in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns  15.7ns  32ns)
* resistance de court-circuit
Rbr out1 out2 br
.param  br=10
* portes impliquees dans le cc
X1 in1 out1 vdd2 0 inv
X2 in2 in3 in4 out2 vdd2 0 nand3
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
 OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
 TRAN    1n 32n
*.print i(vdd2) v(out1) v(out2)
* alter
*.param  br=15000.END
```

- Résultats de la simulation :

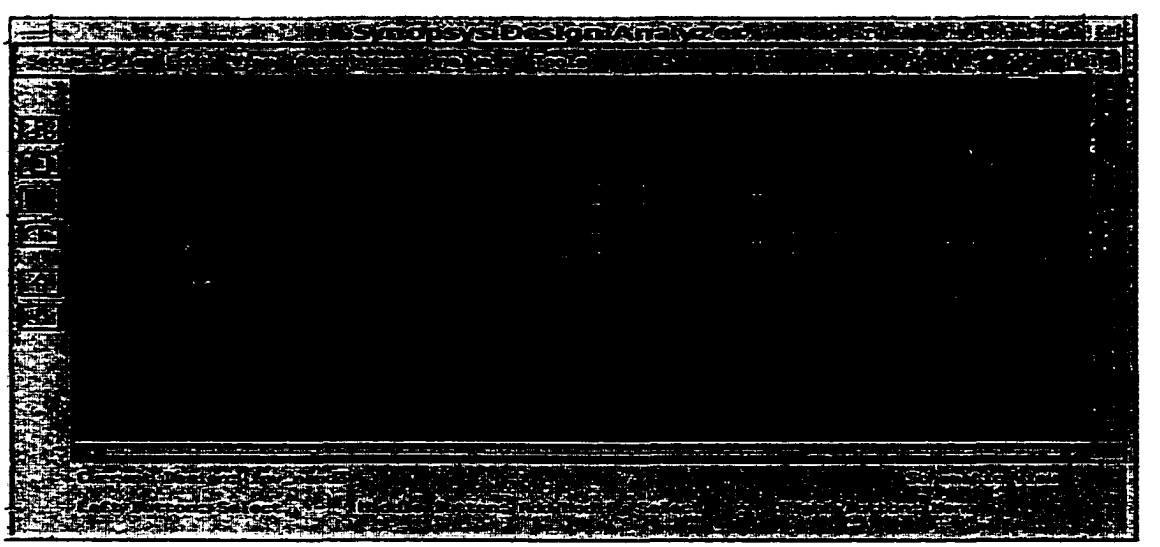


- Modélisation :

Table de vérité

Décimal	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	**1**	**1**	1
1	0	0	0	1	0	1	1
2	0	0	1	0	1	1	1
3	0	0	1	1	0	1	1
4	0	1	0	0	1	1	1
5	0	1	0	1	0	1	1
6	0	1	1	0	1	1	1
7	0	1	1	1	0	1	0
8	1	0	0	0	1	1	1
9	1	0	0	1	0	1	1
10	1	0	1	0	1	1	1
11	1	0	1	1	0	1	0
12	1	1	0	0	1	1	1
13	1	1	0	1	0	1	0
14	1	1	1	0	1	0	1
15	1	1	1	1	0	0	0

✓ Comportement d'une porte complexe :

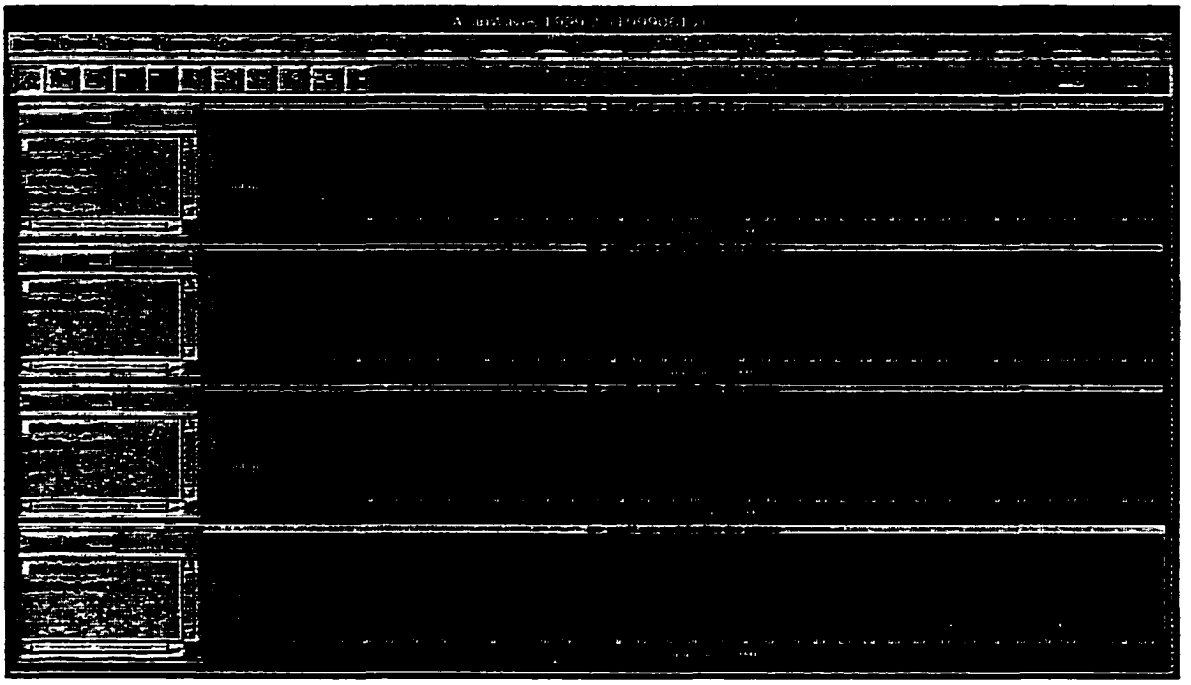


Faute - 4- court circuit INV-NAND4:

- Code source :

```
inv_nand4.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise   fall   high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
Vin4    in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns   15.7ns  32ns)
Vin4    in5  0  PULSE(0   3.3  32ns  0.15ns  0.15ns   31.7ns  64ns)

* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 out1 vdd2 0 inv
X2 in2 in3 in4 in5 out2 vdd2 0 nand4
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 64n
*.print i(vdd2) v(out1) v(out2)*.alter*.param   br=15000 .END
```

- Résultats de la simulation :



- Modélisation :

Table de vérité

Décimal	In5	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	0	**1**	**0**	1
1	0	0	0	0	1	0	0	1
2	0	0	0	1	0	1	0	1
3	0	0	0	1	1	0	0	1
4	0	0	1	0	0	1	0	1
5	0	0	1	0	1	0	0	1
6	0	0	1	1	0	1	0	1
7	0	0	1	1	1	0	0	1
8	0	1	0	0	0	1	0	1
9	0	1	0	0	1	0	0	1
10	0	1	0	1	0	1	0	1
11	0	1	0	1	1	0	0	1
12	0	1	1	0	0	1	0	1
13	0	1	1	0	1	0	0	1
14	0	1	1	1	0	1	0	1
15	0	1	1	1	1	0	0	0
16	1	0	0	0	0	1	0	1
17	1	0	0	0	1	0	0	1
18	1	0	0	1	0	1	0	1
19	1	0	0	1	1	0	0	1
20	1	0	1	0	0	1	0	1
21	1	0	1	0	1	0	0	1
22	1	0	1	1	0	1	0	1
23	1	0	1	1	1	0	0	1
24	1	1	0	0	0	1	0	1
25	1	1	0	0	1	0	0	1
26	1	1	0	1	0	1	0	1
27	1	1	0	1	1	0	0	1
28	1	1	1	0	0	1	0	1
29	1	1	1	0	1	0	0	1
30	1	1	1	1	0	1	1	1
31	1	1	1	1	1	0	1	0

✓ Comportement d'une porte Non Et à 4 entrees (in1,in2,in3,in4).

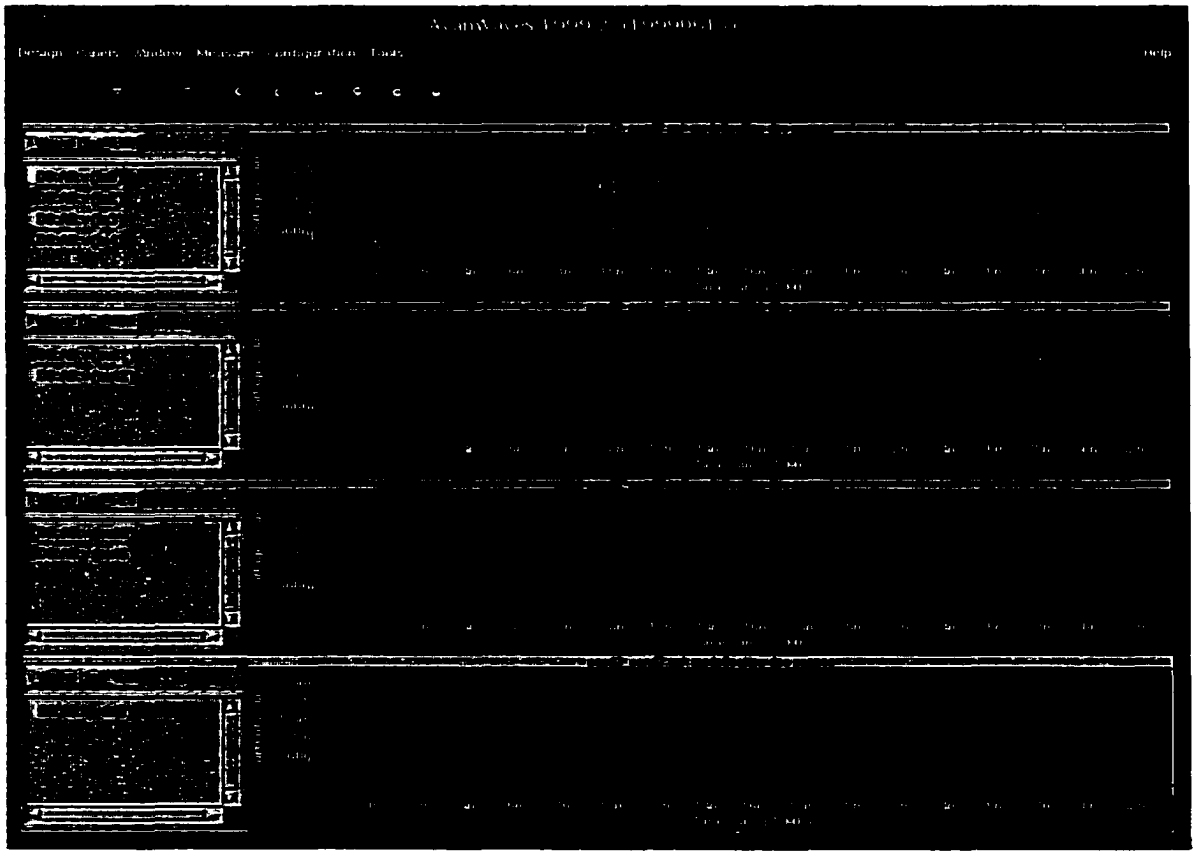
Faute -5- court circuit NAND2-NAND2:

- Code source :

```
nand2_nand2.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise    fall    high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns  1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns  3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns  7.7ns   16ns)
Vin4    in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns  15.7ns  32ns)
* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 in2 out1 vdd2 0 nand2
X2 in3 in4 out2 vdd2 0 nand2
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
```

```
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 32n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param   br=15000
.END
```

- Résultats de la simulation :



- Modélisation :

Table de vérité

Décimal	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	**1**	**1**	1
1	0	0	0	1	1	1	1
2	0	0	1	0	1	1	1
3	0	0	1	1	0	1	1
4	0	1	0	0	1	1	1
5	0	1	0	1	1	1	1
6	0	1	1	0	1	1	1
7	0	1	1	1	0	1	1
8	1	0	0	0	1	1	1
9	1	0	0	1	1	1	1
10	1	0	1	0	1	1	1
11	1	0	1	1	0	1	1
12	1	1	0	0	1	0	1
13	1	1	0	1	1	0	1
14	1	1	1	0	1	0	1
15	1	1	1	1	0	0	0

✓ Comportement d'une porte Ou à 2 entrées (out1 et out2).

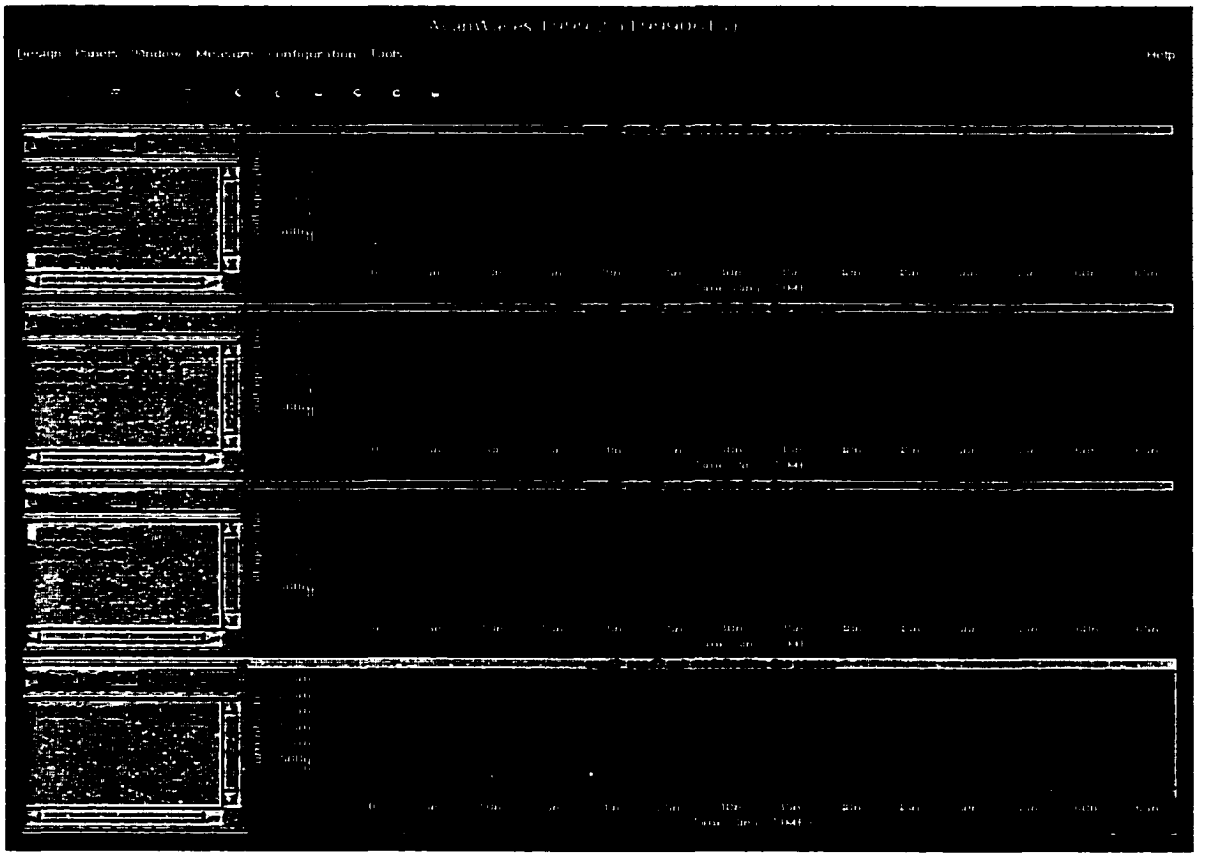
Faute -6- court circuit NAND2-NAND3:

- Code source :

```
nand2_nand3.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1     vdd1     0     DC 3.3
* alimentation secondaire pour courant positif
VDD2     vdd1     vdd2     DC 0
*signaux d'entrees   depart fin  delai1  rise    fall    high  periode
Vin1     in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2     in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3     in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
Vin4     in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns   15.7ns  32ns)
Vin5     in5  0  PULSE(0   3.3  32ns  0.15ns  0.15ns   31.7ns  64ns)
* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 in2     out1 vdd2 0 nand2
X2 in3 in4 in5 out2 vdd2 0 nand3
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
```

```
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 64n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param   br=15000
.END
```

- **Résultats de la simulation :**



- Modélisation :

Table de vérité

Décimal	In5	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	0	**1**	**1**	1
1	0	0	0	0	1	1	1	1
2	0	0	0	1	0	1	1	1
3	0	0	0	1	1	0	1	1
4	0	0	1	0	0	1	1	1
5	0	0	1	0	1	1	1	1
6	0	0	1	1	0	1	1	1
7	0	0	1	1	1	0	1	1
8	0	1	0	0	0	1	1	1
9	0	1	0	0	1	1	1	1
10	0	1	0	1	0	1	1	1
11	0	1	0	1	1	0	1	1
12	0	1	1	0	0	1	1	1
13	0	1	1	0	1	1	1	1
14	0	1	1	1	0	1	1	1
15	0	1	1	1	1	0	1	1
16	1	0	0	0	0	1	1	1
17	1	0	0	0	1	1	1	1
18	1	0	0	1	0	1	1	1
19	1	0	0	1	1	0	1	1
20	1	0	1	0	0	1	1	1
21	1	0	1	0	1	1	1	1
22	1	0	1	1	0	1	1	1
23	1	0	1	1	1	0	1	1
24	1	1	0	0	0	1	1	1
25	1	1	0	0	1	1	1	1
26	1	1	0	1	0	1	1	1
27	1	1	0	1	1	0	1	1
28	1	1	1	0	0	1	0	1
29	1	1	1	0	1	1	0	1
30	1	1	1	1	0	1	0	1
31	1	1	1	1	1	0	0	0

✓ Comportement d'une porte Non Et à 5 entrees (in1,in2,in3,in4,in5).

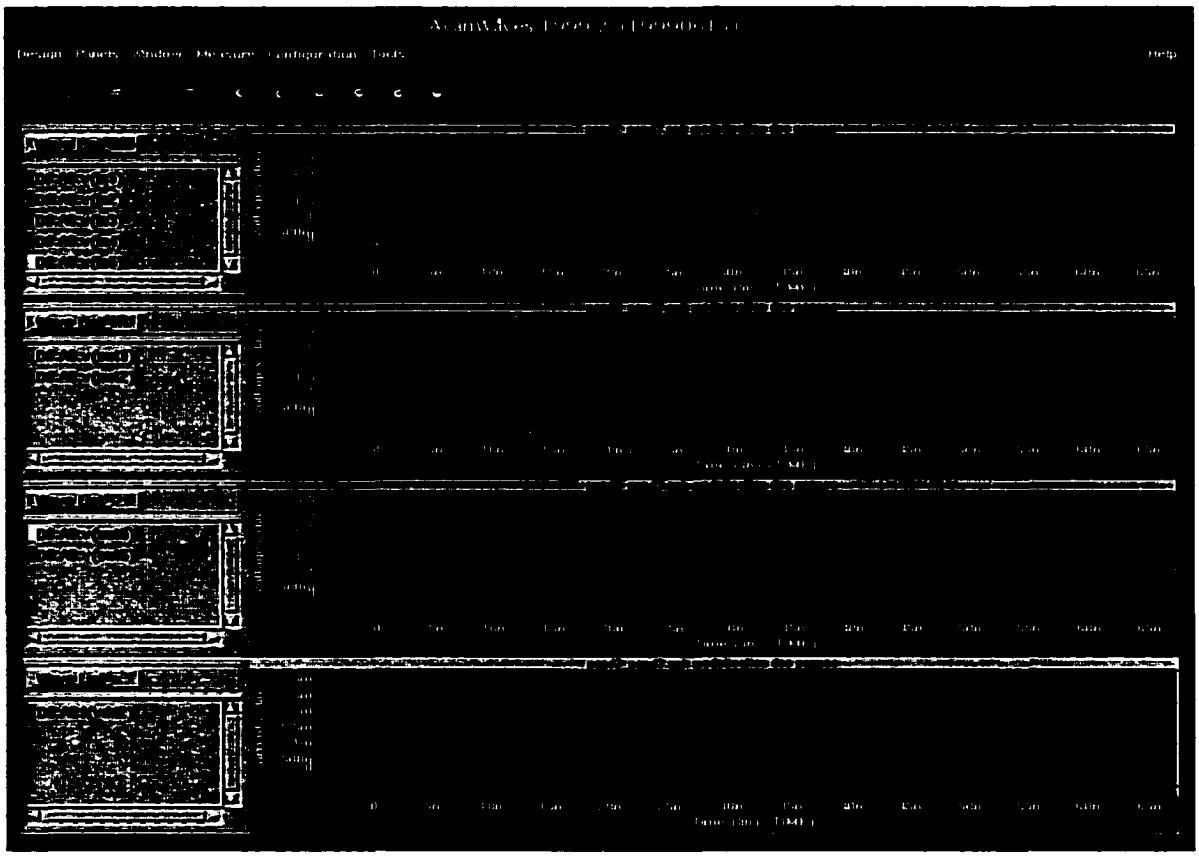
Faute -7- court circuit NAND2-NAND4:

- Code source :

```
nand2_nand4.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise    fall    high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
Vin4    in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns   15.7ns  32ns)
Vin5    in5  0  PULSE(0   3.3  32ns  0.15ns  0.15ns   31.7ns  64ns)
Vin6    in6  0  PULSE(0   3.3  64ns  0.15ns  0.15ns   63.7ns  128ns)

* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 in2  in3   out1 vdd2 0 nand4
X2 in4 in5 in6  out2 vdd2 0 nand3
* portes de charges
X3 out1 out3 vdd2 0 inv
```

```
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 64n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param   br=15000
.END
```

- **Résultats de la simulation :**



- Modélisation :

Table de vérité

Décimal	In6	In5	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	0	0	**1**	**1**	1
1	0	0	0	0	0	1	1	1	1
2	0	0	0	0	1	0	1	1	1
3	0	0	0	0	1	1	1	1	1
4	0	0	0	1	0	0	1	1	1
5	0	0	0	1	0	1	1	1	1
6	0	0	0	1	1	0	1	1	1
7	0	0	0	1	1	1	0	1	1
8	0	0	1	0	0	0	1	1	1
9	0	0	1	0	0	1	1	1	1
10	0	0	1	0	1	0	1	1	1
11	0	0	1	0	1	1	1	1	1
12	0	0	1	1	0	0	1	1	1
13	0	0	1	1	0	1	1	1	1
14	0	0	1	1	1	0	1	1	1
15	0	0	1	1	1	1	0	1	1
16	0	1	0	0	0	0	1	1	1
17	0	1	0	0	0	1	1	1	1
18	0	1	0	0	1	0	1	1	1
19	0	1	0	0	1	1	1	1	1
20	0	1	0	1	0	0	1	1	1
21	0	1	0	1	0	1	1	1	1
22	0	1	0	1	1	0	1	1	1
23	0	1	0	1	1	1	0	1	1
24	0	1	1	0	0	0	1	1	1
25	0	1	1	0	0	1	1	1	1
26	0	1	1	0	1	0	1	1	1
27	0	1	1	0	1	1	1	1	1
28	0	1	1	1	0	0	1	0	1
****									
63	1	1	1	1	1	0	0	0	0

✓ Comportement d'une porte ou à deux entrees (out1 et out2).

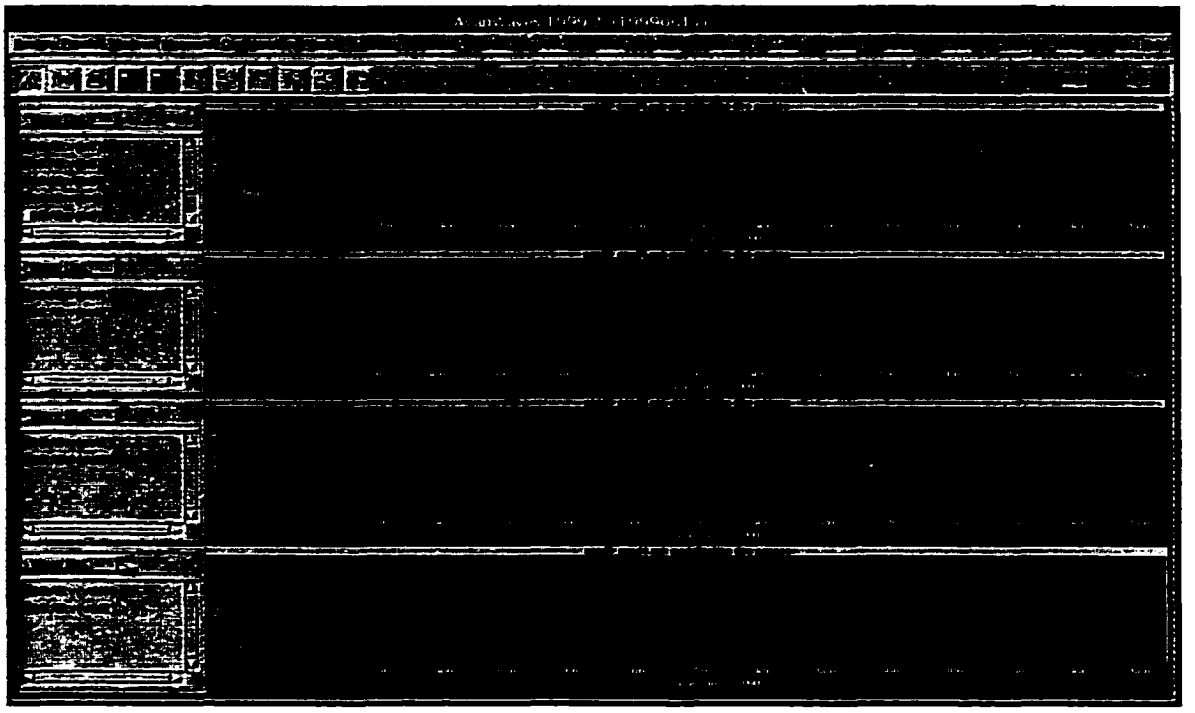
Faute -8 court circuit NAND3-NAND3:

- Code source :

```
nand3_nand3.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise    fall    high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
Vin4    in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns   15.7ns  32ns)
Vin5    in5  0  PULSE(0   3.3  32ns  0.15ns  0.15ns   31.7ns  64ns)
Vin6    in6  0  PULSE(0   3.3  64ns  0.15ns  0.15ns   63.7ns  128ns)
* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 in2 in3 out1 vdd2 0 nand3
X2 in4 in5 in6 out2 vdd2 0 nand3
* portes de charges
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
```

```
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 128n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param   br=15000
.END
```

- résultats de la simulation :



- Modélisation :

Table de vérité

Décimal	In6	In5	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	0	0	**1**	**1**	1
1	0	0	0	0	0	1	1	1	1
2	0	0	0	0	1	0	1	1	1
3	0	0	0	0	1	1	1	1	1
4	0	0	0	1	0	0	1	1	1
5	0	0	0	1	0	1	1	1	1
6	0	0	0	1	1	0	1	1	1
7	0	0	0	1	1	1	0	1	1
8	0	0	1	0	0	0	1	1	1
9	0	0	1	0	0	1	1	1	1
10	0	0	1	0	1	0	1	1	1
11	0	0	1	0	1	1	1	1	1
12	0	0	1	1	0	0	1	1	1
13	0	0	1	1	0	1	1	1	1
14	0	0	1	1	1	0	1	1	1
15	0	0	1	1	1	1	0	1	1
16	0	1	0	0	0	0	1	1	1
17	0	1	0	0	0	1	1	1	1
18	0	1	0	0	1	0	1	1	1
19	0	1	0	0	1	1	1	1	1
20	0	1	0	1	0	0	1	1	1
21	0	1	0	1	0	1	1	1	1
22	0	1	0	1	1	0	1	1	1
23	0	1	0	1	1	1	0	1	1
24	0	1	1	0	0	0	1	1	1
25	0	1	1	0	0	1	1	1	1
26	0	1	1	0	1	0	1	1	1
27	0	1	1	0	1	1	1	1	1
28	0	1	1	1	0	0	1	1	1
29	0	1	1	1	0	1	1	1	1
30	0	1	1	1	1	0	1	1	1
31	0	1	1	1	1	1	0	1	1

32	1	0	0	0	0	0	1	1	1
33	1	0	0	0	0	1	1	1	1
34	1	0	0	0	1	0	1	1	1
35	1	0	0	0	1	1	1	1	1
36	1	0	0	1	0	0	1	1	1
37	1	0	0	1	0	1	1	1	1
38	1	0	0	1	1	0	1	1	1
39	1	0	0	1	1	1	0	1	1
40	1	0	1	0	0	0	1	1	1
41	1	0	1	0	0	1	1	1	1
42	1	0	1	0	1	0	1	1	1
43	1	0	1	0	1	1	1	1	1
44	1	0	1	1	0	0	1	1	1
45	1	0	1	1	0	1	1	1	1
46	1	0	1	1	1	0	1	1	1
47	1	0	1	1	1	1	0	1	1
48	1	1	0	0	0	0	1	1	1
49	1	1	0	0	0	1	1	1	1
50	1	1	0	0	1	0	1	1	1
51	1	1	0	0	1	1	1	1	1
52	1	1	0	1	0	0	1	1	1
53	1	1	0	1	0	1	1	1	1
54	1	1	0	1	1	0	1	1	1
55	1	1	0	1	1	1	0	1	1
56	1	1	1	0	0	0	1	0	1
57	1	1	1	0	0	1	1	0	1
58	1	1	1	0	1	0	1	0	1
59	1	1	1	0	1	1	1	0	1
60	1	1	1	1	0	0	1	0	1
61	1	1	1	1	0	1	1	0	1
62	1	1	1	1	1	0	1	0	1
63	1	1	1	1	1	1	0	0	0

✓ Comportement d'une porte Ou à 2 entrées (out1 et out2).

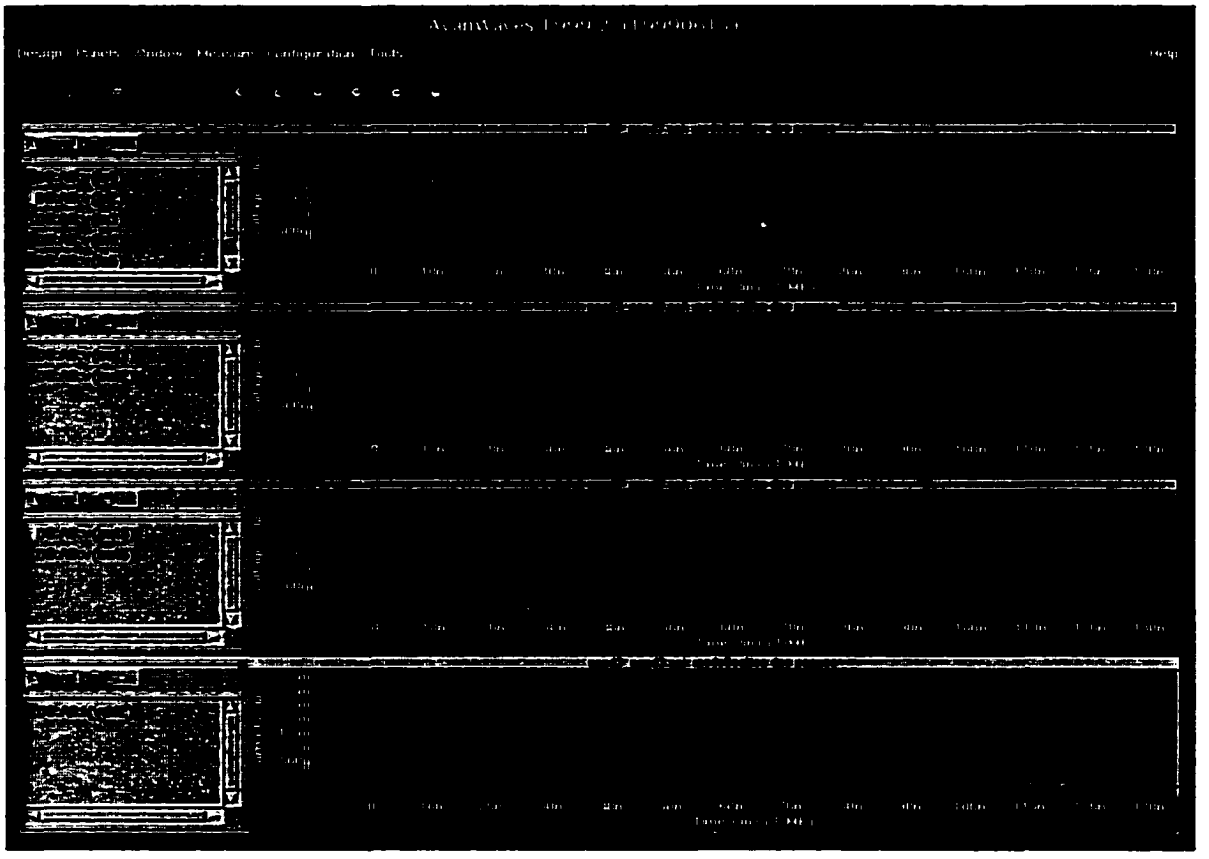
Faute -9 court circuit NAND3-NAND4:

- Code source :

```
nand3_nand4.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin delai1 rise    fall    high periode
Vin1    in1 0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2    in2 0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3    in3 0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
Vin4    in4 0  PULSE(0   3.3  16ns  0.15ns  0.15ns   15.7ns  32ns)
Vin5    in5 0  PULSE(0   3.3  32ns  0.15ns  0.15ns   31.7ns  64ns)
Vin6    in6 0  PULSE(0   3.3  64ns  0.15ns  0.15ns   63.7ns  128ns)
Vin7    in7 0  PULSE(0   3.3  128ns 0.15ns  0.15ns   83.7ns  256ns)
* resistance de court-circuit
Rbr out1 out2 br
.param   br=10
* portes impliquees dans le cc
X1 in1 in2 in3 out1 vdd2 0 nand3
X2 in4 in5 in6 in7out2 vdd2 0 nand4
* portes de charges
X3 out1 out3 vdd2 0 inv
```

```
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 256n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param   br=15000
.END
```

- résultats de la simulation :



- Modélisation :

Table de vérité

Décimal	In7	In6	In5	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	0	0	0	**1**	**1**	1
1	0	0	0	0	0	0	1	1	1	1
2	0	0	0	0	0	1	0	1	1	1
3	0	0	0	0	0	1	1	1	1	1
4	0	0	0	0	1	0	0	1	1	1
5	0	0	0	0	1	0	1	1	1	1
6	0	0	0	0	1	1	0	1	1	1
7	0	0	0	0	1	1	1	0	1	1
8	0	0	0	1	0	0	0	1	1	1
9	0	0	0	1	0	0	1	1	1	1
10	0	0	0	1	0	1	0	1	1	1
11	0	0	0	1	0	1	1	1	1	1
12	0	0	0	1	1	0	0	1	1	1
13	0	0	0	1	1	0	1	1	1	1
14	0	0	0	1	1	1	0	1	1	1
15	0	0	0	1	1	1	1	0	1	1
16	0	0	1	0	0	0	0	1	1	1
17	0	0	1	0	0	0	1	1	1	1
18	0	0	1	0	0	1	0	1	1	1
19	0	0	1	0	0	1	1	1	1	1
20	0	0	1	0	1	0	0	1	1	1
21	0	0	1	0	1	0	1	1	1	1
22	0	0	1	0	1	1	0	1	1	1
23	0	0	1	0	1	1	1	0	1	1
24	0	0	1	1	0	0	0	1	1	1
25	0	0	1	1	0	0	1	1	1	1
26	0	0	1	1	0	1	0	1	1	1
27	0	0	1	1	0	1	1	1	1	1
28	0	0	1	1	1	0	0	1	1	1
29	0	0	1	1	1	0	1	1	1	1
30	0	0	1	1	1	1	0	1	1	1

31	0	0	1	1	1	1	1	0	1	1
32	0	1	0	0	0	0	0	1	1	1
33	0	1	0	0	0	0	1	1	1	1
34	0	1	0	0	0	1	0	1	1	1
35	0	1	0	0	0	1	1	1	1	1
36	0	1	0	0	1	0	0	1	1	1
37	0	1	0	0	1	0	1	1	1	1
38	0	1	0	0	1	1	0	1	1	1
39	0	1	0	0	1	1	1	0	1	1
40	0	1	0	1	0	0	0	1	1	1
41	0	1	0	1	0	0	1	1	1	1
42	0	1	0	1	0	1	0	1	1	1
43	0	1	0	1	0	1	1	1	1	1
44	0	1	0	1	1	0	0	1	1	1
45	0	1	0	1	1	0	1	1	1	1
46	0	1	0	1	1	1	0	1	1	1
47	0	1	0	1	1	1	1	0	1	1
48	0	1	1	0	0	0	0	1	1	1
49	0	1	1	0	0	0	1	1	1	1
50	0	1	1	0	0	1	0	1	1	1
51	0	1	1	0	0	1	1	1	1	1
52	0	1	1	0	1	0	0	1	1	1
53	0	1	1	0	1	0	1	1	1	1
54	0	1	1	0	1	1	0	1	1	1
55	0	1	1	0	1	1	1	0	1	1
56	0	1	1	1	0	0	0	1	1	1
57	0	1	1	1	0	0	1	1	1	1
58	0	1	1	1	0	1	0	1	1	1
59	0	1	1	1	0	1	1	1	1	1
60	0	1	1	1	1	0	0	1	1	1
					***					
127	1	1	1	1	1	1	1	0	0	0

✓ Comportement d'une porte logique Ou à deux entreés (out1 et out2).

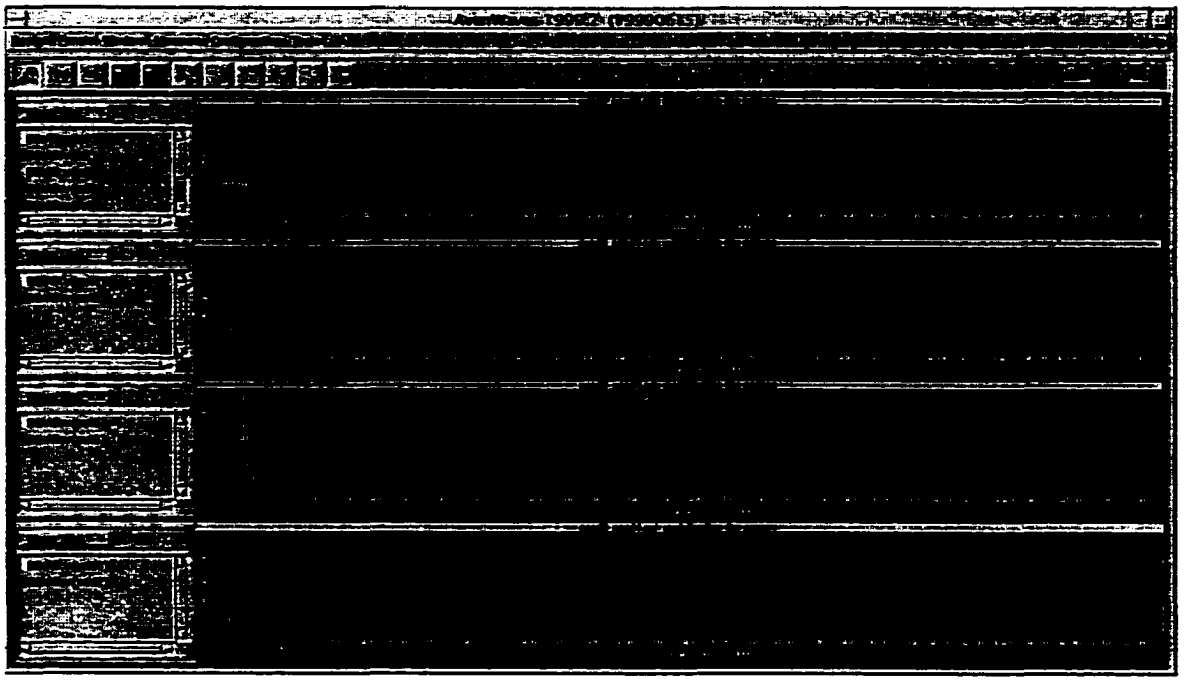
Faute -10 court circuit NAND4-NAND4:

- Code source :

```
Nand4_nand4.spi: modelisation des pannes
*date de creation 27 mars 2001
* fichier de modeles de transistors, technologie 0.35 um
.include logp3v5v_mod_ts.spi
* fichier de modeles de portes
.include modelcmosp35.spi
* alimentation principale
VDD1    vdd1    0    DC 3.3
* alimentation secondaire pour courant positif
VDD2    vdd1    vdd2    DC 0
*signaux d'entrees   depart fin  delai1  rise    fall    high  periode
Vin1    in1  0  PULSE(0   3.3  2ns   0.15ns  0.15ns   1.7ns   4ns)
Vin2    in2  0  PULSE(0   3.3  4ns   0.15ns  0.15ns   3.7ns   8ns)
Vin3    in3  0  PULSE(0   3.3  8ns   0.15ns  0.15ns   7.7ns   16ns)
Vin4    in4  0  PULSE(0   3.3  16ns  0.15ns  0.15ns   15.7ns  32ns)
Vin5    in5  0  PULSE(0   3.3  32ns  0.15ns  0.15ns   31.7ns  64ns)
Vin6    in6  0  PULSE(0   3.3  64ns  0.15ns  0.15ns   63.7ns  128ns)
Vin7    in7  0  PULSE(0   3.3  128ns 0.15ns  0.15ns   83.7ns  256ns)
Vin8    in8  0  PULSE(0   3.3  256ns 0.15ns  0.15ns   103.7ns 512ns)

* resistance de court-circuit
Rbr out1 out2 br
.param  br=10
* portes impliquees dans le cc
X1 in1 in2 in3 in4 out1 vdd2 0 nand4
X2 in5 in6 in7 in8 out2 vdd2 0 nand4
* portes de charges
```

```
X3 out1 out3 vdd2 0 inv
X4 out2 out4 vdd2 0 inv
X5 out3 out5 vdd2 0 inv
X6 out4 out6 vdd2 0 inv
.OPTIONS LIMPTS=350
+ABSTOL=1NA
+NOMOD
+POST
*+PROBE signaux a acceder
.TRAN    .1n 512n
*.print i(vdd2) v(out1) v(out2)
*.alter
*.param   br=15000
.END
```

- résultats de la simulation :



- Modélisation :

Table de vérité

Décimal	In8	In7	In6	In5	In4	In3	In2	In1	Out1	Out2	out
0	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	0	0	0	1	0	1	1	1
3	0	0	0	0	0	0	1	1	1	1	1
4	0	0	0	0	0	1	0	0	1	1	1
5	0	0	0	0	0	1	0	1	1	1	1
6	0	0	0	0	0	1	1	0	1	1	1
7	0	0	0	0	0	1	1	1	0	1	1
8	0	0	0	0	1	0	0	0	1	1	1
9	0	0	0	0	1	0	0	1	1	1	1
10	0	0	0	0	1	0	1	0	1	1	1
11	0	0	0	0	1	0	1	1	1	1	1
12	0	0	0	0	1	1	0	0	1	1	1
13	0	0	0	0	1	1	0	1	1	1	1
14	0	0	0	0	1	1	1	0	1	1	1
15	0	0	0	0	1	1	1	1	0	1	1
16	0	0	0	1	0	0	0	0	1	1	1
17	0	0	0	1	0	0	0	1	1	1	1
18	0	0	0	1	0	0	1	0	1	1	1
19	0	0	0	1	0	0	1	1	1	1	1
20	0	0	0	1	0	1	0	0	1	1	1
21	0	0	0	1	0	1	0	1	1	1	1
22	0	0	0	1	0	1	1	0	1	1	1
23	0	0	0	1	0	1	1	1	0	1	1
24	0	0	0	1	1	0	0	0	1	1	1
25	0	0	0	1	1	0	0	1	1	1	1
26	0	0	0	1	1	0	1	0	1	1	1
27	0	0	0	1	1	0	1	1	1	1	1
28	0	0	0	1	1	1	0	0	1	1	1
29	0	0	0	1	1	1	0	1	1	1	1

30	0	0	0	1	1	1	1	0	1	1	1
31	0	0	0	1	1	1	1	1	0	1	1
32	0	0	1	0	0	0	0	0	1	1	1
33	0	0	1	0	0	0	0	1	1	1	1
34	0	0	1	0	0	0	1	0	1	1	1
35	0	0	1	0	0	0	1	1	1	1	1
36	0	0	1	0	0	1	0	0	1	1	1
37	0	0	1	0	0	1	0	1	1	1	1
38	0	0	1	0	0	1	1	0	1	1	1
39	0	0	1	0	0	1	1	1	0	1	1
40	0	0	1	0	1	0	0	0	1	1	1
41	0	0	1	0	1	0	0	1	1	1	1
42	0	0	1	0	1	0	1	0	1	1	1
43	0	0	1	0	1	0	1	1	1	1	1
44	0	0	1	0	1	1	0	0	1	1	1
45	0	0	1	0	1	1	0	1	1	1	1
46	0	0	1	0	1	1	1	0	1	1	1
47	0	0	1	0	1	1	1	1	0	1	1
48	0	0	1	1	0	0	0	0	1	1	1
49	0	0	1	1	0	0	0	1	1	1	1
50	0	0	1	1	0	0	1	0	1	1	1
51	0	0	1	1	0	0	1	1	1	1	1
52	0	0	1	1	0	1	0	0	1	1	1
53	0	0	1	1	0	1	0	1	1	1	1
54	0	0	1	1	0	1	1	0	1	1	1
55	0	0	1	1	0	1	1	1	0	1	1
56	0	0	1	1	1	0	0	0	1	1	1
57	0	0	1	1	1	0	0	1	1	1	1
58	0	0	1	1	1	0	1	0	1	1	1
59	0	0	1	1	1	0	1	1	1	1	1
60	0	0	1	1	1	1	0	0	1	1	1
***											

✓ Comportement d'un court-circuit avec la ligne d'alimentation (VDD).

ANNEXE 10

Les résultats de simulation sur Verilog-xl

- ***Procédure de simulation sur Verilog-xl***

➢ *Synopsys : étape d'analyse et synthèse du code Hdl*

1) Préparer le fichier vhdl pour l'analyse et la synthèse sur synopsys.
2) Analyse et synthèse sur synopsys en ciblant une technologie ***cmos***.
3) Ajouter les pads d'entrées sorties.

➢ *Cadence : étape de simulation du netlist sur Verilog-xl*

4) Créer une librairie de travail et importer le fichier verilog.
5) Démarrer le logiciel Verilog-xl.
6) Configurer l'environnement de travail et noter le répertoire de simulation sous la forme xxxx.runx.
7) Cliquer sur setup et modifier les paramètres suivant sur la fenêtre qui va s'ouvrir :

Netlist Global Grown Nets : remplacer GND! par VSS!

Simulation library directory : ajouter le path suivant :

/CMC/kits/cmosp35/models/verilog/udp /CMC/kits/cmosp35/

8) Cliquer sur simulation et choisir : créer un textfixture.
9) Cliquer sur Start interactive, durant cette étape le netlist Verilog-xl sera crée, il est situé dans le répertoire :

xxxx.runx/ihnl/cds0/netlist

10) Exécuter la commande :

filtre netlist.v porte1 porte2 .

11) Préparer le fichier ***test bench*** en utilisant les fichiers **txr1.dat** et **txtr2.dat.**
12) Simulation en exécutant la commande :

Verilog netlist.v tb.v lib.v > resultat.txt.

13) Exécuter la commande

Format resultat.txt

- ***Netlist initial format verilog***

```
// Library - multi, Cell - mult_4X4, View - schematic
// LAST TIME SAVED: Mar 13 15:39:58 2001
// NETLIST TIME: Jun 13 19:12:52 2001
`timescale 1ns / 1ns
module mult_4X4 ( ina, inb, prod );
output [7:0]  prod;
input [3:0]  ina;
input [3:0]  inb;
specify
   specparam CDS_LIBNAME  = "multi";
   specparam CDS_CELLNAME = "mult_4X4";
   specparam CDS_VIEWNAME = "schematic";
endspecify
winv_1  U80 ( cds_globals.VDD_, cds_globals.VSS_, n95, n64);
winv_1  U46 ( cds_globals.VDD_, cds_globals.VSS_, n101, n81);
winv_1  U81 ( cds_globals.VDD_, cds_globals.VSS_, n62, n61);
winv_1  U44 ( cds_globals.VDD_, cds_globals.VSS_, n99, n72);
winv_1  U82 ( cds_globals.VDD_, cds_globals.VSS_, n92, n59);
winv_1  U48 ( cds_globals.VDD_, cds_globals.VSS_, n51, n50);
winv_1  U50 ( cds_globals.VDD_, cds_globals.VSS_, n55, n54);
winv_1  U76 ( cds_globals.VDD_, cds_globals.VSS_, n90, n89);
winv_1  U45 ( cds_globals.VDD_, cds_globals.VSS_, n98, n70);
winv_1  U65 ( cds_globals.VDD_, cds_globals.VSS_, n100, n78);
wnand2_1  U22 ( cds_globals.VDD_, cds_globals.VSS_, n70, n131, n136);
wnand2_1  U73 ( cds_globals.VDD_, cds_globals.VSS_, n96, n136, n133);
wnand2_1  U62 ( cds_globals.VDD_, cds_globals.VSS_, n110, n77, n76);
wnand2_1  U60 ( cds_globals.VDD_, cds_globals.VSS_, n108, n57, n73);
wnand2_1  U30 ( cds_globals.VDD_, cds_globals.VSS_, n82, n135, n130);
wnand2_1  U31 ( cds_globals.VDD_, cds_globals.VSS_, n46, n83, n84);
wnand2_1  U52 ( cds_globals.VDD_, cds_globals.VSS_, n75, n98, n102);
wnand2_1  U57 ( cds_globals.VDD_, cds_globals.VSS_, n58, n105, n104);
wnand2_1  U71 ( cds_globals.VDD_, cds_globals.VSS_, n63, n137, n130);
wnand2_1  U54 ( cds_globals.VDD_, cds_globals.VSS_, n103, n55, n67);
wnand2_1  U67 ( cds_globals.VDD_, cds_globals.VSS_, n86, n90, n111);
wnand2_1  U70 ( cds_globals.VDD_, cds_globals.VSS_, n83, n101, n44);
                                        ***
wpadin  U98 ( SUBCORE, SUBESD, SUBRING, VDDCORE, VDDRING, n132,
    ina[1]);
endmodule
```

- ***Txr1.dat : les noms des nœuds à simuler***

top.n95,top.n64,
top.n101,top.n81,
top.n62,top.n61,
top.n99,top.n72,
top.n92,top.n59,
top.n51,top.n50,
top.n55,top.n54,
top.n90,top.n89,
top.n98,top.n70,
top.n100,top.n78,
top.n70,top.n131,top.n136,
top.n96,top.n136,top.n133,
top.n110,top.n77,top.n76,
top.n108,top.n57,top.n73,
top.n82,top.n135,top.n130,
top.n46,top.n83,top.n84,
top.n75,top.n98,top.n102,
top.n58,top.n105,top.n104,
top.n63,top.n137,top.n130,
top.n103,top.n55,top.n67,
top.n86,top.n90,top.n111,
top.n83,top.n101,top.n44,
top.n84,top.n44,top.n109,
top.n54,top.n68,top.n69,
top.n47,top.n85,top.n86,
top.n68,top.n65,top.n51,
top.n81,top.n134,top.n58,
top.n102,top.n53,top.n71,
top.n48,top.n130,top.n134,
top.n72,top.n131,top.n135,
top.n109,top.n108,top.n107,
top.n67,top.n133,top.n134,
top.n77,top.n136,top.n130,
top.n107,top.n99,top.n106,
top.n78,top.n79,top.n80,
top.n73,top.n74,top.n75,

***

top.n138,top.n45,top.n48,
top.SUBRING,top.VDDCORE,top.VDDRING,top.net1268,

- ***Txr2.dat format de sauvegarde des noeuds à simuler***

%b%b %b%b	%b%b %b%b	%b%b %b%b	%b%b %b%b	%b%b %b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b%b	%b%b%b%b	%b%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b%b	%b%b%b%b	%b%b%b%b
%b%b%b%b	%b%b%b%b	%b%b%b%b	%b%b%b%b	%b%b%b%b	%b%b%b%b
%b%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b%b
%b%b%b	%b%b%b	%b%b%b	%b%b%b	%b%b %b%b%b	
%b%b%b	%b%b%b%b	%b%b%b%b			

- ***Netlist final à simuler sur Verilog-xl***

```
// Library - multi, Cell - mult_4X4, View - schematic
// LAST TIME SAVED: Mar 13 15:39:58 2001
// NETLIST TIME: Jun 13 19:12:52 2001
`timescale 1ns / 1ns
module mult_4X4 ( ina, inb, prod );
output [7:0]  prod;
input [3:0]  ina;
input [3:0]  inb;
specify
   specparam CDS_LIBNAME  = "multi";
   specparam CDS_CELLNAME = "mult_4X4";
   specparam CDS_VIEWNAME = "schematic";
endspecify
winv_1  U80 ( VDD, VSS, n95, n64);
winv_1  U46 ( VDD, VSS, n101, n81);
winv_1  U81 ( VDD, VSS, n62, n61);
winv_1  U44 ( VDD, VSS, n99, n72);
winv_1  U82 ( VDD, VSS, n92, n59);
winv_1  U48 ( VDD, VSS, n51, n50);
winv_1  U50 ( VDD, VSS, n55, n54);
winv_1  U76 ( VDD, VSS, n90, n89);
winv_1  U45 ( VDD, VSS, n98, n70);
winv_1  U65 ( VDD, VSS, n100, n78);
wnand2_1  U22 ( VDD, VSS, n70, n131, n136);
wnand2_1  U73 ( VDD, VSS, n96, n136, n133);
wnand2_1  U62 ( VDD, VSS, n110, n77, n76);
wnand2_1  U60 ( VDD, VSS, n108, n57, n73);
wnand2_1  U30 ( VDD, VSS, n82, n135, n130);
wnand2_1  U31 ( VDD, VSS, n46, n83, n84);
wnand2_1  U52 ( VDD, VSS, n75, n98, n102);
wnand2_1  U57 ( VDD, VSS, n58, n105, n104);
wnand2_1  U71 ( VDD, VSS, n63, n137, n130);
wnand2_1  U54 ( VDD, VSS, n103, n55, n67);
wnand2_1  U67 ( VDD, VSS, n86, n90, n111);

                                    ****

wpadin  U98 ( SUBCORE, SUBESD, SUBRING, VDDCORE, VDDRING, n132,
    ina[1]);

endmodule
```

- **Resultat.txt**

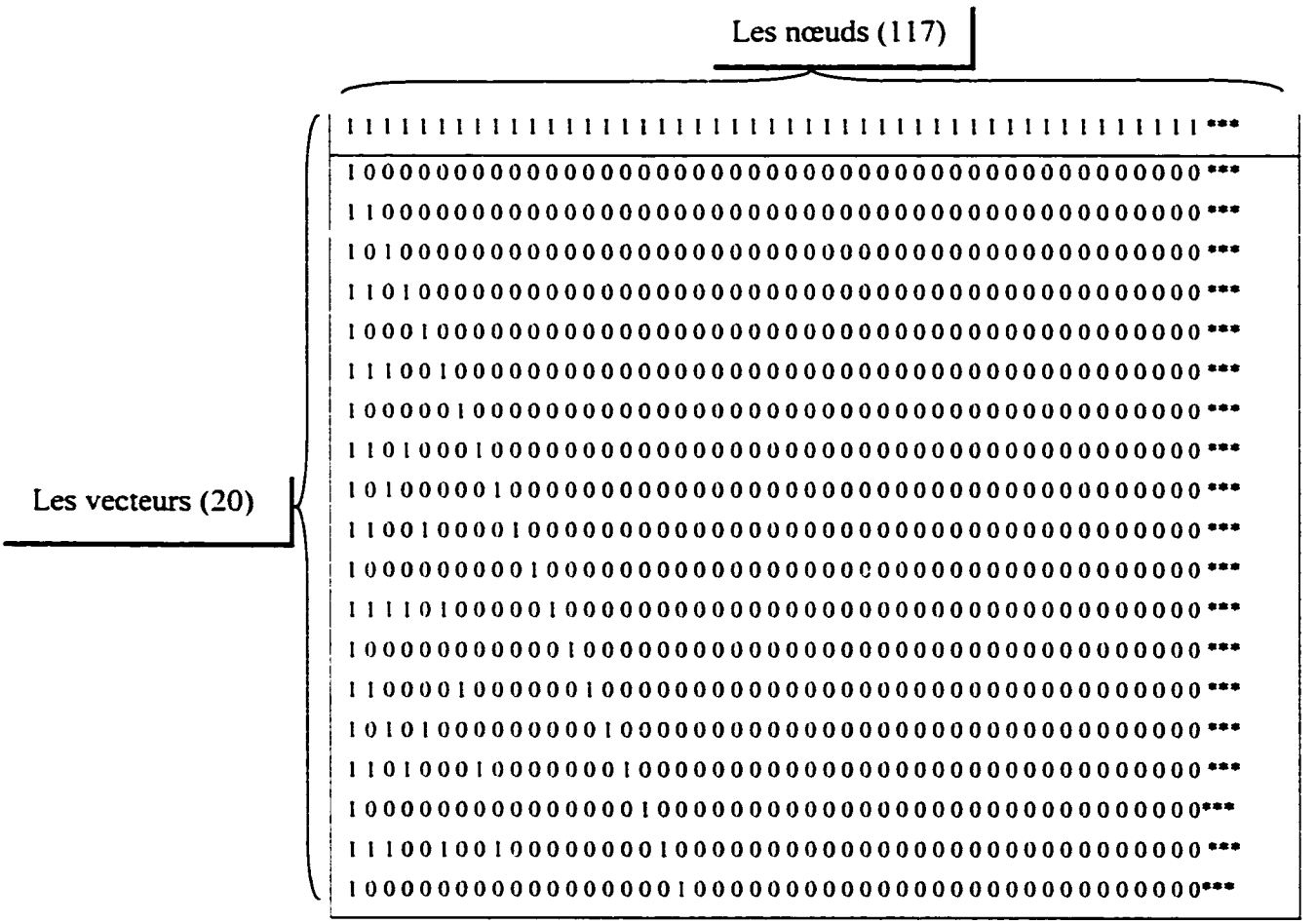
```
VERILOG-XL 3.0.p001   Sep 27, 2001  16:48:16
Copyright (c) 1995 Cadence Design Systems, Inc.  All Rights Reserved.
***
Compiling source file "netlist.v"
Compiling source file "test_bench_yas.v"
***
wand2_4
wand3_2
wand3_4
wand4_1
wand4_2
wand4_4
wbuf_2
wbuf_4
***
```

Les portes logiques

Les résultats logiques des nœuds internes

```
1 - 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 c
2 - 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 c
3 - 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 c
4 - 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 c
5 - 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1
1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1
0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 c
***
17 - 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 0 1 0 0 c

L143 "test_bench_yas.v": $finish at simulation time 1057000
49 warnings
0 simulation events (use +profile or +listcounts option to count) +
48386 accelerated events + 12882 timing check events
CPU time: 0.7 secs to compile + 0.6 secs to link + 1.0 secs in
simulation
End of VERILOG-XL 3.0.p001   Sep 27, 2001  16:48:19
```

- ***Formatage des résultats***



ANNEXE 11

Démonstration du programme de préparation
La première étape d'intégration

➢ ***Netlist du schéma***

```
module mult_4X4 ( ina, inb, prod );
input  [3:0] ina;
input  [3:0] inb;
output [7:0] prod;
   wire n142, n140, n144, n137, n130, n135, n132, n134, n133, n136, n131,
      n143, n139, n138, n141, n43, n44, n45, n46, n47, n48, n49, n50, n51,
      n52, n53, n54, n55, n56, n57, n58, n59, n60, n61, n62, n63, n64, n65,
      n66, n67, n68, n69, n70, n71, n72, n73, n74, n75, n76, n77, n78, n79,
      n80, n81, n82, n83, n84, n85, n86, n87, n88, n89, n90, n91, n92, n93,
      n94, n95, n96, n97, n98, n99, n100, n101, n102, n103, n104, n105, n106,
      n107, n108, n109, n110, n111, n112, n120;
   wand2_1 U6 ( .op(n144), .ip1(n137), .ip2(n133) );
   wand2_1 U7 ( .op(n43), .ip1(n134), .ip2(n132) );
   wand2_1 U8 ( .op(n44), .ip1(n131), .ip2(n134) );
   wnor2_1 U9 ( .op(n45), .ip1(n46), .ip2(n47) );
   wnor2_1 U10 ( .op(n138), .ip1(n45), .ip2(n48) );
   wmux2_2 U11 ( .op(n49), .i0(n50), .i1(n51), .c(n52) );
   wmux2_2 U12 ( .op(n53), .i0(n54), .i1(n55), .c(n56) );
   wxor2_2 U13 ( .op(n57), .ip1(n58), .ip2(n43) );
   wxor2_2 U14 ( .op(n59), .ip1(n57), .ip2(n60) );
   wmux2_2 U15 ( .op(n141), .i0(n61), .i1(n62), .c(n63) );
   wnand2_1 U16 ( .op(n64), .ip1(n132), .ip2(n137) );
   wnand2_1 U17 ( .op(n50), .ip1(n136), .ip2(n132) );
   wand2_1 U18 ( .op(n65), .ip1(n133), .ip2(n135) );
   wand2_1 U19 ( .op(n66), .ip1(n132), .ip2(n135) );
   wnand2_1 U20 ( .op(n67), .ip1(n133), .ip2(n134) );

                                        ***


   winv_2 U91 ( .op(n120), .ip(n87) );
   wpadoutld U92 ( .WORLD(prod[7]), .IP(n138) );
   wpadin U93 ( .OP(n137), .WORLD(inb[0]) );
   wpadin U94 ( .OP(n136), .WORLD(inb[1]) );
   wpadin U95 ( .OP(n135), .WORLD(inb[2]) );
   wpadin U96 ( .OP(n134), .WORLD(inb[3]) );
   wpadin U97 ( .OP(n133), .WORLD(ina[0]) );
   wpadin U98 ( .OP(n132), .WORLD(ina[1]) );
   wpadin U99 ( .OP(n131), .WORLD(ina[2]) );
   wpadin U100 ( .OP(n130), .WORLD(ina[3]) );
endmodule
```

➢ ***Netlist du dessin des masques***

```
xu6 (134 169 137 145 111) wand2_1_g1
xu7 (134 169 143 151 12) wand2_1_g1
xu8 (134 169 153 143 50) wand2_1_g1
xu18 (134 169 145 141 25) wand2_1_g1
xu19 (134 169 151 141 29) wand2_1_g1
xu72 (134 169 153 137 14) wand2_1_g1
xu9 (134 169 21 22 10) wnor2_1_g2
xu10 (134 169 10 23 130) wnor2_1_g2
xu11 (134 169 15 66 65 54) wmux2_2_g3
xu12 (134 169 19 67 97 95) wmux2_2_g3
xu15 (134 169 51 82 79 121) wmux2_2_g3
xu35 (134 169 11 74 72 125) wmux2_2_g3
xu36 (134 169 20 83 87 123) wmux2_2_g3
xu38 (134 169 53 76 75 113) wmux2_2_g3
xu40 (134 169 94 61 59 17) wmux2_2_g3
xu41 (134 169 103 58 56 13) wmux2_2_g3
xu43 (134 169 109 69 108 11) wmux2_2_g3
xu78 (134 169 50 63 62 18) wmux2_2_g3
xu13 (134 169 31 12 102) wxor2_2_g4
xu14 (134 169 102 13 87) wxor2_2_g4
xu34 (134 169 22 16 132) wxor2_2_g4
xu37 (134 169 14 54 117) wxor2_2_g4
xu39 (134 169 55 25 15) wxor2_2_g4
xu42 (134 169 23 21 16) wxor2_2_g4
xu74 (134 169 95 17 82) wxor2_2_g4
xu75 (134 169 18 48 74) wxor2_2_g4
xu77 (134 169 98 29 19) wxor2_2_g4
xu79 (134 169 105 106 20) wxor2_2_g4
xu16 (134 169 151 137 76) wnand2_1_g5
xu17 (134 169 139 151 66) wnand2_1_g5
xu20 (134 169 145 143 98) wnand2_1_g5
xu21 (134 169 24 55 67) wnand2_1_g5
xu22 (134 169 153 139 61) wnand2_1_g5
xu24 (134 169 153 141 58) wnand2_1_g5

                                  ***

xu91 (134 169 132 131) winv_2_g10
xu93 (137 150 149 148 171 147 136) wpadin_g11
xu94 (139 150 149 148 171 147 138) wpadin_g11
xu95 (141 150 149 148 171 147 140) wpadin_g11
xu96 (143 150 149 148 171 147 142) wpadin_g11
xu97 (145 150 149 148 171 147 144) wpadin_g11
xu98 (151 150 149 148 171 147 146) wpadin_g11
xu99 (153 158 157 156 171 155 152) wpadin_g11
xu100 (159 158 157 156 171 155 154) wpadin_g11
xpvddring1 (167 166 165 171 163 160) wpadvddring_g12
```

➢ *Simplification des portes complexes*

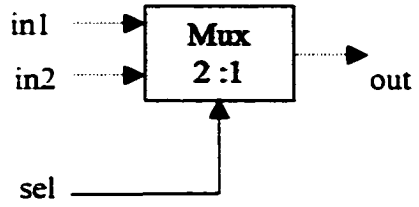
Le programme d'émulation ne peut traiter certains types de portes comme le multiplexeur. Pour résoudre ce type de problème nous avons deux choix :

- Ajouter dans le logiciel de diagnostic des blocs de code permettant de gérer ce type de porte.
- Simplifier les portes multiplexeurs sous forme de portes élémentaire.

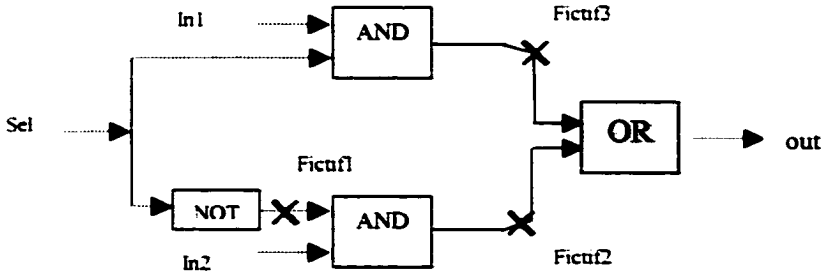
À cause de la complexité du programme d'émulation, on préfère le deuxième choix qui consiste à décomposer chaque porte multiplexeur sous forme de portes élémentaires :

- Multiplexeur

Nombre d'entrée : 3
Entrées : in1 in 2 sel
Nombre de sortie : 1
Sorties : out



➧ Simplification :



- Remarques :

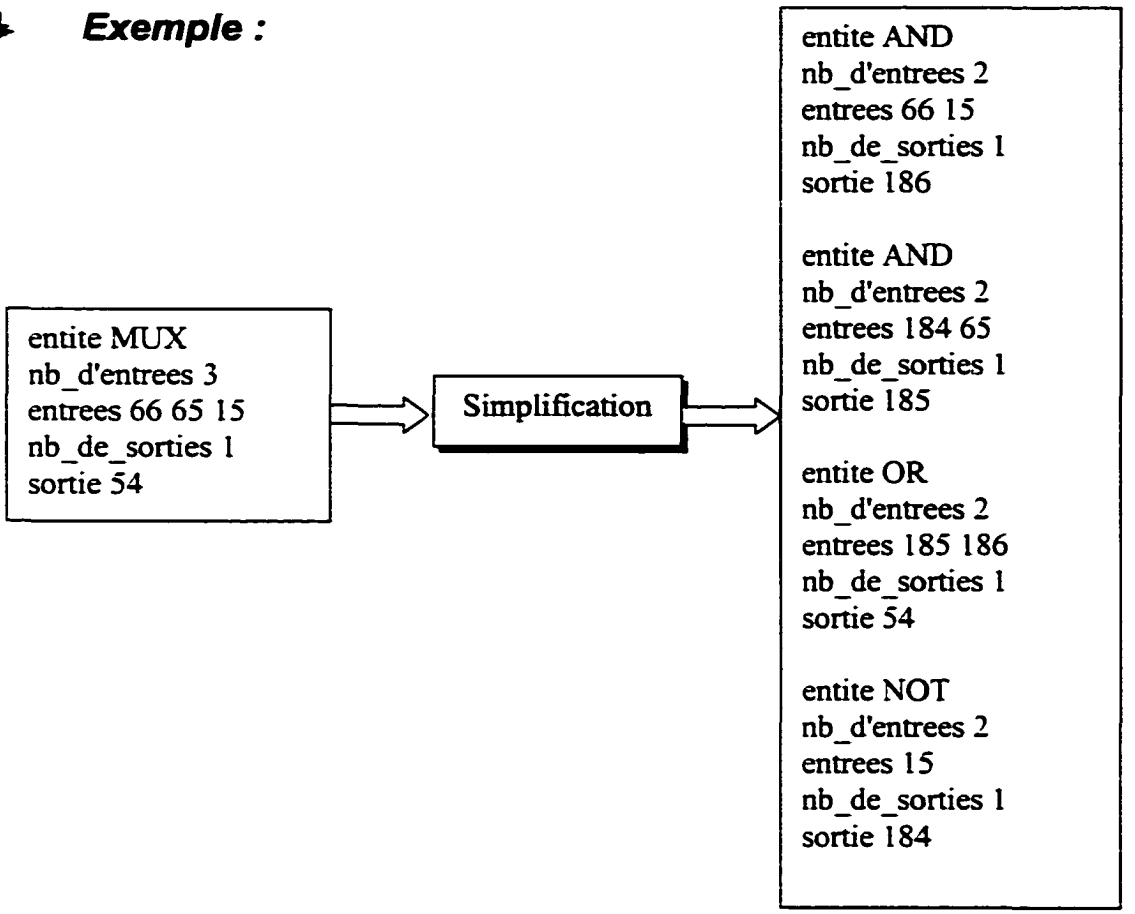
❖ Les nœuds fictifs seront ignorés dans le programme de diagnostic.

❖ Pour chaque multiplexeur on a besoin de 4 portes élémentaires et de 3 nœuds fictifs.

❖ Si on a n multiplexeurs, on aura :

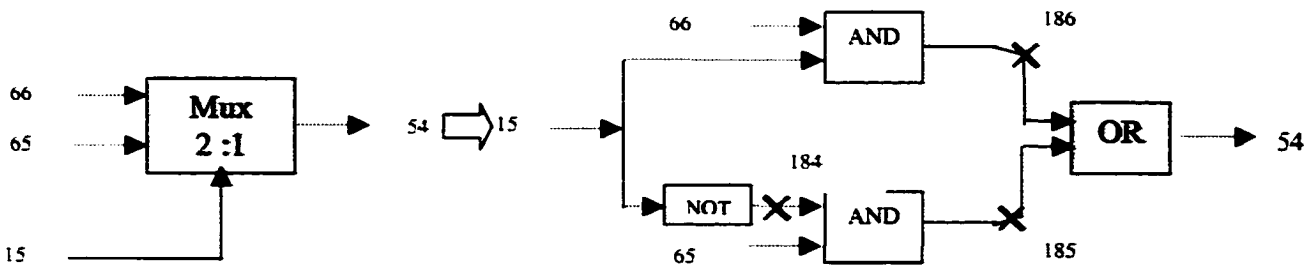
- 3*n nœuds fictifs de plus dans notre netlist
- 3*n portes logiques « élémentaire » de plus dans le netlist.

Exemple :



Schema du mux 2:1 :

Schéma simplifié du mux 2:1 :



➢ ***Liste d'équivalence***

```
n144
111
n43
12
n44
50
n45
10
n138
130
n49
54
n53
95
n57
102
n59
87
n141
121
n64
76
n50
66
n65
25
n66
29
                    ***
n131
153
n130
159
```

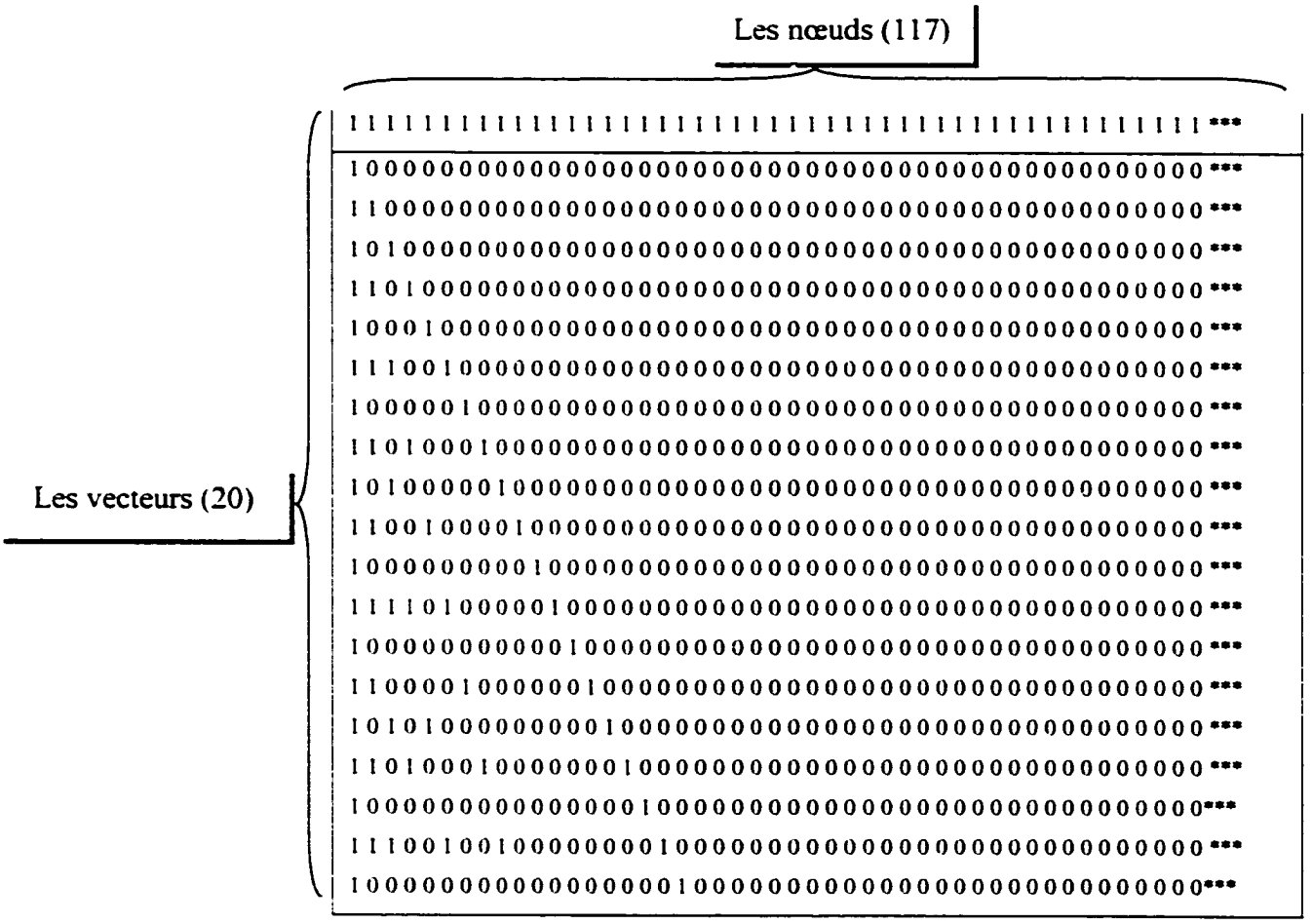
➢ ***Netlist format « TIBO »***

```
nombre noeuds 116
nombre entites 108
nombre entrees primaires 8
entrees primaires 137 139 141 143 145 151 153 159
nombre sorties primaires 8
sorties primaires 111 113 117 121 123 125 131 130

entite AND
nb_d'entrees 2
entrees 145 137
nb_de_sorties 1
sortie 111

entite AND
nb_d'entrees 2
entrees 151 143
nb_de_sorties 1
sortie 12

entite AND
nb_d'entrees 2
entrees 153 143
nb_de_sorties 1
sortie 50

                    ***


entite OR
nb_d'entrees 2
entrees 109 108
nb_de_sorties 1
sortie 107

entite NOT
nb_d'entrees 1
entrees 132
nb_de_sorties 1
sortie 131
```

ANNEXE 12

Exemple de compression d'une matrice de simulation logique de taille 117 nœuds x 20 vecteurs de simulation

➢ ***Résultat de simulation sur Verilog-xl***



➢ ***Résultats de la compression***

noeud<0> mat_comp[0][0]=1048575
noeud<1> mat_comp[0][1]=349525
noeud<2> mat_comp[0][2]=299593
noeud<3> mat_comp[0][3]=69905
noeud<4> mat_comp[0][4]=33825
noeud<5> mat_comp[0][5]=266305
noeud<6> mat_comp[0][6]=16513
noeud<7> mat_comp[0][7]=65793
noeud<8> mat_comp[0][8]=262657
noeud<9> mat_comp[0][9]=1025
noeud<10> mat_comp[0][10]=2049
noeud<11> mat_comp[0][11]=4097
noeud<12> mat_comp[0][12]=8193
noeud<13> mat_comp[0][13]=16385
noeud<14> mat_comp[0][14]=32769
noeud<15> mat_comp[0][15]=65537
noeud<16> mat_comp[0][16]=131073
noeud<17> mat_comp[0][17]=262145
noeud<18> mat_comp[0][18]=524289

***

Exemple :

- Nœud : 0.
- Vacteur : 0.

noeud<0> mat_comp[0][0]=1048575

- La valeur logique du nœud 0 au vecteur 0 = mat_comp[0/32][0]&(1<<0)) = 1.

ANNEXE 13

Ressources du programme diag_total

Circuit 1 : Multiplicateur 4X4

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (um^2)
Multiplicateur 4bits	94	117	8	8	$2x10^5$

➢ Résultats temporels du diagnostic (en s) :

- Nombre de panne à émuler : 25.

Phase	Temps en ms
temps requis pour l'initialisation	0.16634500
temps total pour l'identification	1.85985935
temps moyen pour l'identification	0.07439437
temps total pour la localisation	0.26696503
temps moyen pour la localisation	0.01067860
temps moyen de diagnostic (sans émulation I_{DDQ})	0.25141798
temps total de diagnostic (sans émulation I_{DDQ})	2.12682438
temps moyen (d'émulation I_{DDQ})	0.13133191
temps total (d'émulation I_{DDQ})	3.28329766
temps total de diagnostic (avec émulation I_{DDQ})	5.41012204
temps total du programme	5.57646704

➢ Ressources matérielles :

- Espace mémoire dynamique : 6912 bytes, 54.0000 kb.

Circuit 2 : Filtre Fir 4 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (um^2)
Fir 4 étages	901	986	42	8	$14x10^5$

➢ Résultats temporels du diagnostic :

- Nombre de panne à émuler : 132.

Phase	Temps en ms
temps requis pour l'initialisation	0.36872399
temps total pour l'identification	9.98115003
temps moyen pour l'identification	0.07561477
temps total pour la localisation	18.18829036
temps moyen pour la localisation	0.13779008
temps moyen de diagnostic (sans émulation I_{DDQ})	0.58212884
temps total de diagnostic (sans émulation I_{DDQ})	28.16944039
temps moyen (d'émulation I_{DDQ})	0.16656991
temps total (d'émulation I_{DDQ})	21.98722756
temps total de diagnostic (avec émulation I_{DDQ})	50.15666795
temps total du programme	50.52539194

➢ Ressources matérielles :

- Espace mémoire dynamique : 59648 bytes, 466.0000 kbits

Circuit 3 : Filtre Fir 7 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (um^2)
Fir 7 étages	1641	1772	66	8	$24x10^5$

➢ Résultats temporels du le diagnostic :

- Nombre de panne à émuler : 261.

Phase	Temps en ms
temps requis pour l'initialisation	0.60494602
temps total pour l'identification	19.06745219
temps moyen pour l'identification	0.07305537
temps total pour la localisation	136.23162329
temps moyen pour la localisation	0.52196024
temps moyen de diagnostic (sans émulation I_{DDQ})	1.19996163
temps total de diagnostic (sans émulation I_{DDQ})	155.29907548
temps moyen (d'émulation I_{DDQ})	0.19146333
temps total (d'émulation I_{DDQ})	49.97192848
temps total de diagnostic (avec émulation I_{DDQ})	205.27100396
temps total du programme	205.87594998

➢ Ressources matérielles :

- Espace mémoire dynamique : 112000 bytes, 875.0000 kb.

Circuit 4 : Filtre Fir15 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (um^2)
Fir 15 étages	3521	3759	130	8	$40x10^5$

➢ Résultats temporels du diagnostic :

- Nombre de panne à émuler : 450.

Phase	Temps en ms
temps requis pour l'initialisation	1.00246000
temps total pour l'identification	6.76421857
temps moyen pour l'identification	0.01503160
temps total pour la localisation	693.53179252
temps moyen pour la localisation	1.54118176
temps moyen de diagnostic (sans émulation I_{DDQ})	2.55867336
temps total de diagnostic (sans émulation I_{DDQ})	700.29601109
temps moyen (d'émulation I_{DDQ})	0.29239749
temps total (d'émulation I_{DDQ})	131.57886994
temps total de diagnostic (avec émulation I_{DDQ})	831.87488103
temps total du programme	832.87734103

➢ Ressources matérielles:

- Espace mémoire dynamique : 240128 bytes, 1876.0000 kb.

Circuit 5 : Filtre Fir25 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (um^2)
Fir 25 étages	6468	7962	210	8	$51x10^5$

➢ Résultats temporels du diagnostic :

- Nombre de panne à émuler : 512.

Phase	Temps en ms
temps requis pour l'initialisation	10.54103896
temps total pour l'identification	7.06358778
temps moyen pour l'identification	0.01304220
temps total pour la localisation	1986.40767980
temps moyen pour la localisation	3.8814807
temps moyen de diagnostic (sans émulation I_{DDQ})	14.43162923
temps total de diagnostic (sans émulation I_{DDQ})	1993.465421
temps moyen (d'émulation I_{DDQ})	0.56778218
temps total (d'émulation I_{DDQ})	182.82586348
temps total de diagnostic (avec émulation I_{DDQ})	2175.462111
temps total du programme	2186.94357002

➢ Ressources matérielles:

- Espace mémoire dynamique : 384000 bytes, 3000.0000 kb.

Circuit 6 : Filtre Fir50 étages

	Nombre de cellules	Nombre de nœuds	Nombre d'entrées primaires	Nombre de sorties primaires	Surface du circuit (um^2)
Fir 50 étages	13127	13855	410	8	$89x10^5$

➢ Résultats temporels du diagnostic :

- Nombre de panne à émuler : 671.

Phase	Temps en ms
temps requis pour l'initialisation	25.24830604
temps total pour l'identification	9.91108620
temps moyen pour l'identification	0.01477062
temps total pour la localisation	2189.25894737
temps moyen pour la localisation	3.26268099
temps moyen de diagnostic (sans émulation I_{DDQ})	28.52575765
temps total de diagnostic (sans émulation I_{DDQ})	2199.17003357
temps moyen (d'émulation I_{DDQ})	0.80105231
temps total (d'émulation I_{DDQ})	537.50610149
temps total de diagnostic (avec émulation I_{DDQ})	2736.67613506
temps total du programme	2761.92444110

➢ Ressources matérielles:

- Espace mémoire dynamique : 744000 bytes.

BIBLIOGRAPHIE

[1]: Thibeault, C. (1998). Diagnosis Method Using Probabilistic Signatures Theory and Results. Journal of electronic testing, pp.339-353.

[2]: Brglez, F., Fujiwara, H. (1985). A Neural Netlist of 10 Combinatorial Benchmark Circuits and a Target Translator in FORTRAN. Proceeding ISCAC, pp. 695-698.

[3]: J.G. Proakis, (1983). Digital Communications, 3rd edn., McGraw-Hill, New York.

[4]: Thibeault, C. (2000). On The Adaptation of Viterbi Algorithm for Diagnosis of Multiple Bridging Faults. IEEE TRANSACTION ON COMPUTERS, VOL. 49, NO. 6.

[5]: Soden, J.M., Hawkins, C.F., Miller A.C. (1996). Identifying defects in deep-submicron CMOS ICs. IEEE Spectrum, 33(9), pp. 66-71.

[6]: Maly, W., Gattiker, A.E., (1997). Current Signatures: Application. IEEE international Test Conference, pp. 892-899.

[7]: AVIZIENIS, A. The four-Universe Information System Model For The Study of Fault-Tolerance, Proceedings of the Internatuionnal Symposium on Fault Tolerant Computing, IEEE, Santa Monica (Cal.), 1982, p. 6-13

[8]: Vallet, DP. (1997). IC Falure Analysis : The importance of Test and Daignosics, IEEE Design & Test of computers, 12(4), pp.76-67.

[9]: P. Maxwell, R. Aitken, and L. Huismann, "The Effect on Quality of Non-Uniform fault Coverage and Fault Probability", Proc. IEEE Int'l Test Conf., pp. 739-746, 1994.

[10] C. Ouyang and W. Maly, "Efficient Extraction of Critical Area in Large VLSI ICs", Proc. Int'l Symp. Semiconductor Manufacturing, pp. 301-304, 1996.