MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET DE L'INFORMATIQUE

DEPARTEMENT D'ELECTRONIQUE

Mémoire de Magister

En Electronique,

Option : Microélectronique

Présenté par :

M^elle^ AIT ABDELMALEK Ghania

Sujet :

Étude et modélisation des défauts des circuits intégrés en vue de leur analyse de fiabilité

Devant le jury d'examen composé de :

- M^r^ Belkaid Mohammed Said Professeur à UMMTO — President
- M^r^ Ziani Rezki Maitre de conférences A à UMMTO — Rapporteur
- M^r^ Laghrouche Mourad Maitre de conférences A à UMMTO — Examinateur
- M^r^ Bensidhoum Mohand Outahar Maitre de conférences B à UMMTO — Examinateur

Thèse soutenue le : 25 / 04 / 2011

Dédicaces

A la mémoire de mon grand père

A mes très chers et adorables parents

A ma très chère sœur Zina et son mari Hakim pour leur soutien et leur présence permanente

A mon amie Nora qui a été là à mes côtés du début jusqu'à la fin. Merci Nora.

A mes très chers frères, à mes belles sœurs

A tous les bons cœurs qui travaillent dur, pour faire avancer les choses

Remerciements

En premier lieu, je remercie Dieu, le tout puissant, pour m'avoir donné la patience, la volonté et la force nécessaires pour terminer ce travail.

Je tiens à remercier tout particulièrement mon promoteur Monsieur Rezki Ziani, Maître de conférences A et chef de département électronique à l'UMMTO, de m'avoir confié ce travail et de l'avoir dirigé avec simplicité et objectivité, qu'il trouve ici l'expression de mon respect et de ma profonde reconnaissance.

Je remercie le Professeur Mohammed Said Belkaid, doyen de la faculté de génie électrique et informatique à l'UMMTO, pour m'avoir permis de découvrir le domaine si riche et intéressant de la microélectronique et pour l'honneur qu'il m'a fait en présidant ce jury.

Je remercie également Messieurs Mourad Laghrouche, Maître de conférences A à l'UMMTO et Mohand Outahar Bensidhoum, Maître de conférences B à l'UMMTO, pour l'intérêt qu'ils ont porté à mon travail en acceptant de participer au jury.

Mes remerciements s'adressent aussi à toutes les personnes qui de près ou de loin m'ont aidé à réaliser ce travail, particulièrement :

Monsieur Ahcène Lakhlef, Maître assistant à l'UMMTO, pour son aide et son soutien permanent

Monsieur Mehammed Daoui, Maître de conférences B à l'UMMTO

Monsieur Abdelhakim Khouas, Professeur à l'UMBB

Monsieur Idir Mellal, Master doctorant à l'UMMTO

Mademoiselle Faroudja Abdi, de l'équipe SoC (System On Chip) de la division microélectronique et nanotechnologie du centre de développement des technologies avancées « CDTA ».

Avec tous mes respects, merci encore une fois à toutes ces personnes.

Sommaire

CHAPITRE III : CONCEPTION EN VUE DE FIABILITÉ ET TOLRÉANCE AUX FAUTES

Résumé

De façon à répondre à la demande émergente des industriels pour le développement de méthodes pouvant augmenter et garantir un degré de fiabilité optimal des circuits intégrés tôt dans le flot de conception même en présence de défauts, nous proposons dans le cadre de ma thèse, une méthodologie susceptible d'augmenter la fiabilité de tels circuits via l'augmentation du rendement de fabrication.

Cette dernière se décompose en deux points :

- Méthodologie de test des circuits intégrés dans le but de détecter les défaillances des circuits intégrés notamment, la simulation de fautes, le générateur de vecteurs de test ATPG « Automatic Test Pattern Generation » et la conception en vue du test DFT « Design For Test » ayant les performances pour assurer la fiabilité des circuits, diminuer le coût du test des circuits tout en gardant les performances de ces derniers.
- Méthodologie de tolérance aux fautes et d'amélioration de la fiabilité : une redondance modulaire triple « TMR » renforcée en « TMR double, et TMR triple » par le biais de partitionnement des éléments redondants, qui composent le circuit réalisé afin d'assurer son fonctionnement tout en minimisant sa surface et donc le coût du circuit global.

Cette méthode est ensuite appliquée à des circuits du « benchmarks ISCAS85 et ITC99 », ce qui nous démontre sa faisabilité et son efficacité.

Introduction générale

Introduction générale

Depuis l'avènement de la micro-électronique dans les années 1960, ce domaine n'a cessé de prendre de l'ampleur, aussi bien dans le domaine des sciences et de la recherche que dans la vie quotidienne avec la micro-informatique et la téléphonie mobile, notamment. En 2003, la microélectronique est passée à l'échelle nanométrique en descendant sous la barre des 100 nm de la largeur de grille du transistor (noeud de 90 nm). Depuis 2008, la résolution des circuits intégrés est passée à 45 nm.

Avec l'évolution des techniques de fabrication, la taille des composants continue de décroître. En l'an 2010, il y' a eu une amélioration sur les nœuds 32 à 22 nm en utilisant la technologie de grille de métal et de diélectrique de haute permittivité qui permet de réduire le courant de fuite des transistors et des substrats de films minces de silicium pour réduire les pertes d'électrons dans le circuit. Pour aller plus loin dans les finesses de gravure, les concepteurs prévoient vers l'an 2014, l'adoption de la lithographie EUV « Extreme Ultraviolet » pour avoir des nœuds 18-12nm [Sit06].

Cependant, il arrivera un moment où la simple diminution des dimensions géométriques des transistors pour passer d'une génération à la suivante ne sera plus possible, la technologie à base de silicium va probablement atteindre ses limites ultimes en 2020, lorsque la longueur de canal du MOSFET sera inférieure à 10 nm [Sit06]. Ce seront alors les innovations au niveau des matériaux et des architectures qui permettront d'augmenter les performances des circuits intégrés. Actuellement plusieurs dispositifs nouveaux sont étudiés pour remplacer le silicium. Parmi ces dispositifs, des substrats supraconducteurs avec des molécules à base de diamant, et des transistors basés sur les nanotubes de carbone (CNT) à base de graphène (carbone à 1 atome) [Sit06] font partie des candidats les plus prometteurs car, ayant des propriétés très particulières telle la conductivité électrique et thermique en plus d'être des matériaux résistants et durs.

En contre partie, avec l'arrivée des technologies fortement submicroniques, certains phénomènes physiques qui auparavant étaient négligeables deviennent prépondérants. Nous pouvons citer notamment la variation des paramètres des transistors (dimensions, dopages), les courants de fuite dus à la variation de la tension d'alimentation et de la température. Ces phénomènes pouvant induire des défaillances ont une répercussion directe sur la fiabilité et le rendement de fabrication. L'analyse de défaillance et le test des circuits sont alors une étape indispensable pour améliorer la conception ou le processus de fabrication en vue d'augmenter le rendement de production et la fiabilité.

Aujourd'hui, l'accroissement de complexité que permettent et induisent les technologies de fabrication pose de nouveaux défis du point de vue test. Il s'agit de gérer la complexité non seulement en terme de nombre de composants présents dans les systèmes et volume d'information à manipuler, mais aussi en terme de diversité des composants dans la mesure où les systèmes actuels intègrent de plus en plus de blocs fonctionnels hétérogènes (numérique, analogique, mémoire, RF, MEMS, etc.). Parallèlement à cette gestion de la complexité, la compréhension et la maîtrise des phénomènes physiques mis en jeu dans les nouvelles technologies sont bien entendu indispensables au développement de solutions de test efficaces pour garantir la qualité des produits actuels et à venir.

Le principe général du test d'un circuit intégré consiste à mettre en évidence un mauvais fonctionnement éventuel dû à une défaillance physique. Compte tenu du grand nombre de défaillances physiques possibles dans un circuit, il est impossible de développer des outils rapides et efficaces capables de manipuler directement ces défaillances. Des modèles de fautes ont donc été développés pour représenter les défaillances susceptibles de se manifester dans un circuit, et la plupart des techniques de test s'appuient sur ces modèles de fautes. Un autre aspect important concerne le diagnostic des défauts.

Pour réduire l'effet des défaillances, les solutions proposées par les fondeurs depuis l'an 2000, passage en plaques de silicium « wafer de diamètre 300 mm » ont été d'abord de remplacer les pistes en aluminium par des pistes en cuivre qui permettent d'accélérer les interconnexions entre transistors, de réduire les pertes et de mieux résister à l'électromigration, causant la mortalité précoce des circuits.

Parallèlement aux solutions proposées, les méthodes traditionnelles de tolérance aux fautes « redondance », et durcissement des composants sont utilisées. En effet, les premiers calculateurs étaient si peu fiables que le recours à la redondance permettait à l'ordinateur d'effectuer sans faute un calcul conséquent. L'UNIVAC 1 au tout début des années 1950 mettait en œuvre des unités arithmétiques et logiques dupliquées et utilisait le contrôle de parité. Toujours dans les années 50, d'autres exemples ont vu le jour grâce aux techniques de tolérance aux fautes, comme le calculateur SAPO de l'université de Prague.

Depuis, La tolérance aux fautes inhérente à la conception modulaire procure une nouvelle arme puissante contre les défaillances en intégrant dans des circuits complexes une stratégie de fiabilité non seulement adéquate, mais également supérieure. Elle est surtout utilisée dans le cadre d'application critiques « avionique, spatial, médical, ferroviaire…etc. ». Elle constitue même la meilleure option actuelle pour l'électronique grand public « ordinateurs portables, téléphones mobiles, agenda électronique …etc. ».

Le travail présenté dans ce mémoire porte sur l'étude et la modélisation des défauts des circuits intégrés en vue de leur analyse de fiabilité. La motivation de notre étude est de trouver des solutions structurelles insérées dans le flot de conception des circuits intégrés qui permettent d'améliorer et de garantir leur fiabilité même en présence de défauts.

Ce manuscrit se présente de la façon suivante :

Dans le premier chapitre, nous donnons un bref état de l'art sur la fiabilité des circuits intégrés, la modélisation des défauts ainsi que leurs répartitions dans les circuits. Nous décrivons également dans ce chapitre les méthodologies conceptuelles mises en œuvre en vue de fiabilité DFR « Design For Reliability » des circuits intégrés et tout particulièrement les structures de redondance modulaire triple TMR.

Dans le deuxième chapitre, nous présentons les techniques de test utilisées, notamment la simulation de fautes, la génération automatique de vecteurs de test ATPG « Automatic Test Pattern Generation » et la conception en vue de test DFT « Design For Test ». Elles visent toutes à garantir une fiabilité optimale tôt dans la phase de conception.

Dans le troisième chapitre, nous décrivons dans un premier temps, une méthode basée sur la procédure de test ATPG, pour évaluer la tolérance aux fautes des structures TMR des circuits benchmarks ISCAS85 et ITC99. Dans un deuxième temps, nous traitons ces circuits à l'aide du logiciel de simulation TetraMax. Le logiciel TetraMax donne l'évaluation des performances comportementales des circuits, sous fautes simultanées multiples injectées en phase de conception dans les descriptions de haut niveau (description Verilog). Les tests montrent que les structures TMR ne sont pas suffisamment tolérantes aux fautes pour compenser leur coût en silicium. Pour y remédier, nous proposons d'optimiser ces structures, en renforçant la redondance par voie de partitionnement qui sera réalisé avec l'outil Sh-Metis. Les résultats obtenus démontrent l'intérêt d'une analyse de fiabilité réalisée très tôt dans le processus de conception. Une telle analyse est indispensable pour décider de la stratégie de tolérance la mieux adaptée (types de mécanismes à implémenter et sites à protéger).

Une conclusion générale donnera une synthèse du travail effectué et résumera les principaux résultats obtenus ainsi que les perspectives envisagées.

MOTS-CLÉS : Fiabilité, rendement, défaut, faute, modélisation, tolérance aux fautes.

Chapitre I

Fiabilité et défaillance des circuits intégrés

La loi empirique de Gordon Moore « cofondateur d'Intel » prédit que la densité d'intégration des circuits intégrés double tous les 18 mois. Cependant cette intégration croissante qui permet de réaliser des systèmes intégrés sur une seule puce, systèmes SOC « systèmes on chip », systèmes SIP « systèmes In Package », a un impact négatif sur la fiabilité des circuits. En effet, à cause de la miniaturisation croissante des procédés de fabrication « 90 nm, 65 nm, 45 nm,… » Il est de plus en plus difficile de réaliser un circuit intégré sans aucun défaut de fabrication.

L'objectif de ce chapitre est d'introduire :

- Le concept de la fiabilité des circuits intégrés;
- Les stratégies conceptuelles en vue de fiabilité DFR (Design For Reliability) et les techniques permettant de les mettre en œuvre ;
- La notion de probabilité de tolérance de défauts des circuits intégrés, et les lois mathématiques de modélisation de répartition de défauts.

I.1- Fiabilité et rendement des circuits intégrés VLSI

Le rendement et la fiabilité sont complémentaires, et cette complémentarité est d'autant plus forte en considérant le rendement et la mortalité infantile qui sont fortement corrélés [Chr03, Cim07]. En effet, chacun d'eux est affecté par les mêmes défauts tels que les défauts d'oxyde, d'alignement des masques ou des défauts induits par la faiblesse du processus de fabrication comme le dépôt de particules.

I.1.1-Notion de fiabilité

La fiabilité «R» est un attribut de la sureté de fonctionnement [Cou08, Del10, Vil97] et correspond à la probabilité qu'un système accomplisse la fonction pour laquelle il a été conçu, dans des conditions données et pendant une durée donnée. Un indicateur de bonne fiabilité est caractérisé par un MTBF (Mean Time Between Failure, moyenne des temps de bon fonctionnement) le plus long possible. Dans le cas d'un système non réparable, on parlera plutôt de temps moyen avant une défaillance « MTTF : Mean Time To Failure » [Cou08].

L'équation liant la fiabilité d'un système à son taux de défaillance s'écrit :

$$\lambda(t)\,R(t) + \frac{(\,)}{\qquad} = 0$$

Où t : temps de mission en heure.

λ: taux de défaillance, égal au nombre de défaillances par le nombre d'heures opérationnelles.

La résolution de cette équation donne $R(t) = \exp\left(-\int \lambda(u)du\right)$, qui devient :

$R(t) = e^{-\lambda t}$ et MTBF=1/ λ dans le cas de la loi exponentielle (λ(t) = constante).

La figure 1 montre qu'avec un taux de défaillance constant, la probabilité de fonctionner sans défaillance pour un temps supérieur au MTBF est seulement de 36,78%. En effet, lorsque t = MTBF, la fiabilité du circuit chute et devient égale à :

«R(t)= ——. = e^{-1} = 36,78%».

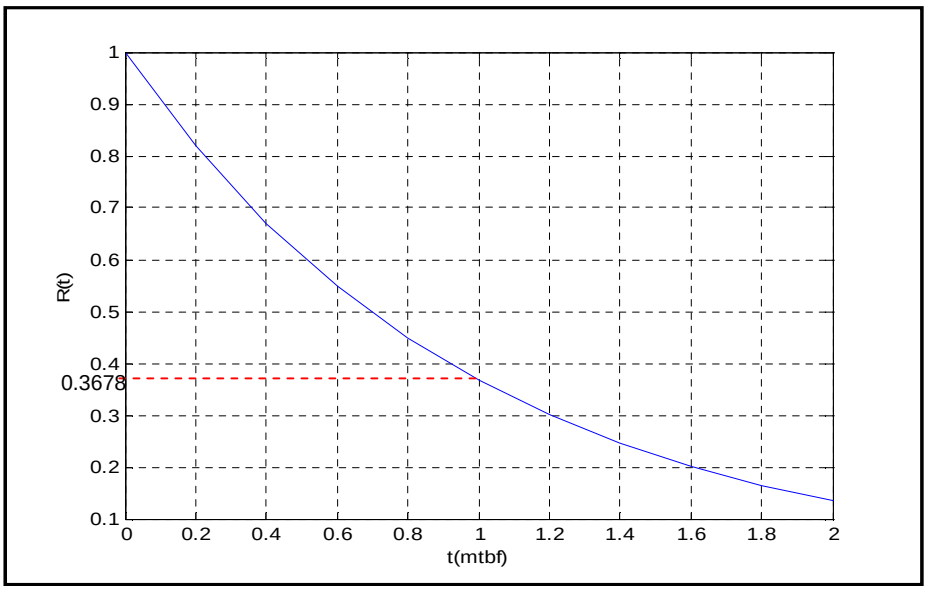


Figure 1. Relation entre fiabilité et le temps.

L'évolution du taux de défaillance d'un circuit VLSI durant son cycle de vie suit une courbe dite « en baignoire » représentée sur la figure 2. Cette courbe montre 3 phases distinctes :

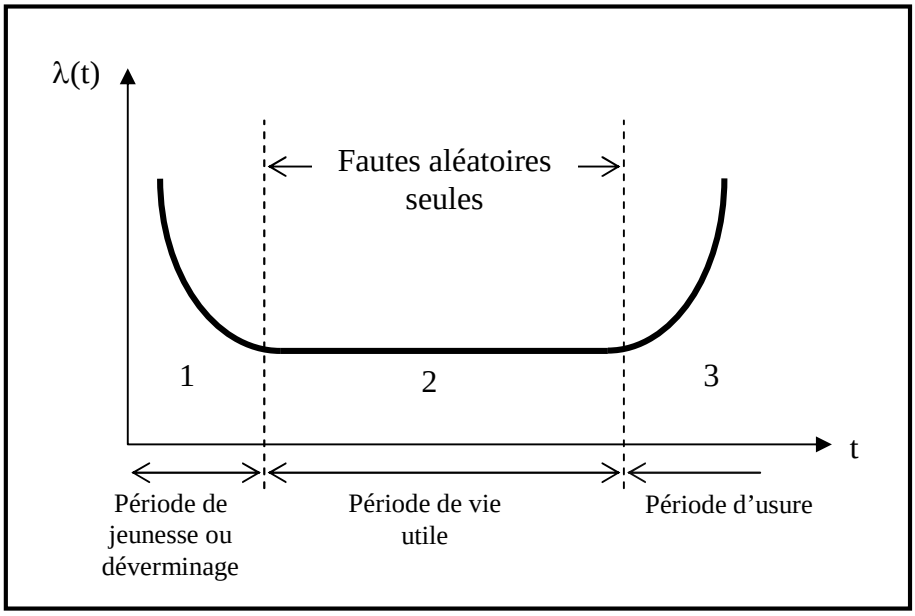


Figure 2. Évolution du taux de panne en fonction du temps [cim07]

Phase 1 : période dite de jeunesse, ou encore de mortalité infantile. Elle se caractérise par un taux de défaillance important mais décroissant. L'occurrence de défaillances durant cette période n'est pas aléatoire au cours du temps mais plutôt le résultat de défauts de conception tels que par exemple des défauts d'isolation de grille. En général, on s'affranchit de l'étude dans cette zone par des tests de déverminage ou rodage.

Phase 2 : période de vie utile caractérisée par un taux de défaillance faible et relativement constant. Les circuits sont affectés par l'apparition de défauts aléatoires.

Phase 3 : période dite de vieillesse ou d'usure, caractérisée par un taux de défaillance croissant. L'occurrence de défaillances durant cette période est due à l'usure critique des circuits. Là aussi, on peut s'affranchir de l'étude dans cette zone, car on suppose que le circuit tombe en panne avant d'atteindre cette zone, ou alors il est remplacé avant.

Ceci justifie l'utilisation de la loi exponentielle (λ constante) dans les études de fiabilité.

I.1.2- Notion de rendement

Noté couramment Y (Yield), le rendement exprime la rentabilité d'un processus de fabrication, il dépend [Bou07, Kho07]:

- Du processus de fabrication utilisé :
- La propreté de la salle blanche ;
- Des variations de température ;
- Des vibrations ;
- De la précision des machines ;
- De la pureté des produits utilisés.
- Essentiellement de la surface du circuit fabriqué.

Rendement : $Y = \frac{\qquad\qquad\qquad\qquad\qquad\qquad}{}$

Exemple

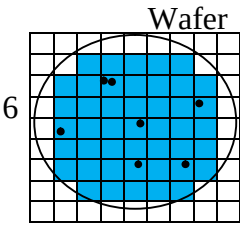
Matrice 9x9

Nombre circuits = 45

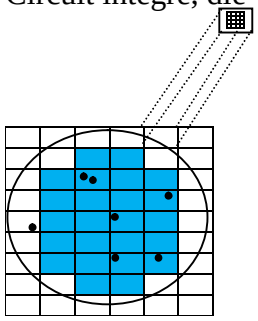
Nombre circuits défectueux « marqués » = 6

Nombre de bons circuits = 39

Y = 39/45 = 87 %



Circuit intégré, die



Matrice 6x9

Nombre circuits =24

Nombre circuits défectueux « marqués » =5

Nb de bons circuits = 19

Y = 19/24 =79 %

I.1.3- Test et fiabilité des circuits intégrés VLSI

La qualité du test influence le degré de fiabilité des circuits électroniques fabriqués. Des estimations ont été faites [Bou07] pour définir la relation entre le rendement « Y », l'efficacité du test caractérisé par la couverture de fautes « FC, Fault Coverage » qui est calculée comme étant le pourcentage de fautes détectées par rapport aux fautes possibles et le niveau de défaut DL « defect level » indiquant le pourcentage de circuits qui ont passé le test mais contiennent toujours des défauts.

Cette relation est exprimée par l'équation empirique de Williams et Brown :

$$DL = 1 - Y^{(\quad)} \times 100\%$$

I.1.4- Analyse de fiabilité, méthodes d'évaluation et prédiction de la fiabilité

L'analyse de fiabilité est utilisée pour étudier les incertitudes et la qualité de n'importe quel circuit électronique, se basant sur le retour d'expérience, l'expertise et les modèles prévisionnels. Cependant, dans un contexte de mondialisation et de compétition industrielle toujours plus difficile, la maîtrise des coûts de développement, des délais de mise sur le marché et des performances du circuit le long du cycle de vie est une des clés du succès commercial de tout projet de lancement de nouveaux systèmes, d'où un besoin crucial de méthodes et outils d'évaluation des performances comportementales au plus tôt dans la phase de conception [Cou08, Ber04].

I.2- Mécanismes de défaillance et Nature de défauts

Avant d'exposer les types de défaillances, il est important de connaitre l'origine de ces défaillances. C'est ce que l'on appelle les « mécanismes ou sources de défaillances ».

I.2.1- Les mécanismes de défaillances

Les mécanismes de défaillances des circuits (figure 3) ont été très étudiés [Cim07, Kho07, Lau07, Mac08]. On distingue ceux dus au processus de fabrication et ceux relatifs à leur utilisation.

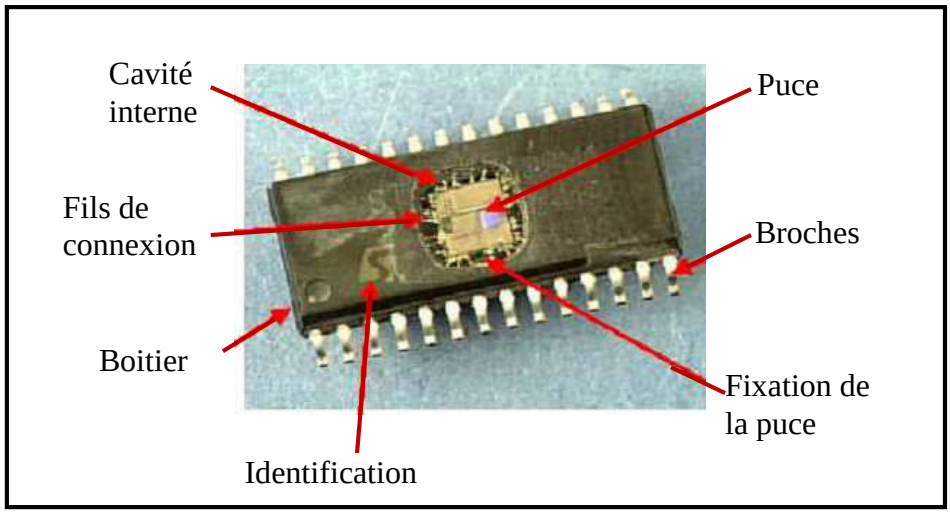


Figure 3. Localisation et mécanismes des défaillances dans un circuit intégré [Mac08]

I.2.1.1- Sources de défauts lors de l'utilisation ou phénomènes physiques naturels

Les sources de défauts survenant lors du fonctionnement du circuit sont [Mac08] :

- Mécanismes de défaillances extrinsèques au niveau boitier final : humidité dans le boitier, particules déposées dans le boitier, fils de connexion cassés ou mal assemblés, boîtier fissuré, radiations…etc.
- Mécanismes de défaillances intrinsèques au niveau de la puce, électromigration (figure 4), décharge électrostatique, excès de courant consommé au repos I_{DDQ} (figure 5),… etc.

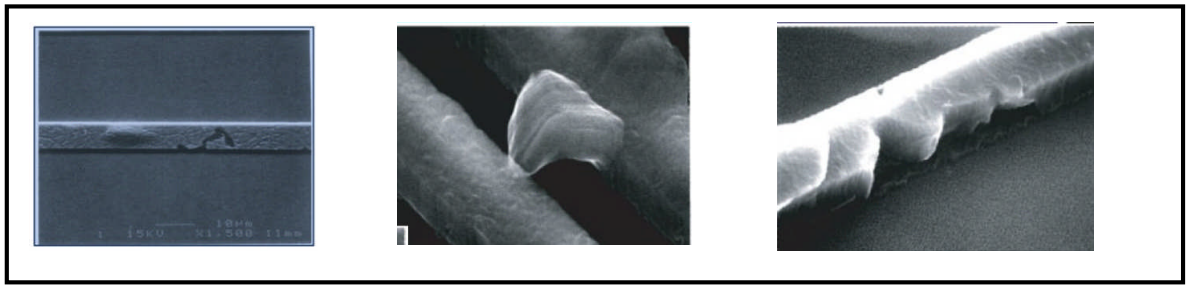
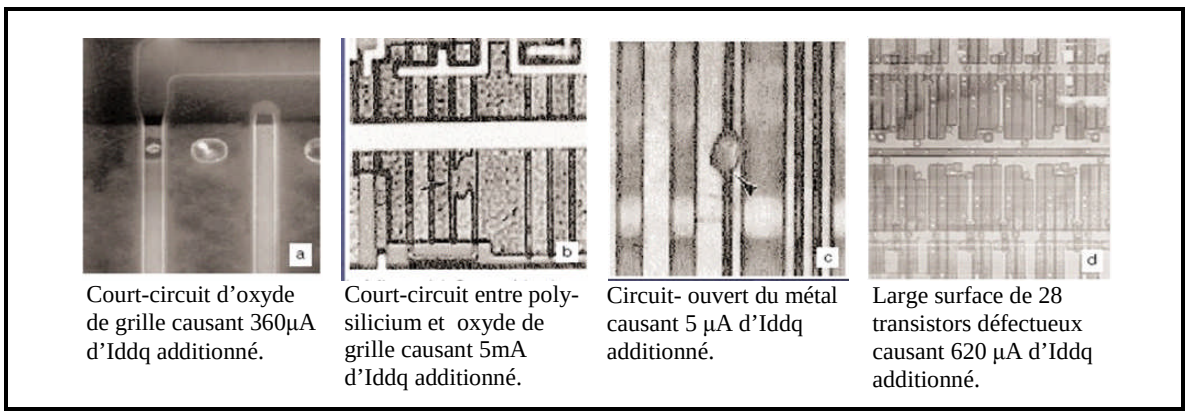


Figure 4. Défauts dus au phénomène d'électromigration [Lau07, Seg04]



Court-circuit d'oxyde de grille causant 360µA d'Iddq additionné.

Court-circuit entre poly-silicium et oxyde de grille causant 5mA d'Iddq additionné.

Circuit- ouvert du métal causant 5 µA d'Iddq additionné.

Large surface de 28 transistors défectueux causant 620 µA d'Iddq additionné.

Figure 5. Quatre exemples de défauts causant surconsommation d'I_{DDQ} [Fer98]

I.2.1.2- Sources de défauts de fabrication

Les défauts de fabrication qui peuvent avoir lieu dans un procédé CMOS sont dus à diverses sources [Kho07, Lau07, Mac08], notamment :

- Erreurs humaines
  - Une partie d'un métal peut être discontinue provoquant un circuit ouvert (fig.6a).
  - Les dopants peuvent ne pas être diffusés avec la bonne concentration (non uniformité) ou aux zones appropriées altérant ainsi les caractéristiques des dispositifs ;
  - Mauvais alignement des masques ou contamination de ces derniers avec des particules de poussières, cheveux …etc.
- Imperfection des matériaux
  - Les contacts et les vias résistifs (fig.6b);
  - Deux fils métalliques séparés peuvent être court-circuités (fig.6c) ;
  - Imperfection et claquage de la couche SiO_2 peuvent causer une augmentation du bruit de la tension ou du courant;
  - Défauts d'oxyde de grille et défauts dans les couches d'interconnexions.
- Équipement défectueux
- Fluctuation de l'environnement de fabrication
  - Température ;
  - Secousses sismiques.

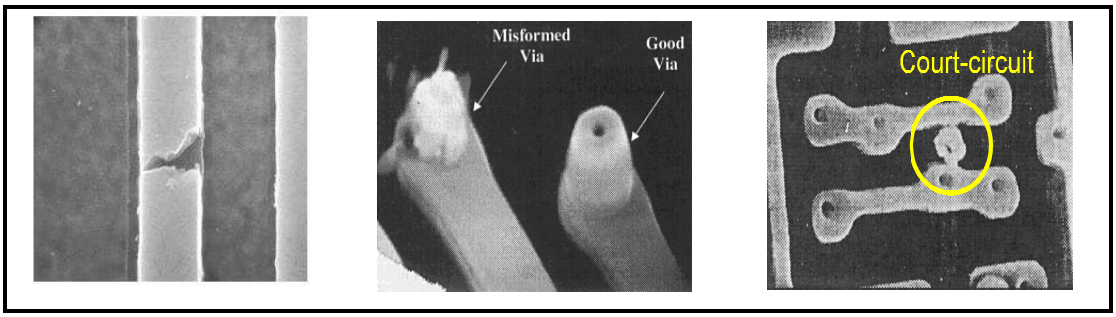


Figure 6 a)-b)-c). Défauts de fabrication [Cim07, Lau07]

I.2.2- Nature des défauts des circuits intégrés

Il existe divers types de défaillances physiques « défauts de fabrication » dans les circuits intégrés. Suivant la nature du défaut pouvant être statique ou dynamique et de ses paramètres, résistifs et capacitifs, le circuit va avoir un comportement différent [Mac08].

I.2.2.1- Définitions

- Défauts :
  - Pannes physiques qui affectent le layout d'un circuit.
- Fautes ou pannes :
  - Représentation des défauts au niveau logique selon un modèle de fautes.

I.2.2.2- Défauts physiques « fabrication »

Selon la surface atteinte par le mécanisme de défaillance, on distingue deux classes de défauts, les défauts globaux et les défauts locaux, et ce sont:

- Défauts catastrophiques qui font que le circuit ne fonctionne pas.
  - Courts- circuits
  - Circuits-ouverts
- Défauts paramétriques
  - Déviations des paramètres en dehors des spécifications
- Défauts permanents
  - Défauts toujours présents
- Défauts temporaires
  - Défauts présents dans certaines conditions « température, tension,…etc.».

I.2.2.3- Répartition des défauts de fabrication

Des lois mathématiques existent pour essayer de modéliser au mieux la répartition des défauts dans les circuits intégrés, les plus utilisées sont :

- La loi de poisson [Lan01] appelée aussi loi des événements rares, utilisée depuis longtemps en microélectronique pour connaître le nombre et la répartition des défauts.
- La loi binomiale négative [Lan98] introduite pour prendre en compte la concentration de défauts à certains endroits du circuit. C'est une distribution de probabilité discrète.

1- Loi de Poisson

Elle s'applique sur de grandes surfaces de silicium. Son équation mathématique dépend de deux paramètres k et et s'écrit [Ham95] :

$$P\{X = k\} = e^{\lambda} \times \frac{\lambda}{!}$$

Avec : $P\{X = k\}$: la probabilité que la structure ait k défauts de fabrication.

: Correspond au nombre moyen de défauts de fabrication et s'écrit :

$$= n \times p$$

n : le nombre de portes logiques (ou de transistors) ;

p : la probabilité moyenne qu'une porte logique (ou un transistor) soit défectueuse ;

En utilisant le logiciel « sinequanon », nous avons représenté sur les figures 7 et 8 deux exemples de répartition de défauts dans les circuits intégrés par le diagramme en bâton de la loi de poisson. Nous constatons sur la figure 7, que pour $\lambda=1$, la probabilité qu'il y'ait 0 ou 1 défaut est la même et égale à 36,78%

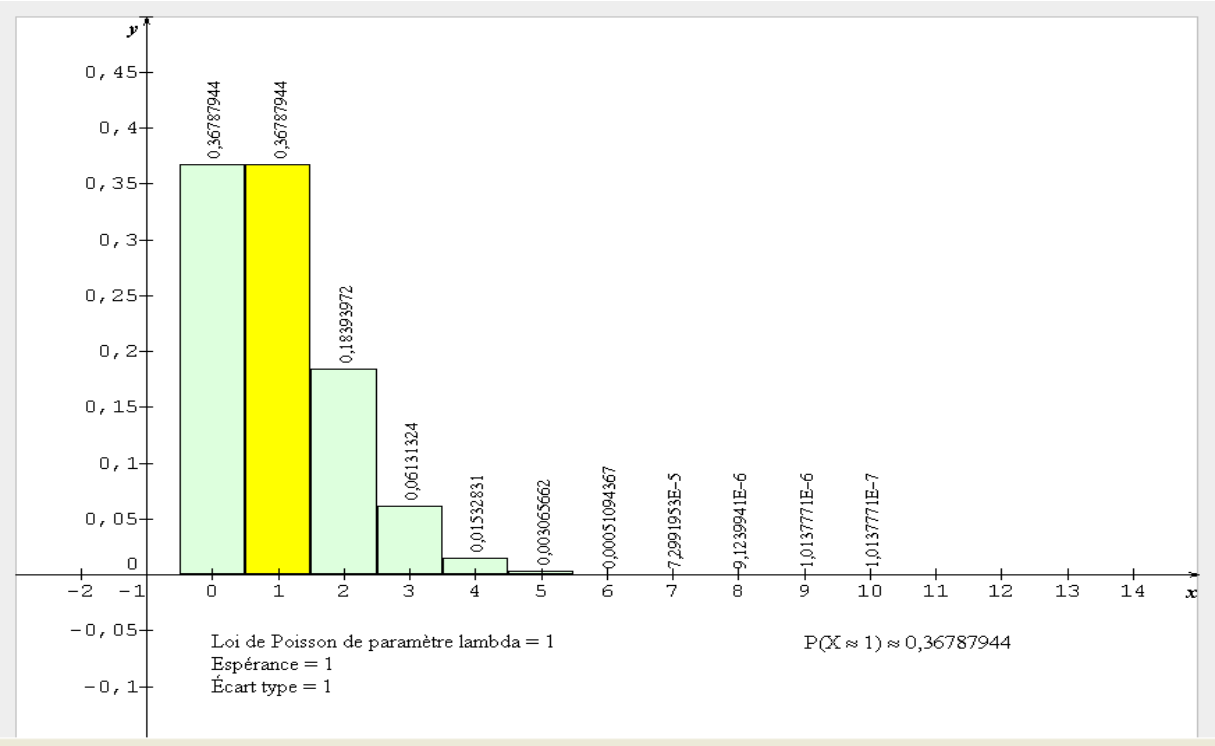


Figure 7. Loi de Poisson pour λ=1

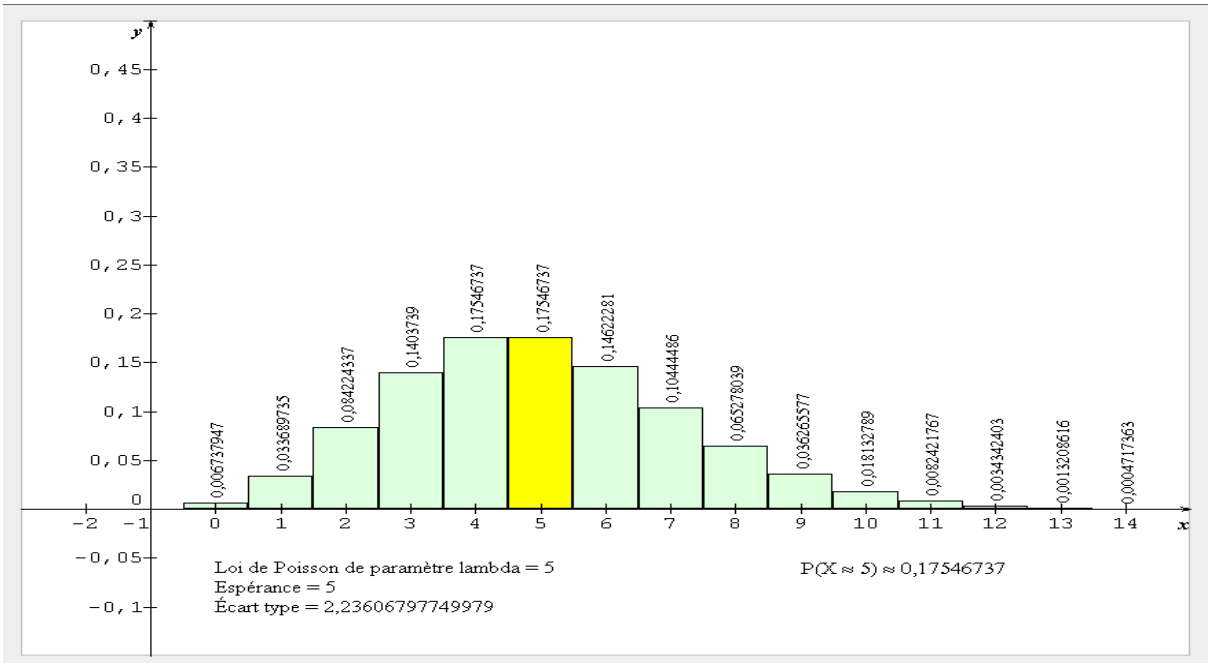


Figure 8. Loi de Poisson pour λ=5

Cependant, lorsque le paramètre λ augmente « λ≥5», on remarque que son diagramme en bâton (figure 8) est correctement approché par l'histogramme d'une loi normale d'espérance et de variance égales à λ. Cette convergence était mise à profit, avant que les moyens informatiques ne se généralisent [Chr03], pour utiliser la loi normale à la place la loi de Poisson dans certains tests.

2- Loi binomiale négative

Elle s'applique sur des blocs de silicium. L'application de cette loi en électronique est caractérisée par deux paramètres « et α » [Str99]. **Elle s'écrit [**Chr03, **Kor93] :**

$$P\{X = k\} = \frac{\Gamma(k+\alpha)}{\Gamma(\alpha).k!} \times \frac{\left(\frac{\lambda}{\alpha}\right)^{k}}{\left(1+\frac{\lambda}{\alpha}\right)^{k\ \alpha}}$$

Avec : a la même signification que dans la loi de poisson ;

α : est le paramètre de partitionnement de défauts.

$\alpha \in [0.5, 10]$ valeurs données pour la loi binomiale par [Sta84, Sta86, Str99].

$\alpha = \infty$ pour la loi de poisson

La fonction Gama $\Gamma(\alpha + 1) = \alpha \times \Gamma(\alpha) = \alpha!$, lorsque α est un nombre entier.

Le tableau 1 présente l'impact du paramètre α sur les fréquences de répartition des défauts [Sta84, Sta86, Str99]. Plus α est petit, plus la probabilité qu'il n'y ait aucun défaut de fabrication est importante, elle est égale au rendement de fabrication d'un circuit sans redondance. Plus α est grand, plus la distribution se rapproche de la loi de Poisson.

Nombre de défauts (k)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Probabilité (en %) qu'il y ait k défauts avec α = 0,5	30,15	13,71	9,34	7,08	5,63	4,61	3,84	3,24	2,76	2,37	2,05
Probabilité (en %) qu'il y ait k défauts avec α = 3	5,27	9,89	12,36	12,87	12,07	10,56	8,80	7,07	5,53	4,22	3,17
Probabilité (en %) qu'il y ait k défauts avec α = 10	1,73	5,78	10,60	14,13	15,31	14,29	11,91	9,07	6,43	4,28	2,71
Probabilité (en %) qu'il y ait k défauts avec la loi de Poisson (α = ∞)	0,85	3,87	8,98	14,14	17,04	16,73	13,94	10,14	6,57	3,85	2,06

Tableau1. Répartition du nombre de défauts par la loi binomiale négative pour λ=5

Les fréquences d'apparition de défauts dans certains circuits intégrés sont données dans les tableaux 2, 3 et 4 par [Cim07, Kho07].

Type de défectuosité	Fréquences d'apparition (%)
Court circuit	39
Circuit ouvert	14
Court circuit diffusion	14
Circuit ouvert diffusion	6
Court circuit Alu-Substrat	2
Pas de défectuosité observable	10

Tableau 2. Répartition des défectuosités dans les circuits nMos

Type de défectuosité	Fréquences d'apparition (%)
Court circuit	51
Circuit ouvert	1
Composant oublié	6
Mauvais composant	13
Composant mal placé	6
Déformation d'un conducteur	8
Mauvaises spécifications analogiques	5
Circuit logique défectueux	5
Autres	5

Tableau 3. Répartition des défectuosités dans les cartes imprimées (PCB)

Classes	Types de défectuosité
Très probable	Court circuit entre la grille et le drain Court circuit entre la grille et la source
Probable	Circuit ouvert sur le drain Circuit ouvert sur la source
Peu probable	Court circuit entre la grille et le substrat Grille flottante

Tableau 4. Classification des défauts catastrophiques pour les transistors MOS

I.3- Méthodologies conceptuelles pour améliorer la fiabilité

Plusieurs méthodes ont été proposées dans la littérature [Ber04, Cim07, Sie92] pour l'amélioration des performances comportementales des circuits intégrés.

Le développement d'un circuit fiable passe par l'utilisation d'un ensemble de méthodologies conceptuelles et structurelles qui peuvent être classées ainsi :

- Prévention de fautes : comment empêcher l'occurrence ou l'introduction de défauts lors de la fabrication du circuit;

- Tolérance aux fautes : comment assurer, par redondance, la continuité du service conduisant à la fiabilité du circuit en dépit de défauts ;
- Élimination de fautes : comment réduire la présence (nombre, sévérité) des défauts ;
- Prévision de fautes : comment estimer la présence, le taux futur et les possibles conséquences des défauts.

I.3.1- Choix d'une structure tolérante aux fautes

Les méthodes de tolérance aux fautes sont classées, selon leur type de redondance, en cinq familles [Kor07]: la redondance matérielle [Cho85], la redondance d'information [Amo88, Ple86, Mak86, Dum07, Maz95], la redondance temporelle [Lah83, Cho85], la redondance logicielle [Arl79, Bri93, Che78, Chr03, Lap90, Ran75] et la redondance mixte.

La structure la plus connue à avoir les qualités requises pour la correction de défauts permanents apparus dès le processus de fabrication et survenant même pendant le fonctionnement du circuit, est la structure NMR « N-Modular Redundancy».

Nous avons alors choisi de nous focaliser sur un type de structure NMR : la TMR «Triple Modular Redundancy».

➢ I.3.2- Tolérance aux défauts de fabrication via la structure TMR

Une structure TMR ou 3MR telle que représentée à la figure 9, est la conception la plus simple et la plus utilisée, c'est un système composé de trois circuits « modules » identiques fonctionnant en parallèle et assurant une même fonction, et d'un voteur majoritaire [Hua03, Kor07, Lyo62] qui permet de détecter et corriger leurs erreurs éventuelles. Le voteur choisit les sorties des modules sains ce qui masquent les modules défectueux.

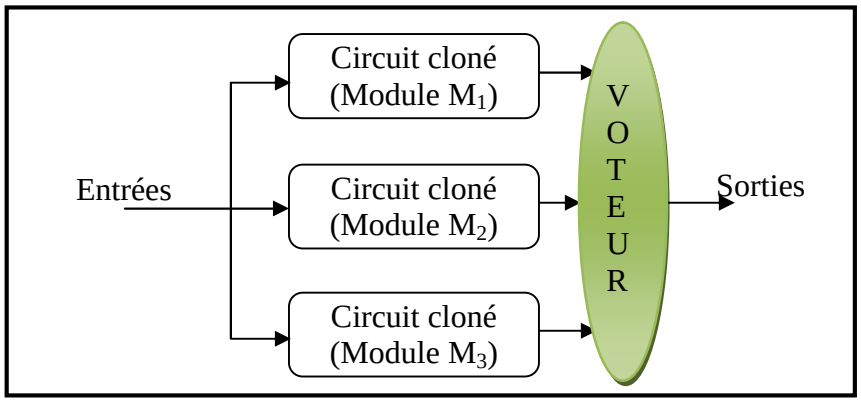


Figure 9. Structure d'un système TMR [Lyo62]

Le voteur est une structure combinatoire dont le nombre est égal au nombre de sorties des modules [Haf90]. Considérons par exemple la structure TMR de la figure 10, où les modules ont trois sorties. $S_{1,1}$, $S_{1,2}$, $S_{1,3}$ sont les sorties du premier module, $S_{2,1}$, $S_{2,2}$, $S_{2,3}$ les sorties du deuxième module et $S_{3,1}$, $S_{3,2}$, $S_{3,3}$ les sorties du troisième module. Les sorties $S_{x,1}$

sont votées ensemble ainsi que les sorties $S_{x,2}$ et $S_{x,3}$. La fonction réalisée par chacun des voteur i est la suivante : $S_i = S_{1,i}.S_{2,i} + S_{1,i}.S_{3,i} + S_{2,i}.S_{3,i}$

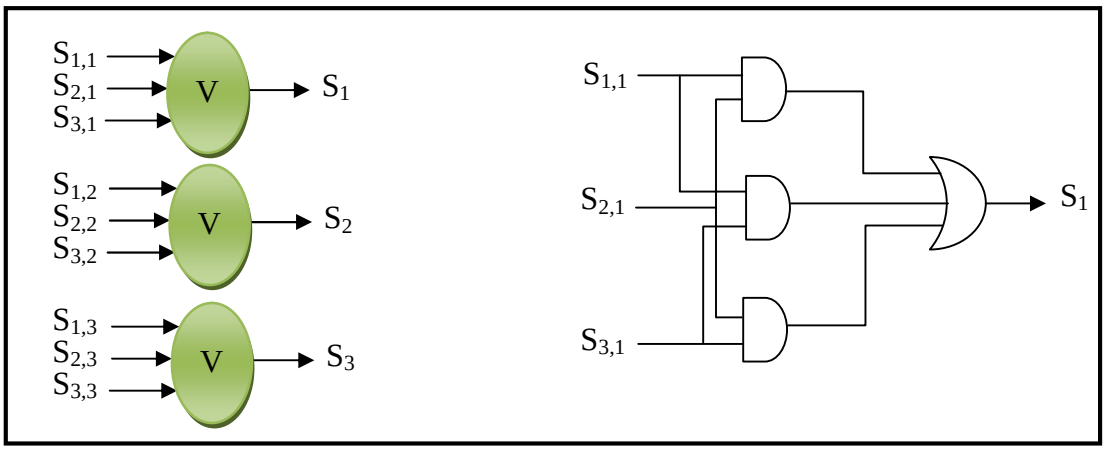


Figure 10. Voteur bit par bit [Lyo62]

Si un seul module est fautif, alors les deux autres seront dominants et la faute sera masquée. Si un module supplémentaire est fautif alors le résultat peut être erroné. Quant au voteur, la moindre faute présente est susceptible de fausser les sorties. Ceci dit, on verra par la suite, que dans certains cas, la structure TMR peut fonctionner correctement même avec deux modules défaillants [Haf90].

Pour estimer la fiabilité de la structure NMR il faut prendre en compte deux cas, qui sont :

- **1er Cas :** Le voteur est sain « les défauts n'affectent que les modules ». les défauts sont indépendants, la fiabilité de la structure NMR s'écrit [Sie92, Lyo62, Haf90, Ham05]:

$$R(t) = \sum C \times R(t)\,[1 - R(t)] \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

R(t) ou R_m, la fiabilité d'un module du système NMR

$R_{NMR}(t)$, la fiabilité du circuit NMR à l'instant t

M, le nombre majoritaire de module tel que :

M= (N+2)/2 si N est pair et ;

M= (N+1)/2 si N est impair.

La figure 11 nous montre que l'utilisation d'un circuit NMR n'est intéressante que si $R \geq 0.5$

- **2ème Cas :** Le voteur peut être défectueux et les défauts sont corrélés, la fiabilité chute

Démonstration : La fiabilité R_{TMR} d'une structure TMR s'écrit :

$$R(t) = R \times \sum C \times R(t)\,[1 - R(t)]$$
$$= R \times (3R(t)\,[1 - R(t)] + R(t)\,)$$
$$= R \times (3R(t) - 2R(t)\,)\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Avec, R_{voteur} : fiabilité du voteur.

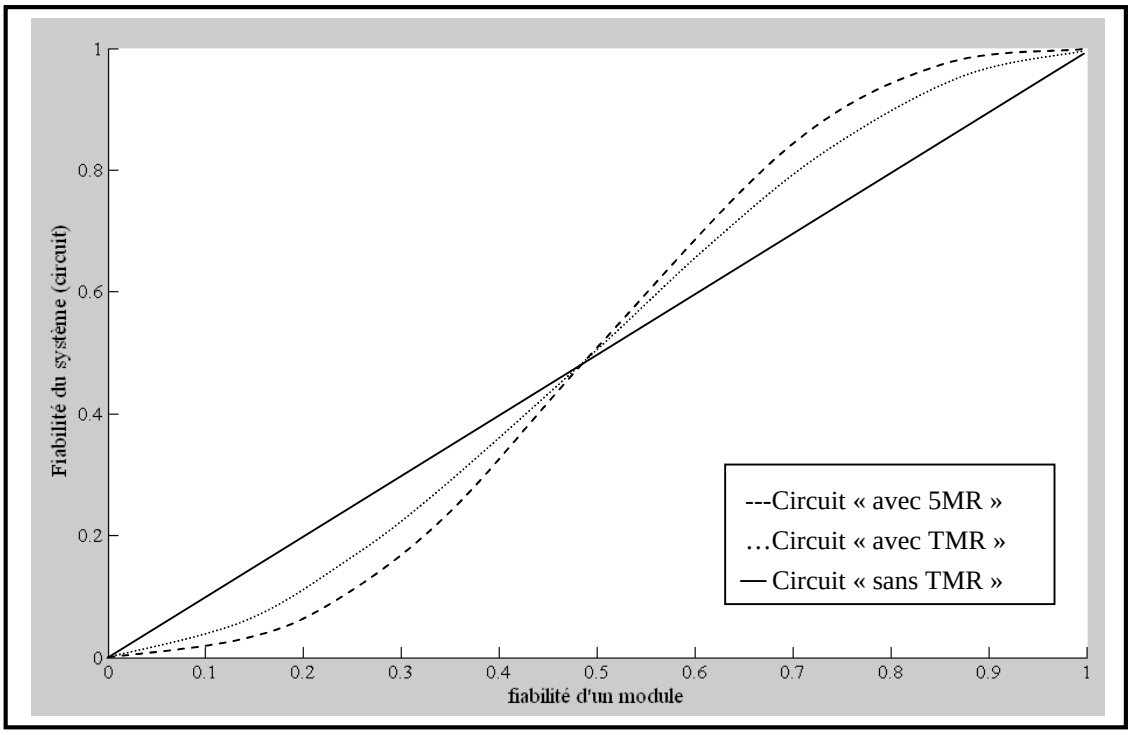


Figure 11. Schéma comparatif de la fiabilité d'un module et des circuits TMR et 5MR avec voteur robuste (R_{voteur}=1).

$R_{TMR} > R_m$ revient à $3 R_m^2 - 2 R_m^3 > R_m / R_v$ ou $3 R_m - 2 R_m^2 - 1/ R_v > 0$

Que l'on peut écrire $R_m^2 - 3/2 R_m + 1/2 R_v < 0$

$$\text{Soit } (R_m - 3/4)^2 < (9 R_v - 8)/16 R_v$$

Pour avoir des solutions le terme $(9 R_v - 8)/16 R_v$ doit être strictement supérieur à zéro.

$$\text{Soit } 9 R_v - 8 > 0 \text{ (car } 16 R_v > 0)$$

$$9R_v - 8 > 0 \Rightarrow R_v > 0, \underline{8} = 0.8888$$

La valeur minimale de R_v est égale à $R_v \simeq 0{,}89$

$$(R_m - 3/4)^2 \geq 0$$

La valeur minimale de $(R_m - 3/4)^2$ est le zéro, $(Rm - 3/4)^2 = 0$ quand $R_m = 3/4 = 0{,}75$

Ainsi, la structure TMR améliore la fiabilité, si et seulement si la fiabilité minimale du voteur est égale à 0,89 et celle du module est égale à 0.75.

I.3.3- Tolérance aux défauts de fabrication

Lorsque les défauts sont corrélés, ils se manifestent par une erreur au même moment et engendrent la diminution de la fiabilité du circuit. Pour savoir quand et comment ces circuits sont aptes à tolérer les défauts de fabrication, on doit distinguer deux cas [Lau09] :

- Les défauts sont présents dans les modules de la structure TMR « partie redondante »;
- Les défauts sont présents dans le voteur « partie non redondante ».

I.3.3.1- Les défauts affectant les modules

Pour étudier la tolérance aux défauts de fabrication affectant les modules d'une structure TMR, nous distinguons plusieurs cas donnés par [Fan06, Lau09]:

1. Cas où un ou plusieurs défauts affectent un seul module ;
2. Cas où deux défauts affectent deux modules différents ;
3. Cas où plus de deux défauts affectent plusieurs modules différents.

- **1er Cas où un ou plusieurs défauts affectent un seul module**

Lorsqu'un ou plusieurs défauts de fabrication affectent un seul et même module (fig.12), alors ce défaut est toléré par la structure TMR. Il reste deux autres modules « sains », qui seront forcément majoritaires lors du vote.

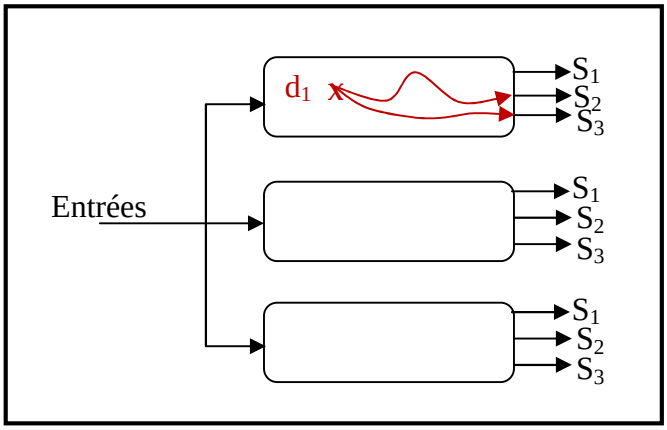


Figure 12: Présence d'un défaut de fabrication [Fan06, Haf90, Lan09]

- **2ème Cas où deux défauts affectent deux modules différents**

Si deux défauts de fabrication (d_1, d_2) sont présents dans deux modules différents (fig.13) alors ils peuvent être tolérés ou pas selon leurs effets, c'est-à-dire :

- Ils sont tolérés si aucun vecteur de test d'entrée ne peut propager deux erreurs jusqu'à deux sorties de modules communes (fig. 13-a).
- Ils sont non tolérés, si deux erreurs sont propagées jusqu'à deux sorties de modules communes (fig. 13-b).

Cependant, selon la nature du défaut si les deux défauts propagent chacun une valeur logique complémentaire alors leurs effets s'annulent [Haf90], dans ce cas là, ce sont les sorties non erronées du troisième module qui seront choisies. Ainsi, pour que deux défauts ne soient pas tolérés, il faut aussi que leurs effets ne se compensent pas.

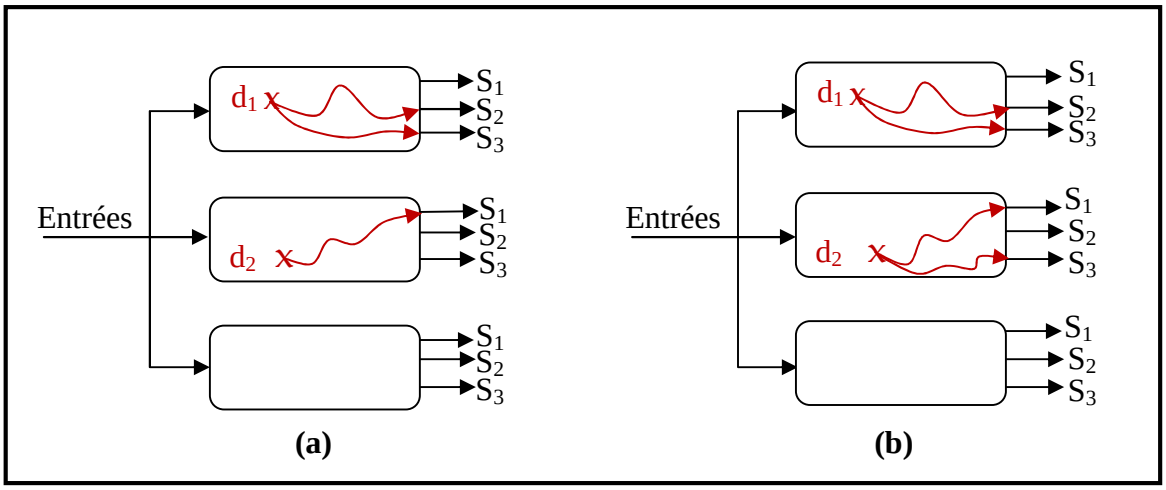


Figure 13. Deux défauts de fabrication sont (a) tolérés(b) non tolérés [Fan06, Lau09]

- **3ème Cas où plus de deux défauts affectent plusieurs modules**

Dans ce cas, nous pouvons regrouper les défauts en paires de défauts. Par exemple quatre défauts (d_1, d_2, d_3, d_4) peuvent être regroupés en six paires de défaut {(d_1, d_2), (d_1, d_3), (d_1, d_4), (d_2, d_3), (d_2, d_4) et (d_3, d_4)}.Ainsi, n défauts de fabrication peuvent être regroupés en C paires de défauts et sont tolérés par la structure TMR si toutes les paires de défauts sont masquées par le voteur.

1.3.3.2- Défauts affectant le voteur

Le voteur ne contient pas de redondance. Il est donc susceptible de propager une erreur pour chacun des défauts de fabrication l'affectant. Le voteur est le point faible de la structure TMR puisque c'est la seule partie à ne pas être tolérante aux défauts de fabrication. Pour réduire son manque de fiabilité, plusieurs voteurs redondants peuvent être utilisés. Il peut aussi être réalisé soit avec une technologie plus robuste (par exemple, des règles de dessin plus larges) [Caz04] ou bien de manière logicielle lorsque cela est possible.

I.3.4- Étude du rendement de fabrication des structures « TMR »

Dans cette partie, nous déterminons le rendement des structures TMR, afin de le comparer à celui des structures originales sans redondance et voir si elles permettent d'augmenter la fiabilité des circuits via l'amélioration du rendement de fabrication.

I.3.4.1- Calcul du rendement de fabrication d'une structure TMR

Soit A_c, la surface du circuit original sans redondance et A_v, la surface du voteur de majorité. En négligeant la taille des interconnexions, la surface A de la structure TMR est donnée par :

$$A = 3A + A$$

Le coût en surface A_O (Area Overhead) de l'implémentation d'un circuit transformé en structure « TMR » à la place d'une version originale du circuit « sans TMR » est égal :

$$A \;=\; \frac{\ }{\ } = 3 + \frac{\ }{\ }$$

Sur une même surface de silicium, nous pouvons réaliser soit :

- N structures TMR
- $A_O \times$ N structures sans redondance

En appelant Y_{TMR} le rendement de fabrication d'une structure TMR et Y le rendement de fabrication de la structure sans redondance, sur une même surface de silicium, on peut donc réaliser soit:

- $N \times Y_{TMR}$ structure TMR qui fonctionnent
- $A_O \times N \times Y_C$ structures sans redondance qui fonctionnent

La condition pour que les structures TMR augmentent le rendement est donné par [Sta84, Goe80] :

$$N \times Y_{TMR} > A_O \times N \times Y_C$$

$$\Longrightarrow Y_{TMR} > A_O \times Y_C \quad \text{(3)}$$

Comme : $Y_{TMR} \leq 1$, alors cette condition devient : $Y \leq 1/Ao$

Ainsi, les structures TMR sont nécessaires pour améliorer le rendement lorsqu'un faible rendement de fabrication est attendu. Ces rendements faibles s'inscrivent dans les perspectives de l'ITRS (International Technology Roadmap for Semiconductors) pour les prochaines années [ITR07]. Ils peuvent être dus à des technologies nouvelles ou des technologies non matures.

Calcul du rendement Y_C et Y_{TMR}

Pour calculer Y_C et Y_{TMR}, nous allons utiliser deux lois mathématiques de répartition des défauts : la loi de Poisson et la loi binomiale négative. Le rendement d'un circuit ne contenant pas de redondance est égal à la probabilité qu'il n'ait aucun défaut de fabrication.

Soit X, le nombre de défauts de fabrication, alors :

$Y_C = P\,\{X = 0\}$.

Pour calculer le rendement de fabrication d'une structure TMR (Y_{TMR}), il faut prendre en compte la tolérance aux défauts de la structure. Nous pouvons diviser la structure TMR en deux parties indépendantes :

- La partie redondante correspondant aux trois circuits clonés ;
- La partie voteur non redondante.

Soient :

Y_T : Le rendement des circuits clonés, Y_V : le rendement du voteur, alors :

$$Y_{TMR} = Y_T \times Y_V.$$

Le voteur est non redondant, la probabilité qu'il fonctionne est égale à la probabilité qu'il y ait zéro défaut dans le voteur :

$$Y_V = P\{X = 0\}.$$

Soit T, la probabilité que deux défauts (paire de défauts) soient tolérés, Y_T est donné par :

X défauts peuvent être regroupés en C_X^2 paires de défauts

$$Y_T = \underbrace{P\{X=0\}}_{\text{Probabilité qu'il y ait 0 défaut}} + \underbrace{P\{X=1\}}_{\text{Probabilité qu'il y ait 1 défaut}} + T \times \underbrace{P\{X=2\}}_{\text{Probabilité qu'il y ait 2 défauts}} + T^3 \times \underbrace{P\{X=3\}}_{\text{Probabilité qu'il y ait 3 défauts}} + T^6 \times \underbrace{P\{X=4\}}_{\text{Probabilité qu'il y ait 4 défauts}} + \ldots \quad \ldots\ldots(4)$$

S'il y a plus de deux défauts de fabrication nous pouvons les regrouper en paires de défauts, à savoir:

- La probabilité qu'une paire de défauts soit tolérée est égale à T
- La probabilité que trois paires de défauts (correspondant à la présence de trois défauts soient tolérés est égale à T^3.
- La probabilité que six paires de défauts (correspondant à la présence de quatre défauts) soient tolérés est égale à T^6.

De manière générale, la probabilité que X défauts soient tolérés par la structure TMR est égale à T . Le paramètre T représente la tolérance aux défauts de la structure TMR. Plus T est élevée, plus la structure est tolérante, donc fiable.

I.3.4.2- Étude du rendement en utilisant la loi de Poisson

La capacité à tolérer des défauts pour une structure TMR dépend des caractéristiques des défauts. En effet, suivant le nombre et l'emplacement de ces défauts, une structure peut être plus ou moins tolérante. Des lois mathématiques de répartition des défauts existent et sont censées s'approcher au mieux de la réalité des processus de fabrication. Des modèles complexes de répartition des défauts ont déjà été proposés [Gag97].

Nous utilisons la loi de Poisson qui prend en compte une probabilité de répartition uniforme des défauts, autrement dit chaque partie du circuit a la même probabilité d'être affectée par un ou plusieurs défauts de fabrication.

Dans cette partie, nous exprimons les solutions donnant les rendements respectifs d'un circuit « original » sans redondance et d'un circuit modifié en structure TMR. Les calculs

effectués permettront de savoir si la réalisation des structures TMR permet d'augmenter le rendement et donc la fiabilité.

1- Rendement de la structure sans redondance

Dans un circuit sans redondance, la présence d'un défaut le rend défectueux. Son rendement Y est donné par [Sta86, Str99]:

$$Y = P\{X = 0\} = e^{-\lambda} \times \frac{(\lambda)^0}{0!} \Leftrightarrow Y = e^{-\lambda}$$

λ_C : représente le nombre de défauts présents dans le circuit. Il est proportionnel à la taille du circuit.

2- Rendement de la structure TMR

Le rendement de la structure TMR est le produit entre le rendement des trois circuits (Y_T) et le rendement du voteur (Y_V).

***** Rendement du voteur

Pour le voteur, la présence d'un défaut le rend défectueux, son rendement s'écrit [Caz04]:

$$Y = P\{X = 0\} = e^{-\lambda} \times \frac{(\lambda)^0}{0!} = e^{-\lambda}$$

Comme la taille du voteur est A_V et que la taille d'un circuit est A_C, on peut en déduire qu'il y ait A_C/A_V fois plus de portes logiques (ou de transistors) dans un circuit que dans le voteur. Par conséquent, vu que le paramètre λ est directement proportionnel au nombre de portes logiques (ou de transistors) alors :

$$\lambda = \frac{A}{A} \times \lambda$$

Ainsi le rendement du voteur peut être exprimé en fonction de λ :

$$Y = e^{-\lambda \times}$$

Nous pouvons même exprimer Y_V en fonction d'Y_C, en utilisant l'équation :

$$Y_C = e \quad \Rightarrow \lambda = -\ln Y \ .$$

$$Y = e^{-\lambda \times}$$

***** Rendement des circuits (partie redondante)

A partir de l'équation 4, et en utilisant la loi de poisson le rendement de la partie redondante s'écrit [Fer90, Sta86, Str99] :

$$Y = e^{-\lambda} \times \left[1 + \lambda + T \times \frac{(\lambda)^2}{2} + T \times \frac{(\lambda)^3}{6} + \cdots\right]$$

$$\Leftrightarrow Y = e^{-\lambda} \times \left[1 + \lambda + \sum_{i=2}^{\infty} T \times \frac{(\lambda)^i}{i!}\right]$$

Le nombre moyen de défauts est trois fois plus important dans la partie redondante du TMR que dans le circuit sans redondance, Ainsi :

$$\lambda = 3\lambda .$$

$$Y = e \times 1 - 3\ln Y + \infty T \times \frac{(-3\ln Y)}{i!}$$

Le rendement de la structure TMR complète $Y_{TMR} = Y_T \times Y_V$ devient alors [Sta86, Str99]:

$$Y = e \times 1 - 3\ln Y + \sum^{\infty} T \times \frac{(\quad)}{!} \quad \text{.....(5)}$$

Sur la figure 14 nous avons tracé le rendement d'une structure TMR « Y_{TMR} » en fonction du rendement d'une structure sans redondance « Y_C » avec le logiciel sinequanon. Considérons dans un premier cas, que la taille du voteur de la structure TMR est négligeable devant la taille des circuits de la structure TMR ($A_V << A_C$). Le coût en surface de la structure TMR est donc égal à 3 ($A_O = 3 + —$). La droite en pointillée de la figure 14 représente la condition $Y_{TMR} > A_O \times Y_C$ pour laquelle implémenter une structure TMR pour un circuit donné améliore le rendement.

Les courbes représentant le rendement de la TMR sont paramétrées par T. Les parties de courbe au dessus de la droite en pointillée indiquent que les structures TMR permettent d'augmenter le rendement. Dans le cas où les courbes sont au dessous de la droite en pointillé, les structures TMR abaissent le rendement. Nous remarquons que lorsque T est égale ou supérieure à T_{min} = 92,58 % sur l'intervalle $Y_C \in [0{,}01, 0{,}15[$, la condition $Y_{TMR} > A_O \times Y_C$ est vérifiée, de ce fait les structures TMR améliorent de plus en plus le rendement de fabrication. Pour T=0,95 cela se vérifie quand $Y_C \in [0, 0{,}2[$, aussi quand Y_C inférieur ou égal à 0,32 pour T=0,99.

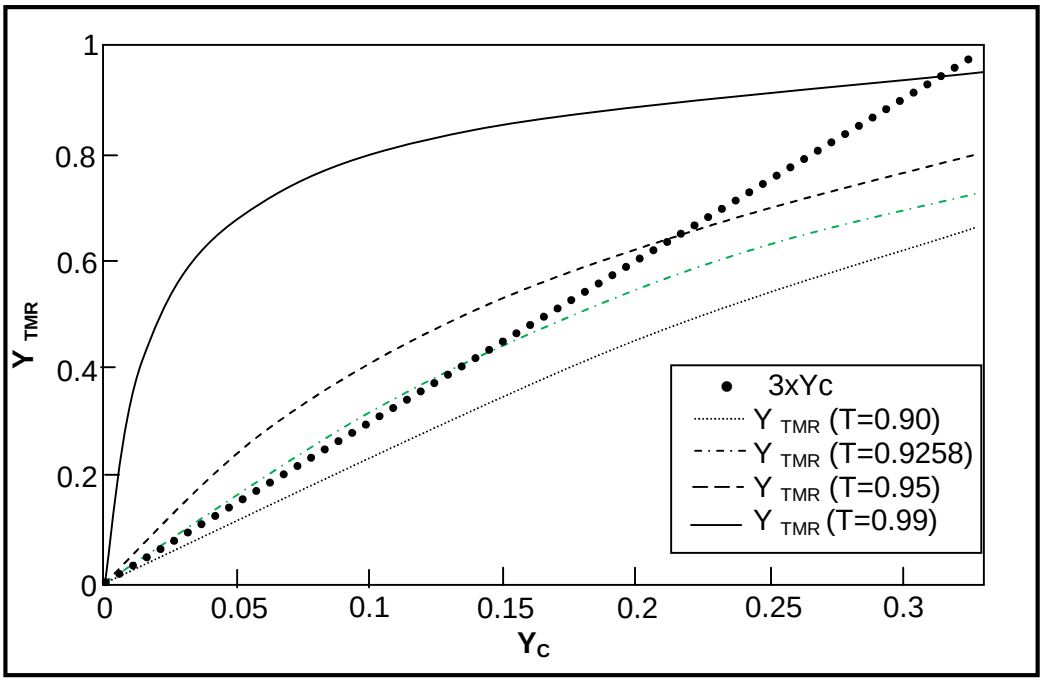


Figure 14. Rendement d'une structure TMR « Y_{TMR} » en fonction de Y_C d'une structure sans redondance.

Pour des tailles de voteurs non négligeables, la valeur de la probabilité T_{min} change. Pour savoir ce que devient cette dernière, nous avons aussi analysé l'effet du coût en surface sur le paramètre T_{min} grâce à l'équation 5. Les résultats sont montrés sur la figure 15. Par exemple, si le coût en surface de la structure TMR est de 3,27. Le coût en surface du voteur est 0,27 ce qui signifie que la taille d'un voteur est égale à 0,27 (A_V/A_C=0,27 / 1=0,27) fois la taille d'un module. La probabilité T_{min} devient égale à 96,77%.

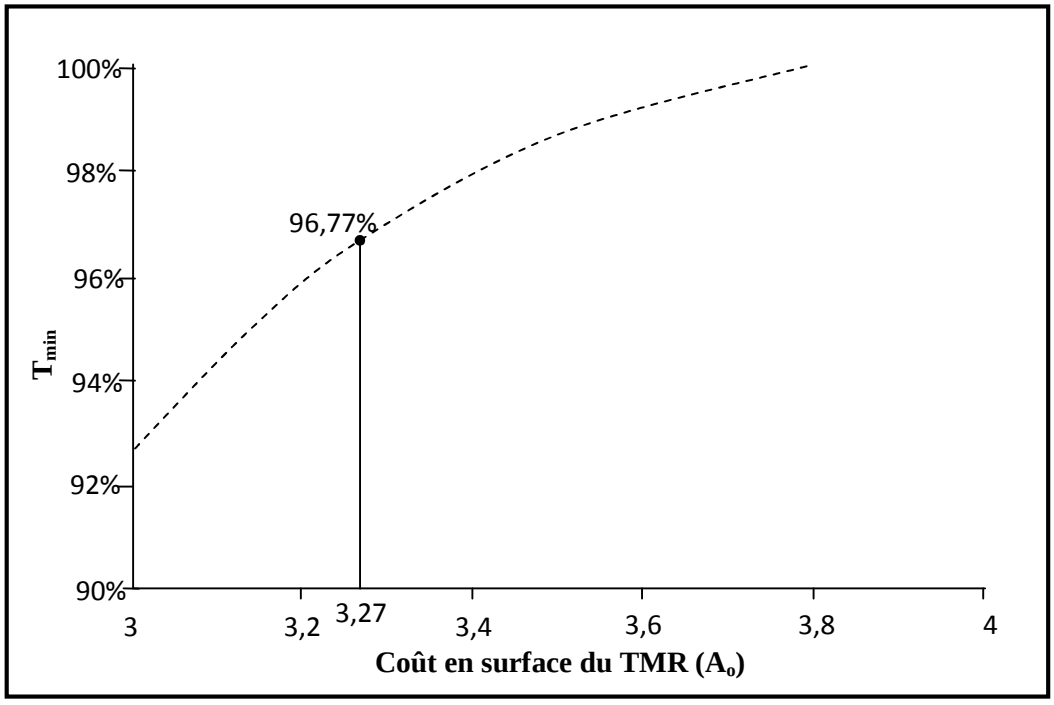


Figure15. T_{min} en fonction du coût en surface d'une structure TMR « A_O »

Nous pouvons tirer deux conclusions de la courbe illustrée sur la figure 15. Premièrement, la valeur T_{min} augmente avec l'augmentation du coût en surface. Autrement dit, plus il y aura de voteurs pour créer la structure TMR et plus il faudra que la structure TMR soit tolérante. Deuxièmement, lorsque A_O >3,8 les structures TMR ne permettent pas d'augmenter le rendement de fabrication car dans ce cas là, même si la structure est totalement tolérante (T=100%), Y_{TMR} restera toujours inférieur à $A_O \times Y_C$.

I.3.4.3- Étude du rendement en utilisant la loi binomiale négative

La loi binomiale négative apparait dans de nombreuses sources scientifiques [Arm72, IBM99, Kor93] comme l'un des modèles le plus puissant et réaliste pour représenter la distribution des défauts de fabrication dans les circuits intégrés.

1-Rendement du circuit version originale « sans redondance »

Le rendement Y_C du circuit est donné par l'équation suivante [Stap86, Str99]:

$$Y \;= P\{X=0\} = \frac{\Gamma(\alpha)}{\Gamma(\alpha).\,0!} \times \frac{\frac{\lambda}{\alpha}}{1+\frac{\lambda}{\alpha}}^{\alpha} = \;1+\frac{\lambda}{\alpha}^{\;\alpha}$$

2-Rendement de la structure TMR

Le rendement de la structure TMR est le produit entre le rendement des trois circuits (Y_T) et le rendement du voteur (Y_V).

*Le rendement du voteur s'écrit :

$$Y \;= P\{X=0\} = \frac{\Gamma(\alpha)}{\Gamma(\alpha).\;!} \times \frac{\frac{\lambda}{\alpha}}{\frac{\lambda}{\alpha}}^{\alpha}$$

$$Y \;= \;1+\text{—}\;\; Y^{\;\bar{\alpha}} - 1^{\;\alpha} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(6)$$

*Le rendement de la partie redondante s'écrit :

$$Y \;= \;1+\frac{3\alpha\;\;Y^{\;\bar{\alpha}}-1}{\alpha}^{\alpha} + (\alpha)\;\;\frac{3\alpha\;\;Y^{\;\bar{\alpha}}-1}{\alpha}\;\;1+\frac{3\alpha\;\;Y^{\;\bar{\alpha}}-1}{\alpha}^{(\;\;\alpha)}$$

$$+\sum^{\infty}\;\;T\;\;\times\frac{\Gamma(\alpha\;\;)}{\Gamma(\alpha).\,!}\times\frac{\left(\frac{\alpha\;\;\;\bar{\alpha}}{\alpha}\right)}{\left(\frac{\alpha\;\;\;\bar{\alpha}}{\alpha}\right)}^{\alpha} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(7)$$

***** Rendement de la structure TMR complète

Le rendement de la structure TMR est $Y_{TMR} = Y_T \times Y_V$. Cette équation est égale au produit des équations 6 et 7. Nous avons tracé à la figure 16, le rendement d'une structure TMR « Y_{TMR} » en fonction du rendement d'une structure sans redondance « Yc » avec «$A_V << A_C$ ». La droite en pointillée est la condition pour laquelle implémenter une structure TMR peut augmenter le rendement. Les courbes représentant le rendement du TMR sont paramétrées par T et sont toutes tracées avec α =3. Les parties de courbes au dessus de la droite indiquent que les structures TMR peuvent augmenter le rendement. A l'opposé de celles qui lui sont au dessous, qui elles le dégradent.

Comme pour la loi de Poisson, nous remarquons sur la figure 16 que plus T est grand plus les structures TMR peuvent augmenter le rendement et que pour α=3, il faut que T> 93,93% pour que la condition $Y_{TMR} > A_O \times Y_C$ soit satisfaite pour certains rendements de fabrication Y_C.

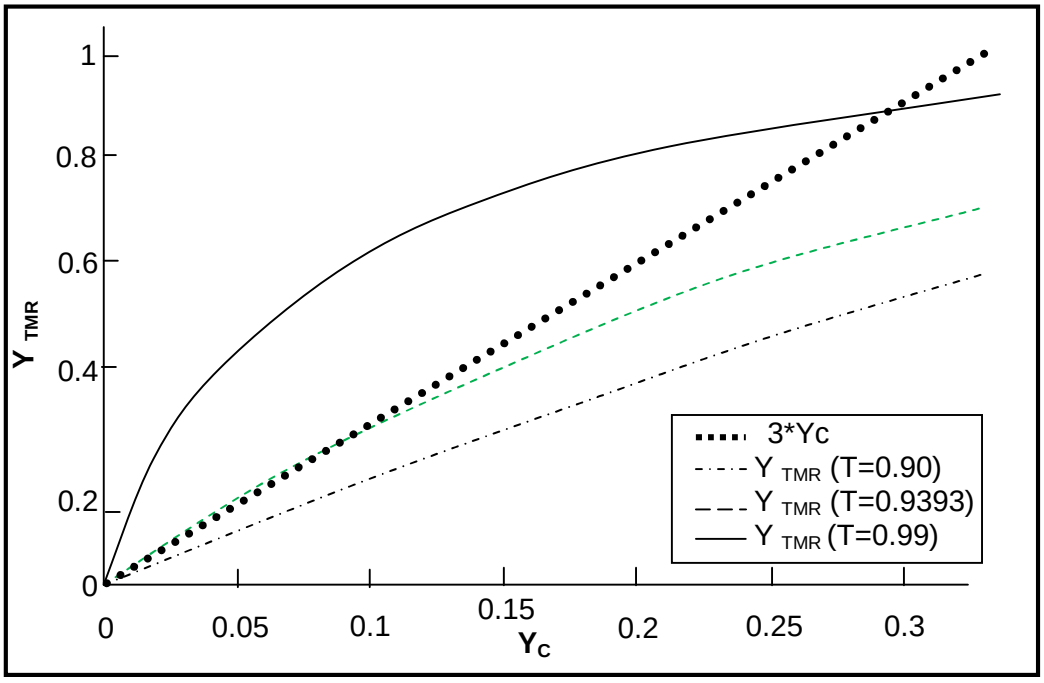


Figure 16. Rendement d'une structure TMR en fonction d'une structure sans redondance avec la loi binomiale négative (α = 3)

Le tableau 5, montre l'effet du paramètre de partitionnement α sur la probabilité T_{min} lorsque le voteur est de taille négligeable. Cette valeur est 92,58% pour la loi de Poisson et augmente lorsque α diminue. Cela signifie que plus le paramètre de partitionnement α est faible plus la tolérance de la structure TMR doit être importante. Par exemple pour α = 10, T_{min}= 93,04% et pour α =3, T_{min}= 93,93%. Pour les valeurs basses de α (α≤ 0,5), T_{min} est très proche de 100%. Par conséquent, lorsque la fabrication d'un circuit intégré entraine une répartition de défauts proche de ces valeurs là, il est impossible d'utiliser les structures TMR pour augmenter le rendement. Cela s'explique par le fait que la probabilité qu'il y ait zéro

défaut augmente lorsque α diminue (tableau 6). Le rendement des structures sans redondance est plus élevé et l'intérêt des structures TMR pour améliorer le rendement est moindre.

Valeur du paramètre α	∞	10	9	8	7	6	5	4	3	2	1	0,5
T_{min} (%)	92,58	93,04	93,09	93,14	93,21	93,30	93,41	93,57	93,93	95,09	98,10	99,84

Tableau 5. Influence du partitionnement des zones de défauts sur la probabilité T_{min}

Valeur du paramètre α	∞	10	3	0,5
Probabilité (en %) qu'il y ait 0 défaut (λ =5)	0,85	1,73	5,27	30,15

Tableau 6. Probabilité qu'il y ait 0 défaut en fonction de *α*.

Comme nous l'avons fait pour la loi de Poisson, nous avons calculé ce que devient la valeur T_{min} si la taille du voteur n'est pas nulle. Les résultats sont montrés sur la figure pour α = 3, 10 et ∞. Sur l'exemple de la figure 17, lorsque le coût en surface de la structure TMR est 3,27, alors la probabilité T_{min}, est égale à 96,93% pour α = ∞ (T_{min}=96,77 pour la loi de Poisson). Plusieurs conclusions peuvent être tirées de la figure 17. Comme pour la loi de Poisson, la valeur T_{min} augmente lorsque le coût en surface A_O augmente. Au-delà d'une certaine valeur du coût en surface A_O, les structures TMR n'ont plus les propriétés suffisantes pour améliorer le rendement. Enfin, le rendement peut être d'autant plus augmenté que le paramètre de partitionnement α est élevé.

Maintenant si la taille du voteur n'est pas nulle, là aussi, les résultats du calcul sont montrés à la figure 17. Lorsque le coût en surface de la structure TMR est 3,27, alors la probabilité T_{min} est égale à : T_{min} = 96,93% pour α =∞ (T_{min} = 96,77% pour la loi de Poisson).

Nous constatons que le rendement augmente avec l'augmentation de α. La probabilité T_{min} croit avec le coût en surface A_O, au-delà d'une certaine valeur (A_O = 3,8), les structures TMR ne peuvent pas améliorer le rendement.

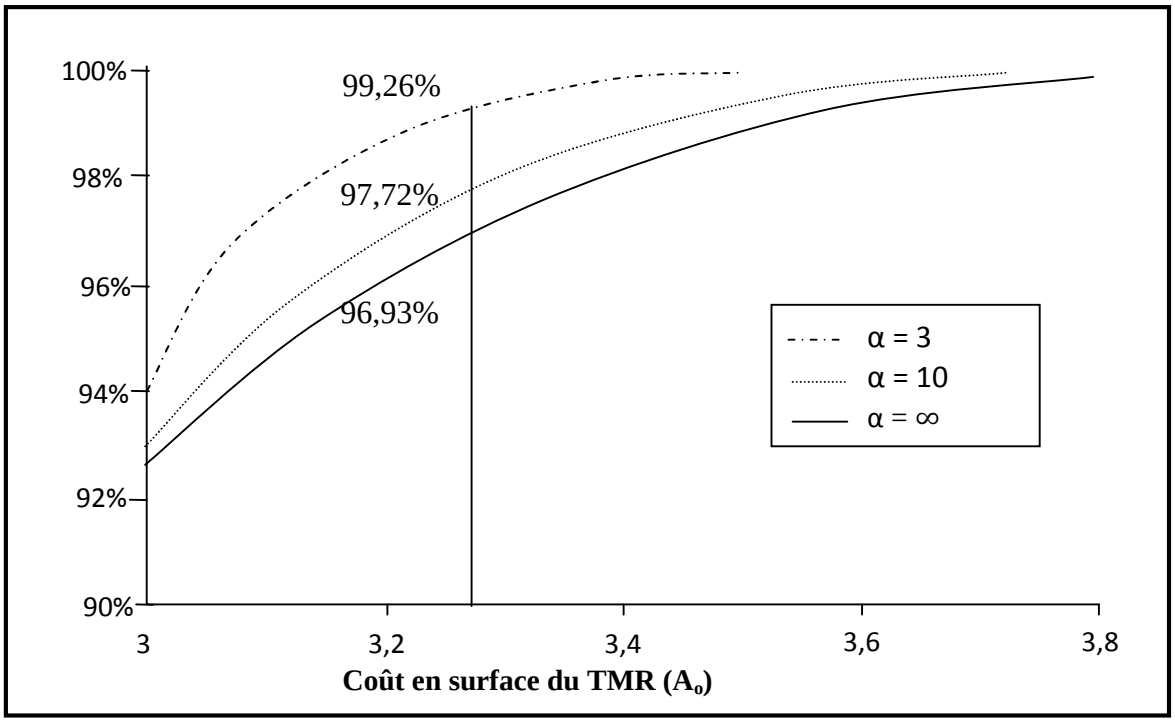


Figure 17. T_{min} en fonction du coût en surface (A_0) et du paramètre de partitionnement α

Conclusion

Dans ce chapitre, on a présenté les bases indispensables à la compréhension du sujet. Nous avons rappelé certaines notions sur la fiabilité des circuits VLSI et sa forte complémentarité double avec le rendement et le test. Puis, on a étudié les défauts de fabrication dans les circuits en analysant leur nombre ainsi que leur répartition limitant leur fiabilité et rendement. On a étudié les stratégies conceptuelles mises en œuvre pour améliorer la fiabilité et le rendement de fabrication des circuits intégrés.

Parmi ces stratégies, les techniques de tolérance, notamment, la redondance d'information utilisant les codes détecteurs d'erreurs, la redondance matérielle ou la redondance temporelle. Quelle que soit la technique utilisée, elle fait intervenir de la redondance qui a bien évidemment un coût (augmentation de la surface, diminution de la fréquence de fonctionnement, augmentation de la consommation,…etc.) mais qui parfois est la seule solution pour garantir un fonctionnement correct tout en préservant les avantages des avancées technologiques.

Nous avons constaté que les structures tolérantes aux fautes via la redondance triple « TMR », sont les plus adéquates pour tolérer les défauts affectant les circuits durant leur cycle de vie.

Cependant, les techniques de réalisation de telles structures sont très coûteuses. C'est pourquoi les motivations des concepteurs sur des designs tolérants aux défauts, ne sont pas tant de fabriquer des designs pour fiabilité ayant le rendement de fabrication le plus élevé possible, mais de déterminer le procédé de fabrication perfectionné qui permettra de concevoir des circuits VLSI avec un coût de production plus bas que possible.

Ceci dit le bon fonctionnement de ces circuits ne peut être garantie qu'à travers la notion de test. C'est donc, une étape indispensable dans la vie d'un circuit intégré que nous allons aborder en détail dans le chapitre suivant.

Chapitre II

Test des circuits intégrés

Un circuit intégré est dit performant s'il est apte à réaliser sa fonction voulue avec un niveau de qualité et fiabilité des plus élevés. Pour garantir cette qualité de bon fonctionnement, il faut tester et vérifier ces circuits dès les premières étapes de conception.

Ce chapitre a pour objectif de représenter le test des circuits VLSI et ne se voit pas exhaustif couvrant tous les aspects liés au domaine de test.

D'où il y a lieu ici d'introduire :

- Les concepts de base du test de production, assumant la validité du design ;
- Les types et techniques de test des circuits intégrés. En particulier, ceux des circuits logiques ;
- La notion de conception en vue du test dite DFT « Design For Test » afin d'améliorer la fiabilité et d'alléger le problème de test en minimisant le coût qui lui est associé.

II.1- Notions de base du test des circuits VLSI

C'est le processus qui permet de déterminer si le circuit est correct ou défectueux. Un circuit est défectueux parce que :

- Soit le circuit conçu ne répond pas aux spécifications du cahier des charges.

  Erreur de conception → test fonctionnel
- Soit le circuit fabriqué ne correspond pas au circuit conçu

  Défaut physique ou de fabrication → test de production

II.1.1- Test fonctionnel

Comme le montre la figure 18, le but d'un test fonctionnel et de vérifier toutes les fonctionnalités du circuit vu comme une boîte noire, avant de l'envoyer en fabrication. Autrement dit, le test fonctionnel est effectué pour déterminer si le circuit réalise bien toutes les spécifications décrites dans le cahier de charges et faire le diagnostic en cas d'erreurs pour modifier sa conception. Il représente une approche de base pour les petites puces analogiques et mixtes [Bou07]. Puisque la fonctionnalité des circuits numériques devient de plus en plus complexe, il apparaît qu'il est pratiquement impossible de vérifier les fonctionnalités d'une puce, en particulier les composants numériques larges tels que les microprocesseurs. Ainsi, dans les circuits VLSI le test fonctionnel tend à être remplacé par le test structurel [Mac08]. Ce test améliore le taux de couverture de fautes.

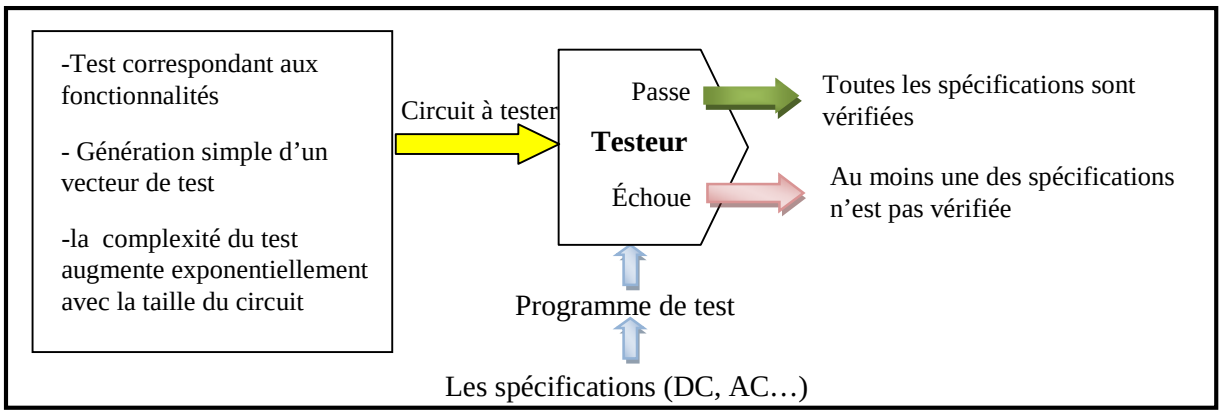


Figure 18. Principe d'un test fonctionnel [Bou07]

II.1.2- Test de production

Son but est de déterminer si le circuit fabriqué ne contient pas de défauts de fabrication et de séparer les circuits défaillants des circuits fonctionnels. Avec un tel test, les circuits défaillants ne peuvent pas être réparés, ils seront donc jetés [Bou07]. Cependant le diagnostic s'effectue dans le but d'améliorer le rendement de la chaîne de fabrication. Le test de production comprend principalement trois types de test :

- Test alternatif a été récemment proposé pour les circuits analogiques, mixtes et radiofréquences RF
- Test structurel (test logique), le circuit est vu comme une boîte blanche par ce type de test, largement adopté pour le test des circuits numériques basé sur les modèles de fautes (fig.19). Il permet d'utiliser un ensemble optimal de vecteurs de test et nécessite un minimum de temps de test, et par conséquent réduit efficacement le coût de test.
- Test paramétrique ou de caractérisation sur plusieurs lots de circuits et prototypes pour déterminer les limites de fonctionnement du circuit. Ce type de test s'effectue à chaque nouvelle conception ou nouveau processus de fabrication sous différentes conditions

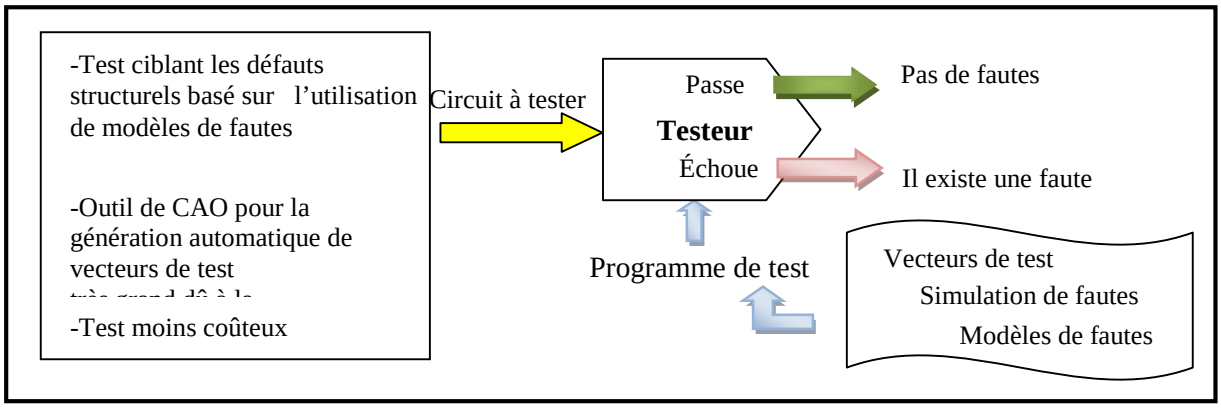


Figure 19.Principe d'un test structurel [Bou07]

II.1.3-Les différentes phases de test

Le circuit VLSI subit différentes phases de test illustrées à la figure 20, pendant son procédé de fabrication (fig.21), dont la durée appelée temps de cycle est d'environ deux mois jusqu'à la mise en boîtier. La fabrication d'un circuit intégré passe par deux étapes principales, l'étape de conception ou de design et l'étape de fabrication. Dans chacune des étapes, les phases de test s'appliquent de la manière suivante :

- Test sur wafer « ∅=300mm » : réalisé par la machine « wafer prober » pour détecter les défauts de fabrication et éviter le montage en boîtier des circuits défectueux;
- Test du circuit encapsulé (Packaged) : détecter les défauts dus au processus d'encapsulation une fois les dices découpées et les circuits montés en boîtier ;
- Test du circuit sur la carte ;
- Test de la carte dans le système.

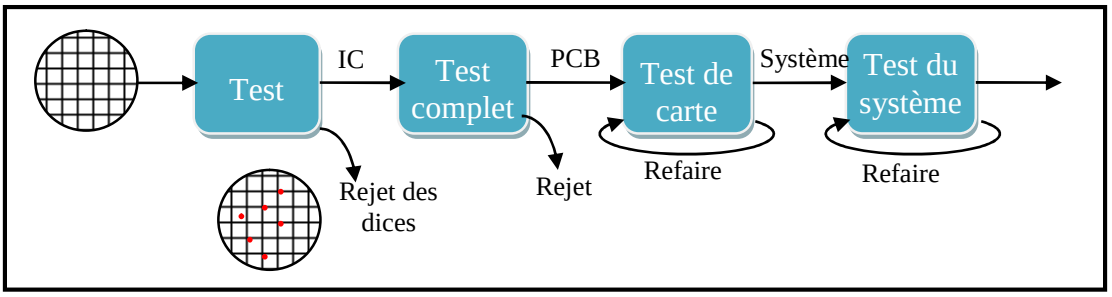


Figure 20. Phases de test

Les principales étapes des deux dernières phases sont [Bou07, Kho07, Mac08, Pac08] :

- Le test de fonctions aux conditions nominales puis aux conditions limites d'environnement ;
- Le test des performances dynamiques ;
- Le test paramétrique (essentiellement les caractéristiques électriques des broches d'entrée/sortie) qui est souvent appliqué aux circuits pour mesurer par exemple, le seuil du courant de fuite I_{DDQ} consommé par le circuit au repos. Si le circuit sous test consomme moins de ce courant de seuil, il est accepté, sinon, il est rejeté. Ce test détecte certaines défaillances que les tests classiques ne perçoivent pas. En effets, certains défauts, tels qu'un court circuit entre polysilicium et oxyde de grille ou encore un défaut de pont entre deux pistes provoquent une surconsommation du courant $_{IDDQ}$. La détection de ce dernier nécessite une minuterie de capture pour effectuer la comparaison requise. Des solutions sont utilisées aujourd'hui et qui s'étendent des capteurs de courants hors puce (off-chip sensors) aux capteurs de courant intégrés (built-in curent sensor BICS) pour des circuits à haute fiabilité [Cim07].

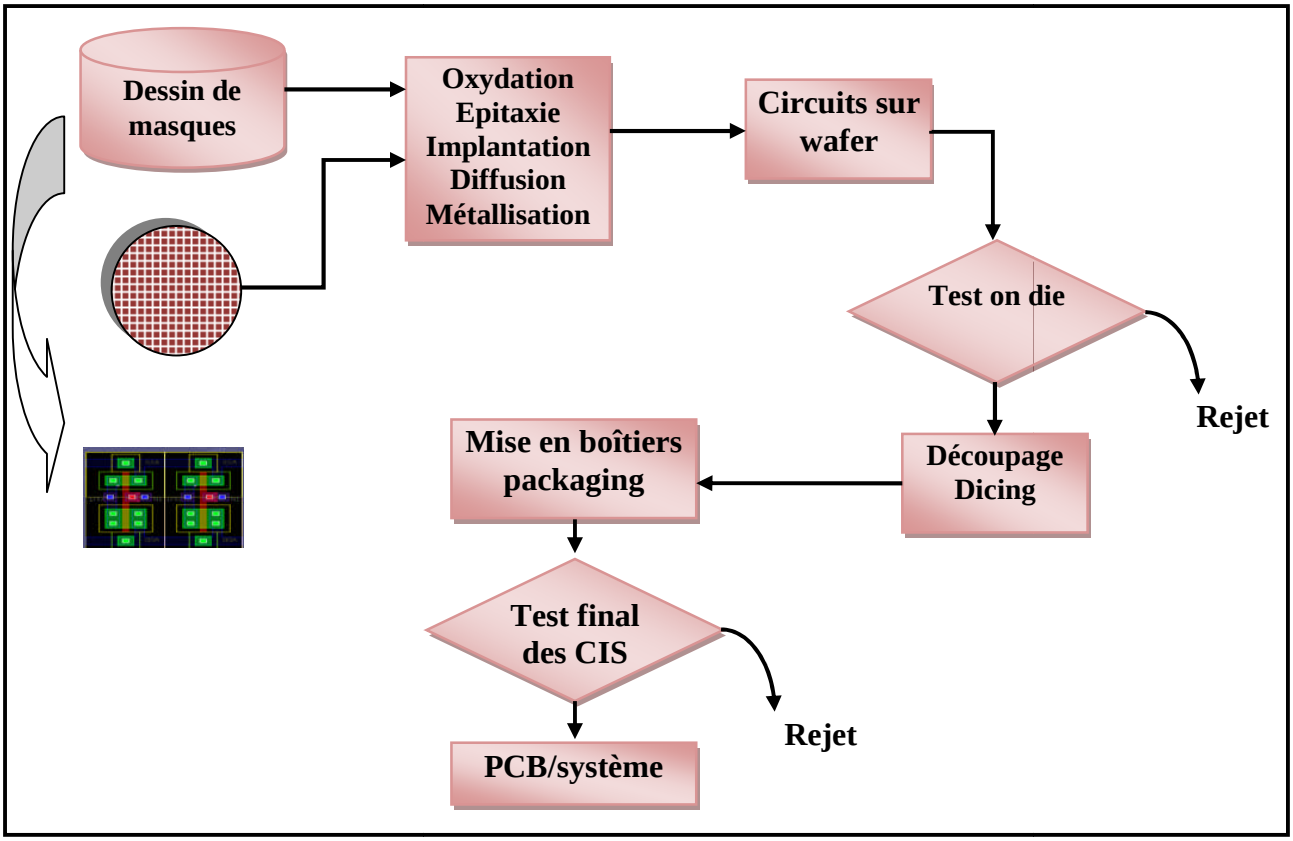


Figure 21. Processus de fabrication des circuits intégrés.

II.1.4- Motivation du test du circuit

Au niveau de leur coût, ces tests suivent la règle de dix (10). Plus un défaut est détecté tôt dans le processus de fabrication, moins le coût qu'il va induire est élevé.

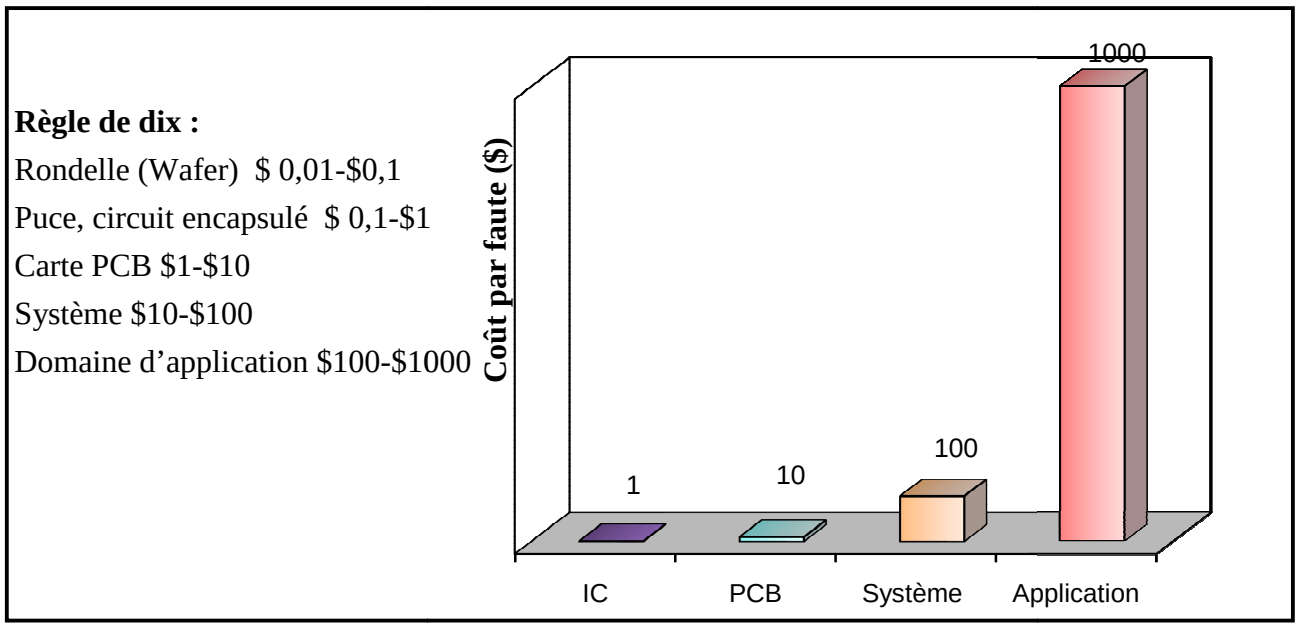


Figure 22. Motivation du test [Cim07, Kho07].

Par exemple, si un circuit intégré est détecté défectueux, le coût du boîtier, de son intégration dans un système et ensuite de la localisation de l'erreur dans ce système sont évités. La figure 22 présente, à titre indicatif, le coût d'un défaut en fonction du moment où il est repéré.

En résumé, le test [Bou07, Cim07, Kho07, Mac08] constitue plus de **70%** de l'effort du développement de circuits (systèmes) électroniques.

II.1.5- Techniques de test utilisées

Les différentes techniques de test (fig.23) utilisées [Kho07, Lak94, Mic87] sont :

1- Utilisation de la simulation de fautes avec les vecteurs de test fonctionnel pour déterminer la liste des fautes non détectées ;
2- Utilisation d'un générateur automatique de vecteurs de test (ATPG) pour détecter les fautes non détectées ;
3- S'il y a encore beaucoup de fautes non détectées, l'utilisation des techniques de conception en vue du test est requise pour améliorer la testabilité.

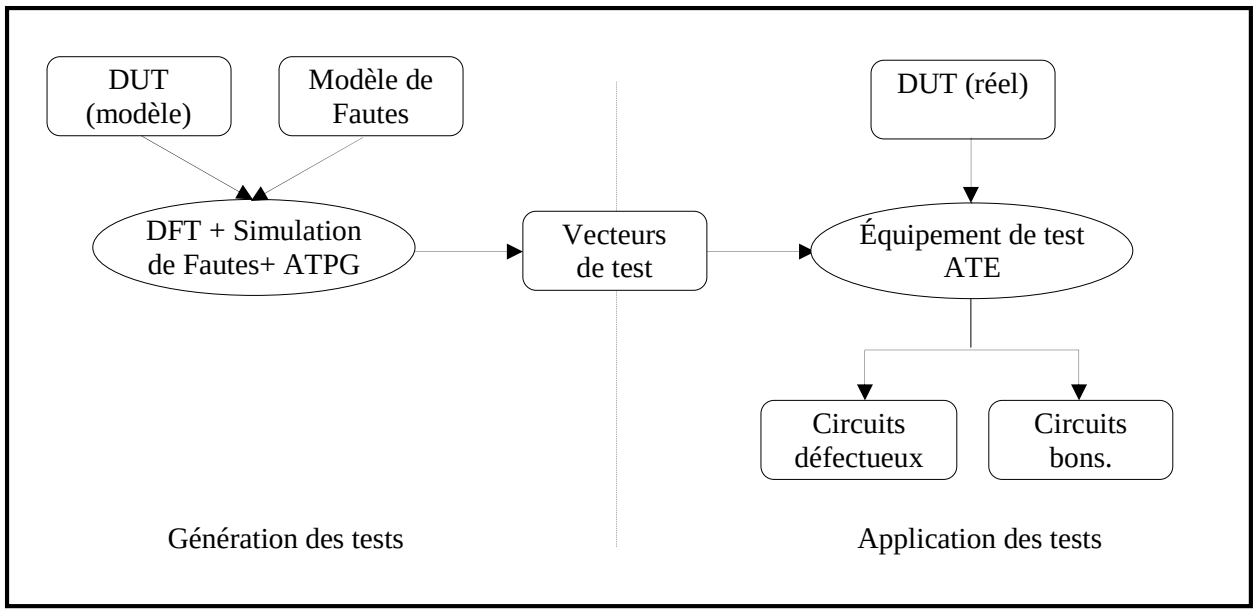


Figure 23. Test des circuits intégrés [Bou07, Kho07].

II.1.6-Test industriel

La figure 24 montre le principe du test industriel d'un circuit VLSI, il consiste à appliquer des vecteurs de test sur les entrées du circuit sous test (DUT) et analyser les sorties.

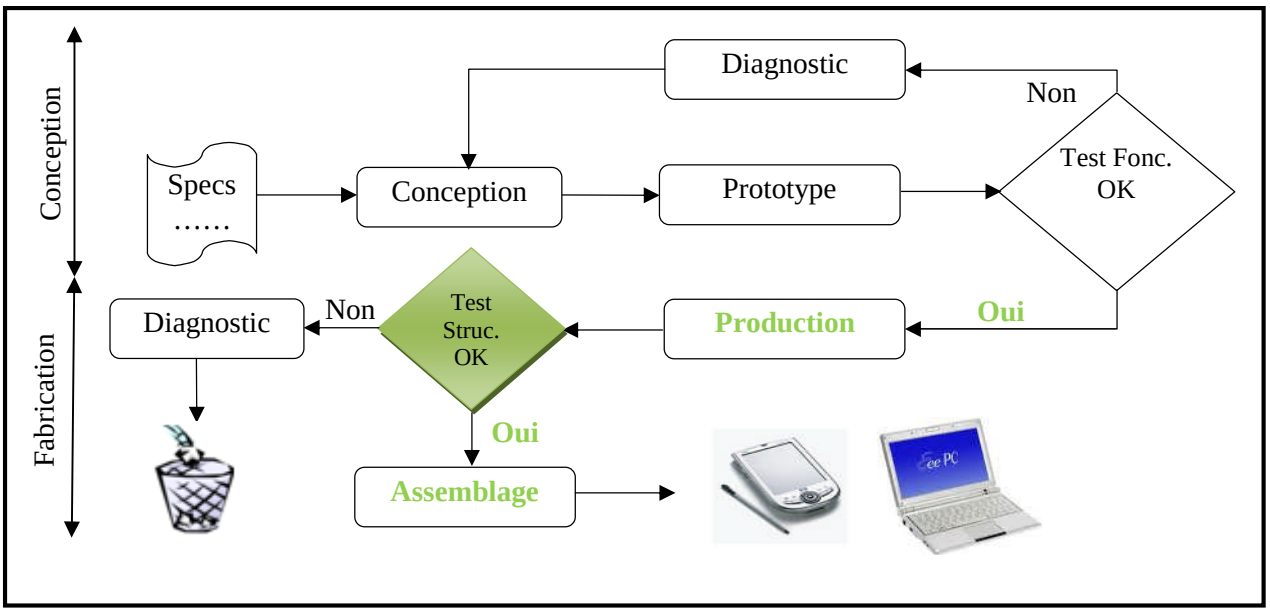


Figure 24. Test industriel d'un circuit VLSI [Bou07, Kho07]

II.1.6.1- Les testeurs de circuits intégrés

Aujourd'hui, les circuits intégrés sont de véritables systèmes intégrés sur une même puce. Ces systèmes « SOC » (System On Chip) peuvent intégrer divers éléments tels que des composants numériques, analogiques ou encore des mémoires.

Pour répondre à la problématique du test de ces SOC, les testeurs souvent désignés par l'acronyme ATE (Automatic Test Equipment) sont de plus en plus complexes et performants. En production, les ATE sont utilisés en fin de fabrication des circuits dans trois domaines suivants [Lau07, Mac08] :

- Le test des motifs de surveillance du procédé de fabrication (chaînes de vias, transistors,…). Ces motifs appelés Test Element Group (TEG) permettent de contrôler que les étapes technologiques du procédé de fabrication se sont déroulées correctement et que les dispositifs intégrés (transistors, interconnexions,…) ont les caractéristiques attendues. Il s'agit donc essentiellement d'un test électrique de caractérisation (mesure de résistances, caractéristiques courant-tension,…)
- Le test des circuits avant découpe : une fois les étapes de fabrication terminées, un premier test des circuits est réalisé sur la plaquette de silicium « wafer » avant sa découpe. Le test sur wafer est appelé test EWS (Electrical Wafer Sort) ou test sous pointes car il est réalisé à l'aide d'une carte à pointes «wafer prober ». Ce test à pour but d'éviter le montage en boîtier des circuits défectueux.

- Le test des circuits en boîtier. Une fois le wafer découpé et les circuits mis en boîtier, les mêmes tests que ceux pratiqués sur wafer sont réalisés mais avec des spécifications plus élevées garantissant la fonctionnalité des circuits avant la livraison chez le client.

II.1.6.2- Architecture d'un testeur

Le schéma de la figure 25 présente l'architecture d'un ATE avec ses principaux composants

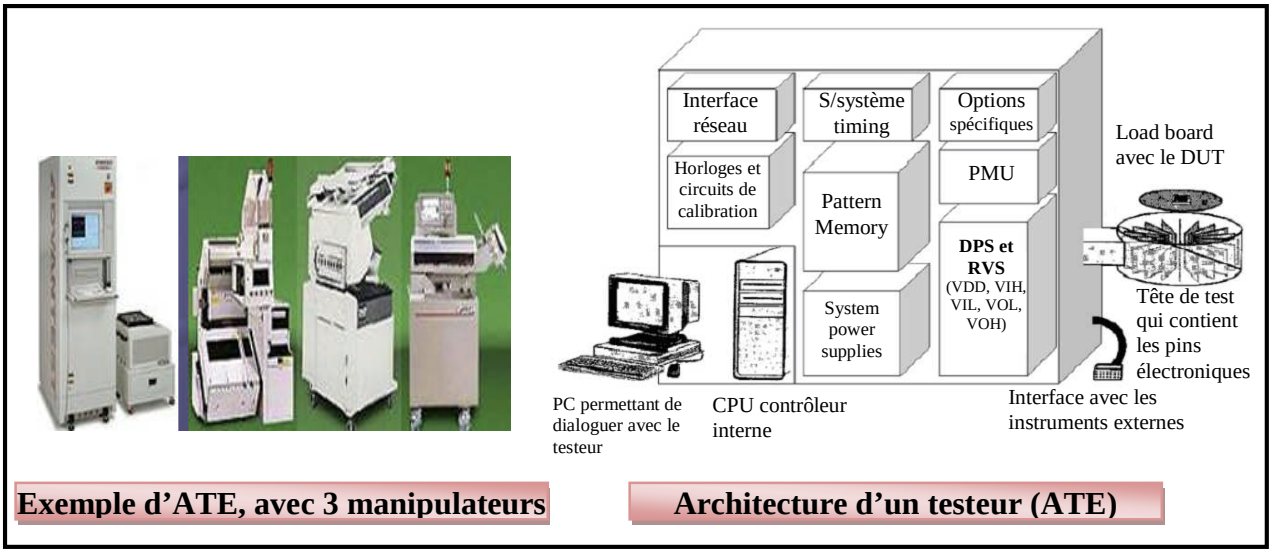


Figure 25. L'équipement de test ATE [Lau07, Kho07, Mac08].

La description des principaux blocs d'un ATE est la suivante [Per96, Mac08] :

- Le contrôleur interne permet la gestion des éléments du testeur en fournissant le moyen de transfert des instructions ;
- Le sous système DC contient les éléments permettant de fournir au circuit les tensions d'alimentation « DPS ». Il contient également les tensions de référence « RVS » nécessaires pour fournir les niveaux logiques haut et bas. Ces tensions sont généralement nommées par les abréviations V_{IH}, V_{IL}, V_{OH} et V_{OL}. On retrouve également dans ce système les unités de mesure de précision appelés « PMU » (Precision Measurement Unit) qui servent à la mesure de courants et de tensions de façon précise.
- Les mémoires de vecteurs de test qui est une mémoire à accès rapide destinée à stocker les groupes de vecteurs de test « patterns ». Ces derniers contiennent les stimuli à envoyer au DUT et la réponse théorique de ces stimuli qui sera comparée à la réponse mesurée.
- Le sous système temporel « timing » reçoit les patterns de test provenant de la mémoire vecteur de test et génère les signaux avec les données temporelles décrites dans le

pattern de test. Ce dernier est un fichier informatique contenant plusieurs vecteurs de test. le sous système temporel peut être inclus dans les pins électroniques.

- La tête du testeur qui contient les cartes auxquelles appartiennent les pins électroniques du testeur et les cartes d'interface avec le DUT. Chaque pads du circuit (analogique ou digital) utilisé lors du test est connecté à un pin électronique. Ces pins reçoivent les données issues du sous système temporel et remettent en forme ces signaux de façon à ce qu'ils présentent les niveaux de tensions décrits dans le pattern de test. Les pins électroniques permettent également de comparer la réponse mesurée avec la réponse théorique contenue dans le pattern de test et de donner la première validation de la « data sheet ».

II.1.6.3- Testeurs de production

Les testeurs de production sont utilisés à l'étape d'EWS. Ces testeurs sont associés à un « prober » pour pouvoir réaliser le test des circuits au niveau wafer. Ce prober a pour rôle d'aligner le wafer avec la carte à pointe et d'indexer le wafer dans le but de se déplacer d'une pièce à une autre ou d'un site de pièce à un autre site de pièce. Il permet également l'encrage des pièces défectueuses qui ne seront pas assemblées en boîtier. Le prober est couplé à la tête du testeur grâce à une interface mécanique spécifique. Les pins électroniques du testeur sont connectées aux « pads » (broches) du DUT à travers des cartes à pointe pour assurer la connexion électrique entre le testeur et le DUT.

II.1.6.4- Testeurs DFT

Avec l'évolution des techniques de test qui sont passées d'une approche fonctionnelle à une approche structurelle, une nouvelle génération de testeurs est apparue ces dernières années. Ces testeurs DFT (Design For Test) [Bed01] sont spécifiquement conçus pour le test des circuits intégrant des dispositifs de test basés sur des méthodes de test structurel. Ces méthodes de test ne nécessitent pas toutes les fonctionnalités coûteuses d'un ATE traditionnel tel que le test fonctionnel à très haute fréquence. Ces testeurs peuvent aussi bien être utilisés en production (exemple : le testeur « Ocelot » de la société Inovys) présenté par la figure 26 pour le développement des patterns de test et l'analyse de défaillance.

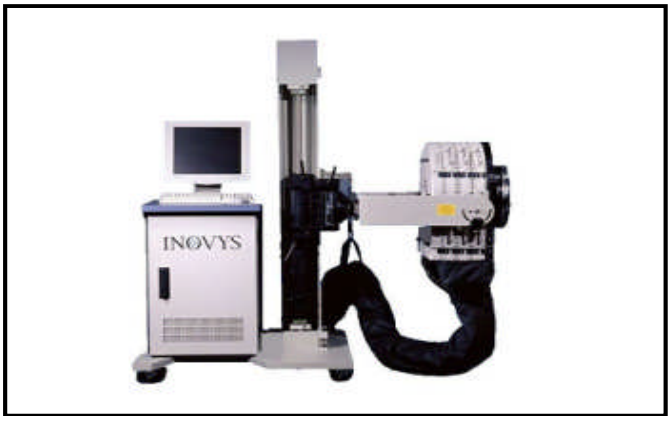


Figure 26. Testeur DFT de production Ocelot de la société Inovys.

II.2 Modèles de fautes « modèles de pannes »

Pour modéliser le comportement logique des défauts physiques et utiliser ainsi des outils de simulation de fautes, de génération de vecteurs de test, ou de diagnostic, des modèles de fautes ont été développés. Un modèle de fautes est un mécanisme d'abstraction de l'état incorrect d'un circuit, permettant de réduire le nombre de défauts à simuler, la complexité du circuit à simuler et d'effectuer le test très tôt dans la phase de conception d'un circuit (réduire le TTM (Time To Market)).

Les modèles couramment utilisés sont :

- Modèle des collages « stuck-at » le plus utilisé pour les circuits numériques.
- Modèle des collages au niveau transistor, toujours fermé/ouvert « stuck-on / open faults ».
- Modèle des court- circuits « Bridging Faults ».
- Modèle des circuits ouverts.
- Modèles pour les fautes de délai « « delay faults »

Autres modèles de fautes :

Modèle de fautes pour les mémoires et pour les circuits configurables (FPGA, PLA...etc.)

II.2.1- Le modèle des collages

Dans le modèle des collages (fig.27) on suppose que le défaut physique se traduit par un collage d'un signal du circuit soit à 0 (Vss) soit à 1 (Vdd). Ainsi, au niveau porte logique une ligne L avec un collage à 0(1) aura toujours l'état logique 0(1) quelle que soit la valeur logique imposée par la sortie de la porte liée à cette ligne L [Mac08, Lau07]. La condition de sensibilisation d'une telle panne est l'application d'une valeur logique opposée au collage sur le site de la panne [Kho07, Lak94, Mac08]. Au niveau transistor, ce modèle de collage est plus complexe (simulation, génération, nombre de fautes) que le modèle de collages au niveau porte.

Pour un signal X : X@0 = collage à 0 = Ca0, X@1= collage à 1 = Ca1.

Autres notations : $X_{S\text{-}a\text{-}0}$ et $X_{S\text{-}a\text{-}1}$ ou X/0 et X/1.

- ✓ **Avantages**
- Facile à utiliser (simulation de fautes et génération de vecteurs de test) ;
- Peut être utilisé au niveau logique et au niveau module ;
- Un nombre de fautes raisonnable (2n fautes), n est le nombre de signaux dans le circuit ;
- Des études [Kho07, Mac08] ont montré que le modèle des collages détecte 90% des défauts de fabrication.
- ✓ **Inconvénient :** ce type de test ne détecte pas tous les défauts de fabrication.

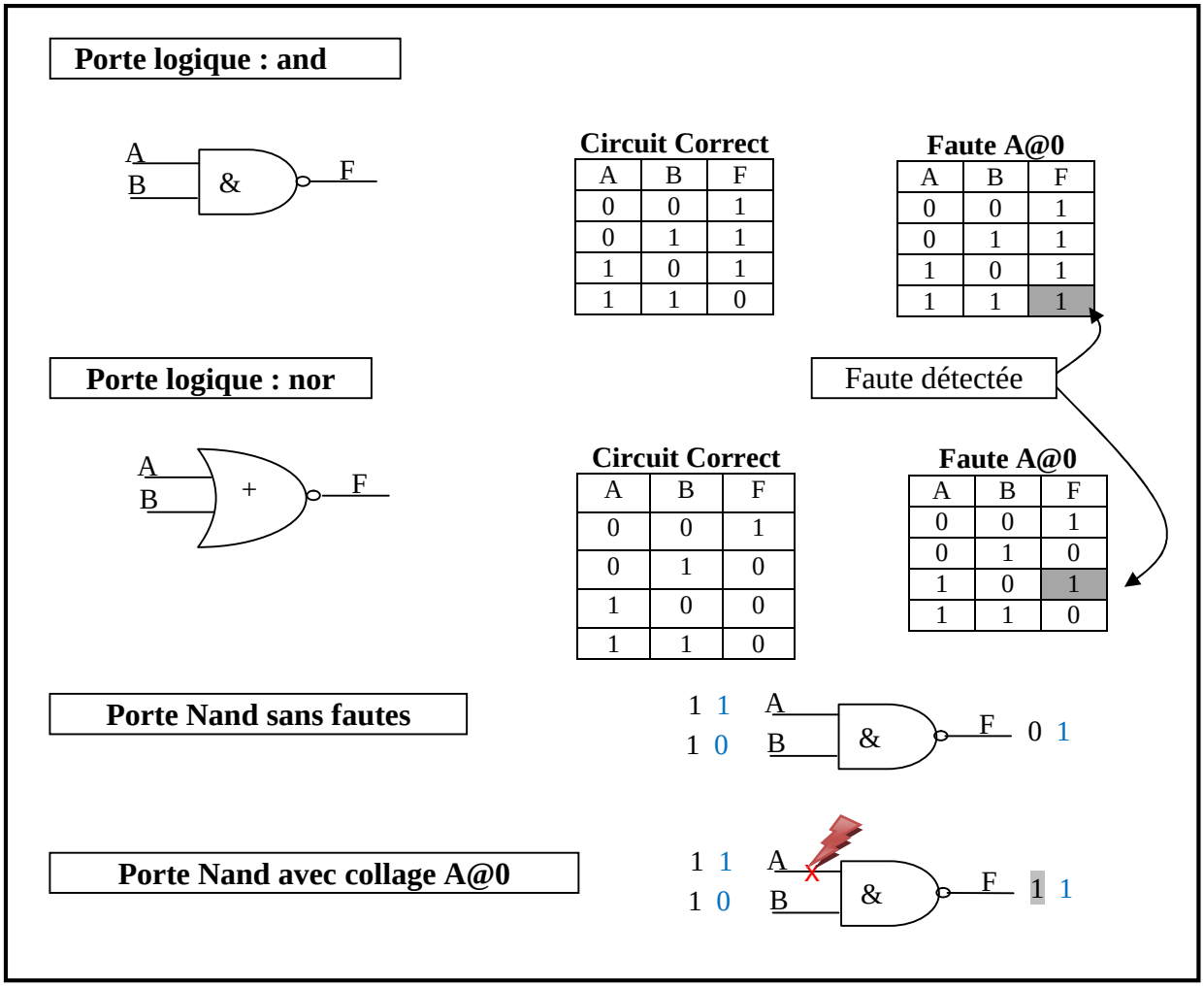


Figure 27. Modèle de collage

II.2.2- Modèle des collages multiples

Le modèle de collage multiple est une extension du modèle de collage simple, dans ce modèle, plusieurs lignes peuvent être colées en même temps à 0 ou à 1[Kho07, Lak94].

✓ **Avantages**

- Ce type de modèle permet de détecter plus de fautes que le modèle du collage simple. Aussi, le modèle de collage multiple présente un intérêt théorique primordial pour les applications à très haute sûreté.

✓ **Inconvénients**

- Très grand nombre de fautes (3^n-1, n=nombre de signaux), mais aussi algorithmes de génération très complexes.

II.2.3 - Modèle des courts- circuits

Dans le modèle des courts-circuits « Bridging Faults » on suppose que le défaut physique se traduit par un court circuit entre deux nœuds E et F (fig.28). Pour modéliser ce défaut au niveau transistor, des éléments résistifs entre ces deux nœuds sont insérés [Cim07]. Au niveau porte certains défauts physiques, peuvent être modélisés par Ca1, Ca0, pont de soudure mais ils ne peuvent pas tous être modélisés de cette façon [Mac08]. Pour ce modèle de court-circuit, on considère qu'il y a contact entre deux (ou plusieurs) équipotentielles de la représentation des portes. Dans le cas de technologies TTL ou ECL, la fonction Z(x, y) réalisée par ce court-circuit peut être soit un ET logique soit un OU logique (fig.28) des valeurs logiques portées initialement par les deux équipotentielles [Mac08].

Ce type de modèle détecte en moyenne 30% des défauts de fabrication. Mais, cela nécessite des algorithmes de génération très complexes, un nombre élevé de fautes à simuler et une description au niveau transistor pour les courts-circuits à l'intérieur des portes.

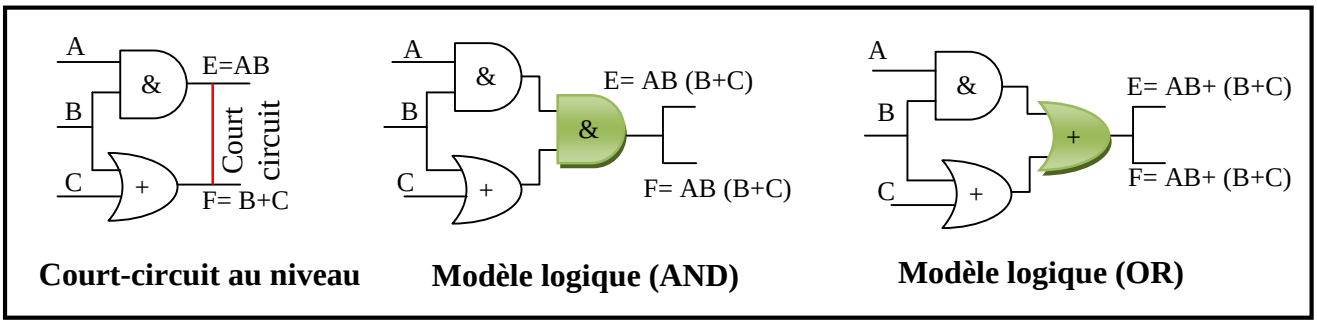


Figure 28. Modèles des court- circuits [Kho07, Lau07, Mac08, Rou08,]

II.2.4 - Modèle des circuits ouverts « Stuck Open Faults »

Dans ce modèle, on suppose que le défaut physique se traduit par un circuit ouvert sur un nœud du circuit, le nœud coupé (fig.29) n'est connecté ni à Vdd ni à Vss « effet mémoire ». Il permet de détecter des défauts physiques qui ne sont pas détectés par le modèle des collages et de tester les fautes par des séquences de test du modèle des collages. Cependant, il utilise des algorithmes de génération très complexes et un nombre élevé de vecteurs de test.

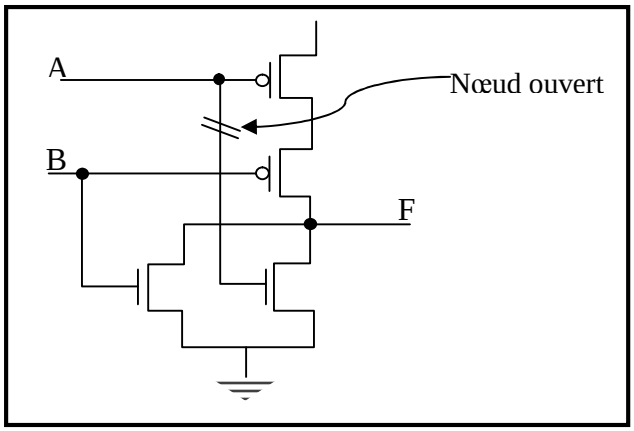


Figure 29. Modèle des circuits ouverts.

II.2.5-Modèle de fautes de délai

Dans le cas des fautes de délai la fonction logique du circuit reste correcte le défaut physique se traduit par un rallongement du temps de réponse du circuit [Kho07, Rou08].
Les deux modèles les plus utilisés pour les fautes de délai sont :

- Modèle des fautes de transition.
- Modèle des fautes de délai sur les chemins « path delay fault model ».

***Modèle des fautes de transition**

La faute de transition se traduit par un temps de transition qui est plus lent que les spécifications. Elle se produit lorsqu'une particule ionisante frappe un nœud sensible d'une cellule mémoire et entraine le basculement de la valeur logique mémorisée.

Il existe deux temps de transitions, temps de montée et temps de descente. Pour chaque signal de sortie on distingue deux fautes:

- Temps de montée plus lent « Slow-to-rise ».
- Temps de descente plus lent « Slow-to-fall ».

*** Modèle des fautes du délai de chemin**

La faute se traduit par un rallongement du temps de propagation entre les entrées et les sorties primaires du circuit. Ce modèle détecte plus de fautes que le modèle des fautes de transition. Cependant, il utilise un nombre élevé de chemins à tester et des algorithmes de génération très complexes.

II.2.6- Solutions pour réduire les fautes

Il existe plusieurs solutions qui peuvent être appliquées pour réduire et supprimer le nombre de fautes dans différents cas, et ce sont :

- Fautes équivalentes

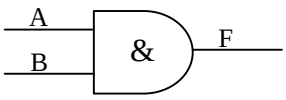
La liste de fautes, dressée à partir de la modélisation, peut être réduite « fault collapsing » grâce aux notions d'équivalence et de dominance de fautes [Kho07, Lak94, Lau07].

- Relation d'implication :

  f1⟹ f2 si tout vecteur de test de f1 est aussi un vecteur de test pour f2, on dit aussi que f2 domine f1.

- Relation d'équivalence :

  f1⟺ f2 si tous les vecteurs de test qui détectent une faute détectent l'autre faute. Il est évident que les fautes sur une même équipotentielle sont équivalentes « cases grisées dans l'exemple du tableau 7 ».



			C.C	A@0	A@1	B@0	B@1	F@0	F@1
A	B		F	F	F	F	F	F	F
0	0		0	0	0	0	0	0	1
0	1		0	0	1	0	0	0	1
1	0		0	0	0	0	1	0	1
1	1		1	0	1	0	1	0	1

Tableau 7. Fautes équivalentes

Liste des vecteurs qui détectent A@0 = {11}
Liste des vecteurs qui détectent A@1 = {01}
Liste des vecteurs qui détectent B@0 = {11}
Liste des vecteurs qui détectent B@1={10}
Liste des vecteurs qui détectent F@0 = {11}
Liste des vecteurs qui détectent F@1= {00, 01, 10}

⟺

A@0<=> B@0<=>F@0
A@1=> F@1
B@1=> F@1
F@1 domine A@1
F@1domine B@1

En résumé, six (06) fautes globales réduites en quatre (04) fautes par équivalence et en trois (03) fautes par dominance.

- Fautes indétectables ou non testables

C'est le type de fautes pour lesquelles on peut démontrer qu'il n'existe pas de vecteur qui les détectent (problème de contrôlabilité ou d'observabilité), elles sont dues à la logique

redondante dans le circuit. Par contre, les fautes non détectées sont les fautes qu'on n'a pas réussi à détecter, cependant une faute non détectée n'est pas forcement non détectable.

On peut supprimer les fautes indétectables en supprimant les redondances (non souhaitées) dans le circuit. Voici quelques règles, qui sont données au tableau 8 pour supprimer des redondances pour les portes (AND, NAND, OR, et NOR).

Fautes indétectables	Règles de simplification
Collage @0 entrée de AND (NAND)	Supprimer la porte, remplacer par 0 (1)
Collage @1 entrée de AND (NAND)	Supprimer l'entrée
Collage @0 entrée de OR (NOR)	Supprimer l'entrée
Collage @1 entrée de OR (NOR)	Supprimer la porte, remplacer par 1 (0)

Tableau 8. Élimination de redondance [Kho07, Lan07].

II.3 - Simulation de fautes

La simulation de fautes est utilisée afin de déterminer la qualité d'une séquence de test T. Cette qualité s'exprime sous la forme d'un taux de couverture de fautes. Ce dernier n'est directement significatif que pour l'ensemble des fautes prises en compte par le simulateur. La simulation joue un rôle important dans le processus de génération de vecteurs de test. Le premier type d'utilisation est donné à la figure 30. Le simulateur de fautes sert à évaluer la qualité de la séquence de test T en cours de production.

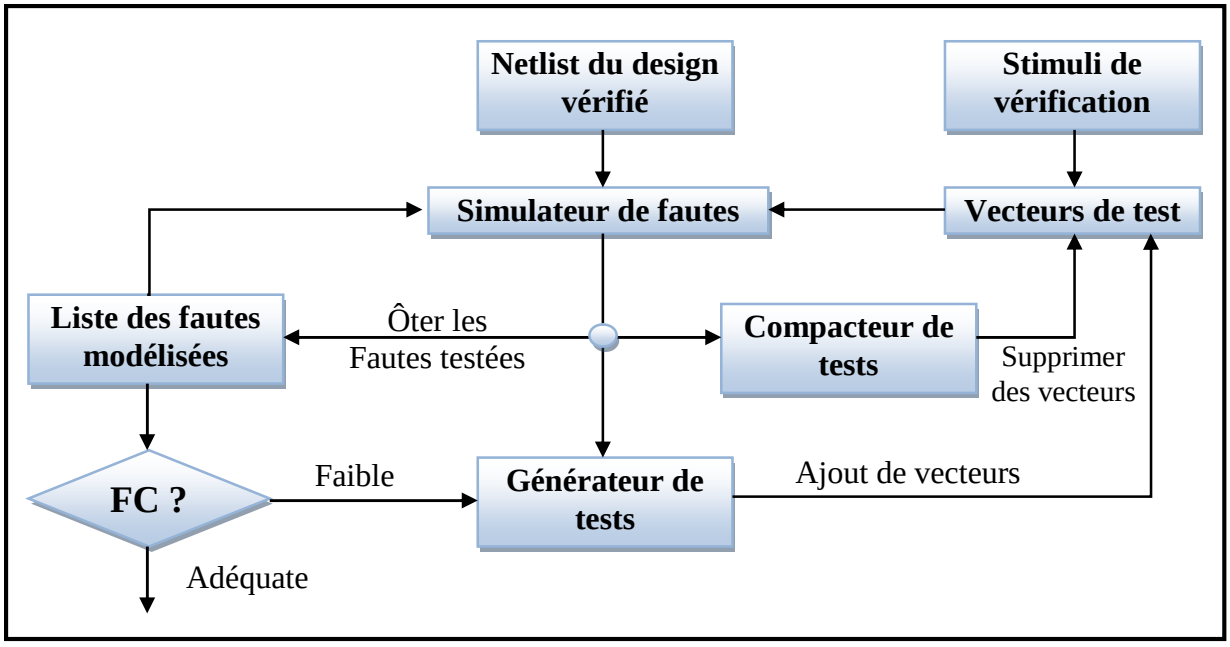


Figure 30. Principe de la simulation de fautes.

Si le résultat n'est pas satisfaisant on modifie la séquence T, soit en rajoutant des vecteurs pour améliorer le taux de couverture soit en supprimant des vecteurs qui ne contribuent plus à améliorer la séquence. Ces modifications peuvent s'effectuer de manière automatique par programme ou bien de manière interactive sous le contrôle de l'ingénieur de test.

La simulation est aussi utilisée du fait que de nombreux générateur automatique de vecteurs de test travaille avec comme objectif de détecter une faute donnée. Une fois trouvé le vecteur de test correspondant à cette faute particulière, on effectue une simulation de fautes afin de supprimer de la liste des fautes à traiter celles détectées par ce nouveau vecteur.

Une dernière utilisation courante de la simulation de fautes consiste à analyser le comportement du circuit en présence de défauts. Ce type d'utilisation est particulièrement courant pour des applications à haute sécurité et évaluer d'autres alternatives de conception et de protection ainsi réduire le temps de la mise au marché [lau07].

II.3.1- Algorithmes de simulation de fautes

Leur but est de déterminer toutes les fautes détectables par chaque vecteur de test.

Le principe est de simuler le circuit correct et les circuits fautifs et comparer les résultats sur les sorties primaires du circuit, autrement dit la simulation du circuit correct, la sauvegarde des résultats obtenus, injection de la faute, puis simulation du circuit fautif et enfin comparaison des résultats des deux simulations.

Il existe plusieurs algorithmes de simulation de fautes à savoir :

II.3.1.1- Simulation de fautes série

C'est la méthode de simulation de fautes la plus simple [Vil97], elle consiste à simuler le circuit correct C, transformer le circuit correct C en C_f en injectant la faute f et simuler le circuit fautif C_f avec le simulateur logique. Cependant, elle nécessite des temps de traitement considérables « CPU » (simulation d'une seule faute à la fois).

II.3.1.2- Simulation de fautes parallèle

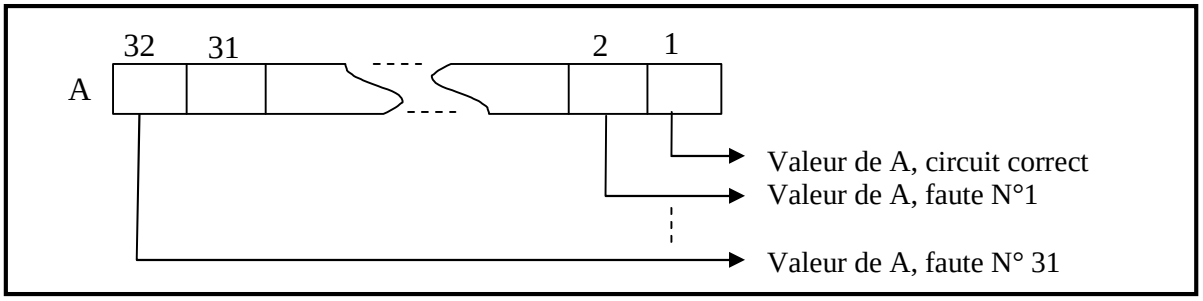
Dans la simulation de fautes parallèle, on simule le circuit correct et N fautes simultanément.

- Les valeurs des signaux du circuit correct et des N circuits fautifs sont stockées en mémoire de l'ordinateur.
- Si W est la taille de la mémoire, alors N=W-1
- Si N_f est le nombre de fautes, il faut N_f/ (W-1) simulations
- L'insertion des fautes est effectuée avec des masques.

Pour chaque signal S du circuit on définit deux masques :

- Un masque M (S) qui indique la position du bit modélisant la faute
  M $(S)_i$ = 1 si le $i^{ème}$ bit représente une faute du signal S, 0 Sinon
- Un masque F(S) qui indique le type de collage
  $F(S)_i$ = 1si la faute est S@1, 0 si la faute est S@0.

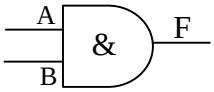
Le nouveau signal : $S' = S. \overline{(\ )} + M(S).F(S)$

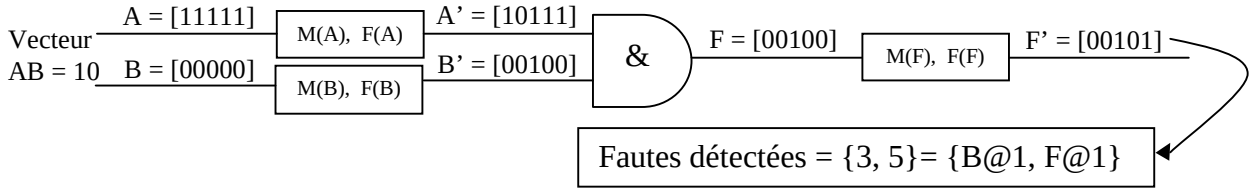


Soit l'exemple d'injection de fautes [Kho07, Lak94] appliqué pour la porte AND suivant :

Position	Fautes
1(MSB)	C.Correct
2	A@0
3	B@1
4	F@0
5(LSB)	F@1

Signal S	M (S)	F (S)
A	01000	00000
B	00100	00100
F	00011	00001



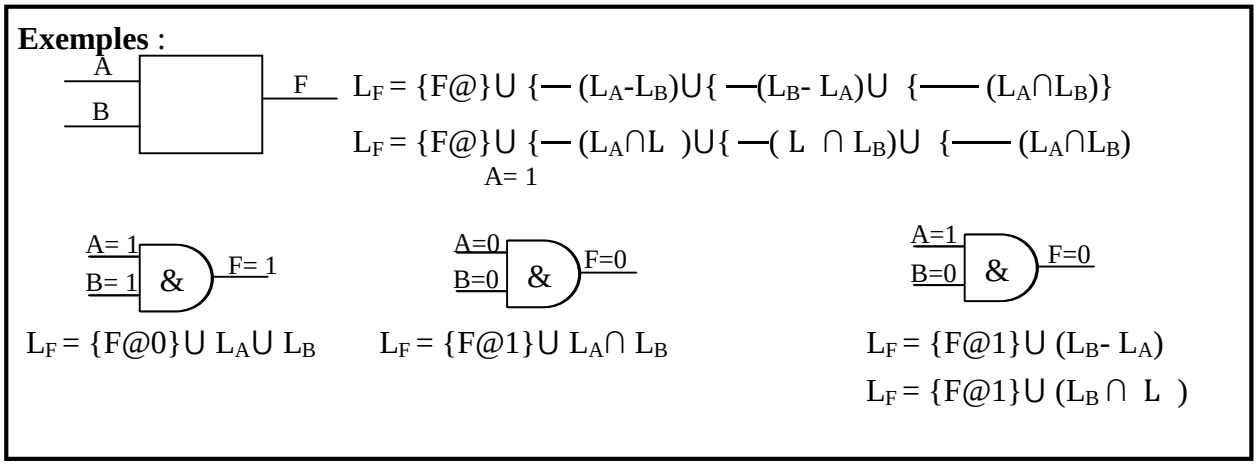


II.3.1.3- Simulation de fautes déductive « DFS »

Elle a été développée par Armstrong en 1972 [Mew78]. Elle consiste à simuler uniquement le comportement du circuit correct et en déduire toutes les fautes détectables en même temps et donc on a besoin d'effectuer uniquement une seule passe de simulation pour chaque vecteur de test appliqué. On associe à chaque signal S, une valeur logique, une liste de fautes notée L_S. Puis on effectue une seule simulation pour chaque vecteur de test (propagation des valeurs logiques « simulation logique » et propagation des listes de fautes). Si le signal S est

une entrée primaire, alors on a : L_S={S@0} si S = 1, et L_S={S@1} si S = 0. Les listes de fautes des sorties primaires contiennent les fautes détectées par le vecteur de test.

Chang et al. [Cha74] ont prouvé expérimentalement que la simulation déductive est plus rapide que celle parallèle seulement pour les circuits dont le nombre de portes est supérieur à 500.



II.3.1.4- Simulation de fautes concurrente

C'est la méthode de simulation la plus générale, elle est rapide car elle est basée sur la simulation simultanée de plusieurs portes fautives. Cependant elle est complexe et demande beaucoup d'espace mémoire que la simulation déductive en début de simulation [lau05].

II.4- La génération de vecteur de test

Elle dépend du modèle de fautes considéré. Une fois le modèle choisi (fig. 31), il faut déterminer les vecteurs d'entrée qui rendent une faute contrôlable et observable.

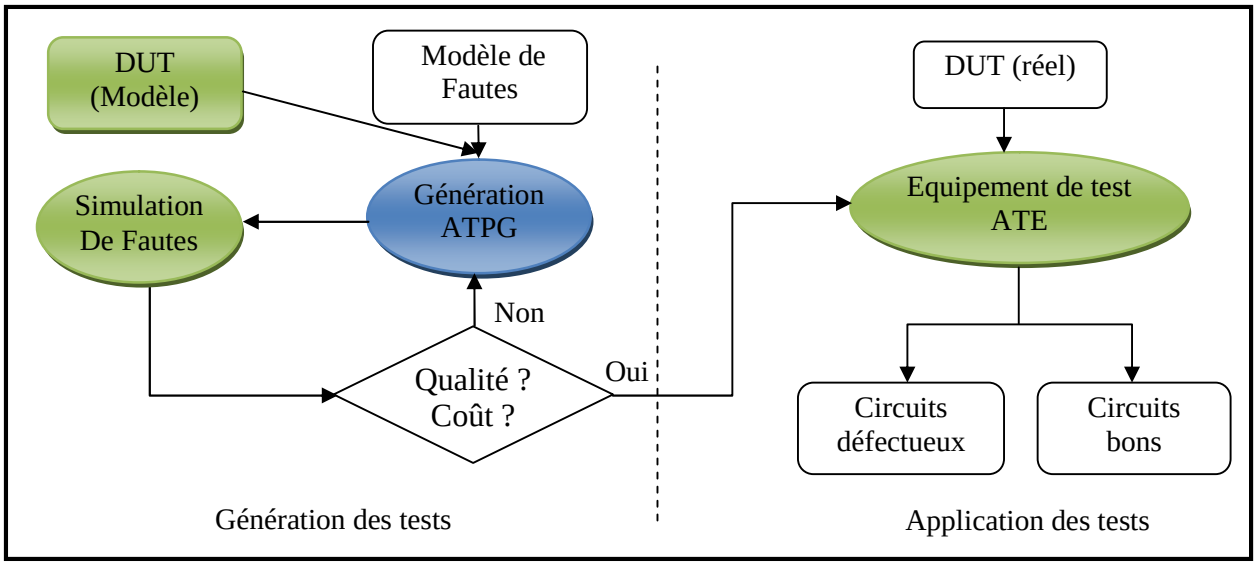


Figure 31. Génération de vecteur de test [Bou07]

La génération de vecteurs de test consiste à définir un programme qui permet de décrire les vecteurs de test spécifiques pour détecter l'ensemble des fautes données dans le but d'avoir une qualité du test et un coût d'application du test optimums [Tri85].

II.4.1- Types de génération

Il existe quatre types de génération de vecteurs de test [Bou07, Kho07, Lau07] :

II.4.1.1- La génération manuelle des vecteurs de test

Les vecteurs de test sont écrits manuellement par l'ingénieur de test. Elle utilise un simulateur de fautes et ne nécessite pas d'algorithme de génération, cependant, il est très difficile d'atteindre un taux de couverture élevé de cette façon.

II.4.1.2- La génération pseudo aléatoire des vecteurs de test

Dans ce cas là, les vecteurs de test sont choisis aléatoirement, en utilisant un simulateur de fautes pour déterminer le taux de couverture, mais elle ne détecte pas les fautes difficiles, ce qui fait que ces techniques de génération ne trouvent pas encore un intérêt industriel important.

II.4.1.3- Génération exhaustive des vecteurs de test

Ce type de génération, applique tous les vecteurs de test possibles, sans avoir à utiliser la simulation de fautes (pas de modèle de fautes). Elle détecte toutes les fautes détectables de tous les modèles de fautes. Cependant elle présente un test très couteux (grand nombre de vecteurs) et s'avère aussi être inutilisable pour les circuits avec un grand nombre d'entrées. A noter que, des techniques de partitionnement du circuit pour réduire le nombre de vecteurs et la génération localement exhaustive peuvent améliorer une telle génération.

II.4.1.4- Génération automatique ATPG « Automatic Test Pattern Generation ».

Dans ce cas là, l'ATPG utilise un simulateur de fautes et des vecteurs de test déterministes, pour chaque faute non détectée un vecteur de test qui la détecte est généré. Il permet d'atteindre un taux de couverture maximum avec un coût minimum. Prenons l'exemple, du circuit ALU 74181-14 entrées, sur lequel l'ATPG détecte 100% des fautes modélisées, en nécessitant seulement 47 vecteurs de test. Cependant, malgré les avantages que présente l'ATPG, il utilise des algorithmes très complexes qui nécessitent donc des temps de traitement considérables.

La figure 32 montre le principe de base de la génération automatique des vecteurs de test.

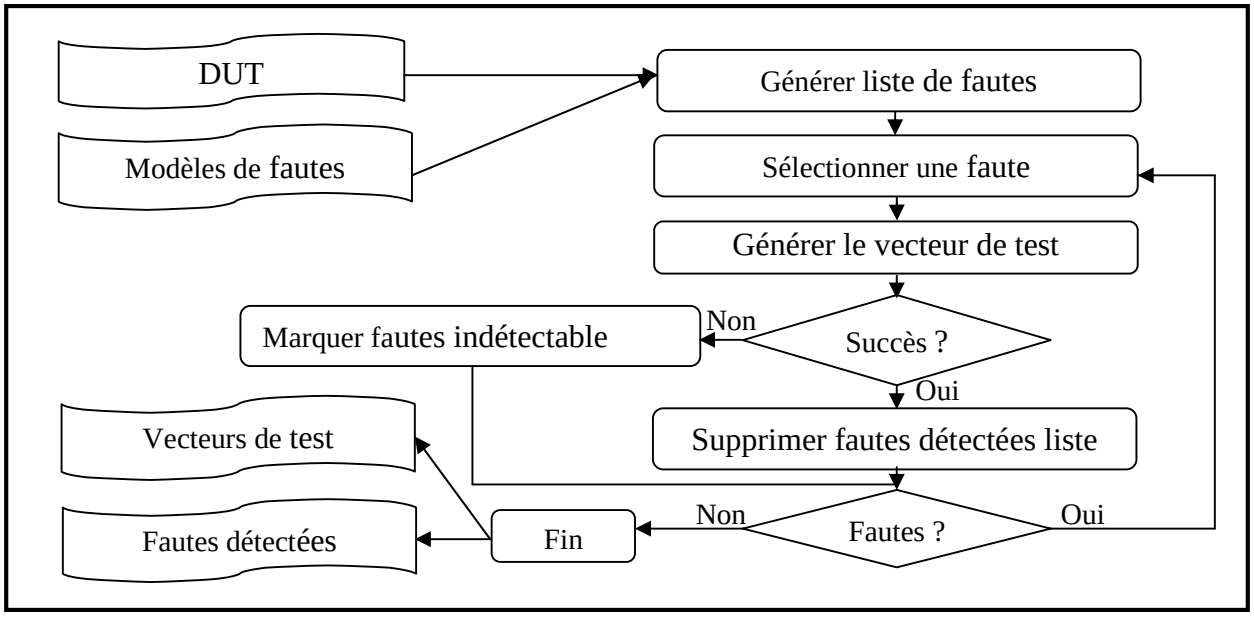


Figure 32. Génération automatique « ATPG » [Kho07, Lak94, Nor89]

Détection de fautes

La recherche d'un vecteur de test T qui détecte la faute F revient à :

- Trouver un vecteur de test T qui :

1- Active la faute F à partir des entrées primaires (PIs).

2- Propage la faute F jusqu'aux sorties primaires (POs).

Exemple : F= S @0

Le vecteur qui détecte la faute **S @0** doit :

- Imposer la valeur 1 au signal S
- Propager la faute **X** =1 jusqu'aux sorties primaires (POs).

Le changement de la valeur du signal S de 1 vers 0 doit être visible sur au moins une des sorties primaires POs du circuit.

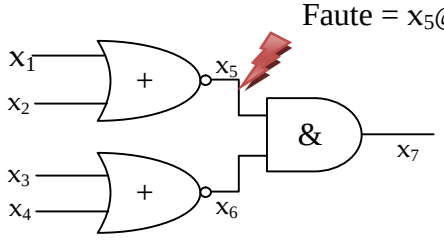
Activation des fautes

Son principe est de ramener le circuit dans l'état ou la faute produit une erreur dans le circuit. Pour le modèle de collage, c'est ramener la valeur opposée du collage sur le signal en question.

Exemple : F = X_5@1

Activation : ($X_5 = 0$) ⇒ ($x_1x_2 = 11$)

Ou ($x_1x_2 = 01$) Ou ($x_1x_2 = 10$)



Propagation des fautes : c'est le processus qui permet de propager la faute à travers toutes les portes du circuit jusqu'aux sorties primaires.

F = X_5@1

Faute = x_5@1 ($x_5 = 0$)

Propagation $\Rightarrow$ ($x_6 = 1$) $\Rightarrow$ ($x_3x_4 = 00$).

Détection de fautes = Activation + propagation

F = X_5@1

Détection $\Rightarrow$ $x_1x_2x_3x_4 = 1100$

Ou $x_1x_2x_3x_4 = 0100$

Ou $x_1x_2x_3x_4 = 1000$

- Algorithmes de génération automatique

Il existe plusieurs algorithmes de génération automatique des vecteurs de test (ATPG), les plus connus et utilisés [Kho07, Lak94] sont : Algorithme D, algorithme PODEM et Algorithme FAN.

1- Algorithme D

Le premier véritable algorithme de génération de test a été développé par IBM en 1966 [Roth66], il utilise la notation D. Pour un signal S, on a :

S = D $\Leftrightarrow$ S =1 dans le bon circuit et S =0 dans le circuit fautif.

S =D $\Leftrightarrow$ S = 0 dans le bon circuit et S =1 dans le circuit fautif.

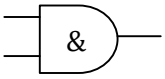
Notation D [Kho07, Lak94]

Valeur du signal pour circuit correct C.C (V)	Valeur du signal pour circuit fautif C.F. (V_f)	Notation D
0	0	0
0	1	D'$\Leftrightarrow$ D
1	0	D
1	1	1

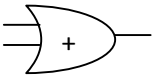
L'algorithme D fonctionne en trois phases :

1- **Phase de « setup** » (activation locale) : on assigne les entrées de la porte pour ramener le signal fautif à la valeur opposée du collage.
2- **Phase de propagation :** on propage la faute jusqu'au sorties primaires.
3- **Phase de justification :** on remonte des valeurs imposées par les deux premières phases sur les signaux internes jusqu'aux entrées primaires.

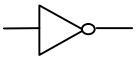
Tables de vérités en notation D



AND	0	1	D	D’
0	0	0	0	0
1	0	1	D	D’
D	0	D	D	0
D’	0	D’	0	D’



OR	0	1	D	D’
0	0	1	D	D’
1	1	1	1	1
D	D	1	D	1
D’	D’	1	1	D’



NOT	0	1	D
	1	0	D’

Chacune des trois phases se décompose en deux étapes :

1- **Étape d’affectation :** c’est l’étape pendant laquelle on fixe des valeurs aux entrées des portes pour contrôler ou propager un autre signal.
2- **Étape d’implication :** c’est l’étape pendant laquelle on simule l’effet des signaux imposés dans l’étape d’affectation sur les autres signaux du circuit (on met à jour les signaux du circuit).

En cas de conflit sur les signaux entre les valeurs à imposer et les valeurs déjà assignées, on revient au dernier choix effectué (pour choisir une autre combinaison des entrées de la porte en question) dans l’ordre suivant : phase de « setup », phase de propagation, et phase de justification et s’il ne reste plus de choix possible, alors la faute est indétectable.

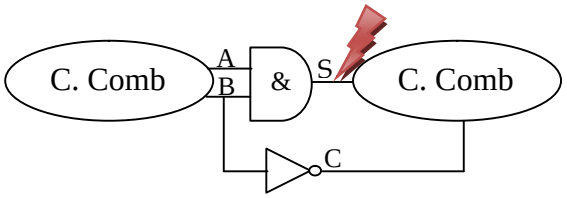
1- Phase de setup : Elle représente le processus pendant lequel on crée la faute (signal D).

Exemple : F = S @0

Setup:

Affectation: => A=1; B=1

Implication :=> S =D; C=0; …



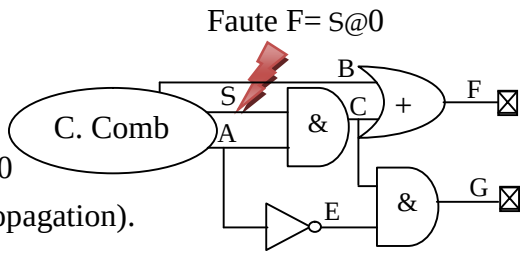
2- Phase de propagation : Elle représente le processus par lequel on propage la faute (signal D) jusqu’aux sorties primaires.

Exemple: F = S @0

Propagation: (S=D)

Affect.1: A=1=> Impl.1: C=D; E=0; G=0

Affect.2: B=0=> Impl.2: F=D (fin de propagation).



L’Algorithme de propagation est donné dans l’annexe A.

3- Phase de justification

Elle représente le processus par lequel on remonte jusqu'aux entrées primaires à partir des signaux imposés pendant les deux premières phases.

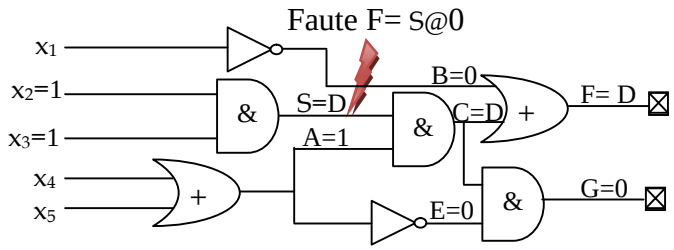
Exemple : F = S @0

Justification :

A=1=>x_4=1 et x_5=x

Ou x_5=1 et x_4=x

B=0 =>x_1=1.



L'algorithme de jstification est donné dans l'annexe A

❖ Problèmes de reconvergence

La reconvergence des signaux de sortie des portes crée des conflits dans les phases de propagation et de justification.

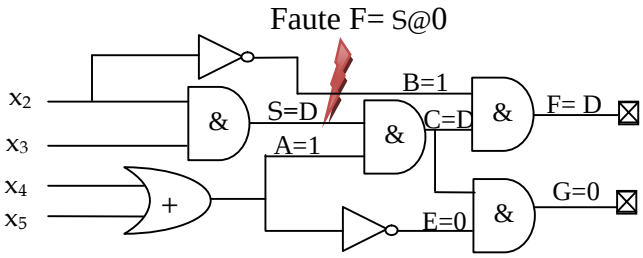
Exemple

setup : S =1⇒ x_2=1 et x_3=1

propagation: A=1; B=1

justification: A=1⇒ x_4=1ou x_5=1

B=1⇒ x2=0 (conflit).



Inconvénients de l'algorithme D

- Nombre très élevé de justifications créant des conflits
  - Choix aléatoire des combinaisons des entrées permettant de justifier la valeur Souhaitée ;
  - Choix aléatoire de la porte qui va propager la faute jusqu'aux sorties primaires.
- Complexité exponentielle.

2- Algorithme PODEM «Path Oriented Decision Making »

L'algorithme PODEM propose plusieurs améliorations , à savoir :

1. Dans les phases de génération, la priorité est donnée à la phase sensibilisation jusqu'aux entrées primaires.
2. L'étape d'implication on essaye la valeur opposée.
3. Pour la propagation de la valeur de test (D ou D') vers les sorties primaires, PODEM utilise un seul niveau logique à la fois.

4. Pour le choix de la porte de propagation, PODEM choisit la porte la plus proche des sorties primaires.

L'algorithme PODEM se décompose en trois phases :

- Détermination de l'objectif (procédure Objectif ()) (voir annexe A).
  Il existe deux types d'objectifs:
  1. Initialement, l'objectif est de ramener la valeur de test (D ou D') à la sortie de la porte fautive.
  2. Propager la valeur de test de l'entrée à la sortie de la porte.
- Remontée (Backtrace) jusqu'aux entrées primaires qui satisfassent l'objectif défini.
- Simulation (implication) de la valeur de l'entrée assignée.

S'il existe toujours un chemin possible, on continue avec un autre objectif sinon, on simule la valeur opposée de l'entrée.

La procédure Backtrace () et la procédure PODEM () sont données dans l'annexe A.

Exemple: Faute = S @0

Obj1: S =1

Bac 1: x_2=1

Sim: S = X; F=X; G=X

Bac 2: x_3=1

Sim: S = 1; F= X; G=X

Obj 2: C=D/D'

Bac 1: A=1

Bac 2: x_4=1

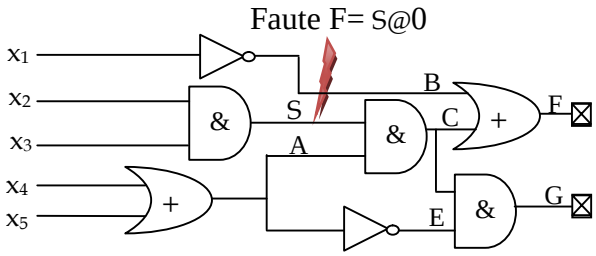
Sim: C=D; F= X; G=0

Obj 3: F=D/D'

Bac1: B=0

Bac 2: x_1=1

Sim: F=D; G=0



Avantages de l'algorithme PODEM

- Moins de backtraces que l'algorithme D
- Utilisation d'heuristiques pour réduire le nombre de backtraces.

Inconvénients

- Là aussi le nombre de backtraces demeure assez élevé créant des conflits et une complexité exponentielle.

3- **Algorithme FAN :** l'algorithme FAN est une amélioration des algorithmes D et PODEM qui ont un inconvénient majeur qui est le nombre élevé de backtraces dû au problème de reconvergence dans les circuits tel illustré à la figure 33.

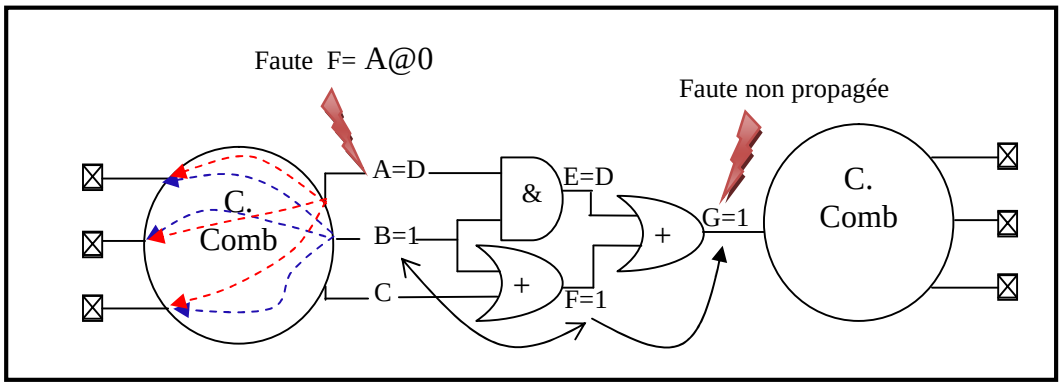


Figure .33 Exemple de reconvergence [Kho07]

Pour y remédier, FAN utilise la topologie du circuit pour réduire le nombre de backtraces, notamment par, identification des nœuds qui sont des sources de reconvergence (check point), et des portes de reconvergence. Mais aussi, dans la phase de génération, une simulation (implication) est effectuée à chaque fois qu'on assigne un signal qui est source de reconvergence (chekpoint). Pour l'étape de justification, on justifie en premier les signaux qui sont des sorties des portes de reconvergence.

II.5- La conception en vue du test (DFT)

La DFT est un ensemble de techniques mises en œuvre en phase de conception du circuit pour en faciliter le test (fig.34). Autrement dit, la DFT sert à trouver une interface entre les communautés de test et de design. Mais aussi de s'assurer que les designs sont économiquement testables. Cela peut aller d'une simple amélioration de l'accessibilité et de l'observabilité de certaines sous parties à une conception autotest intégrée où toutes les fonctions du test sont implémentées dans la puce [Bou07, Cim07, Kho07], permettant ainsi de réaliser le test à la fréquence nominale de fonctionnement du circuit. Mais aussi, elle rend possible la vérification du bon fonctionnement dynamique du circuit, ce qui très important avec l'évolution des technologies et l'augmentation des fréquences d'horloge.

- **L'objectif de la DFT**

Son objectif est de diminuer le temps TTM en rendant automatique la phase de production de vecteurs de test qui diminue le coût d'application des tests et en générant des ensembles de test plus réduits pour diminuer le coût du test. La DFT rend aussi la fiabilité des circuits complexes

optimale, en améliorant la qualité du test par l'augmentation du taux de couverture de fautes (95% -100%) [Lau07].

II.5.1- Les stratégies de la DFT

D'un point de vue historique, les stratégies de la DFT des circuits intégrés peuvent être classées comme suit [Kho07, Lak94, Lau07, Ste00]:

- Les techniques AD-HOC développées pour les circuits imprimés puis étendues aux circuits intégrés;
- Les techniques structurées (systématiques) ;
    - Le balayage (scan path, boundary scan JTAG **(** Joint Test Action Group))
    - BIST (Built In Self Test)

En général, l'approche adoptée est une combinaison des deux techniques.

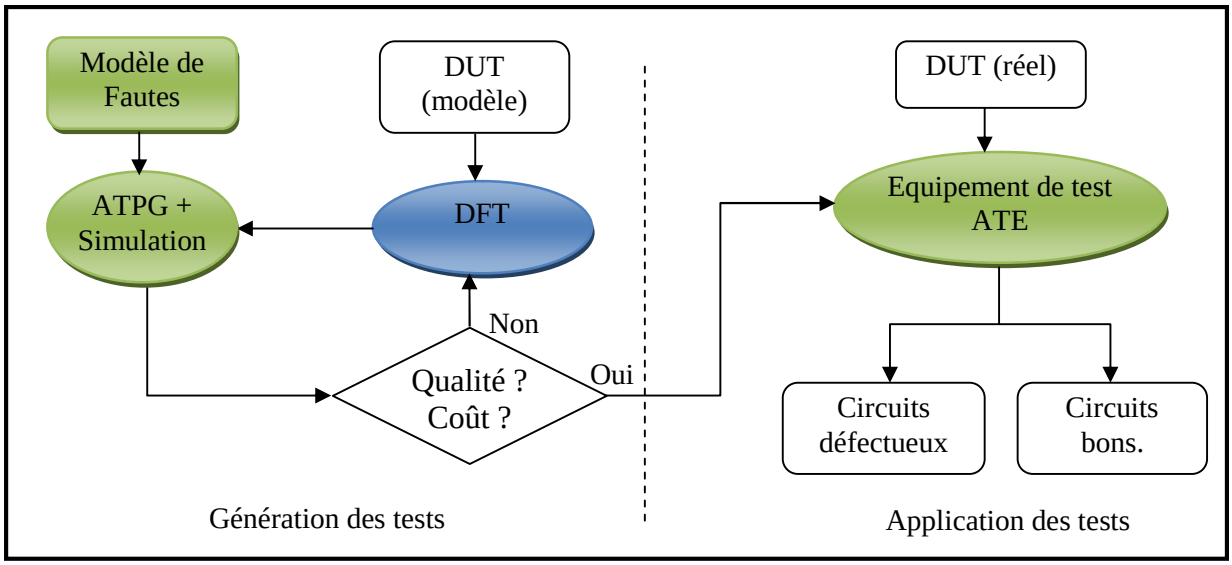


Figure 34. La DFT [Bou07, Kho07]

II.5.1.1- Les techniques AD-HOC

Elles permettent d'augmenter les mesures de testabilité des circuits [Abr90], ce sont des pratiques de conception apprises et utilisées par expérience et sont spécifiques. Les techniques Ad-Hoc sont appliquées pour éliminer les boucles asynchrones, rendre les registres faciles à initialiser, éliminer les portes ayant un grand nombre d'entrées et proposer le contrôle des signaux difficiles à contrôler. Elles varient d'un design à un autre, il n'existe donc pas une méthodologie commune pour appliquer ces techniques. Elles consistent en l'insertion de points de test, l'initialisation et le partitionnement des circuits complexes en sous blocs identiques

1- Insertion de points de test

Dans un circuit combinatoire, ceci permet de réduire le nombre de vecteur de test nécessaire à la détection de toutes les fautes de collage. Il existe deux types de points de test :

- **Points contrôlables**

Un point est dit contrôlable s'il est possible de lui affecter une valeur donnée juste en assignant des valeurs aux entrées primaires. La modification du circuit en ajoutant de la logique et des entrées primaires de contrôlabilité CP et aussi en multiplexant les entrées primaires entre le mode test et le mode fonctionnement via les multiplexeurs ajoutés rend un nœud contrôlable. Deux modes d'opération sont possibles :

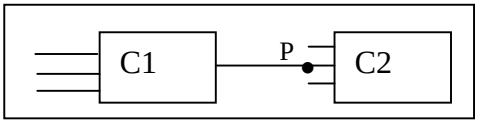
Mode fonctionnel : le circuit fonctionne comme le circuit initial.

Mode test : la valeur du signal contrôlé est déterminée par CP, en utilisant :

Une porte AND ou NOR pour injecter un 0.

Une porte OR ou NAND pour injecter un 1.

Exemple : soit P le signal à contrôler



Circuit initial

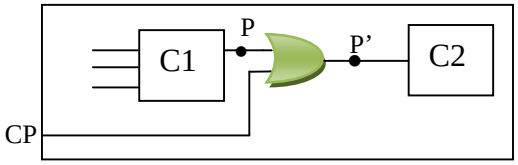
1)-Injection d'un 1

Si CP=0 :

P' = P (mode fonctionnel)

Si CP=1 :

P'=1 (mode test)



1) Injection de 1

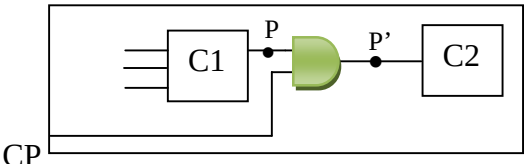
2)-injection d'un 0

Si CP=1 :

P'=P (mode fonctionnel).

CP=0

P'=0 (mode test)



2) Injection de 0

3)- Si CP1 =CP2 = 0

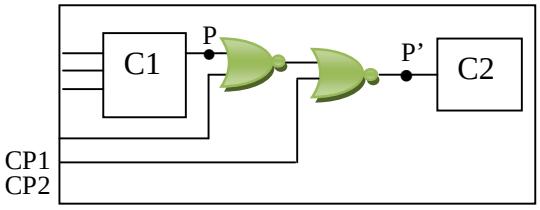
P'= P (mode fonctionnel)

Si CP1=1

P'= not (CP2) (mode test).

Si CP2=0 on a injection de 1

Si CP2=1 on a injection de 0



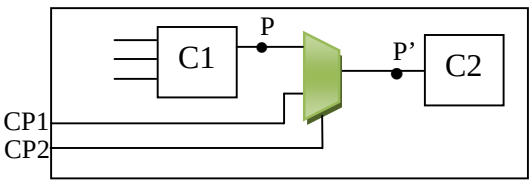
3) Injection de 0/1

4)- Si CP2=0

P'=P (mode fonctionnel)

Si CP2=1

P'=CP1 (mode test)



4) Injection de 0/1

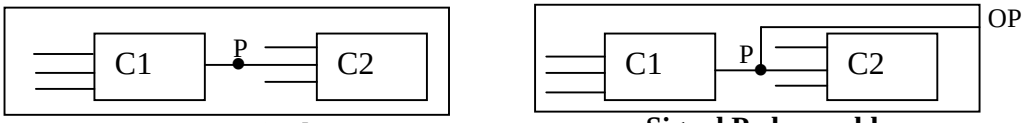
Si CP1=0 on a injection de 0

Si CP1=1 on a injection de 1

- **Points observables**

Un nœud est dit observable s'il est possible de propager sa valeur aux sorties primaires. La modification du circuit initial en ajoutant des sorties primaires d'observabilité OP fonctionnant avec un seul mode d'opération rend le point observable.

Exemple soit P est le signal à contrôler :



Circuit initial **Signal P observable**

2- Initialisation

C'est le processus qui permet de ramener un circuit séquentiel à un état connu à un moment donné (fig.35). Sa méthode consiste en l'utilisation d'un signal reset global pour tout le circuit (reset commandé par un signal d'entrée primaire, et dans le cas de pin limited, utiliser un reset généré par le matériel).

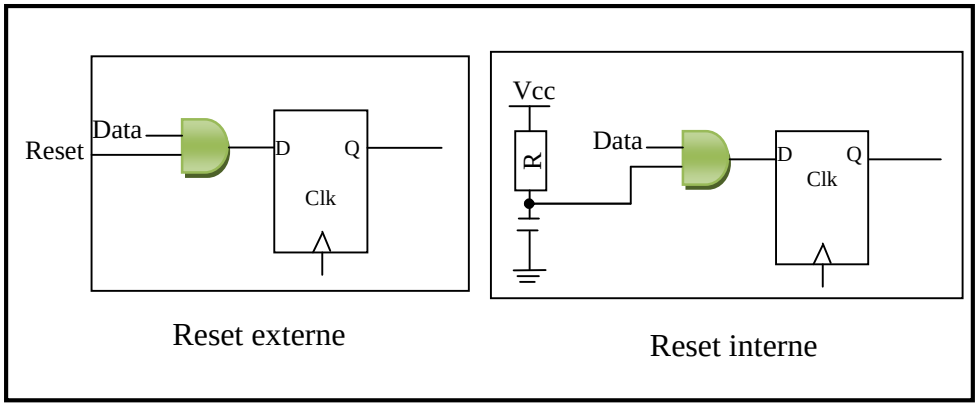


Figure 35. Principe d'initialisation [Kho07, Lau07].

- Reset externe: dans ce cas on distingue deux modes de fonctionnement :

Si reset = 1 : Q = Data (Mode fonctionnel)

Si reset = 0 : Q = 0 (mode reset)

- Reset interne : dans ce cas là, il n'y a pas de pin additionnel.

4- Partitionnement

Le partitionnement consiste à diviser les circuits combinatoires complexes avec multiplexage des entrées et sorties primaires (fig.36), en sous blocs fonctionnellement identiques

de sorte à considérer le test des blocs individuels. En effet, les compteurs et les registres à décalages sont difficiles à tester car ils nécessitent beaucoup de cycles d'horloge. Par exemple, un compteur 16 bits nécessite 65536 cycles pour son test. Alors, que deux compteurs de 8 bits nécessitent moins de 512 cycles [Kho07].

Le Partitionnement des circuits complexes s'effectue en deux phases, choix des sous circuits identiques et insertion des points de test. le coût en surface est exprimé en nombre de lignes communes aux deux sous circuits. Son but est de réduire la complexité des outils de simulation de fautes et de génération automatique des vecteurs de test, améliorer la contrôlabilité et l'observabilité des signaux. La figure 36 montre un exemple de partitionnement d'un circuit, pendant la propagation et la justification, la longueur de test du circuit initial est de (C1+ C2) alors qu'une fois partitionné devient égale à la valeur maximale entre (C1, C2).

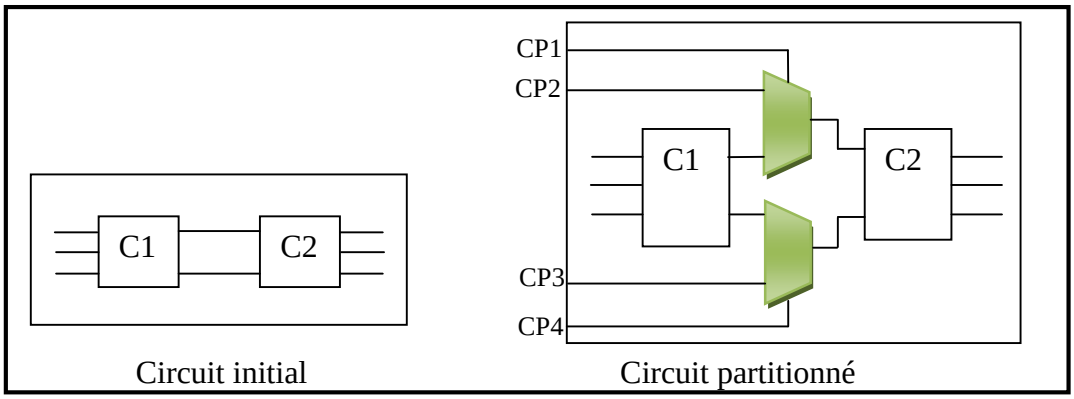


Figure 36. Principe de partitionnement [Kho07, Lan07].

II.5.1.2- Les techniques structurelles (scrutation : scan path)

Elles ont pour objectif l'augmentation de l'observabilité et la contrôlabilité des nœuds internes du circuit, et faciliter le test des circuits séquentiels synchrones [Cho01, Kho07]. Ces techniques sont largement utilisées dans l'industrie, puisque près de 90% des circuits vendus dans le monde possèdent une ou plusieurs chaînes de scan [ITRS05].
Les deux techniques les plus utilisées sont:

- **Scan path seriel**

Chaque élément mémoire du circuit est modifié en le configurant en mode décalage « shift register » pour contrôler et observer les états. Permettant de tester plus facilement les circuits séquentiels. En intégrant ainsi au circuit un mode test qui lorsqu'il est activé relie les points mémoires et les registres à décalage [Ran75] d'un circuit pour former une chaine de scan de la figure 37, directement contrôlable et observable depuis l'interface. Cependant, le Scan path seriel

engendre un coût élevé**,** une surface et un nombre de pins additionnels (PIs/ POs) très élevés, la dégradation des performances et l'augmentation du temps d'application des tests.

Principe de fonctionnement du Scan seriel

Le circuit séquentiel est transformé en circuit combinatoire, chaînage des bascules « éléments mémoires », chaque entrée de bascule est considérée comme une sortie primaire et chaque sortie est considérée comme une entrée primaire.

Le test du circuit est effectué en deux phases, test de bascules chaînées et test de la partie combinatoire.

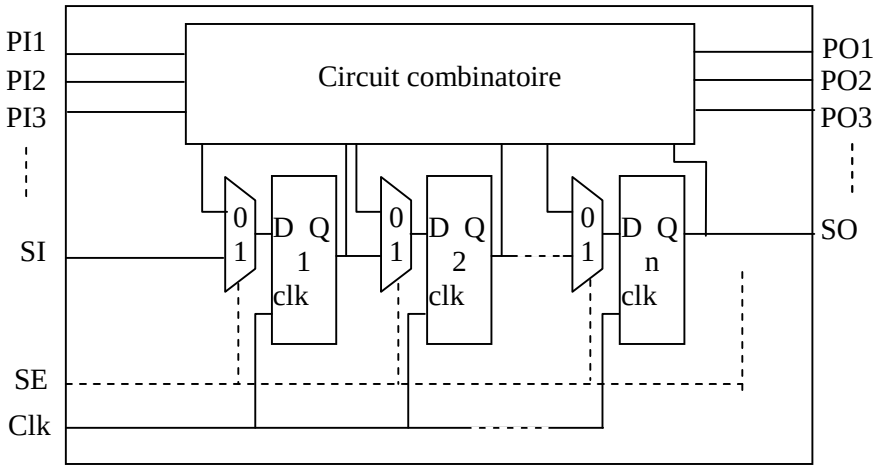


Figure 37. Technique de Scan Sériel [Kho07].

- **Scan périphérique**

La technique dite de **Boundary-Scan « JTAG»** est une extension des techniques de scan path pour le test des interconnexions entre les différentes puces et le test des cartes [Kho07, Par92]. Chaque signal primaire d'entrée et de sortie est complété avec un élément de mémoire appelé Cellule Boundary-Scan « BSC ». Ces cellules sont chainées entre elles sont représentées à la figure 38, pour former le Registre Boundary-Scan « BSR ».

Le scan périphérique est utilisé pour le test des circuits intégrés, la programmation des systèmes programmables (FPGA) et le test des systèmes on chip (SOC). Cependant il nécessité l'ajout de matériel cellules, par exemple 0,3% pour le 68040 de Motorola, « 4 plots obligatoires, routage », perte de rendement du processus de fabrication, dégradation des délais, manque d'outils de CAO incluant le JTAG et rallongement du TTM.

- **Architecture de base du Boundary Scan (BS)**

La figure 38 montre l'architecture générale d'une carte munie du dispositif BS.

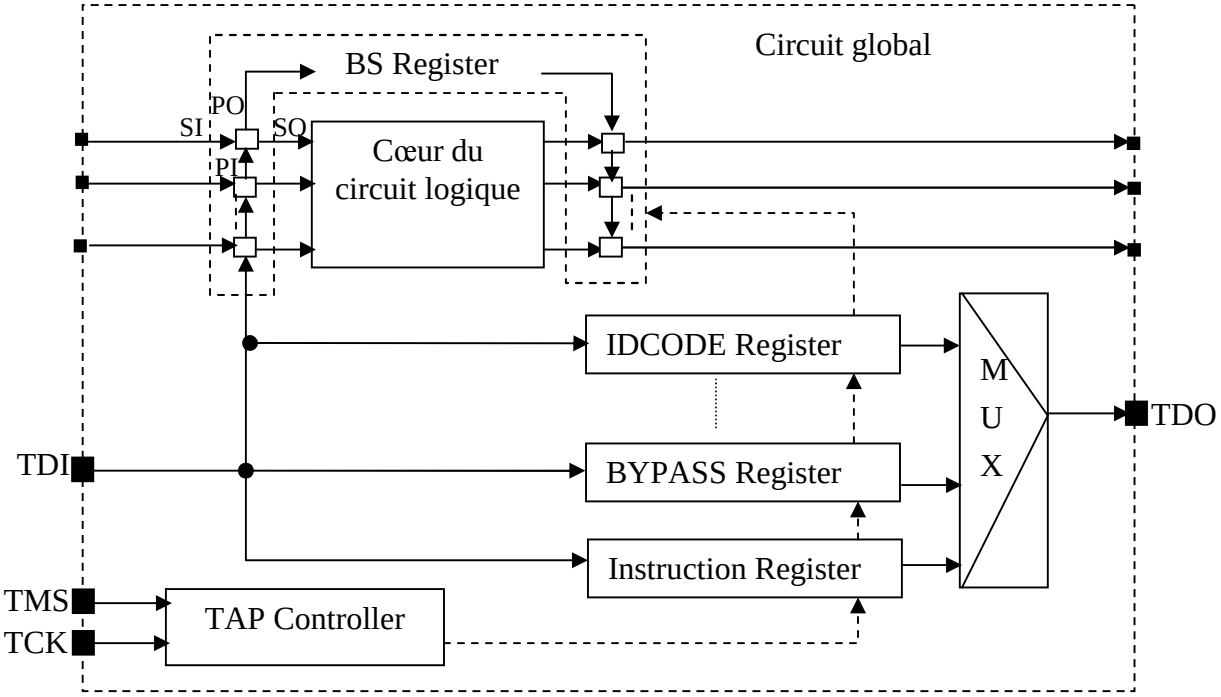


Figure 38. Scan périphérique « JTAG » [Kho07].

Les principaux éléments composant BS, sont les suivants [Kho07] :

- La cellule Boundary Scan de base, plusieurs cellules connectées entre elles en série formant ainsi le registre Boundary Scan.
- Le bus de test : composé au minimum de quatre signaux, notamment :

  TCK (Test Clock) : c'est l'horloge du contrôleur, indépendante de celle du circuit et n'est utilisée qu'au cours du processus de test BS.

  TDI (Test Data Input): les données ou les instructions de test sont reçues via cette ligne et sont dirigées vers le registre approprié.

  TDO (Test Data Output): c'est la sortie par laquelle le contenu d'un registre (instruction ou donnée) est transmis à l'extérieur en mode série.

  TMS (Test Mode Select) : utilisé pour contrôler le contrôleur TAP.

  TRST (Test Reset) : c'est un signal optionnel pour initialiser le TAP contrôleur.
- Le contrôleur de bus de test TAP (Test Access Port) : synchronisé par le signal TCK, comprend une entrée unique (TMS).
- Les registres d'instructions et de données

  Registre Instruction : détermine le mode de fonctionnement du Boundary- Scan

Les registres de données sont :

- Registre Boundary – Scan : C'est le registre le plus important, il permet de connecter tous les plots du circuit « sauf les plots du TAP). Ce registre est obligatoire ;
- Registre Idcode : C'est un registre d'identification, il contient le code d'identification du circuit, la version du circuit et il est optionnel ;
- Registre Bypass : Ce registre contient une seule cellule Boundary- Scan « BSC » (1 bit) réduit le temps de test permettant d'accéder directement à un circuit donné de la carte. Ce registre est aussi obligatoire.

II.5.1.3 - Autotest intégré BIST

Le principe du BIST (fig.39) consiste à ajouter de la circuiterie pour permettre au circuit ou au système de se tester tout seul (Auto-Test) [Ste00, The00] sans avoir recours à un équipement de test cher et complexe. Le BIST est très utilisé pour les mémoires, les circuits analogiques, les systèmes on chip « SOC», les circuits combinatoires et séquentiels.

Les trois principales parties dans le design d'un BIST sont :

- Le générateur de vecteurs de test pseudo- aléatoire « PRPG » qui produit les vecteurs de test pour le circuit ;
- L'analyseur de réponses qui vérifie que la réponse est conforme à la donnée attendue ;
- La partie contrôle qui active les différentes phases de test.

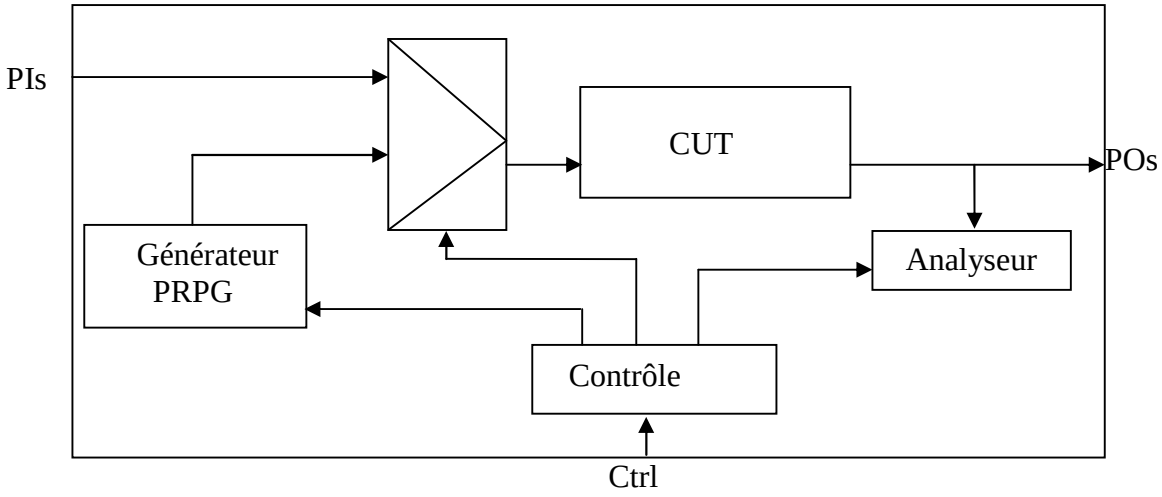


Figure 39. Architecture générale du BIST [Kho07].

✓ **Avantages**

- Génération interne des vecteurs de test, amélioration de contrôlabilité ;
- Analyse interne des réponses, amélioration d'observabilité ;

- Test on - line et off-line;
- Équipements de test simples ou pas d'équipements de test ;
- Test à la fréquence d'utilisation « plus rapide, précis et permet le test réel des délais ».
- Facilite la maintenance

✓ **Inconvénients**

- Augmentation de la surface de la puce et donc son coût « exemple/ Mémoire de 4096 mots de 128 bits de l'ordre de 13% »,
- dégradation des performances « Rom de 256 Kbits de l'ordre de 6% » et ajout de pins d'entrée/sortie. En contre partie, cette augmentation est compensée par la facilité et la rapidité de génération des vecteurs de test. En particulier, lorsque il s'agit de tester un système entier où les méthodes classiques sont trop longues et inefficaces.

▪ **Techniques de génération pour le BIST**

Il existe plusieurs techniques de génération et d'application des vecteurs de test pour le BIST, notamment :

- Registre à décalage « LFSR » pour la génération aléatoire et exhaustive.
- Compteurs binaires, pour la génération exhaustive.
- Roms « Prestored TPG », pour la génération déterministe ou fonctionnelle.
- Automate cellulaire, pour la génération aléatoire.

Deux types de BIST sont utilisés :

- Off- line BIST

  Le test est effectué pendant le mode de test, selon deux modes:

  - Off- line fonctionnel : pour le diagnostic.
  - Off- line structurel : détection des défauts de fabrication.

- On- line BIST [Kho07, Nic98] :

Le test s'effectue selon deux modes durant le fonctionnement normal du circuit pour détecter les défauts :

- On-line concurrent appliqué avec le mode de fonctionnement normal du circuit.

- On-line non concurrent, dit aussi test hors ligne appliqué comme activité séparée.

Conclusion

La principale conclusion que l'on peut tirer de ce chapitre est que les techniques de test ainsi présentées dans le domaine des circuits intégrés permettent d'optimiser les étapes de test de production en termes de coût et fiabilité. Il est maintenant, pratiquement impossible de se passer des techniques de DFT, qui sont la meilleure option actuelle, pour des circuits dépassant les 100 millions de transistors.

Ainsi, le concepteur doit garder à l'esprit les aspects, règles de conception Ad.Hoc suivants afin de faciliter le test final de la puce :

- Établir une liste de nœuds clés durant la phase de conception et vérifier qu'elles soient aisément accessibles ;
- Eviter d'utiliser la logique asynchrone ou redondante sauf si absolument nécessaire ;
- Procurer des moyens pour que tout les latchs et flip flops internes puissent être initialisés pour des fins de test (ajout d'un reset) ;
- Consulter le fondeur durant toutes les phases de conception du circuit intégré pour que des procédures de test mutuellement acceptables soient formulées.

Les paramètres technologiques sont des données difficiles à se procurer et qui dépendent entre autres des circuits fabriqués où d'une technologie donnée. La probabilité T qui caractérise la tolérance aux défauts d'une structure peut être déterminée grâce à une procédure de test. Ainsi pour analyser et prédire la fiabilité de n'importe quel circuit intégré en phase de conception la méthode que nous proposons est basée sur des travaux relatifs à la notion de test utilisant l'ATPG, la simulation de fautes et les techniques de partitionnement. Cette méthode fera l'objet du chapitre III.

Chapitre III

Conception en vue de fiabilité et tolérance aux fautes

En s'inspirant des structures tolérantes aux fautes « TMR », nous pouvons améliorer, voire même procurer une fiabilité étendue aux circuits conçus. La conception du plan d'essais « quels essais faut-il faire pour montrer que le circuit est fiable ? » intervient dés le tout début de la conception, dés que les fonctions requises « le cahier des charges » sont connues.

Pour répondre à cette question, nous allons :

- D'abord rappeler les principes d'évaluation de la tolérance aux fautes.
- Mettre en place une procédure de test spécifique basée sur la génération automatique de vecteurs de test « ATPG » permettant de lier la conception de plusieurs circuits électroniques (en Verilog) [Jal91, Lay80, Sew06], leur simulation une fois réalisés, et pour finir avec le test en vue de garantir leur fiabilité.
- Analyser les résultats expérimentaux du calcul de la probabilité de tolérance aux défauts « T » des circuits. Faire une analyse prédictive de l'impact de la réalisation des structures TMR sur le rendement de fabrication et la fiabilité de tels circuits.

III.1- Principes d'évaluation de la tolérance

La tolérance aux fautes est un moyen utilisé afin d'arriver à la sureté de fonctionnement du circuit, englobant ainsi sa fiabilité [Goe80, Sie92]. Malgré la présence d'un défaut, le circuit continue de fonctionner correctement. La tolérance aux défauts de fabrication des structures TMR est considérée comme la probabilité « T » qu'une paire de défauts soit tolérée.

Pour évaluer cette probabilité, nous nous basons sur deux principes, à savoir :

- Une paire de fautes est testable, donc n'est pas tolérée par la structure TMR, autrement dit, elle n'est pas masquée par le voteur, causant ainsi une erreur en sortie de la structure et remet en cause son bon fonctionnement. Dans ce cas là, il s'agit des fautes multiples.
- Une paire de fautes n'est pas testable, donc tolérée par la structure TMR, elle est masquée par le voteur, l'erreur n'est pas observée en sortie de la structure TMR, donc le bon fonctionnement n'est pas remet en cause.

III.1.1- Méthodologie de test

Dans le souci, de savoir si le circuit conçu en vue de fiabilité fonctionne ou pas en présence de paires de défauts. La procédure de génération de vecteur de test « ATPG » doit prendre en compte le circuit modifié en structure TMR.

III.1.2- Modèle de fautes choisi

Nous avons choisi le modèle de « paires de fautes ». Pour représenter les défauts multiples simultanés, notre choix est porté sur le modèle de fautes de collage doubles [Abr80, Fan06]. Ainsi, l'ATPG peut nous donner la probabilité que deux défauts soit tolérés (T).

- Deux défauts d_1, d_2, modélisés par une paire de fautes $\{f_1, f_2\}$.
- Trois défauts d_1, d_2, d_3, modélisés par trois paires de fautes$\{\{f\ ,f\ \},\{f\ ,f\ \},\{f\ ,f\ \}\}$.

En général, n fautes de collage sont équivalentes à C paires de fautes.

Le nombre de paires de fautes de collage multiples dans les 3 modules de la TMR est:

$$C = \frac{3N!}{(3N-2)! \times 2!} = \frac{9N^2 - 3N}{2}$$

N : le nombre de fautes de collage simple présent dans un module.

III.1.3- Principe de fonctionnement de l'ATPG

Pour connaître la probabilité T qu'une paire de fautes soit tolérée, on doit connaître le nombre de paires de fautes tolérées et non tolérées, un outil de génération de vecteurs de test (ATPG) détectant les paires de fautes est lancé. Ainsi, pour chaque paire de fautes, l'ATPG va chercher un vecteur de test permettant d'observer une erreur en sortie de la structure TMR. Lorsqu'un vecteur de test est trouvé, cela signifie qu'elle est observable sur au moins une sortie et donc qu'elle n'est pas tolérée. A l'opposé si aucun vecteur de test n'est trouvé, cela signifie que la paire de fautes est tolérée.

L'ATPG nous donne ainsi, le taux de couverture de fautes « FC » et la probabilité T de tolérance ou de résistance qu'une paire de fautes soit tolérée par la structure, définie par :

$$T = \frac{\text{Nombre de paires de fautes tolérées}}{\text{Nombre total de paires de fautes}}$$

Pour calculer la probabilité T, nous avons choisi dans un premier temps, le simulateur «ModelSim PE Student Edition 6.6b » de la compagnie Mentor Graphics [Mod05]. On a pris l'exemple d'un « compteur-décompteur », décrit en langage de description de haut niveau VHDL [Del96, Jei09] (voir annexe B) en utilisant le logiciel ISE 9.2 développé par la société Xilinx. Après avoir lancé le testbench, nous obtenons les résultats de simulation de la figure 40.

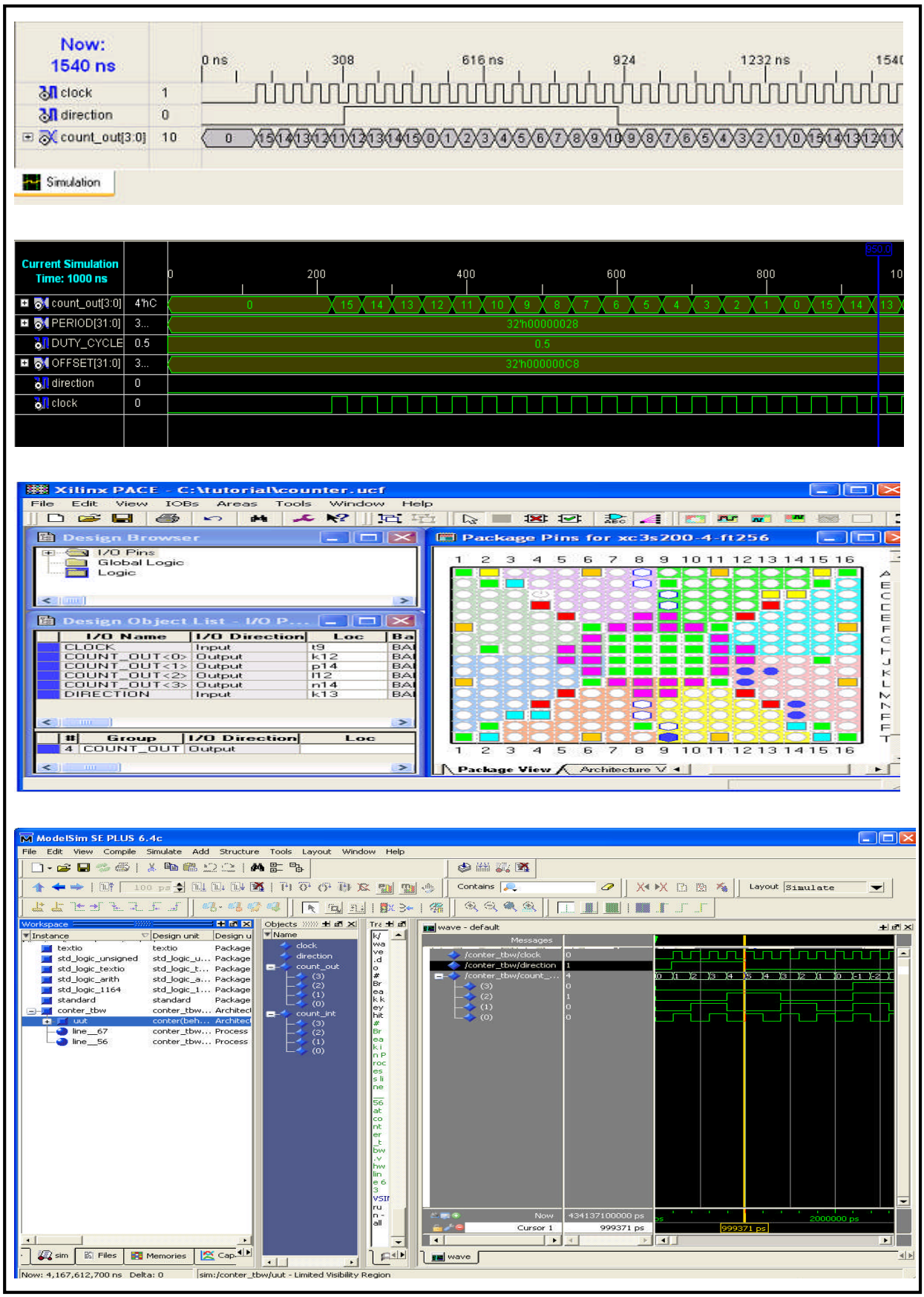


Figure 40. Simulation du circuit sans fautes

Pour voir le comportement du circuit en présence de fautes, on a injecté des fautes de type collage (s@0, s@1) en forçant les points externes de la manière suivante:

- Sélectionner File>New>Source> Do ---> créer un nouveau DO File ;
- Entrer les commandes suivantes dans la source Windows du modelsim :

```
vsim test _conter
add wave cont
add wave clk
add wave Direction
force-freeze clk 0
force Direction 1
run 100
force Direction 0
run 300
force Direction 1
run 400
force Direction 0
run 200
```

- Save the file
- Executer le Do file.

Comme montré sur la figure 41, aucune information sur le nombre de fautes inestables n'est obtenue, donc il est impossible de calculer la probabilité T.

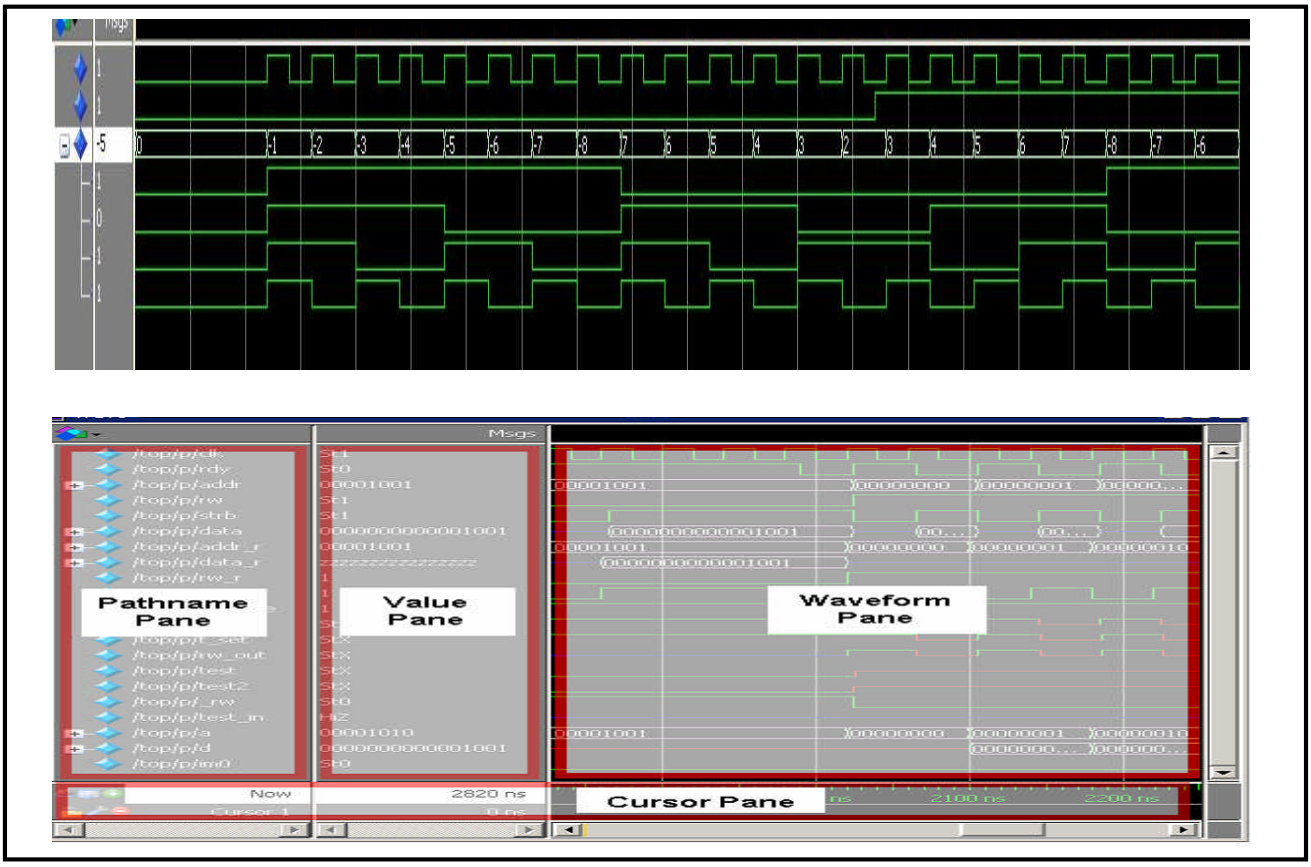


Figure 41. Simulation du circuit avec fautes

Ce type de logiciel ne permet pas également d'injecter des fautes au niveau des lignes internes du circuit. Nous avons alors utilisé dans un deuxième temps le logiciel TetraMax de Synopsys [Sit01, Sit04, Sit05]. Celui-ci travaille avec un modèle sous forme de primitives « AND, OR,… » [Kho07, Lay80] (annexe B). C'est un outil de simulation de fautes et de génération de vecteurs de test est très puissant, cependant les algorithmes utilisés dans les programmes ne sont pas disponibles.

La procédure de détection de paires de fautes est la suivante :

- Soit une paire de fautes $\{f_1, f_2\}$ composée des deux fautes de collage f_1 et f_2.
  - La faute f1 est injectée comme une faute permanente dans la description structurelle, à l'intérieur d'un module de la structure TMR. Cette dernière est ainsi modifiée puisque le module où la faute a été injectée n'est plus identique aux deux autres.
  - L'ATPG détectant les fautes de collage simple est lancé sur cette structure TMR modifiée. De cette manière, on obtient un ATPG ciblant toutes les paires de fautes contenant f1.
  - Nous effectuons la même procédure pour chaque faute de collage de la structure TMR en injectant une faute permanente à chaque fois différente dans un module et en relançant à chaque fois l'ATPG. Ainsi toutes les paires de fautes peuvent être ciblées.

A noter que dans un circuit il y a N fautes de collage simple, donc il y a 3N fautes de collage dans les trois circuits. Ainsi 3N ATPG sont requis.

III.1.4 Réduction de la liste de fautes traitées par l'ATPG

Pour réduire le temps de traitement nécessaire à la procédure de génération de vecteurs de test, la liste de fautes de l'ATPG peut être diminuée. Pour ce faire, nous avons utilisé les caractéristiques et les symétries des structures TMR pour que :

- L'ATPG ne traite pas deux paires de fautes équivalentes
- L'ATPG ne traite pas les paires de fautes qui sont structurellement non testables au vu de leurs emplacements ou de leurs cônes de sorties.

En effet, toutes les paires de fautes dont les deux fautes de collage associées sont présentes dans le même circuit sont non testables puisqu'elles n'affectent qu'un seul circuit. La liste de fautes de l'ATPG peut être ainsi réduite d'un tiers. De même, Les trois circuits étant identiques, injecter une faute permanente dans le premier circuit est équivalent à injecter la même faute permanente dans le deuxième ou le troisième circuit. Grâce à cette symétrie, le temps de simulation est réduit puisqu'il n'y a plus que N ATPG à lancer. Les fautes de collage permanentes seront injectées uniquement dans le premier circuit.

Les symétries de la structure TMR peuvent encore être utilisées pour réduire la taille de la liste de fautes de l'ATPG. Comme montré par la figure 42 deux paires de fautes peuvent être équivalentes. Dans les deux exemples, f_1 est la faute permanente injectée et f_2 se trouve dans la liste de l'ATPG. Or f_2 a le même emplacement et la même nature mais dans deux circuits différents. La recherche de la testabilité de ces deux paires de fautes est équivalente. Cela permet de réduire par deux la liste de fautes de l'ATPG en ne traitant, par exemple, que les paires de fautes avec f_1 toujours présente dans le premier circuit et f_2 dans le deuxième circuit.

Avec ces réductions, la durée du test diminue, le temps d'exécution du programme varie de quelques secondes pour le circuit C 17 et à peu prés 7 heures pour le circuit C 7552.

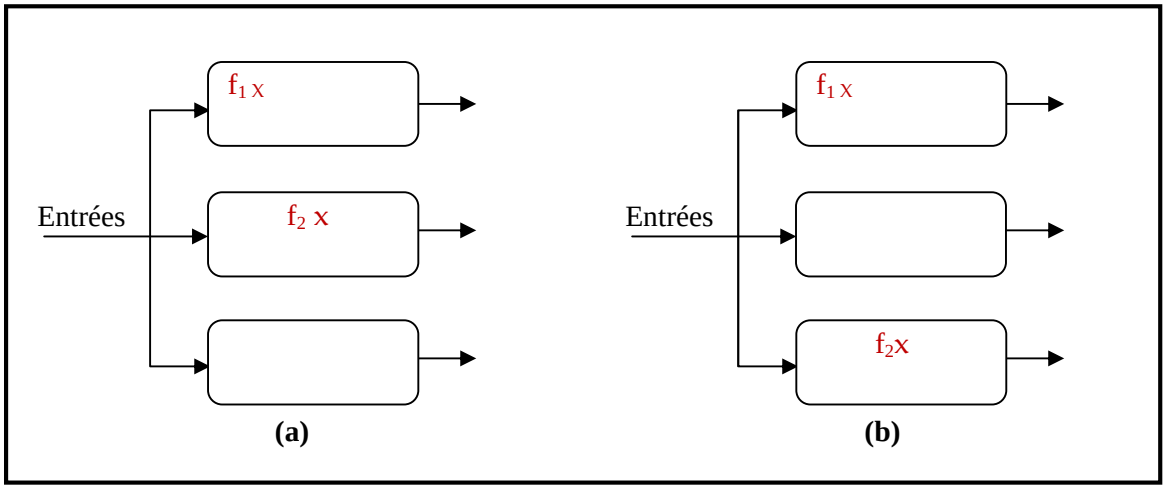


Figure 42. Deux paires de fautes équivalentes [Lau09]

III.2 Expérimentations

La procédure de test permet d'évaluer la tolérance aux fautes des structures TMR.

Pour comprendre le fonctionnement de l'ATPG, nous présentons sa mise en œuvre sur un exemple simple basé sur le circuit c17 de la série ISCAS85 (annexe B). Nous donnons ensuite les résultats obtenus avec un ensemble de circuits des benchmarks ISCAS85 et ITC99 [Kaw85, Lay80, Sit01, Sit03, Sit04, Sit05]. Ces benchmarks sont des ensembles de circuits combinatoires.

III.2.1- Le flot d'utilisation de TetraMax

La génération des vecteurs de test contient 8 phases [Sit01, Sit05] :

Phase 1 : Démarrage de TetraMax

Taper stsyn2005 : le script de configuration Taper tmax32 : le démarrage du logiciel TetraMaxTM en mode console.

Phase 2 : lecture de la description des circuits

Read Netlist <file_name> [Format<Edif/ Verilog/ Vhdl>]
[-Sensitive/ -INSensitive] [-Delete] [-Library][-Master_modules][-Noabort][Verbose]
Exemple:
> read netlist. / ISCAS85/ c17 .v

Dans notre cas, le circuit c17 contient 6 portes Nand, avec 5 entrées « E1, E2, E3, E4, E5 » et deux sorties « S1, S2 ». Son format Verilog est illustré à la figure 43.

```
1   module c17 (E1, E2, E3, E4, E5, S1, S2);
2   input E1, E2, E3, E4, E5;
3   output S1, S2;

4   nand P1 (.Z (N1), .A (E1), .B (E3));
5   nand P2 (.Z (N2), .A (E3), .B (E4));
6   nand P3 (.Z (N3), .A (E2), .B (N2));
7   nand P4 (.Z (N4), .A (N2), .B (E5));
8   nand P5 (.Z (S1), .A (N1), .B (N3));
9   nand P6 (.Z (S2), .A (N3), .B (N4));

10  end module
```

Figure 43. Format Verilog du circuit c17 original « sans DFR ».

Phase 3 : Description de la structure TMR

A partir de la description des circuits numériques des familles des benchmarks ISCAS85 et ITC99, les structures TMR ont été construites en clonant chaque circuit 3 fois, en reliant leurs entrées et en ajoutant un voteur à chaque sortie de circuit [Che78]. Pour fabriquer les structures TMR, nous utilisons les descriptions verilog [Jal91, Lay80, Sit03] des circuits. Un programme codé en Python (option dans tetramax) a été réalisé pour décrire la structure TMR en créant un fichier TOP.

Par exemple, la figure (44. (a)) décrit le format verilog de la structure TMR du circuit c17. Elle fait apparaître une structure TMR formée de trois circuits c17 (lignes 4, 5 et 6) appelés T1, T2 et T3. Les sorties des circuits sont les entrées d'un voteur. Les sorties des deux voteurs (lignes 7 et 8) sont les sorties de la structure TMR. Le format verilog des voteurs est décrit par la figure (44. (b)). Un voteur a trois entrées et une sortie. Il est composé de trois portes ET et d'une porte OU à trois entrées : $S_1 = E_1.E_2 + E_1.E_3 + E_2.E_3$.

```
1 module TMRc17 (E1, E2, E3, E4, E5, S1, S2);
2 input E1, E2, E3, E4, E5;
3 output S1, S2;

4 c17 T1 (E1, E2, E3, E4, E5, S1M1, S2M1);
5 c17 T2 (E1, E2, E3, E4, E5, S1M2, S2M2);
6 c17 T3 (E1, E2, E3, E4, E5, S1M3, S2M3);

7voteur V0 (S1M1, S1M2, S1M3, S1);
8 voteur V1 (S2M1, S2M2, S2M3, S2);

9 end module
```

a)

```
1   module voteur (E1, E2, E3, S1);
2   input E1, E2, E3;
3   output S1;

4   and gate_1 (L1, E1, E2);
5   and gate_2 (L2, E1, E3);
6   and gate_3 (L3, E2, E3);
7   or gate_4 (S1, L1, L2, L3);

8   end module
```

b)

Figure 44. Format Verilog (a)- du circuit c17 modifié en TMR
(b)- du voteur

Phase 4 : Création de la liste de faute de collage d'un module

La liste des fautes de collage simple dans un module est générée par le logiciel Tetramax. Cette génération est très rapide donnant automatiquement les fautes équivalentes.

La liste des fautes de collage générée pour le circuit C17 est présentée par la figure 45. Nous remarquons que certaines fautes de collage sont équivalentes. Le symbole(--) représente le fait que la faute est équivalente à celle du dessus dans la liste. Par exemple, les quatre premières fautes sont équivalentes. La liste de faute peut ainsi être réduite en supprimant toutes les fautes marquées du symbole (--) et nous obtenons 20 fautes au lieu des 36 initiales.

1	sa1 NC P_5/Z	19	sa0 NC P_3/Z
2	sa0 -- P_5/A	20	sa1 NC P_3/B
3	sa0 -- P_5/B	21	sa1 NC P_3/A
4	sa0 -- P_1/Z	22	sa1 NC P_3/Z
5	sa1 NC P_5/B	23	sa0 -- P_3/A
6	sa0 NC P_5/Z	24	sa0 -- P_3/B
7	sa1 NC P_1/B	25	sa1 NC P_6/Z
8	sa1 NC P_1/A	26	sa0 -- P_6/A
9	sa1 NC P_1/Z	27	sa0 -- P_6/B
10	sa0 -- P_1/A	28	sa0 -- P_4/Z
11	sa0 -- P_1/B	29	sa1 NC P_6/A
12	sa1 -- P_5/A	30	sa0 NC P_6/Z
13	sa0 NC P_2/Z	31	sa1 NC P_4/B
14	sa1 NC P_2/B	32	sa1 NC P_4/A
15	sa1 NC P_2/A	33	sa1 NC P_4/Z
16	sa1 NC P_2/Z	34	sa0 -- P_4/B
17	sa0 -- P_2/A	35	sa0 -- P_4/A
18	sa0 -- P_2/B	36	sa1 -- P_6/B

Figure 45. Liste de faute de collage simple du circuit c17

Phase 5: Injection d'une faute pivot

La faute pivot est une faute de collage simple injectée par modification de la netlist (ensemble de portes et de connexions) représentée à la figure 46 [Hsu97, Jei09].

```
1   /*collage a 0 de la sortie de P2 */
2   module c17modifie_13 (E1, E2, E3, E4, E5, S1, S2);
3   input E1, E2, E3, E4, E5;
4   output S1, S2;

5   nand P1 (.Z (N1), .A (E1), .B (E3));
6   nand P2 (.Z (Nremp), .A (E3), .B (E4));
7   nand P3 (.Z (N3), .A (E2), .B (N2));
8   nand P4 (.Z (N4), .A (N2), .B (E5));
9   nand P5 (.Z (S1), .A (N1), .B (N3));
10  nand P6 (.Z (S2), .A (N3), .B (N4));

11  assign N2=0;
12  end module
```

Figure 46. Injection d'un S@0 sur la sortie de la porte P_2 du circuit c17.

Pour injecter par exemple la treizième faute P_2/Z (collage à 0 de la sortie Z de la porte P2) de la liste du circuit c17, les entrées des portes reliées à la sortie de P2 portant un nouveau nom (Nremp) sont forcées à 0 grâce à la ligne 11. La sortie de la porte a un nouveau nom Nremp car il n'est pas possible de forcer une sortie de porte à une valeur logique fixe. Cette connexion Nremp n'est reliée à rien d'autre dans le circuit. La Figure 47 illustre le schéma électrique de l'injection de la faute.

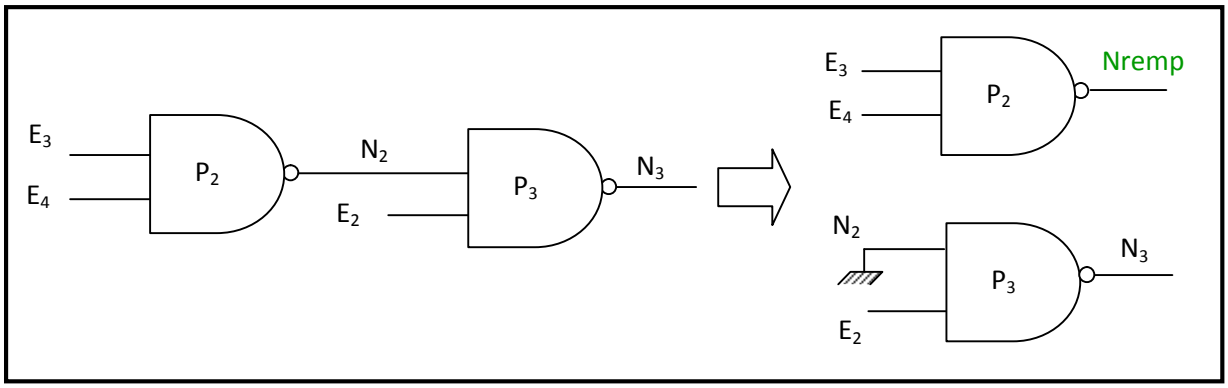


Figure 47. Injection d'une faute à la sortie de la porte NAND P_2

Injecter une faute de collage en sortie d'une porte logique est différent d'injecter une faute de collage en entrée d'une porte. Cela est illustré par un autre exemple des figures 48 et 49. Cette fois, le nom de l'entrée où la faute est injectée est remplacé par Nremp. Il suffit

ensuite de forcer Nremp à "1" logique (ligne 11). Le fil N_3 n'est alors plus relié à la porte P_5 de l'exemple mais peut toujours être relié à l'entrée d'autres portes

```
1   /*collage a 1 de l'entrée B de P5 */
2   module c17modifie_5 (E1, E2, E3, E4, E5, S1, S2);
3   input E1, E2, E3, E4, E5;
4   output S1, S2;

5   nand P1 (.Z (N1), .A (E1), .B (E3));
6   nand P2 (.Z (N2), .A (E3), .B (E4));
7   nand P3 (.Z (N3), .A (E2), .B (N2));
8   nand P4 (.Z (N4), .A (N2), .B (E5));
9   nand P5 (.Z (S1), .A (N1), .B (Nremp));
10  nand P6 (.Z (S2), .A (N3), .B (N4));

11  assign Nremp=1;
12  end module
```

Figure 48. Injection d'un S@1 sur l'entrée B de la porte P_5 du circuit c17

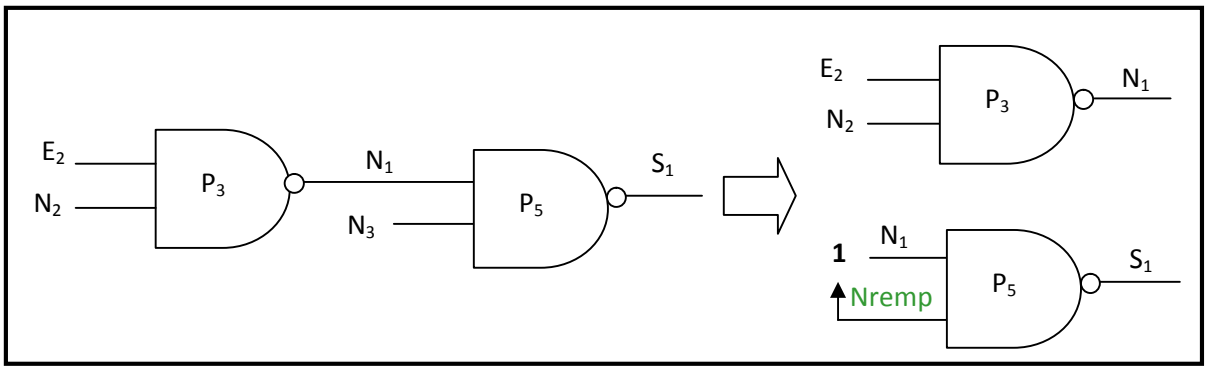


Figure 49. Injection d'une faute à l'entrée de la porte NAND P_5

Phase 6 : Commandes du script Tetramax

Le script Tetramax est crée pour automatiser le test de la structure TMR toute entière. Il est généré grâce à un programme python.

Une partie du script concernant le circuit c17 est montré à la figure 50. C'est le premier des ATPG à lancer traitant toutes les paires de fautes $\{f_1, f_2\}$ avec f_1 = S@1 de P5/Z.

- 1-ligne permet à Tetramax de lire la description Verilog modifiée [Hsu97, Jei09, Lay80, Sew06] du circuit C17 avec une faute pivot représentant le premier circuit de la structure TMR.
- 2-ligne est la description des deux autres circuits.
- 3-ligne décrit la structure TMR toute entière (fig.51).
- 4-ligne décrit les portes logiques utilisées dans la description des circuits.
- 5-6 : lignes informent Tetramax que le test se fait sur le circuit TMR C17.

- Les 16 lignes comprises entre 7 et 22 correspondent aux fautes que l'ATPG va devoir traiter. Quatre fautes sont non testables, donc il ne devrait prendre en compte que 16 paires de fautes au lieu de 20 contenants f_1.
- 23-ligne définit l'option abort limit. Elle représente un temps maximum pour le ciblage d'une paire de fautes. Une fois ce temps expiré cette faute est classée non détectée.
- 24-ligne option-decision NOR signifie que les chemins de propagation utilisés pour détecter la paire de fautes ne sont pas pris aléatoirement (NO Random) et seront toujours les mêmes d'une simulation à l'autre.
- 25-ligne option-norandom fil place des valeurs binaires indifférentes "X" dans les vecteurs de test pour les bits où il n'est pas indispensable de placer un "0" ou un "1" logique. Une fois tous les ATPG lancés, cela permet de réduire le nombre de vecteurs de test.
- 26-ligne lance l'ATPG.
- 27-ligne crée un fichier avec les résultats de l'ATPG.
- 28-ligne réinitialise le logiciel pour l'ATPG suivant.

```
1   read netlist /auto/iscas85/circuits/c17/c17modifie_1.v
2   read netlist /auto/iscas85/circuits/c17/c17.v
3   read netlist /auto/iscas85/circuits/c17/TMRc17.v
4   read netlist /auto/iscas85/circuits/lib_comb.v

5   run build_model TMRc17
6   run drc

7   add faults T2/P5/Z -stuck 1
8   add faults T2/P5/B -stuck 1
9   add faults T2/P5/Z -stuck 0
10  add faults T2/P1/B -stuck 1
11  add faults T2/P1/A -stuck 1
12  add faults T2/P1/Z -stuck 1
13  add faults T2/P2/Z -stuck 0
14  add faults T2/P2/B -stuck 1
15  add faults T2/P2/A -stuck 1
16  add faults T2/P2/Z -stuck 1
17  add faults T2/P3/Z -stuck 0
18  add faults T2/P3/B -stuck 1
19  add faults T2/P3/A -stuck 1
20  add faults T2/P3/Z -stuck 1
21  add faults T2/P6/A -stuck 1
22  add faults T2/P4/A -stuck 1

23  set atpg -abort limit 100
24  set atpg -decision NOR
25  set atpg –norandom _fill
26  run atpg -ndetects 1
27  write patterns /auto/iscas85/circuits/patterns/c17/pat1.txt –
Internal -format verilog_ single _file -parallel 0

28  drc -force
```

Figure 50. Script TetraMax décrivant un ATPG

```
1   module TMR c17 (E1, E2, E3, E4, E5, S1, S2);
2   input E1, E2, E3, E4, E5;
3   output S1, S2;

4   c17modifie_1 T1 (E1, E2, E3, E4, E5, S1M1, S2M1);
5   c17 T2 (E1, E2, E3, E4, E5, S1M2, S2M2);
6   c17 T3 (E1, E2, E3, E4, E5, S1M3, S2M3);

7   voteur V0 (S1M1, S1M2, S1M3, S1);
8   voteur V1 (S2M1, S2M2, S2M3, S2);

9   end module
```

Figure 51. Description Verilog de la structure TMR

Phase 7 : Lancement des ATPG

L'étape de lancement de la totalité des ATPG est l'étape la plus longue. Elle peut varier d'une dizaine de secondes pour le circuit C 17 à plusieurs jours pour de plus gros circuits, tel que le C 7552.

Phase 8 : Analyse et synthèse des résultats

La figure 52 montre les résultats de L'ATPG du circuit c17.

```
// Module tested: TMRc17

//     Uncollapsed Stuck Fault Summary Report
// -----------------------------------------------
// fault class                     code   #faults
// ------------------------------  ----  ---------
// Detected                         DT          7
// possibly detected                PT          0
// Undetectable                     UD          9
// ATPG untestable                  AU          0
// Not detected                     ND          0
// -----------------------------------------------
// total faults                                16
// test coverage                           100.00%
// -----------------------------------------------
//
//            Pattern Summary Report
// -----------------------------------------------
// #internal patterns                           4
//     #basic scan patterns                     4
// -----------------------------------------------
```

Figure 52. Résultat d'un ATPG.

Les paires de fautes sont classées en cinq catégories :

- Fautes «DT», l'ATPG a trouvé un vecteur de test permettant de la tester donc la paire de fautes n'est pas tolérée.
- Fautes éventuellement détectées « PT », la réponse en sortie pour détecter la faute conduit à une valeur indéterminée X. Durant tous les ATPG réalisés, aucune faute n'a été classée dans cette catégorie.
- Fautes indétectables «UD», intestables donc tolérées par la structure TMR.
- Fautes "ATPG untestable" « AU », indétectables par l'ATPG mais pas forcément intestables.
- Fautes non détectées « ND », l'ATPG n'a pas eu le temps de les classer. Pour diminuer leur nombre, le paramètre abort-limit peut être augmenté.

Sur les 16 paires de fautes traitées par l'ATPG, 7 sont détectées « non tolérées » par la structure TMR et 9 sont indétectables donc « tolérées ». La probabilité T qu'une paire de fautes soit tolérée par la structure peut ainsi être déterminée :

T= Nombre de paires de fautes tolérées /Nombre de fautes globales = 9/16=56.25%

La figure 53 représente un résumé des principales phases pour le test des circuits « benchmarks ISCAS85 et ITC99 » simulés sous fautes de collage simultanées multiples. Un exemple de procédure de génération de vecteurs de test plus détaillé est donné dans l'annexe B pour le circuit c432.

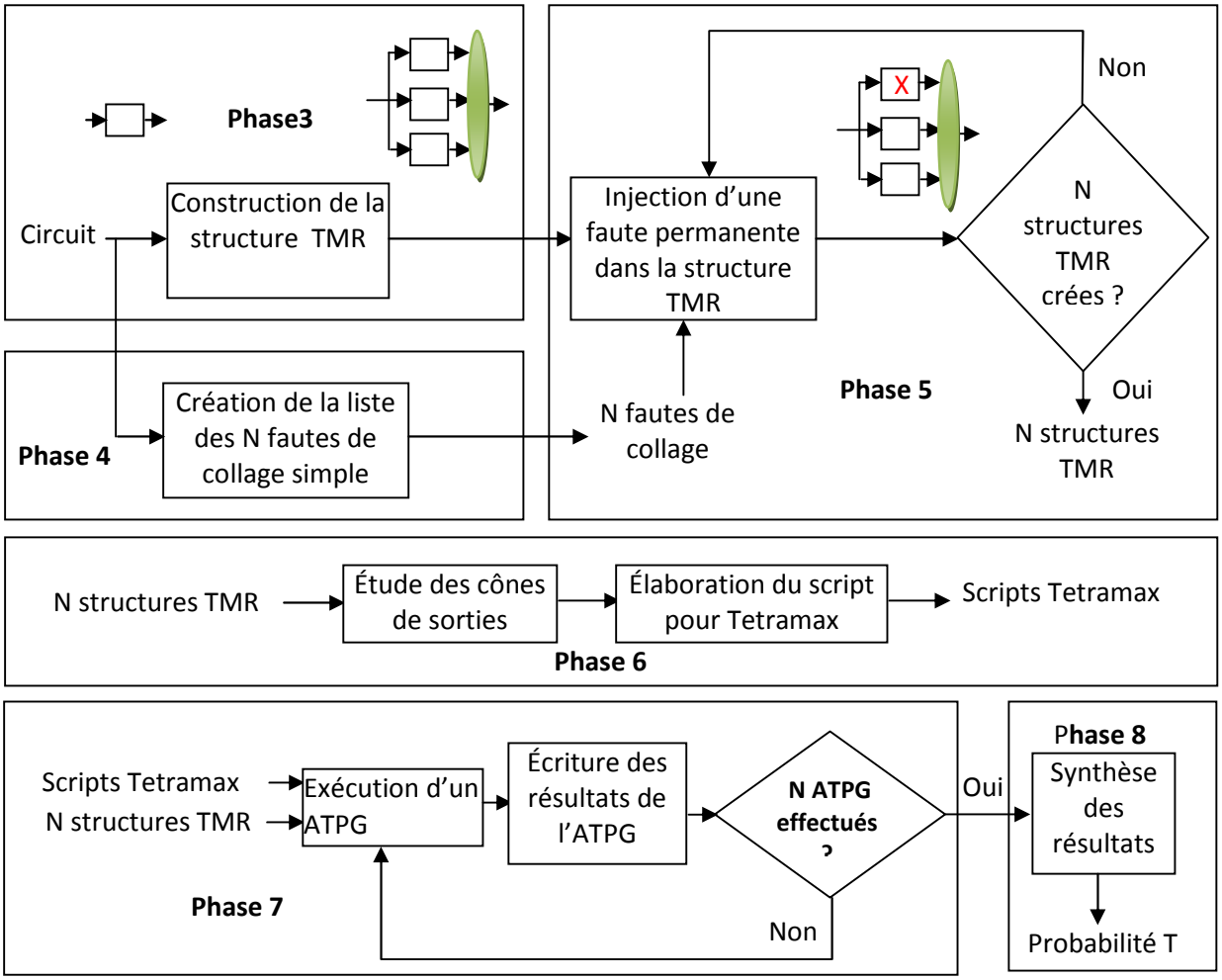


Figure 53. Résumé des phases principales de l'ATPG

III.2.2-Interprétation des résultats

Pour analyser l'impact de la réalisation de structures TMR sur le rendement des circuits intégrés et leur fiabilité, nous avons aussi exécuté la procédure de génération de vecteurs de test sur une partie des « benchmarks ISCAS85 et ITC99 ».

III.2.2.1-Caractéristiques des circuits simulés

Le tableau 9 montre les caractéristiques [Lay80, Sit03, Sit05] de chacun des circuits testés. Le coût en surface A_0 [Gag97, Kar98, Kar97, Sit02] de la structure TMR dépend de la taille du circuit cloné et de la taille du voteur. Il est supérieur ou égal à 3, il est égal à 3 si le voteur a une surface nulle. Il peut être très faible (3,03 pour le circuit c6288) ou très élevé (3,68 pour le circuit b06). Le coût en surface est une donnée très importante puisqu'il intervient largement dans la capacité d'augmentation du rendement de fabrication par la réalisation de la structure TMR. En effet, pour qu'une structure TMR puisse augmenter le rendement de fabrication, le rendement du circuit initial doit être inférieur à $1/A_O$.

Nom du Circuit simulé		E/S P_I/P_O	Nombre de portes	Nombre de fautes de collage	Nombre de paires de fautes	Nombre réduit de paires de fautes traitées par l'ATPG	A_O (coût en surface)
Benchmarks ISCAS85	c432	36/7	160	392	689725	75.10^3	3.10
	c499	41/32	202	486	1062153	95.10^3	3.39
	c1908	33/25	880	1826	15001503	$1.3.10^6$	3.08
	c2670	233/140	1193	2852	36598290	$1.87.10^6$	3.33
	c3540	50/22	1669	3438	53184141	$4.91.\ 10^6$	3.04
	c5315	178/123	2307	4970	111146595	$3.4.\ 10^6$	3.16
	c6288	32/32	2416	6250	175771875	$18.2.10^6$	3.03
	c7552	207/108	3512	7438	248946141	$7.40.\ 10^6$	3.09
Benchmarks ITC99	b02	1/1	25	64	18336	$1.5.\ 10^3$	3.47
	b03	4/4	150	382	656085	$290.\ 10^3$	3.57
	b04	11/8	480	1477	9814665	$535.\ 10^3$	3.32
	b05	1/36	608	2553	29326311	$1.17.10^6$	3.16
	b06	2/6	66	155	107880	$7.73.10^3$	3.68
	b07	1/8	382	1120	5643120	$399.\ 10^3$	3.38
	b09	1/1	131	417	781875	$57.3.\ 10^3$	3.45
	b10	11/6	172	468	984906	$63.5.\ 10^3$	3.31
	b11	7/6	366	1308	7696926	$703.\ 10^3$	3.19
	b12	5/6	1000	2777	34698615	$857.\ 10^3$	3.30
	b13	10/10	309	835	3136260	$59.6.\ 10^3$	3.49

Tableau 9. Caractéristiques des circuits testés [Gag97, Kar98, Kar97, Lay80, Sit02, Sit03, Sit05]

III.2.2.2- Tolérance aux fautes des structures TMR

Le tableau 10 représente les résultats des ATPG, deux points peuvent être déduits :

- Le taux de paires de fautes non testables et donc tolérées représente la tolérance aux fautes de la structure TMR. Il est égal à la probabilité T qu'une paire de fautes soit tolérée. Ce pourcentage varie de 38.03% pour le circuit c6288 à 96,96% pour le circuit b13. Plus la probabilité T est élevée, plus la structure TMR est tolérante.

- L'efficacité du test idéal est égale à 100%. L'ATPG concernant la structure TMR à base de circuits c432 n'a pas réussi à évaluer 25,70 % des paires de fautes. Pour améliorer l'efficacité de l'évaluation de la tolérance aux fautes de la structure TMR, il faut soit donner plus de temps à l'ATPG [Fuj83] (sans garantie pour l'amélioration de l'efficacité), soit choisir un autre ATPG plus puissant et spécialisé dans les structures redondantes (VERIFAULT, MTBF-CARE) [Bqr09].

Circuit		A_O		Résultats globaux			
			$1/A_O$	Fautes intestables « tolérées » T	Fautes Détectées « non-tolérées »	Efficacité de l'ATPG «fautes tolérées +fautes non tolérées»	Nombre de vecteurs de test pour «FD »
Benchmarks ISCAS85	c432	3.10	32,25%	40.00%	34.30%	74.30%	912
	c499	3.39	29,49%	52.73%	34.73%	87.46%	2450
	c1908	3.08	32,46%	56.45%	36.40%	92.85%	3852
	c2670	3.33	30,03%	75.96%	17.50%	93.46%	4987
	c3540	3.04	32.89%	54.09%	23.75%	77.84%	1756
	c5315	3.16	31,64%	93.20%	3.75%	96.95%	642
	c6288	3.03	33,00%	38.03%	54.78%	92.81%	4532
	c7552	3.09	32,36%	84.93%	11.70%	96.63%	8745
Benchmarks ITC99	b02	3.47	28,81%	86.37%	13.63%	100%	12
	b03	3.57	28,01%	87.93%	12.06%	99.99%	114
	b04	3.32	30,12%	84.30%	12.72%	97.02%	3390
	b05	3.16	31,64%	88.66%	8.97%	97.63%	4699
	b06	3.68	27,17%	87.50%	12.50%	100%	16
	b07	3.38	29,58%	81.90%	16.01%	97.91%	1635
	b09	3.45	28,98%	83.07%	16.93%	100%	211
	b10	3.31	30,21%	89.40%	10.59%	99.99%	149
	b11	3.19	31,34%	74.50%	21.16%	95.66%	1551
	b12	3.30	30,30%	95.46%	4.08%	99.54%	1274
	b13	3.49	28,65%	96.96%	3.04%	100%	167

Tableau 10. Résultats de l'ATPG

III.2.2.3-Impact sur la fiabilité et le rendement

Le tableau 11 résume les résultats des ATPG lancés, trois points sont déduits :

- Aucun des circuits étudiés en transformant sa structure en TMR n'est apte à satisfaire la condition $T > T_{min}$, et ce malgré l'utilisation de la loi de Poisson plus favorable pour améliorer le rendement de fabrication car conduisant à des valeurs T_{min} inférieures. Les structures TMR ne sont pas suffisamment tolérantes pour compenser leur coût en silicium. par conséquent, la fiabilité et le rendement ne sont pas améliorés.

- Le symbole NA (cases notées non Applicable) signifie que le coût en surface est très élevé et ce, même si 100% des paires de fautes étaient tolérées. Par exemple, pour le circuit b06, quelque soit la loi mathématique de répartition des défauts, augmenter la fiabilité et le rendement ne sera jamais possible avec les structures TMR.
- La surface du voteur dépend du nombre de sorties de circuits. Plus il y a de sorties à voter, plus la taille du voteur sera importante. Dans le chapitre 1, nous avons même remarqué que si la taille du voteur était trop importante, la réalisation de structures TMR ne permettait pas d'augmenter le rendement (lorsque Ao> 3,8 pour la loi de Poisson). Pour montrer l'importance de la taille du voteur, prenons l'exemple du circuit c 3540 avec une taille du voteur très petite par rapport à la taille de ses circuits puisque son coût en surface est 3,04. La probabilité T_{min} pour ce circuit est 93,35%. Par comparaison, le circuit b06 a une taille de voteur très importante puisque le coût en surface de la structure TMR est 3,68. Sa probabilité T devra être supérieure à 99,56% pour pouvoir augmenter le rendement. Par conséquent, le circuit c3540 est beaucoup plus susceptible de pouvoir vérifier $T > T_{min.}$

Circuit		A_O	T_{min}				Tolérance (T)	Fiabilité (R)
			Loi de Poisson	Loi Binomiale Négative				
				α=10	α=3	α=1		
« ISCAS85 »	c432	3,10	94,42%	95,04%	97,09%	99,70%	40,00%	35,2%
	c499	3,39	97,90%	98,71%	99,76%	NA	52,73%	54,09%
	c1908	3,08	94,07%	94,66%	96,62%	99,58%	56,45%	59,62%
	c2670	3,33	97,44%	98,21%	99,54%	NA	75,96%	85,44%
	c3540	3,04	93,35%	93,79%	95,43%	99,09%	54,09%	56,12%
	c5315	3,16	95,35%	96,13%	98,10%	99,95%	93,20%	98,67%
	c6288	3,03	93,16%	93,55%	95,03%	98,88%	38,03%	32,38%
	c7552	3,09	94,25%	94,83%	96,86%	99,65%	84,93%	93,87%
« ITC99 »	b02	3,47	98,50%	99,21%	99,93%	NA	86,37%	94,93%
	b03	3,57	99,09%	99,67%	NA	NA	87,93%	95,98%
	b04	3,32	97,32%	98,12%	99,49%	NA	84,30%	93,37%
	b05	3,16	95,33%	96,12%	98,10%	99,93%	88,66%	96,43%
	b06	3,68	99,56%	NA	NA	NA	87,50%	95,70%
	b07	3,38	97,80%	98,63%	99,73%	NA	81,90%	91,35%
	b09	3,45	98,36%	99,10%	99,90%	NA	83,07%	92,37%
	b10	3,31	97,21%	98,03%	99,44%	NA	89,40%	96,86%
	b11	3,19	95,77%	96,57%	98,48%	NA	74,50%	83,80%
	b12	3,30	97,10%	97,90%	99,39%	NA	95,46%	99,40%
	b13	3,49	98,04%	99,32%	99,96%	NA	96,96%	99,72%

Tableau 11. Valeurs de « T_{min} » et « R » pour les circuits ISCAS85 et ITC99

III.2.2.4- Impact du Coût en surface de silicium sur la probabilité de tolérance T

Sur la figure 54, représentant l'impact du coût de la réalisation de la TMR sur la probabilité de tolérance T, nous remarquons que de nombreux circuits ont une probabilité T proche de la courbe représentant la valeur T_{min} en fonction du coût en surface (A_O) calculée avec la loi de Poisson. Pour que la réalisation de structures TMR permette d'augmenter le rendement d'un circuit, les circuits doivent être au-dessus de la courbe ($T > T_{min}$) ce qui n'est pas le cas. Par conséquent, il suffit de rendre la structure TMR un peu plus redondante, ce qui signifie plus tolérante.

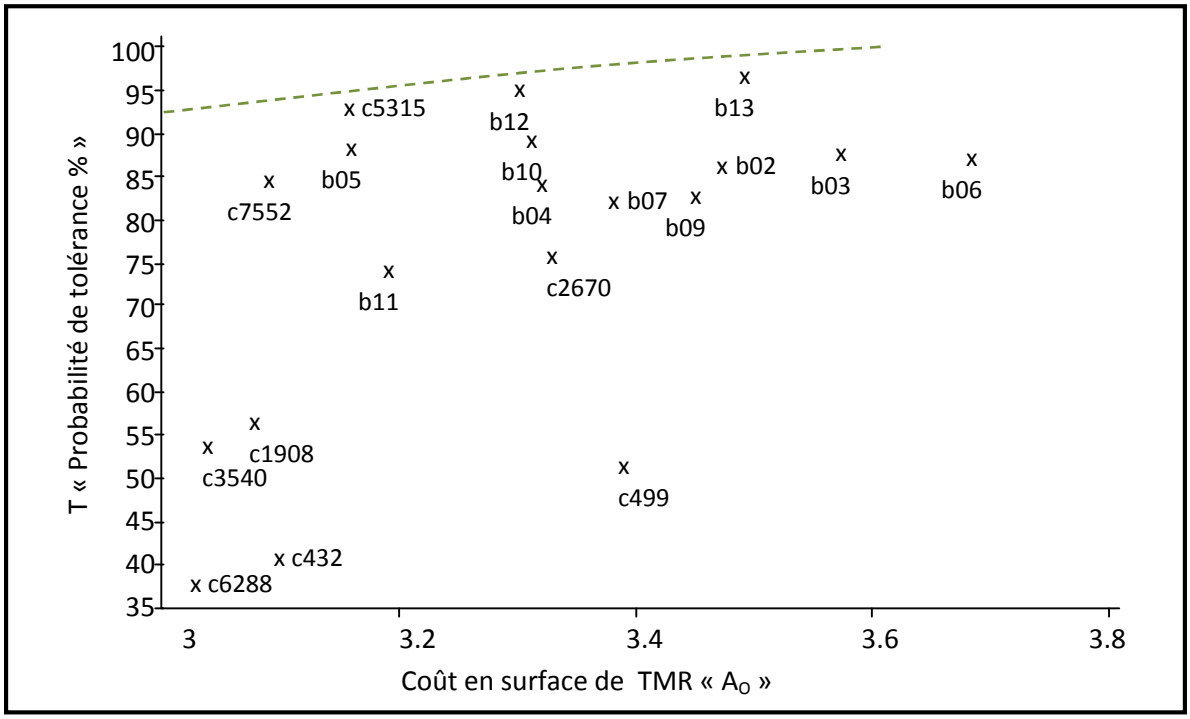


Figure 54. Tolérance aux fautes des circuits en fonction du coût A_O

En résumé, la tolérance T aux défauts d'une structure TMR doit être suffisamment importante pour compenser le coût en surface associé à la réalisation de cette structure TMR. Pour que les structures TMR permettent d'augmenter la fiabilité via l'augmentation du rendement de fabrication, d'autres solutions doivent être trouvées. Deux pistes sont envisageables [Haf90]:

- Diminuer le coût en surface de silicium. autrement dit, utiliser un autre type de structure tolérante aux fautes.
- Utiliser la méthode du partitionnement pour renforcer la redondance des structures TMR. C'est cette deuxième méthode que nous avons choisi de mettre en œuvre.

III.3- Optimisation de la structure TMR

La conception de circuits intégrés est une tâche de plus en plus complexe, car le nombre de composants de ces circuits augmente régulièrement. Le principe du

partitionnement d'un circuit intégré est de rassembler les différents éléments d'un circuit en sous circuits, ou cellules, tels que le nombre de connexions entre ceux-ci soit minimal. Chaque sous circuit peut alors être réalisé indépendamment, ce qui rend la conception et l'intégration du circuit plus rapide [Edm07].

III.3.1- Partitionnement de la structure TMR

La structure TMR est modifiée en partitionnant chaque circuit en deux ou trois partitions. Un voteur est placé entre chaque partition. La Figure 55 montre les trois structures. TMR Simple (figure 55.a), TMR Double, dans laquelle les circuits ont été divisés en deux parties (figure 55.b) et TMR Triple, dans laquelle les circuits ont été divisés en trois parties (figure 55.c)).

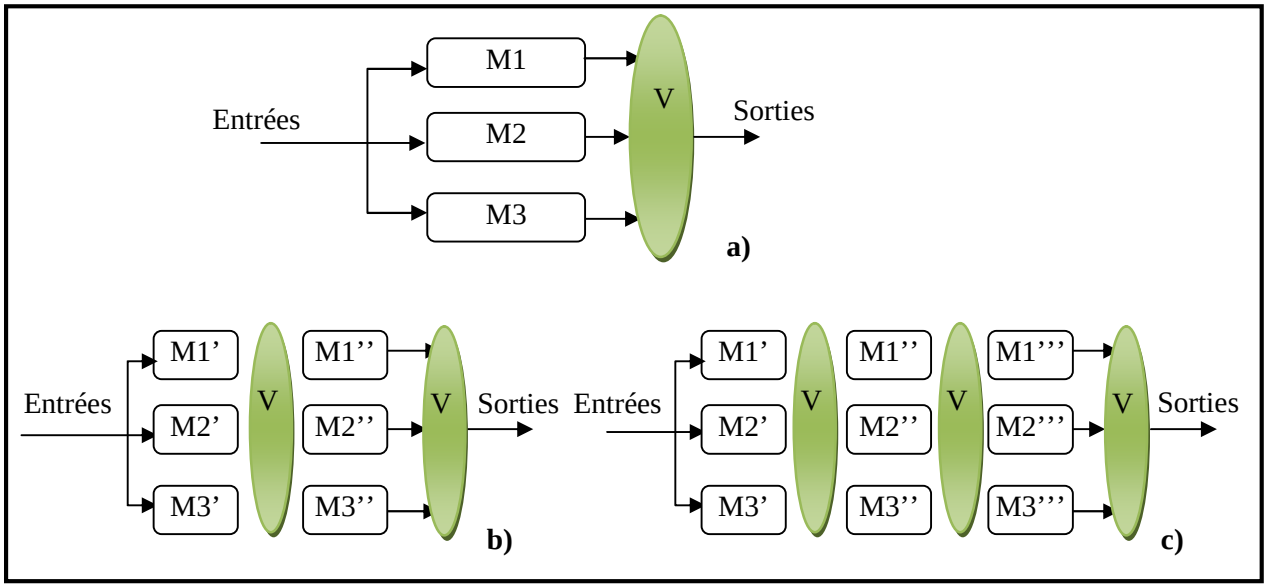


Figure 55. Partitionnement de la structure TMR avec
a) TMR Simple, b) TMR Double et c) TMR Triple

III.3.2- Effets du partitionnement

Le partitionnement a deux effets positifs sur la tolérance aux fautes. Premièrement, il permet de diminuer la profondeur combinatoire des circuits [Edm07, Lau07]. Cela réduit le nombre de chemins de propagation des erreurs. Ainsi, chaque faute n'affecte qu'un nombre réduit de sorties de circuits et la probabilité que deux fautes propagent une erreur jusqu'à deux sorties de circuits identiques est plus faible. Deuxièmement, il rend indépendante chaque partition. Ainsi, un défaut de fabrication dans une partie du partitionnement n'a pas d'impact sur une autre partie du partitionnement.

III.3.2.1-Réduction de la profondeur combinatoire

La profondeur combinatoire d'un circuit est le nombre maximum de portes logiques que le signal électrique peut traverser entre une entrée et une sortie primaire. Par exemple, la comparaison des deux structures TMR données en figure 56, montre que les circuits de la structure de la figure 56-b ont une profondeur combinatoire plus faible, ce qui implique des chemins de propagation des erreurs moins nombreux et des cônes de sorties de faute plus petits et différents. Par conséquent, il y a plus de paires de fautes qui sont structurellement non testables et la tolérance aux fautes de la structure TMR augmente. Il est à noter que, les circuits benchmarks ITC99 ont des profondeurs combinatoires plus petites « 19 couches logiques » que les circuits benchmarks ISCAS85 « 43 couches logiques », d'où une meilleure tolérance aux fautes pour les structures TMR des circuits benchmarks ITC99.

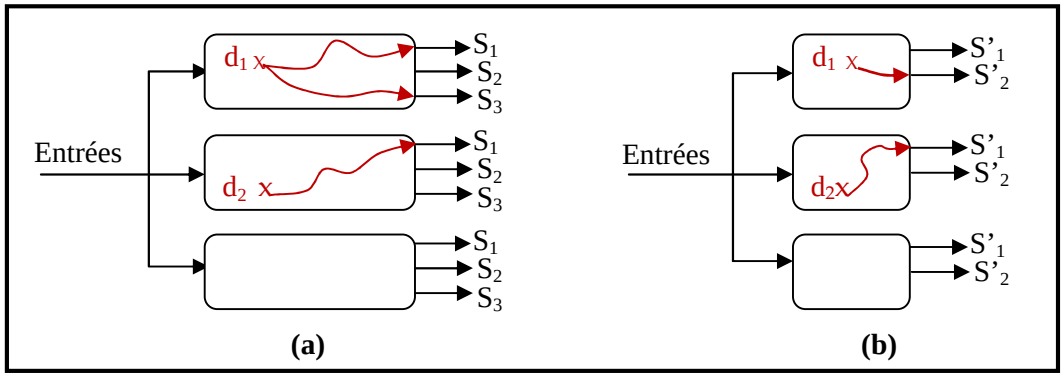


Figure 56. Impact de la profondeur combinatoire sur la propagation des fautes [Lau09]

III.3.2.2-Indépendance des différentes partitions

Lorsque la structure TMR est partitionnée en deux ou trois partitions, toutes les partitions sont indépendantes les unes des autres [Edm07]. Ainsi deux défauts présents dans deux partitions différentes sont forcément tolérés. Par conséquent, un grand nombre de paire de fautes correspondant à deux défauts de fabrication sont tolérées.

Pour rendre optimal le nombre de paires de fautes indétectables par la méthode de partitionnement, les circuits doivent être partitionnés en p partitions de tailles identiques, ainsi leur tolérance aux faute T(%) est toujours supérieure à [Haf90]:

T > 100 × ——

III.3.3- Description du problème du partitionnement

Le problème du partitionnement d'un circuit doit satisfaire deux conditions [Haf90] :

- La taille de chaque partition doit être de taille équivalente pour rendre maximale la probabilité T ;
- Le nombre de coupures doit être minimal pour limiter la surface additionnelle. En effet, une coupure de ligne représente la sortie d'une partition et l'entrée d'une autre.

Entre les deux se trouve un voteur. Le nombre de coupures est donc égal aux nombre de voteurs rajoutés à la structure TMR.

La figure 57 résume le principe du partitionnement, il s'effectue en trois phases :

- Modélisation du circuit sous forme d'un graphe, où les portes du circuit sont représentées par les sommets du graphe et les lignes du circuit par des arcs. Plusieurs types de graphes permettent de modéliser les circuits. Le problème est de déterminer celui qui permet d'effectuer un partitionnement qui répond au mieux au deux conditions citées ci-dessus.
- Partitionnement du graphe représentant le circuit. Il existe de nombreux algorithmes traitant de ce problème. L'algorithme qui convient le mieux d'un point de vue du temps CPU et du nombre de coupures minimal est l'algorithme multi-niveaux [Edm07] représenté à la figure 58. Il contient trois différents algorithmes (annexe B).
- Insertion de voteurs, de sorte que le nombre de voteurs rajoutés à la structure TMR soit proportionnel aux nombre de coupures.

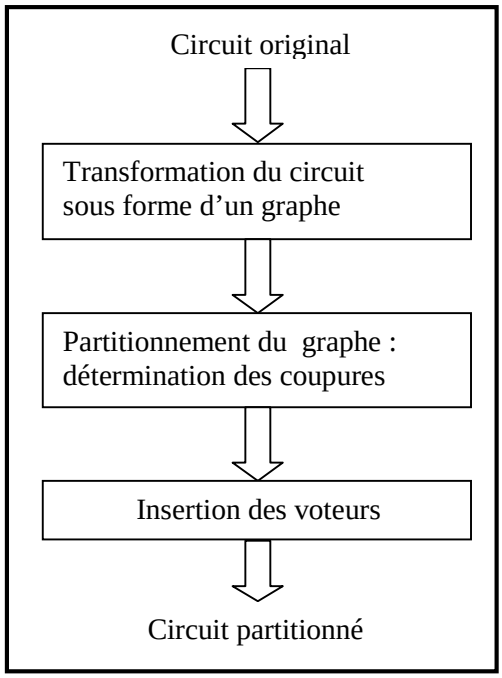


Figure 57. Principe de partitionnement

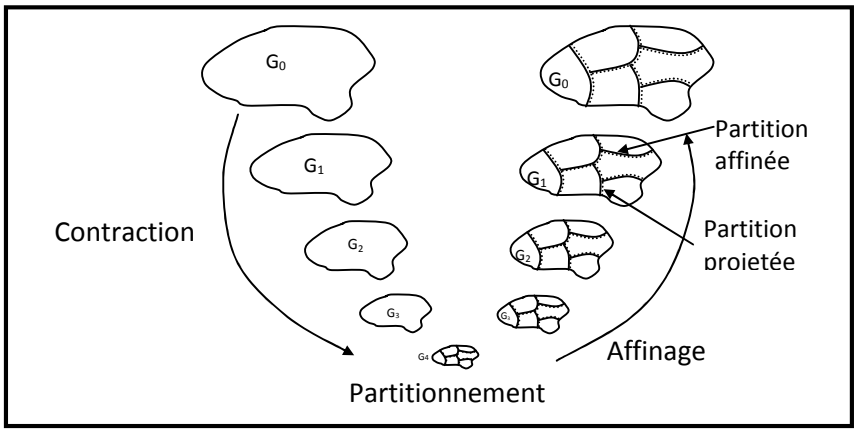


Figure 58. Les trois phases de l'algorithme multi-niveaux [Edm07]

Un circuit VLSI est caractérisé par sa liste de cellules et sa liste de connexions (netlist), il peut être modélisé soit par un graphe, soit par un hypergraphe bien que la modélisation sous forme d'hypergraphes soit plus courante [Edm07]. Ce dernier est la généralisation d'un graphe où l'ensemble des arcs est remplacé par un ensemble d'hyperarcs. Un hyperarc est une extension de la notion d'arcs, de sorte que plus de deux sommets peuvent être connectés par un hyperarc.

La Figure 59 montre un exemple de partition d'un circuit logique sous forme d'un graphe et d'un hypergraphe. Le partitionnement d'un hypergraphe donne de meilleures solutions, en termes de nombre de coupures, que le partitionnement d'un graphe [Edm07]. En effet, la représentation par hypergraphe (figure 60) permet de gérer les arcs divergents d'un

sommet durant la phase de partitionnement d'un point de vue du temps CPU et du nombre de coupures minimal.

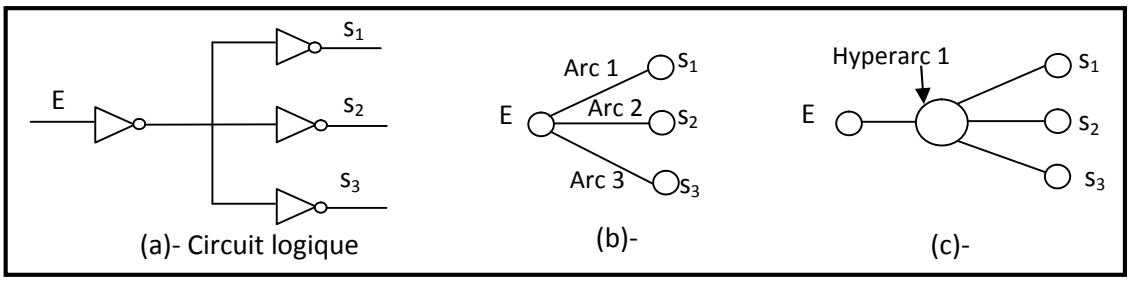


Figure 59 : Représentation d'un circuit sous forme d'un graphe (b) et d'un hypergraphe (c) [Kar98, Kar97]

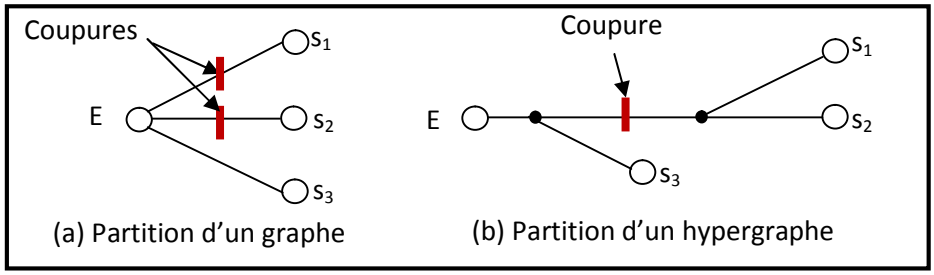


Figure 60. Partition d'un graphe et d'un hypergraphe [Kar98, Kar97]

III.3.3.1- Principe de partitionnement avec le circuit c17

La figure 61 illustre le partitionnement du circuit c17. Comme ce circuit a deux sorties, le nombre de voteur de la structure TMR simple est deux (V_1, V_2). Le partitionnement réalisé à l'aide de l'outil sh-METIS [Kar98, Kar97, Sit02] produit deux partitions de même taille (3 portes logiques dans chaque partition) pour deux coupures. Deux voteurs additionnels (V_3, V_4) sont donc rajoutés. Un total de quatre voteurs est donc requis pour la structure TMR Double.

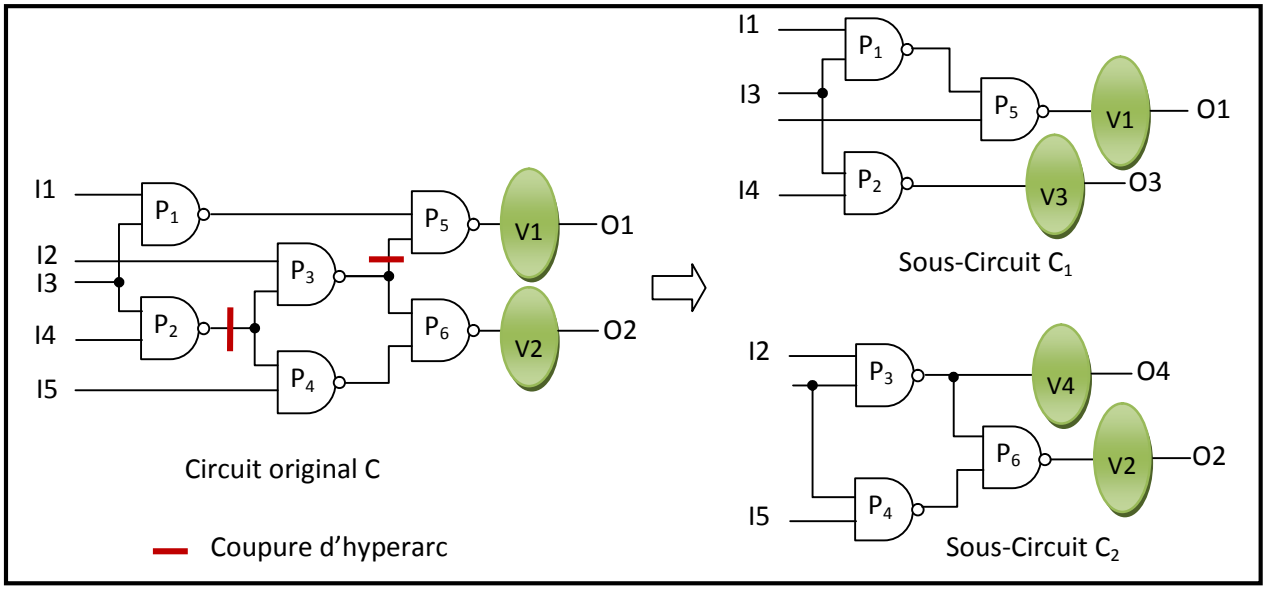


Figure 61. Le circuit c17 partitionné [Kar98, Kar97, Sit02]

III.3.4- Résultats du partitionnement

Dans cette partie, nous étudions les résultats des structures TMR Simple, TMR Double et TMR Triple pour voir si leur réalisation permet d'augmenter le rendement de fabrication et la fiabilité des circuits, grâce aux résultats donnés par Sh-METIS et à un programme élaboré pour transformer les structures TMR simple en structures TMR Double et TMR Triple « option dans le logiciel Tetramax ».

III.3.4.1- Caractéristiques des structures TMR partitionnées

Le tableau 12 représente les caractéristiques des structures TMR partitionnées

Par exemple, le circuit c2670 est caractérisé par :

- 140 voteurs pour la structure TMR simple
- 160 voteurs pour la structure TMR double avec un coût en surface de 1.40% ;

Le partitionnement en deux parties équivalentes exige 20 coupures et donc 20 voteurs à rajouter (140+20=160).

- 179 voteurs pour la structure Triple représentant un coût en surface de 2.72% par rapport à la structure TMR simple.

  Le partitionnement en trois parties équivalentes nécessite 39 coupures et donc 39 voteurs à rajouter (179-140 = 39).

Nous constatons que le coût en surface du partitionnement de la structure TMR est faible pour les plus gros circuits. En effet, moins de 2% pour les plus gros circuits ISCAS85 et moins de 5% pour les plus gros circuits ITC99. Ce faible coût en surface constitue un point positif pour la probabilité T_{min} de chaque structure TMR partitionnée. En effet T_{min} augmente moins, satisfaisant alors la condition « $T > T_{min}$ » recherchée.

Circuit	TMR Simple	TMR Double		TMR Triple	
	Nombre de Voteur	Nombre de Voteur	Coût en surface	nombre de Voteur	Coût en surface
c432	7	29	10.55%	38	14.86%
c499	32	49	6.14%	59	9.75%
c1908	25	53	3.03%	68	4.65%
c2670	140	160	1.40%	179	2.72%
c3540	22	56	1.90%	95	4.09%
c5315	123	149	1.05%	161	1.53%
c6288	32	49	0.58%	64	1.10%
c7552	108	134	0.69%	158	1.32%
b02	5	8	8.15%	9	10.87%
b03	34	43	4.25%	47	6.14%
b04	74	99	3.23%	109	4.52%
b05	60	73	1.08%	91	2.57%
b06	15	22	8.64%	26	13.58%
b07	57	80	4.58%	84	5.38%
b09	29	40	4.96%	44	6.77%
b10	23	36	5.28%	50	10.97%
b11	37	76	6.14%	94	8.97%
b12	127	139	0.85%	167	2.84%
b13	63	66	0.67%	70	1.56%

Tableau 12. Résultats du partitionnement [[Kar98, Kar97, Sit02]

III.3.4.2- Impact sur la fiabilité et le rendement

La procédure de test ATPG effectuée sur les structures TMR simple, TMR Double et TMR Triple nous donne les résultats présentés sur le Tableau 13. Nous avons utilisé la loi de Poisson pour le calcul de T_{min} , en considérant deux cas :

1er Cas : voteurs non robustes réalisés avec la même densité de défauts que les modules.

2ème Cas : voteurs robustes réalisés avec des technologies de fabrication plus matures.

- Dans le premier cas, la condition $T > T_{min}$ est vérifiée pour trois structures TMR Triple « c1908, c5315, c7552 ». Ainsi, la réalisation de TMR Triple peut dans certains cas améliorer le rendement de fabrication et la fiabilité des circuits.
- Dans le deuxième cas, les cases en vert montrent les situations où la condition $T > T_{min}$ est vérifiée, à savoir :

  3 circuits sur 19 pour les structures TMR simple

  7 circuits sur 19 pour les structures TMR double

  17 circuits sur 19 pour les structures TMR Triple avec de très hauts degrés de fiabilité atteignant des valeurs de R= 99,86 pour le circuit b13, et R=99,85 pour le circuit b12.

Ceci nous permet de conclure que le partitionnement améliore grandement la tolérance aux fautes des circuits et par conséquent le rendement de fabrication et la fiabilité.

III.3.4.3 Impact du Coût en surface sur la probabilité T

Pour voir l'impact du coût en surface sur la probabilité T, nous avons reporté les résultats obtenus dans le tableau 13, sur la figure 62 où chaque circuit est représenté par une croix dont les coordonnées sont (A_O, T). Les circuits ont été nommés avec la lettre D ou T pour représenter les structures TMR Double et TMR Triple.

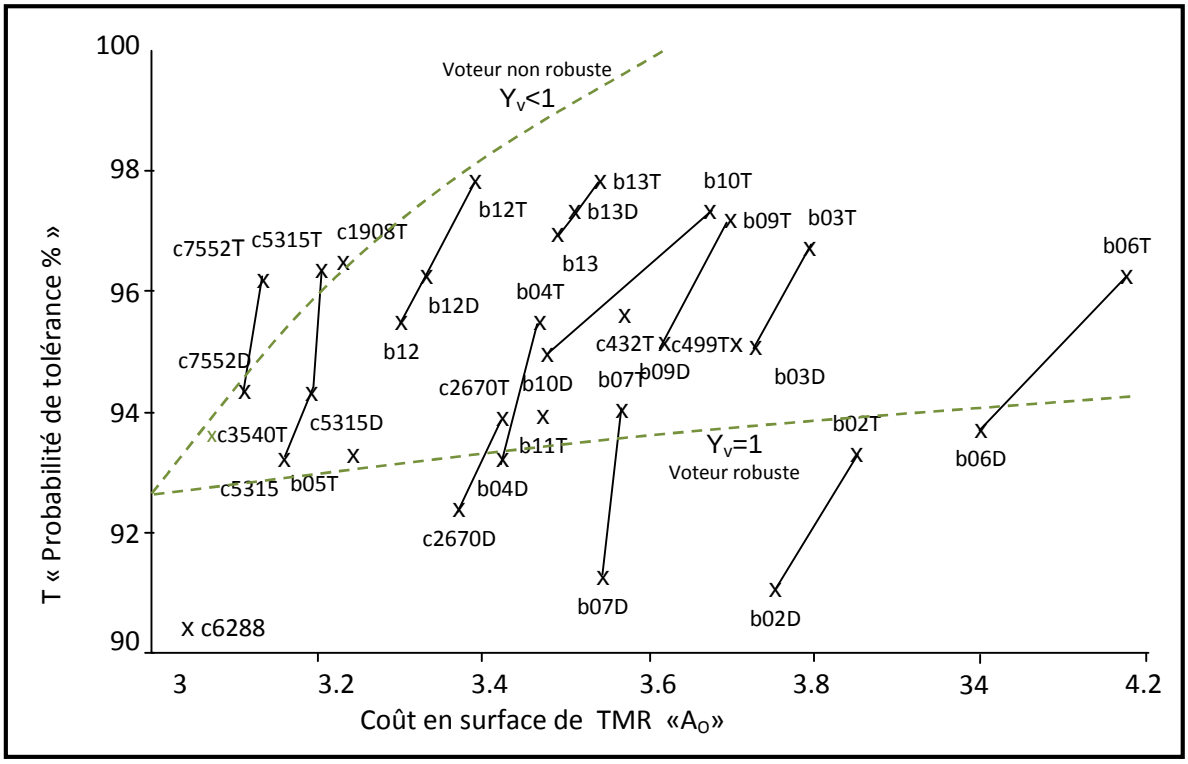


Figure 62. Fiabilité et rendement améliorés grâce au partitionnement

Comme nous pouvons le voir sur cette figure, le partitionnement améliore la tolérance aux fautes d'une structure TMR et la majorité des circuits testés ont leurs caractéristiques comprises entre les courbes correspondant aux voteurs robustes et non robustes. Cependant, si des efforts de conception sont entrepris pour rendre le plus robuste possible les voteurs, la courbe réelle T_{min} correspondant à ces efforts se situerait entre les deux cas extrêmes présentés par la Figure 62 (dans le premier cas, aucun effort n'a été fait pour améliorer le rendement des voteurs ; dans le second cas, les voteurs sont toujours sains et ont un rendement égal à 1). Dans le deuxième cas extrême où le rendement des voteurs est égal à 100%, les structures TMR peuvent même augmenter le rendement pour tous les circuits étudiés.

Circuit	TMR Simple					TMR Double					TMR Triple						
		Voteurs Non robustes		Voteurs robustes			Voteurs Non robustes		Voteurs Robustes			Voteurs Non robustes			Voteurs robustes		
	A_O	T_{min} (%)	T (%)	T_{min} (%)	T (%)	A_O	T_{min} (%)	T (%)	T_{min} (%)	T (%)	A_O	T_{min} (%)	T (%)	R (%)	T_{min} (%)	T (%)	R (%)
c432	3,10	94,42	40,00	92,79	40,00	3,43	98,52	87,03	93,38	87,03	3,57	99,61	95,53	88,48	93,59	95,53	99,41
c499	3,39	97,90	52,73	93,32	52,73	3,60	99,86	89,17	93,63	89,17	3,72	NA	95,11	88,38	93,79	95,11	99,30
c1908	3,08	94,07	56,45	92,75	56,45	3,18	95,60	89,49	92,95	89,49	3,23	96,29	96,50	88,68	93,05	96,50	99,64
c2670	3,33	97,44	75,96	93,22	75,96	3,37	98,83	92,46	93,29	92,46	3,42	98,35	93,90	88,04	93,37	93,90	98,93
c3540	3,04	93,35	54,09	92,67	54,09	3,10	94,40	88,36	92,79	88,36	3,16	95,36	93,79	88,01	92,91	93,79	98,89
c5315	3,16	95,35	93,20	92,90	93,20	3,19	95,75	94,30	92,97	94,30	3,20	95,90	96,38	88,65	92,99	96,38	99,61
c6288	3,03	93,16	38,03	92,65	38,03	3,05	93,53	76,35	92,69	76,35	3,07	93,93	84,14	82,99	92,73	84,14	93,25
c7552	3,09	94,25	84,93	92,77	84,93	3,11	94,57	94,40	92,82	94,40	3,13	94,85	96,22	88,62	92,86	96,22	99,58
b02	3,47	98,50	86,37	93,44	86,37	3,75	NA	91,09	93,83	91,09	3,85	NA	93,28	87,84	93,95	93,28	98,70
b03	3,57	99,09	87,93	93,59	87,93	3,73	NA	95,07	93,80	95,07	3,79	NA	96,75	88,72	93,88	96,75	99,68
b04	3,32	97,32	84,30	93,20	84,30	3,42	98,33	93,19	93,37	93,19	3,47	98,79	95,53	88,48	92,44	95,53	99,41
b05	3,16	95,33	88,66	92,91	88,66	3,19	95,70	89,09	92,97	89,09	3,24	96,41	93,30	87,85	93,06	93,30	98,71
b06	3,68	99,56	87,50	93,74	87,50	4,00	NA	93,68	94,11	93,68	4,18	NA	96,26	88,63	94,30	96,26	99,59
b07	3,38	97,80	81,90	93,30	81,90	3,54	99,40	91,23	93,55	91,23	3,57	99,66	94,11	88,11	92,59	94,11	99,00
b09	3,45	98,36	83,07	93,41	83,07	3,62	NA	95,15	93,66	95,15	3,69	NA	97,13	88,78	93,75	97,13	99,75
b10	3,31	97,21	89,40	93,19	89,40	3,48	98,88	94,92	93,46	94,92	3,67	NA	97,34	88,81	94,73	97,34	99,79
b11	3,19	95,77	74,50	92,97	74,50	3,38	97,94	87,75	93,30	87,75	3,47	98,77	93,91	88,04	93,40	93,91	98,93
b12	3,30	97,10	95,46	93,17	95,46	3,33	97,40	96,27	93,22	96,27	3,39	98,05	97,79	88,87	93,37	97,79	99,85
b13	3,49	98,04	96,96	93,47	96,96	3,51	99,13	97,36	93,50	97,36	3,54	99,40	97,83	88,88	93,54	97,83	99,86

Tableau 13.Comparaisons des résultats des structures TMR Simple, TMR Double et TMR Triple

Conclusion

L'impact de la réalisation de la structure tolérante aux fautes sur le rendement de fabrication et fiabilité peut être positif lorsque deux conditions sont respectées. La première condition concerne le rendement de fabrication qui doit être inférieur à $1/A_O$« A_O : le coût en surface de la réalisation de la TMR ». La deuxième condition est que la probabilité T « probabilité que deux défauts soient tolérés » doit être supérieure à une valeur T_{min} qui dépend des paramètres technologiques de fabrication « répartition, nombre de défauts, concentration des défauts : paramètre α pour la loi binomiale négative ».

La transformation des circuits logiques en structures TMR ne permet pas d'augmenter le rendement de fabrication des circuits, donc leur fiabilité. Le coût en surface dû à leur réalisation est très élevé et leur tolérance n'est pas suffisamment grande pour le compenser.

Finalement nous avons réussi à rendre plus fiables ces circuits en améliorant leur tolérance aux fautes. En renforçant leur redondance, grâce à un partitionnement en deux ou trois parties identiques de chacun des circuits, la tolérance de ces structures augmente de manière importante. En effet plusieurs structures, notamment les TMR triple satisfont les conditions permettant d'augmenter la fiabilité et le rendement de fabrication.

Cependant, pour augmenter le rendement et la fiabilité de la quasi-totalité des circuits, la conception parfaite des voteurs « zéro défaut » s'impose, pour les rendre moins sensibles aux défauts de fabrication, en présentant ainsi un bon compromis entre rendement de fabrication, fiabilité et coût en surface de silicium.

Conclusion générale

Nous avons étudié l'optimisation des circuits intégrés logiques conçus en vue de fiabilité même en présence de défaillances permanentes multiples injectées par modification de leurs descriptions comportementales en langage Verilog grâce à une procédure de test.

Nous avons étudié l'intérêt que pourrait avoir la réalisation des structures tolérantes aux fautes « TMR » vis-à-vis de la fiabilité via leur impact sur le rendement de fabrication. Cet intérêt à réaliser des structures TMR pour augmenter le rendement de fabrication est soumis à deux principales conditions. Premièrement, le rendement initial de fabrication doit être inférieur à $1/A_O$ « A_O étant le coût en surface lié au trois circuits clonés et au voteur ». Deuxièmement, la tolérance aux défauts de fabrication doit être supérieure à une valeur T_{min} « probabilité que deux défauts soient tolérés » calculée avec la loi de Poisson qui dépend des paramètres technologiques de fabrication. Nous avons constaté que la tolérance induisant l'amélioration du rendement peut être d'autant plus augmentée que le paramètre α de partitionnement de défauts est élevé.

On a calculé la probabilité de tolérance T qui caractérise la tolérance aux défauts des circuits des « benchmarks ISCAS85 et ITC99 » transformés en structures TMR grâce à la procédure de test ATPG ciblant les paires de fautes. Cependant d'après les résultats obtenus, aucune structure ne tolère suffisamment de défauts par rapport à son coût élevé en surface. Ainsi les structures TMR étudiées, ne permettent pas de répondre à notre objectif.

Pour y remédier, nous avons appliqué le partitionnement sur les circuits simulés que nous avons transformé en TMR double et TMR triple. Les résultats obtenus avec l'ATPG montrent que ce type de structures améliore considérablement la fiabilité et le rendement de fabrication, en réduisant les facteurs clefs, le coût en surface de silicium et la probabilité T_{min},. Grâce à cela, les structures TMR sont suffisamment tolérantes aux défauts, avec des tolérances optimales de l'ordre T_{b13}=97,83, et aussi de très hauts degrés de fiabilité (R_{b13}= 99,86) en particulier avec les structures TMR triple.

Compte tenu de ces résultats, les circuits intégrés modifiés en structures TMR procurent un bon compromis entre rendement de fabrication, fiabilité et coût en surface de silicium. C'est pourquoi des techniques de tolérance aux fautes sont récemment utilisées pour les cœurs logiques des circuits intégrés regroupant des milliards de transistors sur une même puce SOC (System-on-Chip).

Ce travail ouvre la voie à de riches perspectives de recherche dans le domaine de conception en vue de fiabilité en dépit de défauts, par l'étude et l'implémentation de structures tolérantes aux fautes moins gourmandes en surface de silicium et avec voteurs robustes. Cette voie d'amélioration passe par l'augmentation de la tolérance aux fautes, avec un choix judicieux du partitionnement, et des endroits clefs à l'intérieur des circuits qu'il conviendrait mieux de voter.

ANNEXE A

Algorithme de propagation

- procédure propager **(**S : signal ; V=D ou D')

```
begin
  if (S est une sortie primaire)
      return ;
  else
    trouver une porte P qui a le signal S comme entrée et U en sortie
    trouver une combinaison des entrées de P propageant V
    simuler le circuit   // phase d'implication
    Propager (P, V=D ou D')
    Pour chaque signal d'entrée Si de la porte P
        Justifier (Si, Vi)
  end if
end
```

Algorithme de justification

```
Procédure justifier (S : signal ; V=0 ou 1)
  begin
    if (S est une entrée primaire)
        return ;
    else
      trouver la porte P qui a le signal S comme sortie
      trouver une combinaison des entrées de P permettant d'avoir la valeur V
en sortie de la porte.
      Simuler le circuit   // phase d'implication
      Pour chaque signal d'entrée Si de la porte P
          Justifier (Si, Vi)
    end if
  end
```

Procédure Objectif ()

Algorithme de la procédure Objectif () :

- Procédure Objectif ( ) /* Faute = S @v)*/

```
Begin
    if (la valeur de S est x
        return (S, v') ;
   else
      trouver toutes les portes qui ont D ou D'en entrée et x en sortie.
      Choisir la porte P la plus proche des sorties primaires
      Choisir une entrée Ei= x de la porte P
      Soit ci la valeur qu'on veut assigner à l'entrée Ei
       return (E, c)
  end if
end
```

Procédure Backtrace ()

Algorithme de la procédure Backtrace () :

```
Procédure Backtrace (E, c)        /*Faute = S @v)*/
        Begin
          if  (E est une entrée primaire
                return (E, c) ;
           else
            Choisir une combinaison des entrées de la porte E permettant d'avoir la valeur
      c en sortie.
             Pour la combinaison choisie, choisir une entrée Ei de la porte E
             Soit ci la valeur qu'on veut assigner à l'entrée Ei
             Return (Backtrace (Ei, ci))
          end if
        end
```

Procédure PODEM ()

Algorithme PODEM ():

- Procédure PODEM ()

```
        Begin
          If (une des sorites primaires est à D ou D')
               Return succès ;
          While (il existe encore un chemin)
                (S, v) = objectif ()
                (E, c) = backtrace (S, v)
                 Simuler la valeur c sur l'entrée primaire E
                 if (PODEM ()=succès) return succès
                 Simuler la valeur c' sur l'entrée primaire E
                 if (PODEM ()=succès) return succès
                 Simuler la valeur X sur l'entrée primaire E
                 return echec
          end while
        end
```

ANNEXE B

in this code

```
-- library UNISIM; Description comportementale du conteur/ déconteur par le langage de description VHDL.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- uncomment the following library declaration if instantiating
-- any Xilinx primitive
-- use UNISIM.VComponents.all;
entity conter is
  Port (CLOCK : in STD_LOGIC ;
 DIRECTION : in STD_LOGIC;
COUNT_OUT : OUT_ STD_LOGIC VECTOR ( 3 downto 0));
end conter;
architecture Behavioral of conter is
signal count_int : std_logic_vector (3 downto 0) := “0000”;
begin
processs (CLOCK)
begin
if CLOCK=1 and CLOCK’ event then
   if  Direction =’1’ then
        count_int <= count_int + 1;
        else
       count_int <= count_int -1;
   end if;
end if;
end process;
count_out<= count_int;
end behavioral;
```

La figure 63 illustre l’utilisation du logiciel Tetramax pour injecter des fautes simultanées multiples même au niveau des points internes du circuit et calculer ainsi la probabilité de tolérance T.

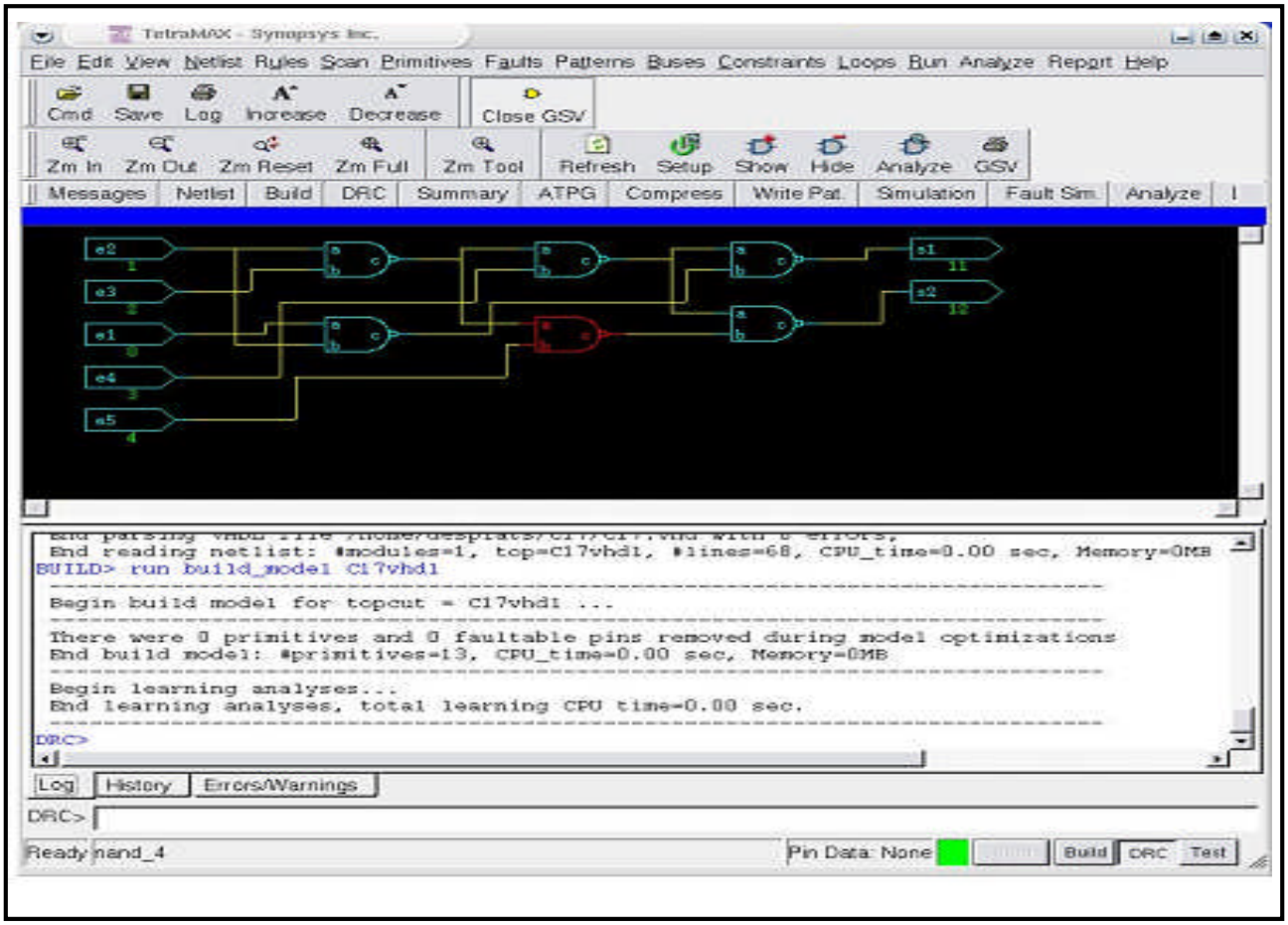


Figure 63. Génération automatique de vecteurs de test (ATPG) avec l'outil Tetramax de synopsys.

Algorithme multi-niveaux pour le partitionnement

Procédure MULTILEVEL (G,*k*)

$G_1 = G$

$i = 1$

répète

$i \longleftarrow i + 1$

G_i = contraction (G_{i-1})

jusqu'à ce que contraction faible ou G_{i+1} suffisamment petit

= partitionner (G_i,k)

pour j=i-1 à 1 faire

P_k = projection (, G_j)

= affinage (P_k)

fin pour

retourner

fin Procédure

Version étendue de l'algorithme de contraction de sommets

Procédure CONTRACTION *($G_i = (S_i , A_i)$)*

$A \leftarrow A_i$

$A_{i+1} \leftarrow A_i$

$S_{i+1} \leftarrow S_i$

tant que A≠Ø faire

Choisir au hasard une arête (s_1, s_2) dans A

Supprimer l'arête (s_1, s_2) de A et de A_{i+1}

Supprimer s_1 et s_2 de S_{i+1}

Ajouter le nouveau sommet $s = \{s_1, s_2\}$ à S_{I+1}

$poids\ (s) = poids\ (s_1) + poids\ (s_2)$

pour tout $(s_1, u) \in A$ **faire**

si $(s_2, u) \in A$ **alors**

$poids\ (s_1, u) \leftarrow poids\ (s_1, u) + poids\ (s_2, u)$

Supprimer (s_2, u) de A_{i+1}

fin si

fin pour

fin tant que

retourner $\mathbf{G_{i+1} = (S_{i+1}, A_{i+1})}$

fin Procédure

Algorithme de contraction de sommets

Procédure CONTRACTION *($G_i = (S_i , A_i)$)*

$S \leftarrow S_i$

$A_{i+1} \leftarrow A_i$

$S_{i+1} \leftarrow S_i$

tant que S≠Ø faire

Tirer au hasard un sommet $s_1 \in S_i$

Prendre l'arrête $(s_1, s_2) \in A_i$ de poids maximal

Supprimer s_1 et s_2 de S

Ajouter le nouveau sommet $s = \{s_1, s_2\}$ à S_{i+1}

$poids\ (s) = poids\ (s_1) + poids\ (s_2)$

pour tout arrête (s_1, u) dans A **faire**

si $(s_2, u) \in A$ **alors**

$poids\ (s_1, u) \leftarrow poids\ (s_1, u) + poids\ (s_2, u)$

Supprimer (s_2, u) de A_{i+1}

fin si

fin pour

fin tant que

retourner $G_{i+1} = (S_{i+1}, A_{i+1})$

fin Procédure

La procédure de test du circuit c432

Le circuit c432 de la famille ISCAS85 comporte 36 entrées et 7 sorties. La séquence de test appliquée à ce circuit est celle générée par l'ATPG (Tetramax, Synopsys) pour des pannes de collage multiples cette séquence comporte 76 vecteurs.

Le nombre total de lignes de la netlist : 432
Le nombre de fautes réduit : 524
Le nombre de lignes des entrées primaires des portes : 36
Le nombre de lignes de sorties des portes : 7
Lignes intérieures des portes : 153

La description verilog du circuit C432 est la suivante :

```
module c432 (N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
      N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
      N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
      N99, N102, N105, N108, N112, N115, N223, N329, N370, N421,
      N430, N431, N432);

input N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
      N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
      N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
      N99, N102, N105, N108, N112, N115;

Output N223, N329, N370, N421, N430, N431, N432;

wire N118,N119,N122,N123,N126,N127,N130,N131,N134,N135,
     N138, N139, N142, N143, N146, N147, N150, N151, N154, N157,
     N158, N159, N162, N165, N168, N171, N174, N177, N180, N183,
     N184, N185, N186, N187, N188, N189, N190, N191, N192, N193,
     N194, N195, N196, N197, N198, N199, N203, N213, N224, N227,
     N230, N233, N236, N239, N242, N243, N246, N247, N250, N251,
     N254, N255, N256, N257, N258, N259, N260, N263, N264, N267,
```

```
     N270, N273, N276, N279, N282, N285, N288, N289, N290, N291,
     N292, N293, N294, N295, N296, N300, N301, N302, N303, N304,
     N305, N306, N307, N308, N309, N319, N330, N331, N332, N333,
     N334, N335, N336, N337, N338, N339, N340, N341, N342, N343,
     N344, N345, N346, N347, N348, N349, N350, N351, N352, N353,
     N354, N355, N356, N357, N360, N371, N372, N373, N374, N375,
     N376, N377, N378, N379, N380, N381, N386, N393, N399, N404,
     N407, N411, N414, N415, N416, N417, N418, N419, N420, N422,
     N425, N428, N429;

inv_1 U36 (.Z (N118), .A (N1));
inv_1 U37 (.Z (N119), .A (N4));
inv_1 U38 (.Z (N122), .A (N11));
inv_1 U39 (.Z (N123), .A (N17));
inv_1 U40 (.Z (N126), .A (N24));
inv_1 U41 (.Z (N127), .A (N30));
inv_1 U42 (.Z (N130), .A (N37));
inv_1 U43 (.Z (N131), .A (N43));
inv_1 U44 (.Z (N134), .A (N50));
inv_1 U45 (.Z (N135), .A (N56));
inv_1 U46 (.Z (N138), .A (N63));
inv_1 U47 (.Z (N139), .A (N69));
inv_1 U48 (.Z (N142), .A (N76));
inv_1 U49 (.Z (N143), .A (N82));
inv_1 U50 (.Z (N146), .A (N89));
inv_1 U51 (.Z (N147), .A (N95));
inv_1 U52 (.Z (N150), .A (N102));
inv_1 U53 (.Z (N151), .A (N108));
nand2_1 U54 (.Z (N154), .A (N118), .B (N4));
nor2_1 U55 (.Z (N157), .A (N8), .B (N119));
nor2_1 U56 (.Z (N158), .A (N14), .B (N119));
nand2_1 U57 (.Z (N159), .A (N122), .B (N17));
nand2_1 U58 (.Z (N162), .A (N126), .B (N30));
nand2_1 U59 (.Z (N165), .A (N130), .B (N43));
nand2_1 U60 (.Z (N168), .A (N134), .B (N56));
nand2_1 U61 (.Z (N171), .A (N138), .B (N69));
nand2_1 U62 (.Z (N174), .A (N142), .B (N82));
nand2_1 U63 (.Z (N177), .A (N146), .B (N95));
```

```
nand2_1 U64 (.Z (N180), .A (N150), .B (N108));
nor2_1 U65 (.Z (N183), .A (N21), .B (N123));
nor2_1 U66 (.Z (N184), .A (N27), .B (N123));
nor2_1 U67 (.Z (N185), .A (N34), .B (N127));
nor2_1 U68 (.Z (N186), .A (N40), .B (N127));
nor2_1 U69 (.Z (N187), .A (N47), .B (N131));
nor2_1 U70 (.Z (N188), .A (N53), .B (N131));
nor2_1 U71 (.Z (N189), .A (N60), .B (N135));
nor2_1 U72 (.Z (N190), .A (N66), .B (N135));
nor2_1 U73 (.Z (N191), .A (N73), .B (N139));
nor2_1 U74 (.Z (N192), .A (N79), .B (N139));
nor2_1 U75 (.Z (N193), .A (N86), .B (N143));
nor2_1 U76 (.Z (N194), .A (N92), .B (N143));
nor2_1 U77 (.Z (N195), .A (N99), .B (N147));
nor2_1 U78 (.Z (N196), .A (N105), .B (N147));
nor2_1 U79 (.Z (N197), .A (N112), .B (N151));
nor2_1 U80 (.Z (N198), .A (N115), .B (N151));
and9_1 U81 ( .Z(N199), .A(N154), .B(N159), .C(N162), .D(N165), .E(N168),
.F(N171), .G(N174), .H(N177), .I(N180) );
inv_1 U82 (.Z (N203), .A (N199));
inv_1 U83 (.Z (N213), .A (N199));
inv_1 U84 (.Z (N223), .A (N199));
xor2 U85 (.Z (N224), .A (N203), .B (N154));
xor2 U86 (.Z (N227), .A (N203), .B (N159));
xor2 U87 (.Z (N230), .A (N203), .B (N162));
xor2 U88 (.Z (N233), .A (N203), .B (N165));
xor2 U89 (.Z (N236), .A (N203), .B (N168));
xor2 U90 (.Z (N239), .A (N203), .B (N171));
nand2_1 U91 (.Z (N242), .A (N1), .B (N213));
xor2 U92 (.Z (N243), .A (N203), .B (N174));
nand2_1 U93 (.Z (N246), .A (N213), .B (N11));
xor2 U94 (.Z (N247), .A (N203), .B (N177));
nand2_1 U95 (.Z (N250), .A (N213), .B (N24));
xor2 U96 (.Z (N251), .A (N203), .B (N180));
nand2_1 U97 (.Z (N254), .A (N213), .B (N37));
nand2_1 U98 (.Z (N255), .A (N213), .B (N50));
nand2_1 U99 (.Z (N256), .A (N213), .B (N63));
nand2_1 U100 (.Z (N257), .A (N213), .B (N76));
```

```
nand2_1 U101 (.Z (N258), .A (N213), .B (N89));
nand2_1 U102 (.Z (N259), .A (N213), .B (N102));
nand2_1 U103 (.Z (N260), .A (N224), .B (N157));
nand2_1 U104 (.Z (N263), .A (N224), .B (N158));
nand2_1 U105 (.Z (N264), .A (N227), .B (N183));
nand2_1 U106 (.Z (N267), .A (N230), .B (N185));
nand2_1 U107 (.Z (N270), .A (N233), .B (N187));
nand2_1 U108 (.Z (N273), .A (N236), .B (N189));
nand2_1 U109 (.Z (N276), .A (N239), .B (N191));
nand2_1 U110 (.Z (N279), .A (N243), .B (N193));
nand2_1 U111 (.Z (N282), .A (N247), .B (N195));
nand2_1 U112 (.Z (N285), .A (N251), .B (N197));
nand2_1 U113 (.Z (N288), .A (N227), .B (N184));
nand2_1 U114 (.Z (N289), .A (N230), .B (N186));
nand2_1 U115 (.Z (N290), .A (N233), .B (N188));
nand2_1 U116 (.Z (N291), .A (N236), .B (N190));
nand2_1 U117 (.Z (N292), .A (N239), .B (N192));
nand2_1 U118 (.Z (N293), .A (N243), .B (N194));
nand2_1 U119 (.Z (N294), .A (N247), .B (N196));
nand2_1 U120 (.Z (N295), .A (N251), .B (N198));
and9_1 U121 ( .Z(N296), .A(N260), .B(N264), .C(N267), .D(N270), .E(N273),
.F(N276), .G(N279), .H(N282), .I(N285) );
inv_1 U122 (.Z (N300), .A (N263));
inv_1 U123 (.Z (N301), .A (N288));
inv_1 U124 (.Z (N302), .A (N289));
inv_1 U125 (.Z (N303), .A (N290));
inv_1 U126 (.Z (N304), .A (N291));
inv_1 U127 (.Z (N305), .A (N292));
inv_1 U128 (.Z (N306), .A (N293));
inv_1 U129 (.Z (N307), .A (N294));
inv_1 U130 (.Z (N308), .A (N295));
inv_1 U131 (.Z (N309), .A (N296));
inv_1 U132 (.Z (N319), .A (N296));
inv_1 U133 (.Z (N329), .A (N296));
xor2 U134 (.Z (N330), .A (N309), .B (N260));
xor2 U135 (.Z (N331), .A (N309), .B (N264));
xor2 U136 (.Z (N332), .A (N309), .B (N267));
xor2 U137 (.Z (N333), .A (N309), .B (N270));
```

```
nand2_1 U138 (.Z (N334), .A (N8), .B (N319));
xor2 U139 (.Z (N335), .A (N309), .B (N273));
nand2_1 U140 (.Z (N336), .A (N319), .B (N21));
xor2 U141 (.Z (N337), .A (N309), .B (N276));
nand2_1 U142 (.Z (N338), .A (N319), .B (N34));
xor2 U143 (.Z (N339), .A (N309), .B (N279));
nand2_1 U144 (.Z (N340), .A (N319), .B (N47));
xor2 U145 (.Z (N341), .A (N309), .B (N282));
nand2_1 U146 (.Z (N342), .A (N319), .B (N60));
xor2 U147 (.Z (N343), .A (N309), .B (N285));
nand2_1 U148 (.Z (N344), .A (N319), .B (N73));
nand2_1 U149 (.Z (N345), .A (N319), .B (N86));
nand2_1 U150 (.Z (N346), .A (N319), .B (N99));
nand2_1 U151 (.Z (N347), .A (N319), .B (N112));
nand2_1 U152 (.Z (N348), .A (N330), .B (N300));
nand2_1 U153 (.Z (N349), .A (N331), .B (N301));
nand2_1 U154 (.Z (N350), .A (N332), .B (N302));
nand2_1 U155 (.Z (N351), .A (N333), .B (N303));
nand2_1 U156 (.Z (N352), .A (N335), .B (N304));
nand2_1 U157 (.Z (N353), .A (N337), .B (N305));
nand2_1 U158 (.Z (N354), .A (N339), .B (N306));
nand2_1 U159 (.Z (N355), .A (N341), .B (N307));
nand2_1 U160 (.Z (N356), .A (N343), .B (N308));
and9_1 U161 ( .Z(N357), .A(N348), .B(N349), .C(N350), .D(N351), .E(N352),
.F(N353), .G(N354), .H(N355), .I(N356) );
inv_1 U162 (.Z (N360), .A (N357));
inv_1 U163 (.Z (N370), .A (N357));
nand2_1 U164 (.Z (N371), .A (N14), .B (N360));
nand2_1 U165 (.Z (N372), .A (N360), .B (N27));
nand2_1 U166 (.Z (N373), .A (N360), .B (N40));
nand2_1 U167 (.Z (N374), .A (N360), .B (N53));
nand2_1 U168 (.Z (N375), .A (N360), .B (N66));
nand2_1 U169 (.Z (N376), .A (N360), .B (N79));
nand2_1 U170 (.Z (N377), .A (N360), .B (N92));
nand2_1 U171 (.Z (N378), .A (N360), .B (N105));
nand2_1 U172 (.Z (N379), .A (N360), .B (N115));
nand4_1 U173 (.Z (N380), .A (N4), .B (N242), .C (N334), .D (N371));
nand4_1 U174 (.Z (N381), .A (N246), .B (N336), .C (N372), .D (N17));
```

```
nand4_1 U175 (.Z (N386), .A (N250), .B (N338), .C (N373), .D (N30));
nand4_1 U176 (.Z (N393), .A (N254), .B (N340), .C (N374), .D (N43));
nand4_1 U177 (.Z (N399), .A (N255), .B (N342), .C (N375), .D (N56));
nand4_1 U178 (.Z (N404), .A (N256), .B (N344), .C (N376), .D (N69));
nand4_1 U179 (.Z (N407), .A (N257), .B (N345), .C (N377), .D (N82));
nand4_1 U180 (.Z (N411), .A (N258), .B (N346), .C (N378), .D (N95));
nand4_1 U181 (.Z (N414), .A (N259), .B (N347), .C (N379), .D (N108));
inv_1 U182 (.Z (N415), .A (N380));
and8_1 U183 ( .Z(N416), .A(N381), .B(N386), .C(N393), .D(N399), .E(N404),
.F(N407), .G(N411), .H(N414) );
inv_1 U184 (.Z (N417), .A (N393));
inv_1 U185 (.Z (N418), .A (N404));
inv_1 U186 (.Z (N419), .A (N407));
inv_1 U187 (.Z (N420), .A (N411));
nor2_1 U188 (.Z (N421), .A (N415), .B (N416));
nand2_1 U189 (.Z (N422), .A (N386), .B (N417));
nand4_1 U190 (.Z (N425), .A (N386), .B (N393), .C (N418), .D (N399));
nand3_1 U191 (.Z (N428), .A (N399), .B (N393), .C (N419));
nand4_1 U192 (.Z (N429), .A (N386), .B (N393), .C (N407), .D (N420));
nand4_1 U193 (.Z (N430), .A (N381), .B (N386), .C (N422), .D (N399));
nand4_1 U194 (.Z (N431), .A (N381), .B (N386), .C (N425), .D (N428));
nand4_1 U195 (.Z (N432), .A (N381), .B (N422), .C (N425), .D (N429));

end module
```

La description VERILOG de la structure TMR du circuit c432

1ère étape : la description de la structure TMR du circuit C432

```
1 module TMRc432 (N1, N4, N8, N11, N14, N17, N21, N24, N27, N30,
  N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
  N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
  N99, N102, N105, N108, N112, N115, N223, N329, N370, N421,
  N430, N431, N432);
2 Input N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
  N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
  N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
  N99, N102, N105, N108, N112, N115;
```

```
3 Output N223, N329, N370, N421, N430, N431, N432;

4 c 432 T1 (N1, N4, N8, N11, N14, N17, N21, N24, N27, N30, N34, N37, N40,
  N43, N47, N50, N53, N56, N60, N63, N66, N69, N73, N76, N79, N82, N86,
  N89, N92, N95, N99, N102, N105, N108, N112, N115, N223M1, N329M1, N370M1,
  N421M1, N430M1, N431M1, N432M1) ;
5 c432T2 (N1, N4, N8, N11, N14, N17, N21, N24, N27, N30, N34, N37, N40,
  N43, N47, N50, N53, N56, N60, N63, N66, N69, N73, N76, N79, N82, N86,
  N89, N92, N95, N99, N102, N105, N108, N112, N115, N223M2, N329M2, N370M2,
  N421M2, N430M2, N431M2, N432M2) ;
6 c432T3 (N1, N4, N8, N11, N14, N17, N21, N24, N27, N30, N34, N37, N40,
  N43, N47, N50, N53, N56, N60, N63, N66, N69, N73, N76, N79, N82, N86,
  N89, N92, N95, N99, N102, N105, N108, N112, N115, N223M3, N329M3, N370M3,
  N421M3, N430M3, N431M3, N432M3) ;

7 voteur V0 (N223M1, N223M2, N223M3, N223);
8 voteur V1 (N329M1, N329 M2, N329M3, N329);

9 voteur V2 (N370M1, N370M2, N370M3, N370);
10 voteur V3 (N421M1, N421M2, N421M3, N421);
11 voteur V4 (N430M1, N430M2, N430M3, N430);
12 voteur V5 (N431M1, N431M2, N431M3, N431);
13 voteur V6 (N432M1, N432M2, N432M3, N432);

endmodule
```

La description VERILOG du voteur

```
1  module voteur (E1, E2, E3, S1);
2  input E1, E2, E3;
3  output S1;
4  and gate_1 (L1, E1, E2);
5  and gate_2 (L2, E1, E3);
6  and gate_3 (L3, E2, E3);
7  or gate_4 (S1, L1, L2, L3);
8  endmodule
```

2ème étape : la liste de fautes de collage simple du circuit c432:

```
   1     1gat inpt    2   0 >sa0 >sa1
   2     1f01 from    1gat
   3     1f02 from    1gat      >sa1
   4     4gat inpt    3   0 >sa0 >sa1
   5     4f01 from    4gat      >sa1
   6     4f02 from    4gat
   7     4f03 from    4gat      >sa1
   8     8gat inpt    2   0 >sa0 >sa1
   9     8f01 from    8gat >sa0
  10     8f02 from    8gat      >sa1
  11    11gat inpt    2   0 >sa0 >sa1
  12    11f01 from    11gat
  13    11f02 from    11gat     >sa1
  14    14gat inpt    2   0 >sa0 >sa1
  15    14f01 from    14gat >sa0
  16    14f02 from    14gat     >sa1
  17    17gat inpt    3   0 >sa0 >sa1
  18    17f01 from    17gat     >sa1
  19    17f02 from    17gat
  20    17f03 from    17gat     >sa1
  21    21gat inpt    2   0 >sa0 >sa1
  22    21f01 from    21gat >sa0
  23    21f02 from    21gat     >sa1
  24    24gat inpt    2   0 >sa0 >sa1
  25    24f01 from    24gat
  26    24f02 from    24gat     >sa1
  27    27gat inpt    2   0 >sa0 >sa1
  28    27f01 from    27gat >sa0
  29    27f02 from    27gat     >sa1
  30    30gat inpt    3   0 >sa0 >sa1
  31    30f01 from    30gat     >sa1
  32    30f02 from    30gat
  33    30f03 from    30gat     >sa1
  34    34gat inpt    2   0 >sa0 >sa1
  35    34f01 from    34gat >sa0
  36    34f02 from    34gat     >sa1
```

```
37    37gat inpt    2   0 >sa0 >sa1
38    37f01 from    37gat
39    37f02 from    37gat      >sa1
40    40gat inpt    2   0 >sa0 >sa1
41    40f01 from    40gat >sa0
42    40f02 from    40gat      >sa1
43    43gat inpt    3   0 >sa0 >sa1
44    43f01 from    43gat      >sa1
45    43f02 from    43gat
46    43f03 from    43gat      >sa1
47    47gat inpt    2   0 >sa0 >sa1
48    47f01 from    47gat >sa0
49    47f02 from    47gat      >sa1
50    50gat inpt    2   0 >sa0 >sa1
51    50f01 from    50gat
52    50f02 from    50gat      >sa1
53    53gat inpt    2   0 >sa0 >sa1
54    53f01 from    53gat >sa0
55    53f02 from    53gat      >sa1
56    56gat inpt    3   0 >sa0 >sa1
57    56f01 from    56gat      >sa1
58    56f02 from    56gat
59    56f03 from    56gat      >sa1
60    60gat inpt    2   0 >sa0 >sa1
61    60f01 from    60gat >sa0
62    60f02 from    60gat      >sa1
63    63gat inpt    2   0 >sa0 >sa1
64    63f01 from    63gat
65    63f02 from    63gat      >sa1
66    66gat inpt    2   0 >sa0 >sa1
67    66f01 from    66gat >sa0
68    66f02 from    66gat      >sa1
69    69gat inpt    3   0 >sa0 >sa1
70    69f01 from    69gat      >sa1
71    69f02 from    69gat
72    69f03 from    69gat      >sa1
73    73gat inpt    2   0 >sa0 >sa1
74    73f01 from    73gat >sa0
```

```
 75    73f02 from    73gat      >sa1
 76    76gat inpt    2   0 >sa0 >sa1
 77    76f01 from    76gat
 78    76f02 from    76gat      >sa1
 79    79gat inpt    2   0 >sa0 >sa1
 80    79f01 from    79gat >sa0
 81    79f02 from    79gat      >sa1
 82    82gat inpt    3   0 >sa0 >sa1
 83    82f01 from    82gat      >sa1
 84    82f02 from    82gat
 85    82f03 from    82gat      >sa1
 86    86gat inpt    2   0 >sa0 >sa1
 87    86f01 from    86gat >sa0
 88    86f02 from    86gat      >sa1
 89    89gat inpt    2   0 >sa0 >sa1
 90    89f01 from    89gat
 91    89f02 from    89gat      >sa1
 92    92gat inpt    2   0 >sa0 >sa1
 93    92f01 from    92gat >sa0
 94    92f02 from    92gat      >sa1
 95    95gat inpt    3   0 >sa0 >sa1
 96    95f01 from    95gat      >sa1
 97    95f02 from    95gat
 98    95f03 from    95gat      >sa1
 99    99gat inpt    2   0 >sa0 >sa1
100    99f01 from    99gat >sa0
101    99f02 from    99gat      >sa1
102   102gat inpt    2   0 >sa0 >sa1
103   102f01 from   102gat
104   102f02 from   102gat      >sa1
105   105gat inpt    2   0 >sa0 >sa1
106   105f01 from   105gat >sa0
107   105f02 from   105gat      >sa1
108   108gat inpt    3   0 >sa0 >sa1
109   108f01 from   108gat      >sa1
110   108f02 from   108gat
111   108f03 from   108gat      >sa1
112   112gat inpt    2   0 >sa0 >sa1
```

```
113   112f01 from   112gat >sa0
114   112f02 from   112gat      >sa1
115   115gat inpt    2   0 >sa0 >sa1
116   115f01 from   115gat >sa0
117   115f02 from   115gat      >sa1
118   118gat not     1   1      >sa1
   2
119   119gat not     2   1 >sa0 >sa1
   6
120   119f01 from   119gat >sa0
121   119f02 from   119gat >sa0
122   122gat not     1   1      >sa1
  12
123   123gat not     2   1 >sa0 >sa1
  19
124   123f01 from   123gat >sa0
125   123f02 from   123gat >sa0
126   126gat not     1   1      >sa1
  25
127   127gat not     2   1 >sa0 >sa1
  32
128   127f01 from   127gat >sa0
129   127f02 from   127gat >sa0
130   130gat not     1   1      >sa1
  38
131   131gat not     2   1 >sa0 >sa1
  45
132   131f01 from   131gat >sa0
133   131f02 from   131gat >sa0
134   134gat not     1   1      >sa1
  51
135   135gat not     2   1 >sa0 >sa1
  58
136   135f01 from   135gat >sa0
137   135f02 from   135gat >sa0
138   138gat not     1   1      >sa1
  64
139   139gat not     2   1 >sa0 >sa1
```

```
  71
140   139f01 from   139gat >sa0
141   139f02 from   139gat >sa0
142   142gat not     1   1      >sa1
  77
143   143gat not     2   1 >sa0 >sa1
  84
144   143f01 from   143gat >sa0
145   143f02 from   143gat >sa0
146   146gat not     1   1      >sa1
  90
147   147gat not     2   1 >sa0 >sa1
  97
148   147f01 from   147gat >sa0
149   147f02 from   147gat >sa0
150   150gat not     1   1      >sa1
 103
151   151gat not     2   1 >sa0 >sa1
 110
152   151f01 from   151gat >sa0
153   151f02 from   151gat >sa0
154   154gat nand    2   2 >sa0 >sa1
 118     5
155   154f01 from   154gat      >sa1
156   154f02 from   154gat >sa0 >sa1
157   157gat nor     1   2      >sa1
   9   120
158   158gat nor     1   2      >sa1
  15   121
159   159gat nand    2   2 >sa0 >sa1
 122    18
160   159f01 from   159gat      >sa1
161   159f02 from   159gat >sa0 >sa1
162   162gat nand    2   2 >sa0 >sa1
 126    31
163   162f01 from   162gat      >sa1
164   162f02 from   162gat >sa0 >sa1
165   165gat nand    2   2 >sa0 >sa1
```

```
 130    44
166   165f01 from   165gat      >sa1
167   165f02 from   165gat >sa0 >sa1
168   168gat nand    2   2 >sa0 >sa1
 134    57
169   168f01 from   168gat      >sa1
170   168f02 from   168gat >sa0 >sa1
171   171gat nand    2   2 >sa0 >sa1
 138    70
172   171f01 from   171gat      >sa1
173   171f02 from   171gat >sa0 >sa1
174   174gat nand    2   2 >sa0 >sa1
 142    83
175   174f01 from   174gat      >sa1
176   174f02 from   174gat >sa0 >sa1
177   177gat nand    2   2 >sa0 >sa1
 146    96
178   177f01 from   177gat      >sa1
179   177f02 from   177gat >sa0 >sa1
180   180gat nand    2   2 >sa0 >sa1
 150   109
181   180f01 from   180gat      >sa1
182   180f02 from   180gat >sa0 >sa1
183   183gat nor     1   2      >sa1
  22   124
184   184gat nor     1   2      >sa1
  28   125
185   185gat nor     1   2      >sa1
  35   128
186   186gat nor     1   2      >sa1
  41   129
187   187gat nor     1   2      >sa1
  48   132
188   188gat nor     1   2      >sa1
  54   133
189   189gat nor     1   2      >sa1
  61   136
190   190gat nor     1   2      >sa1
```

```
  67   137
191   191gat nor     1   2      >sa1
  74   140
192   192gat nor     1   2      >sa1
  80   141
193   193gat nor     1   2      >sa1
  87   144
194   194gat nor     1   2      >sa1
  93   145
195   195gat nor     1   2      >sa1
 100   148
196   196gat nor     1   2      >sa1
 106   149
197   197gat nor     1   2      >sa1
 113   152
198   198gat nor     1   2      >sa1
 116   153
199   199gat and     3   9 >sa0 >sa1
 155   160   163   166   169   172   175   178   181
200   199f01 from   199gat
201   199f02 from   199gat
202   199f03 from   199gat
203   203gat not     9   1 >sa0 246 >sa1
 200
204   203f01 from   203gat >sa0 >sa1
205   203f02 from   203gat >sa0 >sa1
206   203f03 from   203gat >sa0 >sa1
207   203f04 from   203gat >sa0 >sa1
208   203f05 from   203gat >sa0 >sa1
209   203f06 from   203gat >sa0 >sa1
210   203f07 from   203gat >sa0 >sa1
211   203f08 from   203gat >sa0 >sa1
212   203f09 from   203gat >sa0 >sa1
213   213gat not     9   1 >sa0 >sa1
 201
214   213f01 from   213gat      >sa1
215   213f02 from   213gat      >sa1
216   213f03 from   213gat      >sa1
```

```
217   213f04 from   213gat      >sa1
218   213f05 from   213gat      >sa1
219   213f06 from   213gat      >sa1
220   213f07 from   213gat      >sa1
221   213f08 from   213gat      >sa1
222   213f09 from   213gat      >sa1
223   223gat not     0   1 >sa0 >sa1
 202
224   224gat xor     2   2 >sa0 >sa1
 204   156
225   224f01 from   224gat      >sa1
226   224f02 from   224gat      >sa1
227   227gat xor     2   2 >sa0 >sa1
 205   161
228   227f01 from   227gat      >sa1
229   227f02 from   227gat      >sa1
230   230gat xor     2   2 >sa0 >sa1
 206   164
231   230f01 from   230gat      >sa1
232   230f02 from   230gat      >sa1
233   233gat xor     2   2 >sa0 >sa1
 207   167
234   233f01 from   233gat      >sa1
235   233f02 from   233gat      >sa1
236   236gat xor     2   2 >sa0 >sa1
 208   170
237   236f01 from   236gat      >sa1
238   236f02 from   236gat      >sa1
239   239gat xor     2   2 >sa0 >sa1
 209   173
240   239f01 from   239gat      >sa1
241   239f02 from   239gat      >sa1
242   242gat nand    1   2      >sa1
   3   214
243   243gat xor     2   2 >sa0 >sa1
 210   176
244   243f01 from   243gat      >sa1
245   243f02 from   243gat      >sa1
```

```
246   246gat nand    1   2      >sa1
 215    13
247   247gat xor     2   2 >sa0 >sa1
 211   179
248   247f01 from   247gat      >sa1
249   247f02 from   247gat      >sa1
250   250gat nand    1   2      >sa1
 216    26
251   251gat xor     2   2 >sa0 >sa1
 212   182
252   251f01 from   251gat      >sa1
253   251f02 from   251gat      >sa1
254   254gat nand    1   2      >sa1
 217    39
255   255gat nand    1   2      >sa1
 218    52
256   256gat nand    1   2      >sa1
 219    65
257   257gat nand    1   2      >sa1
 220    78
258   258gat nand    1   2      >sa1
 221    91
259   259gat nand    1   2      >sa1
 222   104
260   260gat nand    2   2 >sa0 >sa1
 225   157
261   260f01 from   260gat      >sa1
262   260f02 from   260gat >sa0 >sa1
263   263gat nand    1   2
 226   158
264   264gat nand    2   2 >sa0 >sa1
 228   183
265   264f01 from   264gat      >sa1
266   264f02 from   264gat >sa0 >sa1
267   267gat nand    2   2 >sa0 >sa1
 231   185
268   267f01 from   267gat      >sa1
269   267f02 from   267gat >sa0 >sa1
```

```
270   270gat nand    2   2 >sa0 >sa1
 234   187
271   270f01 from   270gat      >sa1
272   270f02 from   270gat >sa0 >sa1
273   273gat nand    2   2 >sa0 >sa1
 237   189
274   273f01 from   273gat      >sa1
275   273f02 from   273gat >sa0 >sa1
276   276gat nand    2   2 >sa0 >sa1
 240   191
277   276f01 from   276gat      >sa1
278   276f02 from   276gat >sa0 >sa1
279   279gat nand    2   2 >sa0 >sa1
 244   193
280   279f01 from   279gat      >sa1
281   279f02 from   279gat >sa0 >sa1
282   282gat nand    2   2 >sa0 >sa1
 248   195
283   282f01 from   282gat      >sa1
284   282f02 from   282gat >sa0 >sa1
285   285gat nand    2   2 >sa0 >sa1
 252   197
286   285f01 from   285gat      >sa1
287   285f02 from   285gat >sa0 >sa1
288   288gat nand    1   2
 229   184
289   289gat nand    1   2
 232   186
290   290gat nand    1   2
 235   188
291   291gat nand    1   2
 238   190
292   292gat nand    1   2
 241   192
293   293gat nand    1   2
 245   194
294   294gat nand    1   2
 249   196
```

```
295   295gat nand    1   2
 253   198
296   296gat and     3   9 >sa0 >sa1
 261   265   268   271   274   277   280   283   286
297   296f01 from   296gat
298   296f02 from   296gat
299   296f03 from   296gat
300   300gat not     1   1      >sa1
 263
301   301gat not     1   1      >sa1
 288
302   302gat not     1   1      >sa1
 289
303   303gat not     1   1      >sa1
 290
304   304gat not     1   1      >sa1
 291
305   305gat not     1   1      >sa1
 292
306   306gat not     1   1      >sa1
 293
307   307gat not     1   1      >sa1
 294
308   308gat not     1   1      >sa1
 295
309   309gat not     9   1 >sa0 >sa1
 297
310   309f01 from   309gat >sa0 >sa1
311   309f02 from   309gat >sa0 >sa1
312   309f03 from   309gat >sa0 >sa1
313   309f04 from   309gat >sa0 >sa1
314   309f05 from   309gat >sa0 >sa1
315   309f06 from   309gat >sa0 >sa1
316   309f07 from   309gat >sa0 >sa1
317   309f08 from   309gat >sa0 >sa1
318   309f09 from   309gat >sa0 >sa1
319   319gat not     9   1 >sa0 >sa1
 298
```

```
320   319f01 from   319gat      >sa1
321   319f02 from   319gat      >sa1
322   319f03 from   319gat      >sa1
323   319f04 from   319gat      >sa1
324   319f05 from   319gat      >sa1
325   319f06 from   319gat      >sa1
326   319f07 from   319gat      >sa1
327   319f08 from   319gat      >sa1
328   319f09 from   319gat      >sa1
329   329gat not     0   1 >sa0 >sa1
 299
330   330gat xor     1   2      >sa1
 310   262
331   331gat xor     1   2      >sa1
 311   266
332   332gat xor     1   2      >sa1
 312   269
333   333gat xor     1   2      >sa1
 313   272
334   334gat nand    1   2      >sa1
  10   320
335   335gat xor     1   2      >sa1
 314   275
336   336gat nand    1   2      >sa1
 321    23
337   337gat xor     1   2      >sa1
 315   278
338   338gat nand    1   2      >sa1
 322    36
339   339gat xor     1   2      >sa1
 316   281
340   340gat nand    1   2      >sa1
 323    49
341   341gat xor     1   2      >sa1
 317   284
342   342gat nand    1   2      >sa1
 324    62
343   343gat xor     1   2      >sa1
```

```
 318   287
344   344gat nand    1   2      >sa1
 325    75
345   345gat nand    1   2      >sa1
 326    88
346   346gat nand    1   2      >sa1
 327   101
347   347gat nand    1   2      >sa1
 328   114
348   348gat nand    1   2      >sa1
 330   300
349   349gat nand    1   2      >sa1
 331   301
350   350gat nand    1   2      >sa1
 332   302
351   351gat nand    1   2      >sa1
 333   303
352   352gat nand    1   2      >sa1
 335   304
353   353gat nand    1   2      >sa1
 337   305
354   354gat nand    1   2      >sa1
 339   306
355   355gat nand    1   2      >sa1
 341   307
356   356gat nand    1   2      >sa1
 343   308
357   357gat and     2   9 >sa0 >sa1
 348   349   350   351   352   353   354   355   356
358   357f01 from   357gat
359   357f02 from   357gat
360   360gat not     9   1 >sa0 >sa1
 358
361   360f01 from   360gat      >sa1
362   360f02 from   360gat      >sa1
363   360f03 from   360gat      >sa1
364   360f04 from   360gat      >sa1
365   360f05 from   360gat      >sa1
```

```
366   360f06 from   360gat      >sa1
367   360f07 from   360gat      >sa1
368   360f08 from   360gat      >sa1
369   360f09 from   360gat      >sa1
370   370gat not     0   1 >sa0 >sa1
 359
371   371gat nand    1   2      >sa1
  16   361
372   372gat nand    1   2      >sa1
 362    29
373   373gat nand    1   2      >sa1
 363    42
374   374gat nand    1   2      >sa1
 364    55
375   375gat nand    1   2      >sa1
 365    68
376   376gat nand    1   2      >sa1
 366    81
377   377gat nand    1   2      >sa1
 367    94
378   378gat nand    1   2      >sa1
 368   107
379   379gat nand    1   2      >sa1
 369   117
380   380gat nand    1   4
   7   242   334   371
381   381gat nand    4   4 >sa0 >sa1
 246   336   372    20
382   381f01 from   381gat      >sa1
383   381f02 from   381gat      >sa1
384   381f03 from   381gat      >sa1
385   381f04 from   381gat      >sa1
386   386gat nand    6   4 >sa0 >sa1
 250   338   373    33
387   386f01 from   386gat      >sa1
388   386f02 from   386gat      >sa1
389   386f03 from   386gat      >sa1
390   386f04 from   386gat      >sa1
```

```
391   386f05 from   386gat      >sa1
392   386f06 from   386gat      >sa1
393   393gat nand    5   4 >sa0 >sa1
 254   340   374    43
394   393f01 from   393gat      >sa1
395   393f02 from   393gat
396   393f03 from   393gat      >sa1
397   393f04 from   393gat      >sa1
398   393f05 from   393gat      >sa1
399   399gat nand    4   4 >sa0 >sa1
 255   342   375    59
400   399f01 from   399gat      >sa1
401   399f02 from   399gat      >sa1
402   399f03 from   399gat      >sa1
403   399f04 from   399gat      >sa1
404   404gat nand    2   4 >sa0 >sa1
 256   344   376    72
405   404f01 from   404gat      >sa1
406   404f02 from   404gat
407   407gat nand    3   4 >sa0 >sa1
 257   345   377    85
408   407f01 from   407gat      >sa1
409   407f02 from   407gat
410   407f03 from   407gat      >sa1
411   411gat nand    2   4 >sa0 >sa1
 258   346   378    98
412   411f01 from   411gat      >sa1
413   411f02 from   411gat
414   414gat nand    1   4      >sa1
 259   347   379   111
415   415gat not     1   1 >sa0
 380
416   416gat and     1   8 >sa0
 382   387   394   400   405   408   412   414
417   417gat not     1   1      >sa1
 395
418   418gat not     1   1      >sa1
 406
```

```
419   419gat not     1   1      >sa1
 409
420   420gat not     1   1      >sa1
 413
421   421gat nor     0   2 >sa0 >sa1
 415   416
422   422gat nand    2   2 >sa0 >sa1
 388   417
423   422f01 from   422gat      >sa1
424   422f02 from   422gat      >sa1
425   425gat nand    2   4 >sa0 >sa1
 389   396   418   401
426   425f01 from   425gat      >sa1
427   425f02 from   425gat      >sa1
428   428gat nand    1   3      >sa1
 402   397   419
429   429gat nand    1   4      >sa1
 391   398   410   420
430   430gat nand    0   4 >sa0 >sa1
 383   390   423   403
431   431gat nand    0   4 >sa0 >sa1
 384   392   426   428
432   432gat nand    0   4 >sa0 >sa1
 385   424   427   429
```

3ème étape : injection d'une faute pivot

***Exemple 1** :*

Injection de la faute n° 508 de la liste de fautes, collage à 0 de la sortie (N421) de la porte nor « U188 »

```
1   /*collage a 0 de la sortie de U188 */
2   module c432modifie_508 (N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
                N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
                N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
                N99, N102, N105, N108, N112, N115, N223, N329, N370, N421,
                N430, N431, N432;
3   input N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
                N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
```

```
            N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
            N99, N102, N105, N108, N112, N115;
4   outputs N223, N329, N370, N421, N430, N431, N432;

5   inv_1 U36 (.Z (N118), .A (N1));
6   inv_1 U37 (.Z (N119), .A (N4));
7   inv_1 U38 (.Z (N122), .A (N11));
8   inv_1 U39 (.Z (N123), .A (N17));
9   inv_1 U40 (.Z (N126), .A (N24));
10  inv_1 U41 (.Z (N127), .A (N30));
11  inv_1 U42 (.Z (N130), .A (N37));
12  inv_1 U43 (.Z (N131), .A (N43));
13  inv_1 U44 (.Z (N134), .A (N50));
14  inv_1 U45 (.Z (N135), .A (N56));
15  inv_1 U46 (.Z (N138), .A (N63));
16  inv_1 U47 (.Z (N139), .A (N69));
17  inv_1 U48 (.Z (N142), .A (N76));
18  inv_1 U49 (.Z (N143), .A (N82));
19  inv_1 U50 (.Z (N146), .A (N89));
20  inv_1 U51 (.Z (N147), .A (N95));
21  inv_1 U52 (.Z (N150), .A (N102));
22  inv_1 U53 (.Z (N151), .A (N108));
23  nand2_1 U54 (.Z (N154), .A (N118), .B (N4));
24  nor2_1 U55 (.Z (N157), .A (N8), .B (N119));
25  nor2_1 U56 (.Z (N158), .A (N14), .B (N119));
26  nand2_1 U57 (.Z (N159), .A (N122), .B (N17));
27  nand2_1 U58 (.Z (N162), .A (N126), .B (N30));
28  nand2_1 U59 (.Z (N165), .A (N130), .B (N43));
29  nand2_1 U60 (.Z (N168), .A (N134), .B (N56));
30  nand2_1 U61 (.Z (N171), .A (N138), .B (N69));
31  nand2_1 U62 (.Z (N174), .A (N142), .B (N82));
32  nand2_1 U63 (.Z (N177), .A (N146), .B (N95));
33  nand2_1 U64 (.Z (N180), .A (N150), .B (N108));
34  nor2_1 U65 (.Z (N183), .A (N21), .B (N123));
35  nor2_1 U66 (.Z (N184), .A (N27), .B (N123));
36  nor2_1 U67 (.Z (N185), .A (N34), .B (N127));
37  nor2_1 U68 (.Z (N186), .A (N40), .B (N127));
38  nor2_1 U69 (.Z (N187), .A (N47), .B (N131));
```

```
39   nor2_1 U70 (.Z (N188), .A (N53), .B (N131));
40   nor2_1 U71 (.Z (N189), .A (N60), .B (N135));
41   nor2_1 U72 (.Z (N190), .A (N66), .B (N135));
42   nor2_1 U73 (.Z (N191), .A (N73), .B (N139));
43   nor2_1 U74 (.Z (N192), .A (N79), .B (N139));
44   nor2_1 U75 (.Z (N193), .A (N86), .B (N143));
45   nor2_1 U76 (.Z (N194), .A (N92), .B (N143));
46   nor2_1 U77 (.Z (N195), .A (N99), .B (N147));
47   nor2_1 U78 (.Z (N196), .A (N105), .B (N147));
48   nor2_1 U79 (.Z (N197), .A (N112), .B (N151));
49   nor2_1 U80 (.Z (N198), .A (N115), .B (N151));
50   and9_1 U81 (.Z (N199), .A (N154), .B (N159), .C (N162), .D (N165),
   .E (N168), .F (N171), .G (N174), .H (N177), .I (N180));
51   inv_1 U82 (.Z (N203), .A (N199));
52   inv_1 U83 (.Z (N213), .A (N199));
53   inv_1 U84 (.Z (N223), .A (N199));
54   xor2 U85 (.Z (N224), .A (N203), .B (N154));
55   xor2 U86 (.Z (N227), .A (N203), .B (N159));
56   xor2 U87 (.Z (N230), .A (N203), .B (N162));
57   xor2 U88 (.Z (N233), .A (N203), .B (N165));
58   xor2 U89 (.Z (N236), .A (N203), .B (N168));
59   xor2 U90 (.Z (N239), .A (N203), .B (N171));
60   nand2_1 U91 (.Z (N242), .A (N1), .B (N213));
61   xor2 U92 (.Z (N243), .A (N203), .B (N174));
62   nand2_1 U93 (.Z (N246), .A (N213), .B (N11));
63   xor2 U94 (.Z (N247), .A (N203), .B (N177));
64   nand2_1 U95 (.Z (N250), .A (N213), .B (N24));
65   xor2 U96 (.Z (N251), .A (N203), .B (N180));
66   nand2_1 U97 (.Z (N254), .A (N213), .B (N37));
67   nand2_1 U98 (.Z (N255), .A (N213), .B (N50));
68   nand2_1 U99 (.Z (N256), .A (N213), .B (N63));
69   nand2_1 U100 (.Z (N257), .A (N213), .B (N76));
70   nand2_1 U101 (.Z (N258), .A (N213), .B (N89));
71   nand2_1 U102 (.Z (N259), .A (N213), .B (N102));
72   nand2_1 U103 (.Z (N260), .A (N224), .B (N157));
73   nand2_1 U104 (.Z (N263), .A (N224), .B (N158));
74   nand2_1 U105 (.Z (N264), .A (N227), .B (N183));
75   nand2_1 U106 (.Z (N267), .A (N230), .B (N185));
```

```
76   nand2_1 U107 (.Z (N270), .A (N233), .B (N187));
77   nand2_1 U108 (.Z (N273), .A (N236), .B (N189));
78   nand2_1 U109 (.Z (N276), .A (N239), .B (N191));
79   nand2_1 U110 (.Z (N279), .A (N243), .B (N193));
80   nand2_1 U111 (.Z (N282), .A (N247), .B (N195));
81   nand2_1 U112 (.Z (N285), .A (N251), .B (N197));
82   nand2_1 U113 (.Z (N288), .A (N227), .B (N184));
83   nand2_1 U114 (.Z (N289), .A (N230), .B (N186));
84   nand2_1 U115 (.Z (N290), .A (N233), .B (N188));
85   nand2_1 U116 (.Z (N291), .A (N236), .B (N190));
86   nand2_1 U117 (.Z (N292), .A (N239), .B (N192));
87   nand2_1 U118 (.Z (N293), .A (N243), .B (N194));
88   nand2_1 U119 (.Z (N294), .A (N247), .B (N196));
89   nand2_1 U120 (.Z (N295), .A (N251), .B (N198));
90   and9_1 U121 (.Z (N296), .A (N260), .B (N264), .C (N267), .D (N270),
    .E (N273), .F (N276), .G (N279), .H (N282), .I (N285));
91   inv_1 U122 (.Z (N300), .A (N263));
92   inv_1 U123 (.Z (N301), .A (N288));
93   inv_1 U124 (.Z (N302), .A (N289));
94   inv_1 U125 (.Z (N303), .A (N290));
95   inv_1 U126 (.Z (N304), .A (N291));
96   inv_1 U127 (.Z (N305), .A (N292));
97   inv_1 U128 (.Z (N306), .A (N293));
98   inv_1 U129 (.Z (N307), .A (N294));
99   inv_1 U130 (.Z (N308), .A (N295));
100   inv_1 U131 (.Z (N309), .A (N296));
101   inv_1 U132 (.Z (N319), .A (N296));
102   inv_1 U133 (.Z (N329), .A (N296));
103   xor2 U134 (.Z (N330), .A (N309), .B (N260));
104   xor2 U135 (.Z (N331), .A (N309), .B (N264));
105   xor2 U136 (.Z (N332), .A (N309), .B (N267));
106   xor2 U137 (.Z (N333), .A (N309), .B (N270));
107   nand2_1 U138 (.Z (N334), .A (N8), .B (N319));
108   xor2 U139 (.Z (N335), .A (N309), .B (N273));
109   nand2_1 U140 (.Z (N336), .A (N319), .B (N21));
110   xor2 U141 (.Z (N337), .A (N309), .B (N276));
111   nand2_1 U142 (.Z (N338), .A (N319), .B (N34));
112   xor2 U143 (.Z (N339), .A (N309), .B (N279));
```

```
113   nand2_1 U144 (.Z (N340), .A (N319), .B (N47));
114   xor2 U145 (.Z (N341), .A (N309), .B (N282));
115   nand2_1 U146 (.Z (N342), .A (N319), .B (N60));
116   xor2 U147 (.Z (N343), .A (N309), .B (N285));
117   nand2_1 U148 (.Z (N344), .A (N319), .B (N73));
118   nand2_1 U149 (.Z (N345), .A (N319), .B (N86));
119   nand2_1 U150 (.Z (N346), .A (N319), .B (N99));
120   nand2_1 U151 (.Z (N347), .A (N319), .B (N112));
121   nand2_1 U152 (.Z (N348), .A (N330), .B (N300));
122   nand2_1 U153 (.Z (N349), .A (N331), .B (N301));
123   nand2_1 U154 (.Z (N350), .A (N332), .B (N302));
124   nand2_1 U155 (.Z (N351), .A (N333), .B (N303));
125   nand2_1 U156 (.Z (N352), .A (N335), .B (N304));
126   nand2_1 U157 (.Z (N353), .A (N337), .B (N305));
127   nand2_1 U158 (.Z (N354), .A (N339), .B (N306));
128   nand2_1 U159 (.Z (N355), .A (N341), .B (N307));
129   nand2_1 U160 (.Z (N356), .A (N343), .B (N308));
130   and9_1 U161 (.Z (N357), .A (N348), .B (N349), .C (N350), .D (N351),
   .E (N352), .F (N353), .G (N354), .H (N355), .I (N356));
131   inv_1 U162 (.Z (N360), .A (N357));
132   inv_1 U163 (.Z (N370), .A (N357));
133   nand2_1 U164 (.Z (N371), .A (N14), .B (N360));
134   nand2_1 U165 (.Z (N372), .A (N360), .B (N27));
135   nand2_1 U166 (.Z (N373), .A (N360), .B (N40));
136   nand2_1 U167 (.Z (N374), .A (N360), .B (N53));
137   nand2_1 U168 (.Z (N375), .A (N360), .B (N66));
138   nand2_1 U169 (.Z (N376), .A (N360), .B (N79));
139   nand2_1 U170 (.Z (N377), .A (N360), .B (N92));
140   nand2_1 U171 (.Z (N378), .A (N360), .B (N105));
141   nand2_1 U172 (.Z (N379), .A (N360), .B (N115));
142   nand4_1 U173 (.Z (N380), .A (N4), .B (N242), .C (N334), .D (N371));
143   nand4_1 U174 (.Z (N381), .A (N246), .B (N336), .C (N372), .D (N17));
144   nand4_1 U175 (.Z (N386), .A (N250), .B (N338), .C (N373), .D (N30));
145   nand4_1 U176 (.Z (N393), .A (N254), .B (N340), .C (N374), .D (N43));
146   nand4_1 U177 (.Z (N399), .A (N255), .B (N342), .C (N375), .D (N56));
147   nand4_1 U178 (.Z (N404), .A (N256), .B (N344), .C (N376), .D (N69));
148   nand4_1 U179 (.Z (N407), .A (N257), .B (N345), .C (N377), .D (N82));
149   nand4_1 U180 (.Z (N411), .A (N258), .B (N346), .C (N378), .D (N95));
```

```
150   nand4_1 U181 (.Z (N414), .A (N259), .B (N347), .C (N379), .D (N108));
151   inv_1 U182 (.Z (N415), .A (N380));
152   and8_1 U183 (.Z (N416), .A (N381), .B (N386), .C (N393), .D (N399),
   .E (N404), .F (N407), .G (N411), .H (N414));
153   inv_1 U184 (.Z (N417), .A (N393));
154   inv_1 U185 (.Z (N418), .A (N404));
155   inv_1 U186 (.Z (N419), .A (N407));
156   inv_1 U187 (.Z (N420), .A (N411));
157   nor2_1 U188 (.Z (Nremp), .A (N415), .B (N416));
158   nand2_1 U189 (.Z (N422), .A (N386), .B (N417));
159   nand4_1 U190 (.Z (N425), .A (N386), .B (N393), .C (N418), .D (N399));
160   nand3_1 U191 (.Z (N428), .A (N399), .B (N393), .C (N419));
161   nand4_1 U192 (.Z (N429), .A (N386), .B (N393), .C (N407), .D (N420));
162   nand4_1 U193 (.Z (N430), .A (N381), .B (N386), .C (N422), .D (N399));
163   nand4_1 U194 (.Z (N431), .A (N381), .B (N386), .C (N425), .D (N428));
164   nand4_1 U195 (.Z (N432), .A (N381), .B (N422), .C (N425), .D (N429));

165   assign N421=0;
166   endmodule
```

Exemple 2 :

Injection de la faute n°136 de la liste des fautes, collage à 1 de l'entrée A de l'inverseur U36

```
1   /*collage a 1 de l'entrée A de U36 */
2   module c432modifie_136 (N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
              N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
              N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
              N99, N102, N105, N108, N112, N115, N223, N329, N370, N421,
              N430, N431, N432);
3   input N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
              N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
              N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
              N99, N102, N105, N108, N112, N115;
4   outputs N223, N329, N370, N421, N430, N431, N432;

   5    inv_1 U36 (.Z (N118), .A ((Nremp)) ;
6    inv_1 U37 (.Z (N119), .A (N4));
7    inv_1 U38 (.Z (N122), .A (N11));
```

```
8    inv_1 U39 (.Z (N123), .A (N17));
9    inv_1 U40 (.Z (N126), .A (N24));
10   inv_1 U41 (.Z (N127), .A (N30));
11   inv_1 U42 (.Z (N130), .A (N37));
12   inv_1 U43 (.Z (N131), .A (N43));
13   inv_1 U44 (.Z (N134), .A (N50));
14   inv_1 U45 (.Z (N135), .A (N56));
15   inv_1 U46 (.Z (N138), .A (N63));
16   inv_1 U47 (.Z (N139), .A (N69));
17   inv_1 U48 (.Z (N142), .A (N76));
18   inv_1 U49 (.Z (N143), .A (N82));
19   inv_1 U50 (.Z (N146), .A (N89));
20   inv_1 U51 (.Z (N147), .A (N95));
21   inv_1 U52 (.Z (N150), .A (N102));
22   inv_1 U53 (.Z (N151), .A (N108));
23   nand2_1 U54 (.Z (N154), .A (N118), .B (N4));
24   nor2_1 U55 (.Z (N157), .A (N8), .B (N119));
25   nor2_1 U56 (.Z (N158), .A (N14), .B (N119));
26   nand2_1 U57 (.Z (N159), .A (N122), .B (N17));
27   nand2_1 U58 (.Z (N162), .A (N126), .B (N30));
28   nand2_1 U59 (.Z (N165), .A (N130), .B (N43));
29   nand2_1 U60 (.Z (N168), .A (N134), .B (N56));
30   nand2_1 U61 (.Z (N171), .A (N138), .B (N69));
31   nand2_1 U62 (.Z (N174), .A (N142), .B (N82));
32   nand2_1 U63 (.Z (N177), .A (N146), .B (N95));
33   nand2_1 U64 (.Z (N180), .A (N150), .B (N108));
34   nor2_1 U65 (.Z (N183), .A (N21), .B (N123));
35   nor2_1 U66 (.Z (N184), .A (N27), .B (N123));
36   nor2_1 U67 (.Z (N185), .A (N34), .B (N127));
37   nor2_1 U68 (.Z (N186), .A (N40), .B (N127));
38   nor2_1 U69 (.Z (N187), .A (N47), .B (N131));
39   nor2_1 U70 (.Z (N188), .A (N53), .B (N131));
40   nor2_1 U71 (.Z (N189), .A (N60), .B (N135));
41   nor2_1 U72 (.Z (N190), .A (N66), .B (N135));
42   nor2_1 U73 (.Z (N191), .A (N73), .B (N139));
43   nor2_1 U74 (.Z (N192), .A (N79), .B (N139));
44   nor2_1 U75 (.Z (N193), .A (N86), .B (N143));
45   nor2_1 U76 (.Z (N194), .A (N92), .B (N143));
```

```
46   nor2_1 U77 (.Z (N195), .A (N99), .B (N147));
47   nor2_1 U78 (.Z (N196), .A (N105), .B (N147));
48   nor2_1 U79 (.Z (N197), .A (N112), .B (N151));
49   nor2_1 U80 (.Z (N198), .A (N115), .B (N151));
50   and9_1 U81 (.Z (N199), .A (N154), .B (N159), .C (N162), .D (N165),
   .E (N168), .F (N171), .G (N174), .H (N177), .I (N180));
51   inv_1 U82 (.Z (N203), .A (N199));
52   inv_1 U83 (.Z (N213), .A (N199));
53   inv_1 U84 (.Z (N223), .A (N199));
54   xor2 U85 (.Z (N224), .A (N203), .B (N154));
55   xor2 U86 (.Z (N227), .A (N203), .B (N159));
56   xor2 U87 (.Z (N230), .A (N203), .B (N162));
57   xor2 U88 (.Z (N233), .A (N203), .B (N165));
58   xor2 U89 (.Z (N236), .A (N203), .B (N168));
59   xor2 U90 (.Z (N239), .A (N203), .B (N171));
60   nand2_1 U91 (.Z (N242), .A (N1), .B (N213));
61   xor2 U92 (.Z (N243), .A (N203), .B (N174));
62   nand2_1 U93 (.Z (N246), .A (N213), .B (N11));
63   xor2 U94 (.Z (N247), .A (N203), .B (N177));
64   nand2_1 U95 (.Z (N250), .A (N213), .B (N24));
65   xor2 U96 (.Z (N251), .A (N203), .B (N180));
66   nand2_1 U97 (.Z (N254), .A (N213), .B (N37));
67   nand2_1 U98 (.Z (N255), .A (N213), .B (N50));
68   nand2_1 U99 (.Z (N256), .A (N213), .B (N63));
69   nand2_1 U100 (.Z (N257), .A (N213), .B (N76));
70   nand2_1 U101 (.Z (N258), .A (N213), .B (N89));
71   nand2_1 U102 (.Z (N259), .A (N213), .B (N102));
72   nand2_1 U103 (.Z (N260), .A (N224), .B (N157));
73   nand2_1 U104 (.Z (N263), .A (N224), .B (N158));
74   nand2_1 U105 (.Z (N264), .A (N227), .B (N183));
75   nand2_1 U106 (.Z (N267), .A (N230), .B (N185));
76   nand2_1 U107 (.Z (N270), .A (N233), .B (N187));
77   nand2_1 U108 (.Z (N273), .A (N236), .B (N189));
78   nand2_1 U109 (.Z (N276), .A (N239), .B (N191));
79   nand2_1 U110 (.Z (N279), .A (N243), .B (N193));
80   nand2_1 U111 (.Z (N282), .A (N247), .B (N195));
81   nand2_1 U112 (.Z (N285), .A (N251), .B (N197));
82   nand2_1 U113 (.Z (N288), .A (N227), .B (N184));
```

```
83   nand2_1 U114 (.Z (N289), .A (N230), .B (N186));
84   nand2_1 U115 (.Z (N290), .A (N233), .B (N188));
85   nand2_1 U116 (.Z (N291), .A (N236), .B (N190));
86   nand2_1 U117 (.Z (N292), .A (N239), .B (N192));
87   nand2_1 U118 (.Z (N293), .A (N243), .B (N194));
88   nand2_1 U119 (.Z (N294), .A (N247), .B (N196));
89   nand2_1 U120 (.Z (N295), .A (N251), .B (N198));
90   and9_1 U121 (.Z (N296), .A (N260), .B (N264), .C (N267), .D (N270),
    .E (N273), .F (N276), .G (N279), .H (N282), .I (N285));
91   inv_1 U122 (.Z (N300), .A (N263));
92   inv_1 U123 (.Z (N301), .A (N288));
93   inv_1 U124 (.Z (N302), .A (N289));
94   inv_1 U125 (.Z (N303), .A (N290));
95   inv_1 U126 (.Z (N304), .A (N291));
96   inv_1 U127 (.Z (N305), .A (N292));
97   inv_1 U128 (.Z (N306), .A (N293));
98   inv_1 U129 (.Z (N307), .A (N294));
99   inv_1 U130 (.Z (N308), .A (N295));
100   inv_1 U131 (.Z (N309), .A (N296));
101   inv_1 U132 (.Z (N319), .A (N296));
102   inv_1 U133 (.Z (N329), .A (N296));
103   xor2 U134 (.Z (N330), .A (N309), .B (N260));
104   xor2 U135 (.Z (N331), .A (N309), .B (N264));
105   xor2 U136 (.Z (N332), .A (N309), .B (N267));
106   xor2 U137 (.Z (N333), .A (N309), .B (N270));
107   nand2_1 U138 (.Z (N334), .A (N8), .B (N319));
108   xor2 U139 (.Z (N335), .A (N309), .B (N273));
109   nand2_1 U140 (.Z (N336), .A (N319), .B (N21));
110   xor2 U141 (.Z (N337), .A (N309), .B (N276));
111   nand2_1 U142 (.Z (N338), .A (N319), .B (N34));
112   xor2 U143 (.Z (N339), .A (N309), .B (N279));
113   nand2_1 U144 (.Z (N340), .A (N319), .B (N47));
114   xor2 U145 (.Z (N341), .A (N309), .B (N282));
115   nand2_1 U146 (.Z (N342), .A (N319), .B (N60));
116   xor2 U147 (.Z (N343), .A (N309), .B (N285));
117   nand2_1 U148 (.Z (N344), .A (N319), .B (N73));
118   nand2_1 U149 (.Z (N345), .A (N319), .B (N86));
119   nand2_1 U150 (.Z (N346), .A (N319), .B (N99));
```

```
120   nand2_1 U151 (.Z (N347), .A (N319), .B (N112));
121   nand2_1 U152 (.Z (N348), .A (N330), .B (N300));
122   nand2_1 U153 (.Z (N349), .A (N331), .B (N301));
123   nand2_1 U154 (.Z (N350), .A (N332), .B (N302));
124   nand2_1 U155 (.Z (N351), .A (N333), .B (N303));
125   nand2_1 U156 (.Z (N352), .A (N335), .B (N304));
126   nand2_1 U157 (.Z (N353), .A (N337), .B (N305));
127   nand2_1 U158 (.Z (N354), .A (N339), .B (N306));
128   nand2_1 U159 (.Z (N355), .A (N341), .B (N307));
129   nand2_1 U160 (.Z (N356), .A (N343), .B (N308));
130   and9_1 U161 (.Z (N357), .A (N348), .B (N349), .C (N350), .D (N351),
   .E (N352), .F (N353), .G (N354), .H (N355), .I (N356));
131   inv_1 U162 (.Z (N360), .A (N357));
132   inv_1 U163 (.Z (N370), .A (N357));
133   nand2_1 U164 (.Z (N371), .A (N14), .B (N360));
134   nand2_1 U165 (.Z (N372), .A (N360), .B (N27));
135   nand2_1 U166 (.Z (N373), .A (N360), .B (N40));
136   nand2_1 U167 (.Z (N374), .A (N360), .B (N53));
137   nand2_1 U168 (.Z (N375), .A (N360), .B (N66));
138   nand2_1 U169 (.Z (N376), .A (N360), .B (N79));
139   nand2_1 U170 (.Z (N377), .A (N360), .B (N92));
140   nand2_1 U171 (.Z (N378), .A (N360), .B (N105));
141   nand2_1 U172 (.Z (N379), .A (N360), .B (N115));
142   nand4_1 U173 (.Z (N380), .A (N4), .B (N242), .C (N334), .D (N371));
143   nand4_1 U174 (.Z (N381), .A (N246), .B (N336), .C (N372), .D (N17));
  144   nand4_1 U175 (.Z (N386), .A (N250), .B (N338), .C (N373, .D (N30))
;
145   nand4_1 U176 (.Z (N393), .A (N254), .B (N340), .C (374), .D (N43));
146   nand4_1 U177 (.Z (N399), .A (N255), .B (N342), .C (N375), .D (N56));
147   nand4_1 U178 (.Z (N404), .A (N256), .B (N344), .C (N376), .D (N69));
148   nand4_1 U179 (.Z (N407), .A (N257), .B (N345), .C (N377), .D (N82));
149   nand4_1 U180 (.Z (N411), .A (N258), .B (N346), .C (N378), .D (N95));
150   nand4_1 U181 (.Z (N414), .A (N259), .B (N347), .C (N379), .D (N108));
151   inv_1 U182 (.Z (N415), .A (N380));
152   and8_1 U183 (.Z (N416), .A (N381), .B (N386), .C (N393), .D (N399),
   .E (N404), .F (N407), .G (N411), .H (N414));
153   inv_1 U184 (.Z (N417), .A (N393));
154   inv_1 U185 (.Z (N418), .A (N404));
```

```
155   inv_1 U186 (.Z (N419), .A (N407));
156   inv_1 U187 (.Z (N420), .A (N411));
157   nor2_1 U188 (.Z (N421), .A (N415), .B (N416));
158   nand2_1 U189 (.Z (N422), .A (N386), .B (N417));
159   nand4_1 U190 (.Z (N425), .A (N386), .B (N393), .C (N418), .D (N399));
160   nand3_1 U191 (.Z (N428), .A (N399), .B (N393), .C (N419));
161   nand4_1 U192 (.Z (N429), .A (N386), .B (N393), .C (N407), .D (N420));
162   nand4_1 U193 (.Z (N430), .A (N381), .B (N386), .C (N422), .D (N399));
163   nand4_1 U194 (.Z (N431), .A (N381), .B (N386), .C (N425), .D (N428));
164   nand4_1 U195 (.Z (N432), .A (N381), .B (N422), .C (N425), .D (N429));
```

```
165   assign Nremp=1;
166   endmodule
```

4ème Étape : Création du script pour TetraMax

Puisque les trois circuits sont identiques, on injecte des fautes seulement dans T1, la première faute injectée est f1 « collage de l'entrée du premier inverseur à 0 ». Avec f2 une faute présente dans la liste de faute générée par TetraMax et qui se trouve dans T2. Le script TetraMax décrivant l'ATPG du circuit c432 est le suivant :

```
1   read netlist /auto//ait /c432/c432modifie_1.v
2   read netlist /auto/ait /circuits/c432/c432.v
3   read netlist /auto/ait /circuits/c432/TMRc432.v
4   read netlist /auto/ait/circuits/lib_comb.v

5   run build_model TMRc432
6   run drc

7   add faults T2/1 gat inpt /A –stuck-at  0
8   add faults T2/1gat inpt /A –stuck 1
9   add faults T2/4 gat inpt /A –stuck 0
10  add faults T2/ 4 gat inpt/ A –stuck1
11  add faults T2/ 8 gat inpt/ A –stuck0
12  add faults T2/ 8 gat inpt/ A –stuck1
13  add faults T2/ 11gat inpt/A –stuck0
14  add faults T2/11gat inpt/ A –stuck1
15  add faults T2/14gat inpt /A –stuck0
16  add faults T2/14gat inpt /A –stuck1
```

17 add faults T2/17gat inpt /A –stuck0
18 add faults T2/17gat inpt /A –stuck1
19 add faults T2/21gat inpt /A –stuck0
20 add faults T2/21gat inpt /A –stuck1
21 add faults T2/24gat inpt /A –stuck0
22 add faults T2/24gat inpt /A –stuck1
23 add faults T2/27gat inpt /A –stuck0
24 add faults T2/27gat inpt /A –stuck1
25 add faults T2/30gat inpt /A –stuck0
26 add faults T2/30gat inpt /A –stuck1
27 add faults T2/34gat inpt /A –stuck0
28 add faults T2/34gat inpt /A –stuck1
29 add faults T2/37gat inpt /A –stuck0
30 add faults T2/37gat inpt /A –stuck1

31 add faults T2/40gat inpt /A –stuck0
32 add faults T2/40gat inpt /A –stuck1
33 add faults T2/43gat inpt/A–stuck0
34 add faults T2/43gat inpt/A–stuck1
35 add faults T2/47gat inpt/A–stuck0
36 add faults T2/47gat inpt/A–stuck1
37 add faults T2/50gat inpt/A–stuck0
38 add faults T2/50gat inpt/A–stuck1
39 add faults T2/53gat inpt/A–stuck0
40 add faults T2/53gat inpt/A–stuck1
41 add faults T2/56gat inpt/A–stuck0
42 add faults T2/56gat inpt/A–stuck1
43 add faults T2/60gat inpt/A–stuck0
44 add faults T2/60gat inpt/A–stuck1
45 add faults T2/63gat inpt/A–stuck0
46 add faults T2/63gat inpt/A–stuck1
47 add faults T2/66gat inpt/A–stuck0
48 add faults T2/66gat inpt/A–stuck1
49 add faults T2/69gat inpt/A–stuck0
50 add faults T2/69gat inpt/A–stuck1
51 add faults T2/73gat inpt/A–stuck0
52 add faults T2/73gat inpt/A–stuck1
53 add faults T2/76gat inpt/A–stuck0

```
54  add faults T2/76gat inpt/A–stuck1
55  add faults T2/79gat inpt/A–stuck0
56  add faults T2/79gat inpt/A–stuck1
57  add faults T2/82gat inpt/A–stuck0
58  add faults T2/82gat inpt/A–stuck1
59  add faults T2/86gat inpt/A–stuck0
60  add faults T2/86gat inpt/A–stuck1
61  add faults T2/89gat inpt/A–stuck0
62  add faults T2/89gat inpt/A–stuck1
63  add faults T2/92gat inpt/A–stuck0
64  add faults T2/92gat inpt/A–stuck1
65  add faults T2/95gat inpt/A–stuck0
66  add faults T2/95gat inpt/A–stuck1
67  add faults T2/99gat inpt/A–stuck0
68  add faults T2/99gat inpt/A–stuck1
69  add faults T2/102gat inpt/A–stuck0
70  add faults T2/102gat inpt/A–stuck1
71  add faults T2/105gat inpt/A–stuck0
72  add faults T2/105gat inpt/A–stuck1
73  add faults T2/108gat inpt/A–stuck0
74  add faults T2/108gat inpt/A–stuck1
75  add faults T2/112gat inpt/A–stuck0
76  add faults T2/112gat inpt/A–stuck1
77  add faults T2/115gat inpt/A–stuck0
78  add faults T2/115gat inpt/A–stuck1
79  add faults T2/118gat not/Z–stuck1
80  add faults T2/119gat not /Z–stuck0
81  add faults T2/119gat not /Z–stuck1
82  add faults T2/122gat not /Z–stuck1
83  add faults T2/123gat not /Z–stuck0
84  add faults T2/123gat not /Z–stuck1
85  add faults T2/126gat not /Z–stuck1
86  add faults T2/127gat not /Z–stuck0
87  add faults T2/127gat not /Z–stuck1
88  add faults T2/130gat not /Z–stuck1
89  add faults T2/131gat not /Z–stuck0
90  add faults T2/131gat not /Z–stuck1
91  add faults T2/134gat not /Z–stuck1
```

92 add faults T2/135gat not /Z–stuck0
93 add faults T2/135gat not /Z–stuck1
94 add faults T2/138gat not /Z–stuck1
95 add faults T2/139gat not /Z–stuck0
96 add faults T2/139gat not /Z–stuck1
97 add faults T2/142gat not /Z–stuck1
98 add faults T2/143gat not /Z–stuck0
99 add faults T2/143gat not /Z–stuck1
100 add faults T2/146gat not /Z–stuck1
101 add faults T2/147gat not /Z–stuck0
102 add faults T2/147gat not /Z–stuck1
103 add faults T2/150gat not /Z–stuck1
104 add faults T2/151gat not /Z–stuck0
105 add faults T2/151gat not /Z–stuck1
106 add faults T2/154gat nand /Z–stuck0
107 add faults T2/154gat nand /Z–stuck1
108 add faults T2/157gat nor /Z–stuck1
109 add faults T2/158gat nor/Z–stuck1
110 add faults T2/159gat nand/ Z–stuck0
111 add faults T2/159gat nand/ Z–stuck1
112 add faults T2/162gat nand/ Z–stuck0
113 add faults T2/162gat nand/ Z–stuck1
114 add faults T2/165gat nand/ Z–stuck0
115 add faults T2/165gat nand/ Z–stuck1
116 add faults T2/168gat nand/ Z–stuck0
117 add faults T2/168gat nand/ Z–stuck1
118 add faults T2/171gat nand/ Z–stuck0
119 add faults T2/171gat nand/ Z–stuck1
120 add faults T2/174gat nand/ Z–stuck0
121 add faults T2/174gat nand/ Z–stuck1
122 add faults T2/177gat nand/ Z–stuck0
123 add faults T2/177gat nand/ Z–stuck1
124 add faults T2/180gat nand/ Z–stuck0
125 add faults T2/180gat nand/ Z–stuck1
126 add faults T2/183gat nor/ Z–stuck1
127 add faults T2/184gat nor/ Z–stuck1
128 add faults T2/185gat nor/ Z–stuck1
129 add faults T2/186gat nor/ Z–stuck1

```
130  add faults T2/187gat nor/ Z–stuck1
131  add faults T2/188gat nor/ Z–stuck1
132  add faults T2/189gat nor/ Z–stuck1
133  add faults T2/190gat nor/ Z–stuck1
134  add faults T2/191gat nor/ Z–stuck1
135  add faults T2/192gat nor/ Z–stuck1
136  add faults T2/193gat nor/ Z–stuck1
137  add faults T2/194gat nor/ Z–stuck1
138  add faults T2/195gat nor/ Z–stuck1
139  add faults T2/196gat nor/ Z–stuck1
140  add faults T2/197gat nor/ Z–stuck1
141  add faults T2/198gat nor/ Z–stuck1
142  add faults T2/199gat and/ Z–stuck0
143  add faults T2/199gat and/ Z–stuck1
144  add faults T2/203gat not/ Z–stuck0
145  add faults T2/203gat not/ Z–stuck1
146  add faults T2/213gat not/ Z–stuck0
147  add faults T2/213gat not/ Z–stuck1
148  add faults T2/223gat not/ Z–stuck0
149  add faults T2/223gat not/ Z–stuck1
150  add faults T2/224gat xor/ Z–stuck0
151  add faults T2/224gat xor/ Z–stuck1
152  add faults T2/227gat xor/ Z–stuck0
153  add faults T2/227gat xor/ Z–stuck1
154  add faults T2/230gat xor/ Z–stuck0
155  add faults T2/230gat xor/ Z–stuck1
156  add faults T2/233gat xor/ Z–stuck0
157  add faults T2/233gat xor/ Z–stuck1
158  add faults T2/236gat xor/ Z–stuck0
159  add faults T2/236gat xor/ Z–stuck1
160  add faults T2/239gat xor/ Z–stuck0
161  add faults T2/239gat xor/ Z–stuck1
162  add faults T2/242gat nand/ Z–stuck1
163  add faults T2/243gat xor/ Z–stuck0
164  add faults T2/243gat xor/ Z–stuck1
165  add faults T2/246gat nand/ Z–stuck1
166  add faults T2/247gat xor/ Z–stuck0
167  add faults T2/247gat xor/ Z–stuck1
```

168 add faults T2/250gat nand/ Z–stuck1
169 add faults T2/251gat xor/ Z–stuck0
170 add faults T2/251gat xor/ Z–stuck1
171 add faults T2/254gat nand/ Z–stuck1
172 add faults T2/255gat nand/ Z–stuck1
173 add faults T2/256gat nand/ Z–stuck1
174 add faults T2/257gat nand/ Z–stuck1
175 add faults T2/258gat nand/ Z–stuck1
176 add faults T2/259gat nand/ Z–stuck1
177 add faults T2/260gat nand/ Z–stuck0
178 add faults T2/260gat nand/ Z–stuck1
179 add faults T2/264gat nand/ Z–stuck0
180 add faults T2/264gat nand/ Z–stuck1
181 add faults T2/267gat nand/ Z–stuck0
182 add faults T2/267gat nand/ Z–stuck1
183 add faults T2/270gat nand/ Z–stuck0
184 add faults T2/270gat nand/ Z–stuck1
185 add faults T2/273gat nand/ Z–stuck0
186 add faults T2/273gat nand/ Z–stuck1
187 add faults T2/276gat nand/ Z–stuck0
188 add faults T2/276gat nand/ Z–stuck1
189 add faults T2/279gat nand/ Z–stuck0
190 add faults T2/279gat nand/ Z–stuck1
191 add faults T2/282gat nand/ Z–stuck0
192 add faults T2/282gat nand/ Z–stuck1
193 add faults T2/285gat nand/ Z–stuck0
194 add faults T2/285gat nand/ Z–stuck1
195 add faults T2/296gat nand/ Z–stuck0
196 add faults T2/296gat nand/ Z–stuck1
197 add faults T2/300gat not/ Z–stuck1
198 add faults T2/301gat not/ Z–stuck1
199 add faults T2/302gat not/ Z–stuck1
200 add faults T2/303gat not/ Z–stuck1
201 add faults T2/304gat not/ Z–stuck1
202 add faults T2/305gat not/ Z–stuck1
203 add faults T2/306gat not/ Z–stuck1
204 add faults T2/307gat not/ Z–stuck1
205 add faults T2/307gat not/ Z–stuck1

206 add faults T2/307gat not/ Z–stuck1
207 add faults T2/308gat not/ Z–stuck1
208 add faults T2/309gat not/ Z–stuck0
209 add faults T2/309gat not/ Z–stuck1
210 add faults T2/319gat not/ Z–stuck0
211 add faults T2/319gat not/ Z–stuck1
212 add faults T2/329gat not/ Z–stuck0
213 add faults T2/329gat not/ Z–stuck1
214 add faults T2/330gat xor/ Z–stuck1
215 add faults T2/331gat xor/ Z–stuck1
216 add faults T2/332gat xor/ Z–stuck1
217 add faults T2/333gat xor/ Z–stuck1
218 add faults T2/334gat nand/ Z–stuck1
219 add faults T2/335gat xor/ Z–stuck1
220 add faults T2/336gat nand/ Z–stuck1
221 add faults T2/337gat xor/ Z–stuck1
222 add faults T2/338gat nand/ Z–stuck1
223 add faults T2/339gat xor/ Z–stuck1
224 add faults T2/340gat nand/ Z–stuck1
225 add faults T2/341gat xor/ Z–stuck1
226 add faults T2/342gat nand/ Z–stuck1
227 add faults T2/343gat xor/ Z–stuck1
228 add faults T2/344gat nand/ Z–stuck1
229 add faults T2/345gat nand/ Z–stuck1
230 add faults T2/346gat nand/ Z–stuck1
231 add faults T2/347gat nand/ Z–stuck1
232 add faults T2/348gat nand/ Z–stuck1
233 add faults T2/349gat nand/ Z–stuck1
234 add faults T2/350gat nand/ Z–stuck1
235 add faults T2/351gat nand/ Z–stuck1
236 add faults T2/352gat nand/ Z–stuck1
237 add faults T2/353gat nand/ Z–stuck1
238 add faults T2/354gat nand/ Z–stuck1
239 add faults T2/355gat nand/ Z–stuck1
240 add faults T2/356gat nand/ Z–stuck1
241 add faults T2/357gat nand/ Z–stuck0
242 add faults T2/357gat nand/ Z–stuck1
243 add faults T2/360gat not/ Z–stuck0

```
244  add faults T2/360gat not/ Z–stuck1
245  add faults T2/370gat not/ Z–stuck0
246  add faults T2/370gat not/ Z–stuck1
247  add faults T2/371gat nand/ Z–stuck1
248  add faults T2/372gat nand/ Z–stuck1
249  add faults T2/373gat nand/ Z–stuck1
250  add faults T2/374gat nand/ Z–stuck1
251  add faults T2/375gat nand/ Z–stuck1
252  add faults T2/376gat nand/ Z–stuck1
253  add faults T2/377gat nand/ Z–stuck1
254  add faults T2/378gat nand/ Z–stuck1
255  add faults T2/379gat nand/ Z–stuck1
256  add faults T2/381gat nand/ Z–stuck0
257  add faults T2/386gat nand/ Z–stuck0
258  add faults T2/386gat nand/ Z–stuck1
259  add faults T2/393gat nand/ Z–stuck0
260  add faults T2/393gat nand/ Z–stuck1
261  add faults T2/399gat nand/ Z–stuck0
262  add faults T2/399gat nand/ Z–stuck1
263  add faults T2/404gat nand/ Z–stuck0
264  add faults T2/404gat nand/ Z–stuck1
265  add faults T2/407gat nand/ Z–stuck0
266  add faults T2/407gat nand/ Z–stuck1
267  add faults T2/411gat nand/ Z–stuck0
268  add faults T2/411gat nand/ Z–stuck1
269  add faults T2/414gat nand/ Z–stuck1
270  add faults T2/415gat not/ Z–stuck0
271  add faults T2/416gat and/ Z–stuck0
272  add faults T2/417gat not/ Z–stuck1
273  add faults T2/418gat not/ Z–stuck1
274  add faults T2/419gat not/ Z–stuck1
275  add faults T2/420gat not/ Z–stuck1
276  add faults T2/421gat nor/Z–stuck0
277  add faults T2/421gat nor/Z–stuck1
278  add faults T2/422gat nand/ Z–stuck0
279  add faults T2/422gat nand/ Z–stuck1
280  add faults T2/425gat nand/ Z–stuck0
281  add faults T2/425gat nand/ Z–stuck1
```

```
 282  add faults T2/428gat nand/ Z–stuck1
 283  add faults T2/429gat nand/ Z–stuck1
 284  add faults T2/430gat nand/ Z–stuck0
 285  add faults T2/430gat nand/ Z–stuck1
 286  add faults T2/431gat nand/ Z–stuck0
 287  add faults T2/431gat nand/ Z–stuck1
 288  add faults T2/432gat nand/Z–stuck0
 289  add faults T2/432gat nand/Z–stuck1
290   set atpg -abort_limit 7200
291   set atpg -decision NOR
292   set atpg -norandom_fill
293   run atpg -ndetects 1

294   write patterns /auto/iscas/circuits/patterns/c432/pat1.txt –Internal -format verilog_single_file -
        parallel 0
295   drc -force
```

La description Verilog de la structure TMR, après l'injection de la première faute de la liste de fautes dans T1, collage à 0 de l'entrée N1est la suivante :

```
1  module TMRc432 (N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
                N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
                N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
                N99, N102, N105, N108, N112, N115, N223, N329, N370, N421,
                N430, N431, N432);

2   input N1,N4,N8,N11,N14,N17,N21,N24,N27,N30,
                N34, N37, N40, N43, N47, N50, N53, N56, N60, N63,
                N66, N69, N73, N76, N79, N82, N86, N89, N92, N95,
                N99, N102, N105, N108, N112, N115;
3   output N223, N329, N370, N421, N430, N431, N432;


4    c432modifie_1 T1 (E1, E2, E3, E4, E36, S1M1, S2M1,…S7M1);
5    c432 T2 (E1, E2, E3, E4,…E36, S1M2, S7M2);
6    c432 T3 (E1, E2, E3, E4,…E36, S1M3, S7M3);

7  voteur V0 (N223M1, N223M2, N223M3, N223);
```

```
8 voteur V1 (N329M1, 329 M 2, 329 M 3, N329);
9 voteur V2 (N370M1, N370M2, N370M3, N370);
10voteur V3 (N421M1, N421M2, N421M3, N421);
11voteur V4 (N430M1, N430M2, N430M3, N430);
12voteur V5 (N431M1, N431M2, N431M3, N431);
13voteur V6 (N432M1, N432M2, N432M3, N432);
14 endmodule
```

Références bibliographiques

[Abr80] M Abramovici. and M.A Breuer., "Multiple Fault Diagnosis in Combinational circuits based on an effect –cause analysis", IEEE Trans. On computers, vol. c29, n°6, pp 451-460, June 1980.

[Abr90] M Abramovici., A.B Melvin., D.A Friedman., "Digital systems testing and testable design", Revised, IEEE press, 1990.

[Arl79] J Arlat., " Conception d'un microcalculateur tolérant aux fautes par diversification fonctionnelle", thèse de docteur- ingénieur, INP, Toulouse, April 1979.

[Arm72] D. B Armstrong, IEEE Trans. On Computers, Vol. C-21, no 5, May 1972, pp. 464-471.

[Amo88] P Arnoult., "Minitel, codage de l'information et corps finis 43, revue pour la science,N° 125, mars 1988.

[Ber04] M Berg, "Methodologies for Reliable Design Implementation", Washington, D.C, september 2004.

[Bou07] A. Bounceur, " Plateforme CAO pour le test de circuits mixtes", thèse de doctorat, Institut National Polytechnique de Grenoble, 2007.

[Bqr09] BQR, "Reliable Ingineering Optimized Maintenance, (MTBF- CARE)", BQR, 2009.

[Brg89] F. Brglez, D. Bryant and K Kozminski, " Combinational Profiles of Sequential Benchmark circuits", IEEE Int. Symp. On circuits and systems. pp 1929-1934, 1989.

[Bri93] D Brière, P Traverse, " Airbus A320/A330/ A340 electrical flight controls- a family of tolerant systems", in Proc. 23 rd Int. Symp. On fault tolerant computing (FTCS-23), pp. 616-623, IEEE CS Press, 1993.

[Bur01] M Burns and G.W Roberts, "An Introduction to Mixed-Signal IC Test and Measurement", Oxford University Press, 2001.

[Caz04] J. M Cazeaux, D Rossi et C Metra, "New high speed Cmos self- cheking Voter", Proc. of IEEE international on lin e testing symposium, pp. 58-63, 2004.

[Cha74] Chang et al, "Comparison of parallel and deductive fault simulation methods"

[Che78] L. Chen, A. Avizienis, " N-version programming: a fault tolerance approach to reliability of software operation ", in Proc.8 th Int. symp. On fault tolerant computing (FTCS-8), pp. 3-9, IEEE CS Press, 1978.

[Cho85] Y.H. Choi, M. Malek, "A tolerant FFT processor", proceding of fault tolerant computing symposium, pp. 814-823, 1985.

[Cho01] A Chouki, Engineering Handbooks online, "Scan testing", CRC Press 2001.

[Chr03] M. Chrzanowska Jeske," Between failure, reliability, yield and IC layout", spring 2003.

[Cim07] M. Cimino, " Conception de circuits radiofréquences sous contraintes de fiabilité étendue", thèse de doctorat, Université Bordeaux I, 2007.

[Cou08] A. Coulibaly, "modélisation sémantique et évaluation de performances comportementales de produits en conception", mémoire d'habilitation à diriger des recherches, Université Louis Pasteur, Strasbourg I, 2008.

[Del10] D. Delahaye, "Evaluation quantitative", fascicule de cours sûreté de fonctionnement, 2009-2010.

[Del96] T. Delong et al, "A fault injection technique for VHDL Behavioral- Level Models", IEEE Design and Test of Computers, 1996, pp 24-33.

[Dum07] J.G Dumas, J.L Roch, E Tannier et S Varrette, "Thérie des codes « compression, cryptage, correction »", édition Dunod, ISBN 978-2100506927, 2007.

[Edm07] C. Edmond Bichot, "Élaboration d'une nouvelle métaheuristique pour le partitionnement de graphe : la méthode de fusion-fission, Application au découpage de l'espace aérien", thèse de doctorat, Institut National Polytechnique de Toulouse, 2007.

[Fan06] L Fang, M.S Hsiao, "Bilateral Testing of Nano-scale Fault Tolerant Circuits", Proc. of IEEE Defect and Fault Tolerance in VLSI Systems, pp.309-317, 2006.

[Fer98] A. Ferre, " I_{DDQ} testing: state of the art and future trends", Integration, the VLSI journal 26, pp 167-196, 1998.

[Fer90] A.V. Ferris-Prabhu, " A cluster- modified Poisson model for estimating defect density and yield " Semiconductor manufacturing, IEEE transactions on vol 3, Issue 2, pp.54-59, may 1990.

[Fuj83] H Fujiwra. and T Shimono, "On the acceleration of test generation algorithms", IEEE Trans. Comput, vol. c32, pp 1135-1144, dec. 1983.

[Gag97] Y Gagnon, "Are Defect- Tolerant Circuits with Redundancy Really Cost Effective? Complete and Realistic Cost Model", Proceedings of IEEE defect and fault tolerance of VLSI systems, pp.157-165, 1997.

[Goe80] P. Goel, Fault-Tolerant Computing Symp, Aug.1980, pp. 145-151.

[Gui00] L. Guiller, " Réduction de la consommation durant le test des circuits VLSI", thèse de doctorat, Université Montpellier II, 2000.

[Haf90] M. Hafezparast, "Fault tolerant hardware designs and their reliability analyses", thèse de doctorat, Brunel university, the university of west London, 1990.

[Ham05] K.Hamidi, O. Malassé and J.F.Aubry :"Coupling of information-flow aggregation method and dynamical model for a more accurate evaluation of reliability". In Proc. of the European Safety and Reliability Conference (ESREL05), Poland, June 2005.

[Ham95] K.Hamza, " The smallest uniform upper bound on the distance between the mean and the median of the binomial and Poisson distribution", statistic and probability letters, pp. 21-25, février 1995.

[Hsu97] M.C.Hsueh, T.K.Tsai, and R.K.Iyer, "Fault injection techniques and tools", IEEE Computer, vol. 30, no 4, April 1997, pp.175-182.

[IBM99] IBM Microelectronics Second Quarter 1999, vol. 5, N°. 2.

[ITRS05] International Technology Roadmap for Semiconductors (ITRS), édition 2005.

[ITRS07] International Technology Roadmap for Semiconductors (ITRS), édition 2007.

[Iza86] N.Izawa, "Reliability evaluation of partitioned TMR systems", systems and computers in Japan, 17 (9), pp. 78-86, 1986

[Jal91] P.Jalote, " An integrated approach to software engineering ", Springer-Verilog, New York, NY.,1991.

[Jei09] M.Jeitler, M.Delvai, S.Reichor, " Fuse – A Hardware Accelerated HDL Fault Injection Tool. In: 5 th Southern Conference on Programmable Logic, 2009. SPL. (2009) 89-94.

[Kar98] G.Karypis, R.Aggarwal, V. Kumar, S.Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Domain",Technical Report 1998, http://www.cs.umn.edu/~karypis/metis

[Kar97] G.Karypis, V.Kumar, "METIS 3.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System", Technical Report 97-061, Department of Computer Science, University of Minnesota, 1997.

[Kaw85] M. Kawai and K.Oozeki, "Automatic test pattern generation for large combinational circuits", Proc. Int. Symp. On circuits and systems, Japan, pp 663-666, June 1985.

[Kho07] A. Khouas, "Test de systèmes électroniques" fascicule de cours, École polytechnique Montréal, Automne 2007.

[Kor93] I.Koren, Z.Koren, C.H.Stapper, "A unified negative bonomiale distribution for yield analysis of defect tolerant circuits" IEEE transaction of computers, vol.42, N°6, 99 724-734, 1993.

[Kor07] I. Koren, C. Krishna, "Fault Tolerant Systems", Edition Morgan Kauffman, 2007. ISBN: 987-0-12-088525-1.

[Lah83] S. Laha, J.H. Patel, "Error correction in arithmetic operations using time redundancy", proceeding of fault tolerant computing symposium, pp. 298-305, 1983.

[Lak94] A. Lakhlef, " Génération automatique de vecteurs de test et simulateur logique", thèse de magister, Université Farhat Abbas, 1994.

[Lan01] R.E.Langford, J.J.Liou, V.Raghavan, "The application and validation of a new robust windowing method for the Poisson yield model" Advanced Semiconductor Manufacturing Conference, 2001 IEEE/ SEMI pp. 157-160, april 2001.

[Lan98] R. E.Langford, J.J. Liou," Negative Binomial Yield Model Parameter Extraction Using Wafer Probe Bin Map Data", Electron Devices Meeting, Proceedings of IEEE Hong Kong, pp.130-133, 1998.

[Lap90] J. C. Laprie, J. Arlat, C. Béounes, K. Kanoun, "Definition and analysis of hardware and software fault tolerant architectures", computer, 23(7), pp. 39-51, juillet 1990.

[Lau07] C.Laundrault, " test, testabilité et test intégré des circuits intégrés logiques", fascicule de cours, Université Montpellier II, Master 2 Recherche Systèmes automatiques et microélectronique intégré, 2007.

[Lau09] C. Laundrault, J. Vial, P. Girard, A. Virazel, S. Pravossoudovitch, " Test et Testabilité de structures numériques tolérantes aux fautes", Université Montpellier II, 2009.

[lau05] M. Laurent Capocchi, "Simulation concurrente de fautes comportementales pour des systèmes à événements discrets", thèse de doctorat, Université de Corse, Pasquale Paoli, 2005

[Lay80] Layout and parasitic information for ISCAS circuits, dropzone.tamu.~edu/…/iscas. html-Etats-Unis, 1980.

[Lev03] R. Leveugle, K.Hadjiat, "Multi-level fault injection in VHDL descriptions: alternative approaches and experiments", journal of electronic testing: theory and application: (JETTA), vol. 19, No. 5, October, 2003, pp. 559-575.

[Lyo62] R. E. Lyons, W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability", IBM Journal of Research and Development, 6 (2), pp.200-209, Avril 1962. http://www.ccs.neu.edu/course/csg712/resources/Lyons-Vanderkulk-62.pdf

[Mac08] A.Machouat, "Développement et application d'une méthode d'analyse de défaillances fonctionnelles et contribution à l'amélioration de l'utilisation des techniques optiques statiques et dynamiques ", thèse de doctorat, Université Bordeaux I, 2008.

[Mak86] G. Maki, P. Owsley, K. Cameron, J. Venbrux, "VLSI reed solmon decoder design", military communications conference, IEEE volume 3, pp.46.5.1- 46.5.2, octobre 1986.

[Maz95] R. Mazzaferri, T. Murray, "The connection network class for fault tolerant meshes", computers IEEE transactions, vol 44, Issue 1, pp. 131-138, 1995.

[Mew78] J.Mewon and Chapell, "Deductive fault simulation with functional blocks", IEEE Trans Comput vol.c27, pp 689-695, 1978.

[Mic87] A. Miczo, "Digital logic testing and simulation", John wiley and sons, New York, 1987.

[Mod05] ModelSim, SE Tutorial, version 6.0d, published: April 19, 2005.

[Nic98] M.Nicolaidis, "On line testing for VLSI: state of the art and trends", Integration, the VLSI journal vol 26, Issue 1-2, December 1998, pp, 197-209.

[Nor89] F.E. Norrad and H. Raskard, "An automatic test generation algorithm for hardware description languages", 26 th ACM/IEEE design autom. conf. 1989, pp 429-434.

[Pac08] C.Paccard, " Développement d'outils statistiques pour la mise en place de boucles de régulation en microélectronique", thèse de doctorat, Université de Toulouse III, 2008.

[Par92] K.P. Parker, "The Boundary- Scan Handbook", Kluwer, 1992.

[Ple86] V. Pless, "Decoding the golay codes", Information theory, IEEE transactions on volume 32, Issue 4, pp.561-567, juillet 1986.

[Ran75] B. Randell, "System structure for software fault tolerance", IEEE transactions on software engineering, SE-1, SE-1(2), pp. 220-232, juin 1975.

[Rot66] J. P. Roth, IBM journal of research and Development, vol 10 Issue 4, 1966.

[Rou08] A.Rousset, "Diagnostic de pannes dans les circuits logiques : Développement d'une méthode ciblant un ensemble élargi de modèles de fautes", thèse de doctorat, Montpellier II, 2008.

[Sar89] T.M. Sarfert and R. Markgraf, "Hierarchical test pattern generation based on high level primitives", Proc Int. Test Conf. pp 470-478, 1989.

[Seg04] J. Segura, F. Hawkins, CMOS electronics how it works, how it fails, IEEERESS, 2004.

[Sew06] S.R. Seward and P.K.Lala, "Fault injection for verifying testability at the Verilog level", Department of computer science and computer engineering, Fayetteville, University of Arkansas, 2006.

[Sie92] D. P. Siewiorek, R.S.Swarz, Reliable Computer Systems, Design and Evaluation", second edition, 1992.

[Sta84] C. H. Stapper, "Yield model for clusters within integrated circuits", IBM J. Res and Dev, vol 28, N°5, pp. 636-639, 1984.

[Sta86] C.H. Stapper, "On yield, fault distributions, and clustering of particles", IBM J. Res and Dev, vol.30, N°3, pp. 326-338, 1986.

[Ste00] A. Steininger, "Testing and built in self- test-A survey", Journal of systems Architecture 46, pp. 721-747, 2000.

[Str99] S. E. Stroud, "Yield modeling for majority voting based defect-tolerant VLSI circuits", Southeastcon, pp. 229-236, 1999.

[The00] The VLSI Handbook, Ed. Wai-Kai Chen, "ATPG and BIST", CRC Press LLC, 2000.

[Tri85] E. Trishler and M. Schulz, "Application of testability analysis to ATG: methods and experimental results, Int. test conf, pp.79-82, 1985.

[Vil97] A. Villemeur, "Sureté de fonctionnement des systèmes industriels", Edition Eyrolles, 1997.ISSN : 0399-4198.

[Wil86] T.W. Williams, "VLSI Testing", North-Holland, 1986.

Site internet :

[Sit01]www.Synopsys.com/products/test/tetramax_ds.html .

[Sit02] http://www.glaros.dtc.umn.edu/gkhome/views/metis

[Sit03]http://www.cad.polito.it/download/tools/itc99.html
[Sit04]http://www.synopsys.com/Tools/Implementation/RTLSynthesis/pages/TetraMAXATPG.aspx

[Sit05] http://www.vtvt.ece.vt.edu/vlsidesign/cadtools.php

[sit06] www.euronews.net/sci-tech