TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ACM	Auto Correction Mode
AVF	Architecture Vulnerability Factor
CMOS	Complementary Metal-Oxide-Semiconductor
CTR	Compile Time Reconfiguration
DOM	Detection Only Mode
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
ISA	Instruction Set Architecture
ISS	Instruction Set Simulation
LUT	Look Up Table
MBU	Multiple Bits Upset
MTBF	Mean Time Between Failures
MCS	Minimal Cut Set
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
RAM	Random Access Memory
RTL	Register Transfer Level
RTR	Real Time Reconfiguration
SDF	Standard Delay Format
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEL	Single Event Latch-up
SER	Soft Error Rate
SET	Single Event Transient
SEU	Single Event Upset
SoC	System on Chip
SRAM	Static Random Access Memory
SWIFI	SoftWare Implemented Fault Injection
VHDL	VHSIC Hardware Description Language

INTRODUCTION

Jusqu'à très récemment, presque uniquement la communauté spatiale s'intéressait aux radiations cosmiques et s'inquiétait de leurs effets. Cependant, la diminution de la taille des transistors rend les systèmes électroniques plus sensibles à leurs effets. L'augmentation des cas de dysfonctionnements temporaires de l'électronique embarquée dans les avions est souvent imputée aux radiations (Normand, 1996).

L'augmentation de la sensibilité des composants électroniques aux radiations a pour conséquence que les fournisseurs d'équipement à fonctionnement ininterrompu au niveau du sol s'intéressent de plus en plus aux effets des radiations cosmiques (Santarini, 2005). De plus, les circuits intégrés « résistants aux radiations » sont coûteux et les technologies avec lesquelles ils sont fabriqués sont souvent moins avancées en terme de la finesse des géométries. Les concepteurs sont donc souvent contraints d'utiliser des circuits intégrés traditionnels qui ne sont pas spécifiquement adaptés à tolérer des radiations.

Lors de la conception d'un circuit, il est important d'effectuer une analyse de sureté tout au long du flot de conception. Il est donc nécessaire d'adapter les méthodologies d'intégrations actuelles. Les concepteurs ont besoin d'outils et de méthodologies pour valider les divers choix de stratégies de protection contre les effets des rayons cosmiques afin de se conformer aux exigences de fiabilité du concepteur (Mukherjee, 2008).

Le projet présenté dans ce mémoire fait partie du projet CRIAQ AVIO403 dont les objectifs principaux sont le développement et la validation de techniques et de méthodologies de conception et de vérification qui facilitent le test, le diagnostic et l'émulation de pannes pour les systèmes tolérants aux radiations. Le projet se décompose en quatre sous objectifs :

- l'élaboration et la mise en œuvre d'une stratégie de validation hâtive permettant l'exploration à haut niveau de l'espace de conception ;
- l'élaboration et la mise en œuvre d'une stratégie de vérification à niveaux d'abstraction multiples supportant la simulation de pannes liées à l'effet des radiations ;

- l'élaboration d'une stratégie de conception facilitant les tests en précertification pour la robustesse à l'égard des radiations ;
- l'exploration de stratégies de tolérance des pannes basée sur des systèmes industriels complexes.

Le projet dont les travaux sont présentés dans ce mémoire vise à élaborer, mettre en œuvre et valider une stratégie permettant de déterminer la robustesse inhérente des autres éléments du système global. Le but étant de réaliser les outils nécessaires afin d'effectuer une analyse de sureté à haut niveau d'abstraction.

Cette tâche se décompose en trois sous-objectifs. Le premier est d'investiguer et de mettre en place une ou plusieurs approches permettant l'injection de pannes similaires à l'effet des radiations sur un FPGA. Pour cela, les différents outils d'injection de pannes par simulation existants seront étudiés. Aussi, une méthodologie d'injection de pannes par émulation utilisant les outils Xilinx disponibles sera proposée. Les performances et les limites de ces deux approches seront estimées et ensuite comparées.

Le second sous-objectif consiste à proposer un format permettant de capturer le comportement fautif d'un circuit et d'appliquer cette information à plus haut niveau d'abstraction. Différents formats seront proposés afin d'être adaptés à plusieurs types d'utilisations et de répondre aux besoins du concepteur. La précision et la compression des données obtenues avec ces différents formats seront caractérisées.

Le sous-objectif final vise à proposer et caractériser une méthodologie d'injection de pannes sous Simulink émulant le comportement fautif de circuits dû aux radiations. Pour cela, un bloc/fonction permettant de lire et d'appliquer les Signatures sous Simulink sera créé. Ensuite, les performances de la simulation ainsi que la précision de l'injection seront caractérisées.

Pour résumer, voici les objectifs numérotés afin de faciliter le renvoi et le référencement tout au long de ce mémoire. Les objectifs sont donc :

1. D'investiguer et de mettre en place une ou plusieurs méthodologies permettant l'injection de pannes similaires à l'effet des radiations sur un FPGA ;
2. De proposer un format permettant de capturer le comportement fautif d'un circuit et d'appliquer cette information à plus haut niveau d'abstraction ;
3. De proposer et de caractériser une méthodologie d'injection de pannes sous Simulink émulant le comportement fautif de circuits dû aux radiations.

Dans un premier chapitre, la revue de littérature effectuée sera détaillée en expliquant les différentes connaissances nécessaires à la compréhension de ce sujet de recherche. L'origine et les effets des rayons cosmiques ainsi que les variables nécessaires pour décrire leurs effets et les modéliser seront définies et détaillées. Ensuite, la classification et la quantification des erreurs causées par les radiations seront présentées. Pour finir, les différentes techniques et outils d'injection de pannes applicables à divers niveaux d'abstraction seront analysées. Les outils qui ont été utilisés sont, IEEE Xplore, Acm Digital Library, Compendex, et la bibliothèque de l'ÉTS.

Ensuite, dans un deuxième chapitre, la méthodologie proposée afin de répondre aux objectifs du projet sera détaillée. Le flux ainsi que les différentes étapes de la méthodologie proposée seront décrits. Les limites de cette méthodologie qui est une contribution majeure du projet seront détaillées.

Dans un troisième chapitre, la première étape de ce projet, à savoir l'injection de pannes, sera présentée. Le processus d'injection de pannes par simulation, les paramètres spécifiés ainsi que les résultats obtenus seront détaillés. Cette méthodologie sera critiquée et une piste alternative, utilisant l'injection de pannes par émulation, sera proposée. Les résultats obtenus dans plusieurs études de cas seront aussi détaillés et critiqués.

Ensuite, dans un quatrième chapitre, le nouveau concept de Signature du comportement fautif d'un circuit, proposé dans ce mémoire, sera détaillé. La génération de Signatures à travers plusieurs études de cas sera présentée et analysée.

Dans un cinquième chapitre, la troisième et dernière étape de la méthodologie, à savoir l'instrumentation sous Simulink, sera détaillée. Le processus et les objectifs seront détaillés. Ensuite, les résultats obtenus seront critiqués et différents exemples seront présentés à travers plusieurs études de cas.

Pour finir, nous conclurons sur le travail effectué dans ce mémoire et nous discuterons des ouvertures et pistes envisageables à la suite de ce projet.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Les rayons cosmiques dans l'atmosphère

Le terme rayons cosmiques est générique et englobe plusieurs types de rayonnements. Il peut s'agir de rayonnement électromagnétique, d'électrons, de protons, d'ions lourds couvrant un large spectre d'énergie, allant des faibles énergies jusqu'aux hauts niveaux d'énergie venant de différentes sources telles que le soleil, les étoiles dans l'univers, ou des objets célestes plus énergétiques comme les super novas. À l'entrée des rayons cosmiques dans l'atmosphère, on assiste à une atténuation des niveaux d'énergie ainsi qu'à la désintégration de ces rayons en différentes particules. Ceci est causé notamment par l'interaction avec les atomes d'azote et d'oxygène (Bolchini et Sandionigi, 2010).

Comme observé à la Figure 1.1, à l'entrée de l'atmosphère, les rayons cosmiques se désintègrent en différentes particules. On parle d'« averses » de particules secondaires qui produisent des protons, électrons, neutrons, ions lourds, muons et pions. La majorité des particules produites sont des neutrons qui n'ont pas tendance à se recombiner, elles sont causées par des collisions très rapides. Dans l'atmosphère, leur niveau d'énergie peut atteindre quelques centaines de mégaélectronvolts. On peut les mesurer dès 330 km d'altitude, et leur nombre donc leur densité de flux augmente quand l'altitude diminue. On parle de flux de neutrons, ce qui correspond à leur nombre. Le maximum d'intensité du flux de neutrons ainsi produit se situe à environ 20 km d'altitude. À partir de cette distance, leur densité diminue, au niveau du sol elle atteint 1/500 de la densité maximale. La latitude est aussi un facteur important à cause du champ magnétique de la terre. Notons que les neutrons ne sont pas directement sensibles au champ magnétique terrestre, mais que ce dernier agit directement sur les particules chargées dont les neutrons sont issus.

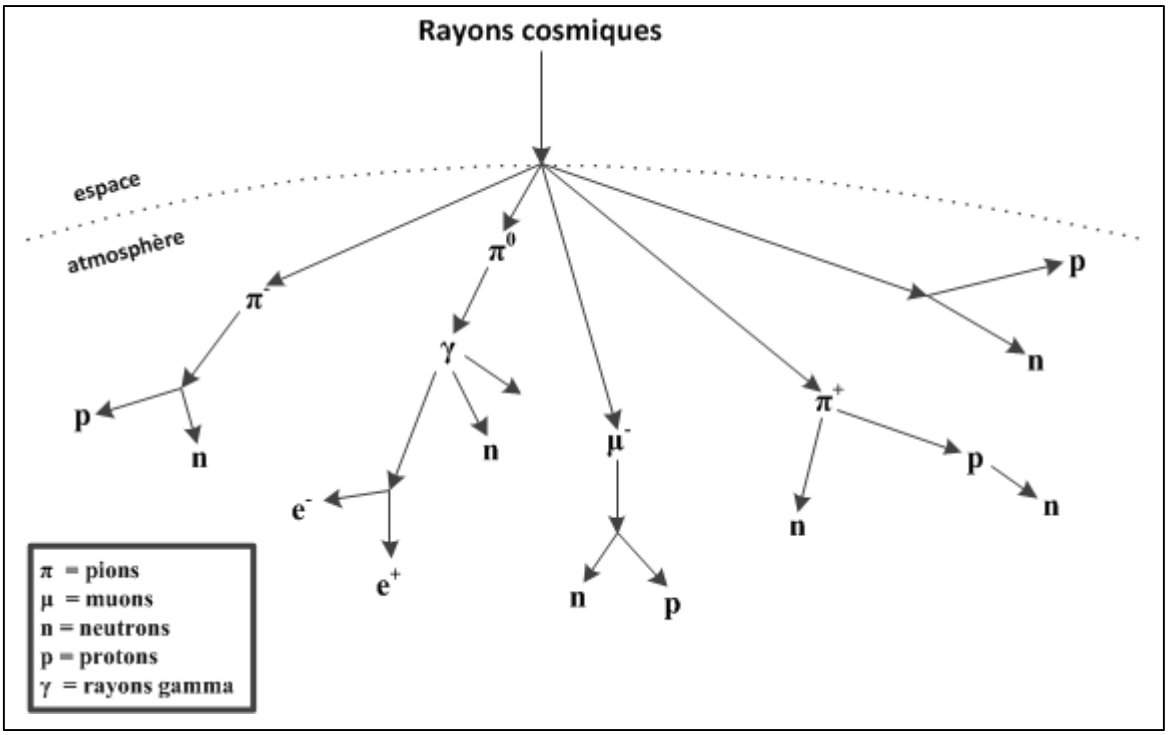

Figure 1.1 Schéma de l'averse de particules secondaires dans l'atmosphère
Adaptée de Santarini (2005)

1.1.1 Effet des neutrons énergétiques sur les circuits intégrés

Les neutrons, avec un haut niveau d'énergie, peuvent générer des particules chargées dans n'importe quel circuit intégré (Actel, Decembre 2002). La plupart des neutrons passeront à travers le silicium sans interagir avec le circuit, mais il est possible qu'un neutron vienne interagir et perturber un atome. Des particules secondaires sont alors créés ce qui va engendrer une trainée de paires électrons-trous pouvant atteindre quelques dizaines de microns de longueur. Si cela a lieu proche d'une jonction PN, ces charges induisent un courant momentané qui peut générer un pic de tension transitoire.

Les neutrons ne génèrent pas directement d'ionisation dans le silicium. On parle de réaction élastique et inélastique. Lors d'une réaction inélastique, le noyau du silicium peut se fragmenter et ainsi créer des ions plus légers et des particules supplémentaires (neutrons et protons). L'énergie cinétique résiduelle est partagée entre les autres particules, ce qui a pour

conséquence qu'une seule réaction inélastique peut causer une erreur dans un circuit électronique (Baumann, 2005).

La Figure 1.2 montre une coupe d'un transistor au niveau physique, en présence d'un neutron qui vient interagir avec un noyau, le fragmente et génère une trainée de paires électrons/trous proche de la jonction PN de la source d'un transistor. Dans cette situation, un pic de tension pourrait être mesuré sur le drain du transistor si ce dernier est adéquatement polarisé.

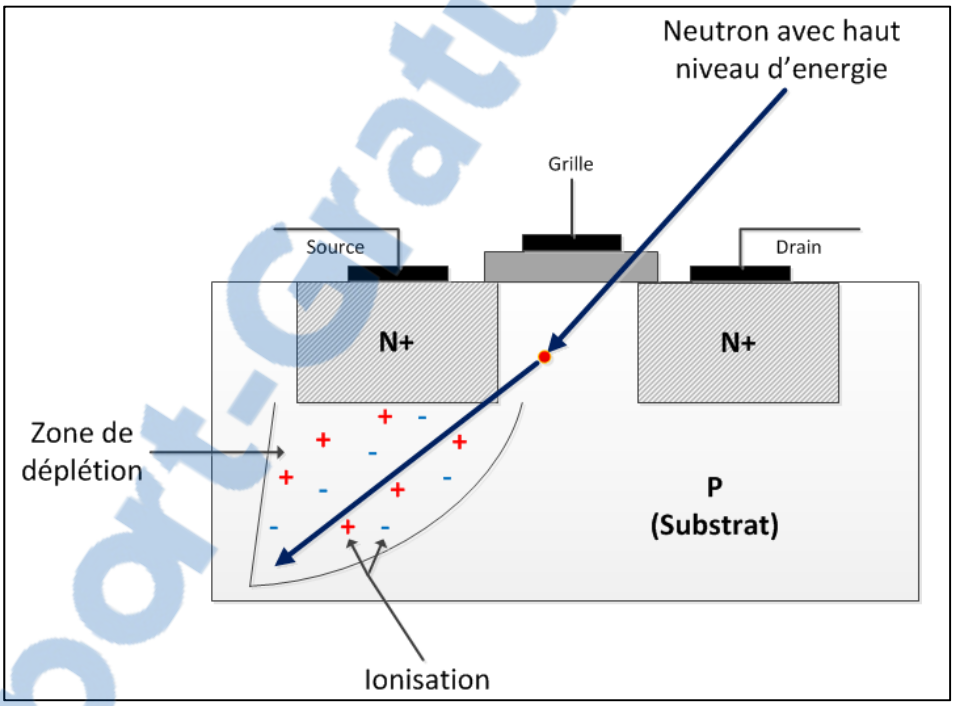

Figure 1.2 Interaction d'un Neutron avec le silicium d'un circuit intégré
Adaptée de Actel (2002)

Plus l'énergie du neutron incident est élevée, plus la probabilité d'une réaction élastique ou inélastique est faible. Toutefois, lors d'une réaction inélastique, plus l'énergie du neutron incident est élevée plus la propagation s'effectuera dans différentes directions.

La probabilité qu'une erreur causée par l'impact d'un neutron apparaisse dépend de plusieurs facteurs. Tout d'abord, il faut que ce neutron possède une énergie suffisante pour engendrer

la génération de paires électron-trou. Ensuite, il faut que ces particules produites possèdent elles-mêmes une énergie suffisante et qu'elles se propagent dans la bonne direction pour déposer suffisamment de charges dans une zone sensible du substrat.

Il a été prouvé que la sensibilité d'un circuit intégré dépendait aussi de la technologie utilisée. Plus la technologie est fine plus la quantité de particules nécessaires à perturber la jonction est faible (Hazucha et Svensson, 2000).

La Figure 1.3 représente le taux d'erreur mesuré en fonction du procédé de fabrication utilisé. Plus le procédé est fin, plus le taux d'erreur, ou Soft Error Rate (SER), est élevé. Pour une faible différence de finesse de 0.25 micron à 0.18, le SER est multiplié par 10, entre 0.25 et 0.05 micron il est multiplié par plus de 1000. Le procédé de fabrication est donc un facteur important à prendre en compte lorsque l'on souhaite estimer la sensibilité d'un circuit aux radiations.

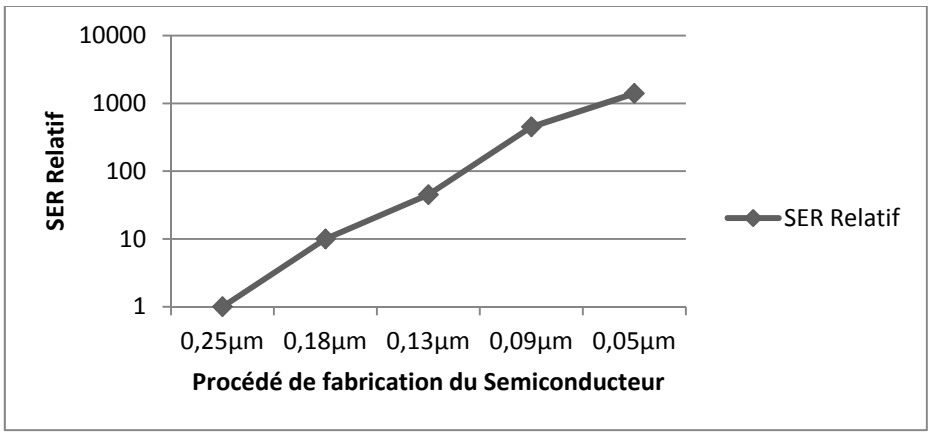

Figure 1.3 SER en fonction de la finesse du procédé
Adaptée de Hazucha et Svensson (2000)

1.1.2 Modélisation du flux de neutrons

Le flux de neutrons dans l'atmosphère dépend de trois paramètres : l'énergie, l'altitude et la latitude (Normand, 1996). Généralement, la variation d'énergie est représentée en traçant le flux différentiel en fonction de l'énergie, appelé le spectre. La variation dans l'espace est quant à elle représentée en fonction de l'altitude et de la latitude. D'autres chercheurs, comme O. C. Allkofer et P. K. Grieder (Allkofer et Grieder, 1984), qui ont mesuré le spectre des neutrons dans l'atmosphère ont obtenu des résultats similaires. Différentes formules ont été proposées afin d'approximer la variation du spectre en fonction de la latitude (Normand, 1996).

Comme illustré à la Figure 1.4, on note que le flux de neutrons dépend effectivement de l'altitude et de la latitude. Les gaz présents dans l'atmosphère atténuent le flux de neutrons. De plus, le champ magnétique de la terre piège les rayons cosmiques et diminue leurs interactions avec les gaz atmosphériques. En effet, les lignes de champs étant plus éloignées à l'équateur, le flux de neutrons y est plus faible (Actel, Decembre 2002).

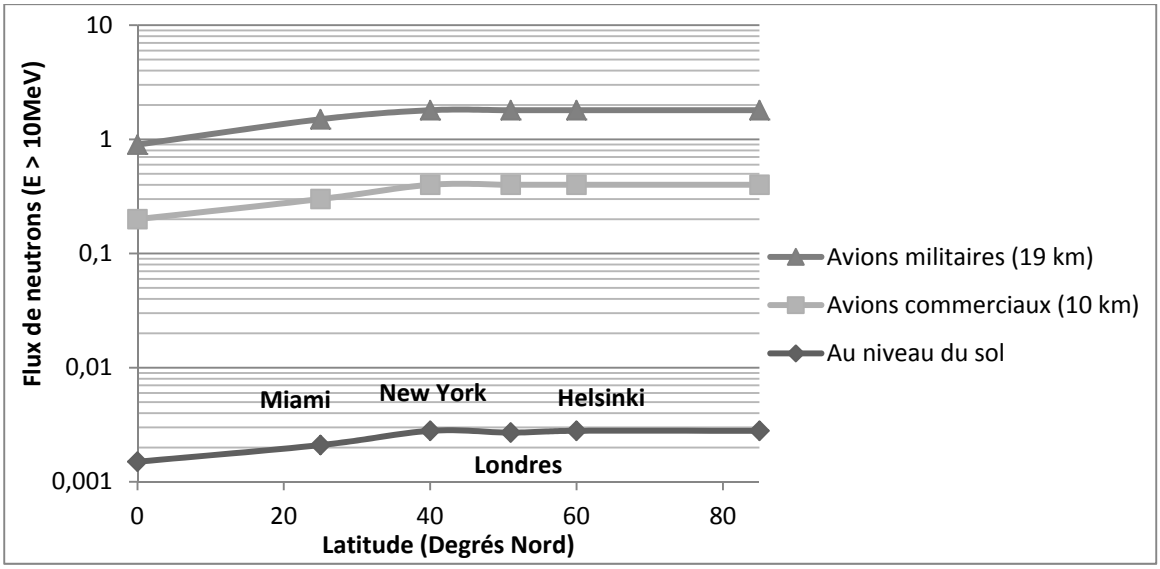

Figure 1.4 Variation du flux de neutrons en fonction de l'altitude de la latitude
Adaptée de Actel (2002)

Il existe un autre modèle, indépendant du spectre d'énergie, il s'agit d'un modèle Wilson Nealy. Ce modèle à la particularité d'utiliser deux nouveaux paramètres : la profondeur atmosphérique notée x (gm/cm²) qui remplacera l'altitude, et la rigidité verticale notée R (GV) qui remplacera la latitude (Normand, 1996). Cet autre modèle prend aussi en compte la modulation solaire du flux de neutrons dans l'atmosphère, ce qui le rend plus précis.

1.1.3 Efficacité d'un écran de protection

Il est très difficile de réaliser un bouclier anti neutron. En effet, même un épais mur de béton n'est pas efficace pour diminuer le flux de neutrons. Les neutrons sont ralentis et donc absorbés par les noyaux légers. Un matériau riche en hydrogène, comme l'eau, le polyéthylène, la paraffine, ou encore le béton, est donc la meilleure solution pour bloquer ces radiations. Pour le faire de manière efficace lorsque le flux de neutrons est élevé cela demande une épaisseur de matériau conséquente.

La Figure 1.5 représente l'atténuation du flux de neutrons en fonction de l'épaisseur de béton utilisée. On remarque que pour atténuer le flux de moitié il faut utiliser une épaisseur d'environ 1.3 m de béton, ce qui est très difficile à proposer pour de nombreuses applications.

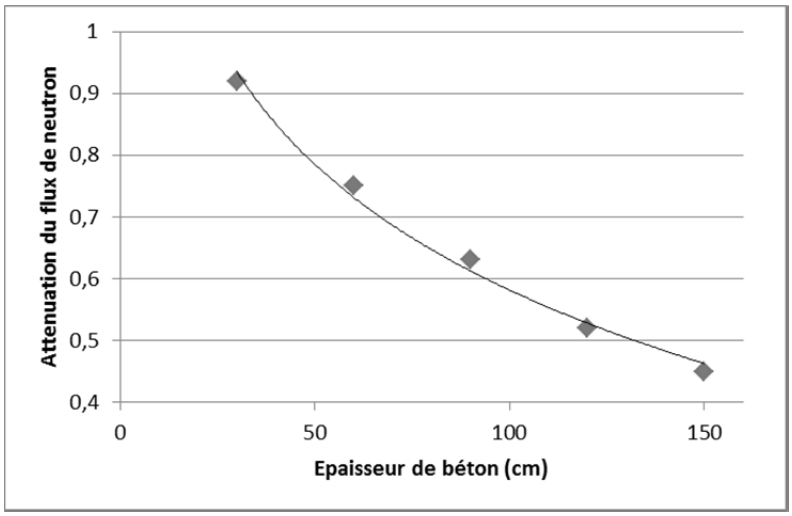

Figure 1.5 Atténuation du flux de neutrons en fonction de l'épaisseur de béton en cm
Adaptée de Dirk *et al.* (2003)

1.1.4 Bilan de l'introduction aux rayons cosmiques dans l'atmosphère

Les rayons cosmiques sont présents avec un flux élevé entre 8000 et 13 500 mètres d'altitude, ce qui correspond aux altitudes auxquelles les avions volent. Nous avons vu que les neutrons peuvent créer des perturbations en réagissant avec les noyaux de silicium. Une simple réaction inélastique avec un noyau de silicium peut causer une perturbation fortement énergétique. La réduction de la taille des transistors rend les circuits électroniques plus sensibles aux radiations. Le taux d'erreur sera par exemple dix fois plus élevé pour une différence de finesse faible de 0.25 à 0.18 micron. De plus, il est très difficile de se protéger des neutrons. Cela nécessite une épaisseur importante de matériau riche en hydrogène, ce qui est très contraignant. Une épaisseur de béton de 1.3m ne réduira le flux de neutrons que de moitié. Il est donc nécessaire et important de prendre en compte les erreurs causées par la réaction de ces neutrons avec le silicium.

1.2 Les erreurs causées par les rayons cosmiques

De nos jours, l'effet des rayons cosmiques sur l'électronique embarquée est connu. De nombreuses données ont été collectées et démontrent que les neutrons sont la première cause d'erreurs non destructives sur l'électronique embarquée dans les avions (Normand, 1996).

En 1993, une étude non intentionnelle menée par Olsen (Olsen *et al.*, 1993), a montré un mauvais fonctionnement d'un ordinateur commercial lorsqu'une accumulation d'erreurs a été observée sur un module de mémoire SRAM CMOS. Le taux d'erreur observé à une altitude de 10 km était de 4.8×10^{-8} perturbations (ou upsets) par bit par jour. Une seconde expérimentation, cette fois intentionnelle, réalisée par Taber et Normand en embarquant une grande quantité de mémoire SRAM CMOS à différentes altitudes a démontré un taux d'erreur de 1.2×10^{-7} par bit par jour à 9 km d'altitude (Taber et Normand, 1993).

Le projet Rosetta (Lesea *et al.*, 2005), réalisé par Xilinx, démontre l'effet des rayons cosmiques sur les FPGA ainsi que l'influence de la position sur la terre. Pour cela, différents

groupes de 100 FPGA de différentes technologies ont été placés à différentes altitudes et leurs configurations ont été vérifiées afin de collecter des données pour mesurer et quantifier l'effet des rayons cosmiques. Le projet Rosetta a été prolongé afin d'estimer la robustesse des nouvelles gammes de FPGA Xilinx.

1.2.1 Classification des pannes causées par les radiations

Comme présenté dans la Figure 1.6, les effets causés par ces radiations sont appelés « Single Event Effects » (SEE) s'ils sont causés par une particule, ou « Total Ionizing Dose effects » (TID effects) s'ils sont causés par l'accumulation d'une dose sur une longue durée (Bolchini et Sandionigi, 2010). Un SEE peut être destructif (permanent) ou non. Les erreurs non destructives ou erreurs passagères (soft errors), se décomposent en plusieurs catégories (JEDEC, 2006) :

- Single Event Upset (SEU) : une erreur non destructive causée par une radiation et capturée par un élément mémoire du circuit ;
- Single Event Latch-up (SEL) : fort courant causé par le passage d'une particule énergétique dans une région sensible causant un dysfonctionnement (peut être destructif);
- Single Event Functional Interupt (SEFI) : erreur non destructive qui cause le mauvais fonctionnement, le blocage ou autre mauvais fonctionnement d'un circuit ;
- Single Event Transient (SET) : augmentation momentanée de la tension d'un nœud.

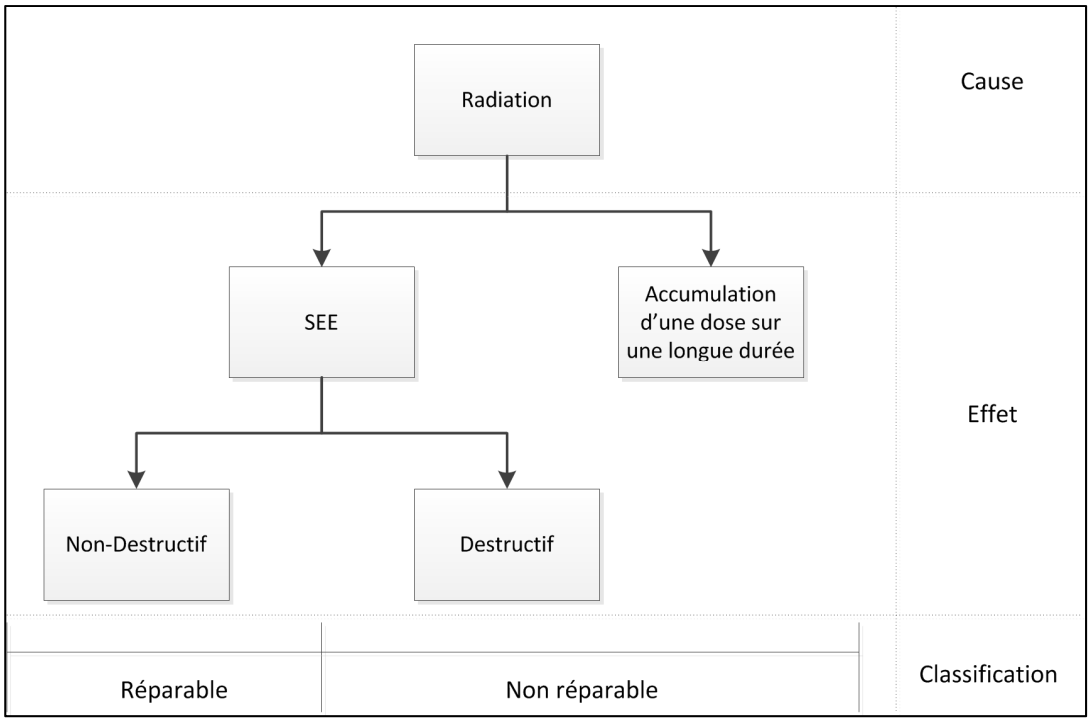

Figure 1.6 Classification des pannes dues aux Radiations
Adaptée de Bolchini et Sandionigi (2010)

Certaines pannes similaires peuvent être causées par le vieillissement du circuit. Ce cas ne sera pas détaillé ici, car il ne rentre pas dans le cadre de ce projet.

Notons qu'il est important de ne pas confondre erreur et panne (Mukherjee, 2008). Une erreur, visible par l'utilisateur est la conséquence d'une panne dans le système qui peut être causée par une interaction avec l'environnement, une imperfection ou un défaut. Une panne n'est pas forcement visible par l'utilisateur, elle peut par exemple être masquée par la logique combinatoire du circuit.

Une erreur est tout simplement la manifestation d'une panne. Cette dernière causera généralement une erreur à une échelle locale, mais elle peut ne pas se voir à une échelle plus globale dans le circuit.

Sur la Figure 1.7, on voit une panne masquée dans le champ intérieur. Il ne s'agit donc pas d'une erreur. Si elle se propage en dehors du champ et qu'elle devient visible, alors elle devient une erreur. Certaines pannes locales ne causeront pas d'erreur sur une échelle plus large, elles seront masquées. Afin de quantifier cette donnée, il est possible de définir le facteur de vulnérabilité architectural (ou AVF). Il s'agit du pourcentage de pannes qui vont se manifester comme une erreur. Plus ce facteur est grand, plus le circuit ou le bit ciblé est vulnérable et est sensible aux SEE non destructifs.

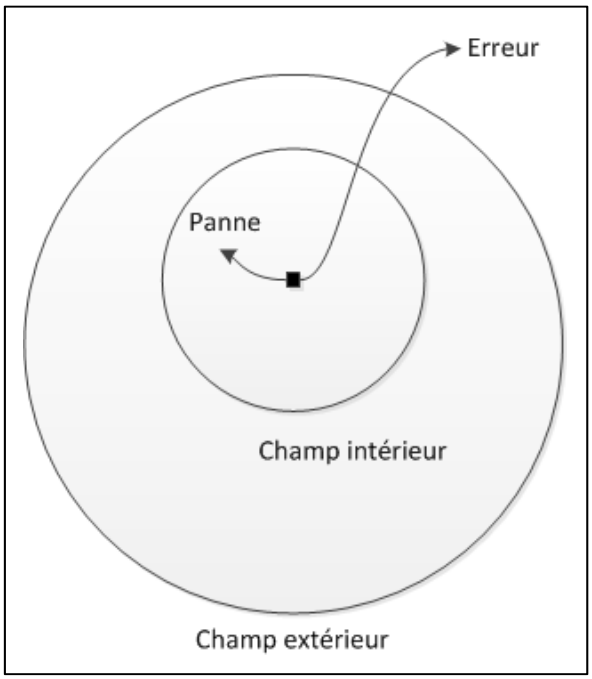

Figure 1.7 Schéma décrivant la différence entre une panne et une erreur
Adaptée de Mukherjee (2008)

1.2.2 Taux de pannes

Les pannes non destructives causées par une radiation sont caractérisées par une fréquence qui dépend de la section efficace (ou cross section) du circuit bombardé et du taux de radiations dans l'environnement (λ_{SEU}) (Hazucha et Svensson, 2000). La section efficace correspond à la partie du circuit qui est sensible aux radiations. On distingue deux types de

sections efficaces, la dynamique qui correspond à la partie sensible du circuit et la statique qui correspond à la partie opérationnelle.

Le taux de pannes non destructives est donc calculé de la manière suivante (Bolchini et Sandionigi, 2010) :

$$\lambda rec = \frac{Cross\ section\ dynamique}{Cross\ section\ statique} \times \lambda SEU \tag{1.1}$$

En utilisant cette équation, on peut alors calculer le temps moyen entre chaque panne, en anglais Mean Time To Failure (MTTF), avec la formule suivante :

$$MTTFrec = \int_0^{lifetime} t \times Flifetime(t)dt = \frac{1}{\lambda \mathrm{rec}} \tag{1.2}$$

Avec Flifetime(t) la loi exponentielle décrivant la durée de vie d'un composant électronique. On en déduit donc que le nombre de pannes non destructives qui ont lieu est :

$$Nrec = durée\ de\ vie\ \times\ \lambda \mathrm{rec} \tag{1.3}$$

Pour calculer le MTTF d'un système complet, on utilise la formule suivante :

$$MTTF_{system} = \frac{1}{\frac{1}{MTTF1} + \frac{1}{MTTF2} + \cdots} \tag{1.4}$$

Comme observé sur la Figure 1.8, le temps entre deux pannes, ou Mean Time Between Failures (MTBF), est quantifié de la manière suivante : $MTBF = MTTF + MTTR$ (Mukherjee, 2008).

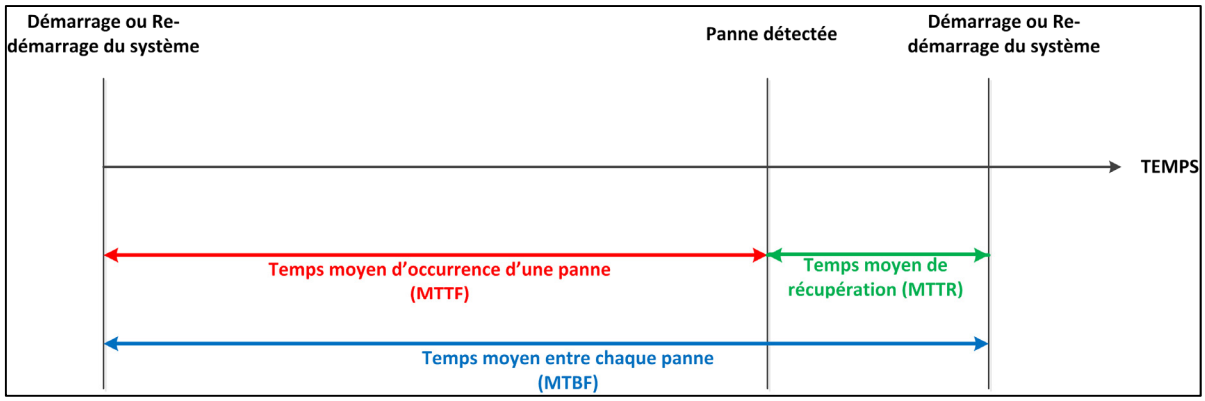

Figure 1.8 Représentation schématique des variables MTTF, MTTR, et MTBF
Adaptée de Mukherjee (2008)

Afin qu'un neutron change l'état d'une cellule, il faut que la charge accumulée dépasse un certain seuil. Cette charge minimale nécessaire est appelée charge critique ou Qcrit (Mukherjee, 2008). Hazucha et Svensson ont proposé le modèle empirique suivant pour prévoir le taux d'erreur appelé Soft Eror Rate (SER) :

$$SER = C \times Flux \times Surface \times e^{\frac{Q_{crit}}{Q_{coll}}} \tag{1.5}$$

Où C est une constante dépendant du procédé de fabrication et du choix du design et Qcoll est le rapport entre la charge collecté et la charge générée par unité de volume.

Plus la technologie est fine plus la surface diminue, ce qui devrait diminuer le SER. Par contre, cela impose aussi que la tension d'alimentation est plus faible, ce qui va entrainer une diminution de la charge critique Qcrit (Hazucha et Svensson, 2000).

1.3 Vérification d'un circuit par injection de pannes

Afin de vérifier le bon fonctionnement d'un circuit, un concepteur a besoin de techniques et d'outils afin de réaliser une analyse de sureté. Il existe de nombreuses techniques pour analyser la sureté d'un circuit déjà fabriqué. Il est possible d'effectuer l'analyse par corruption de la mémoire et du logiciel, de perturber l'alimentation du circuit, d'effectuer un

balayage par laser de la surface du circuit, d'exposer le circuit à des ions lourds, ainsi que de perturber les signaux au niveau des broches. Mais, une analyse de sureté peut aussi être réalisée beaucoup plus tôt dans le procédé de conception d'un circuit. On parle d'approches par simulation, émulation matérielle ou prototypage (Vanhauwaert, 2008).

1.3.1 Les types de pannes

Une panne correspond à un défaut physique, ou une altération qui a lieu au niveau matériel ou logiciel (Ziade, Ayoubi et Velazco, 2004). Les pannes peuvent être classifiées selon leur durée, on va alors distinguer les pannes transitoires, permanentes et intermittentes. Une panne transitoire peut être causée par une fluctuation de l'alimentation, une interférence électromagnétique ou une radiation. Généralement, ces pannes n'endommagent pas le circuit physique, sauf dans le cas de radiations très énergétiques répétées. Elles peuvent bien évidemment induire un état erroné dans le système. Il a été démontré que ces pannes étaient plus difficiles à détecter, mais aussi plus probables que les pannes permanentes.

Une panne permanente peut être causée par le vieillissement d'un composant, un transistor endommagé, un problème de fabrication, etc. La réparation de ces erreurs nécessite de changer ou de réparer le composant endommagé.

Une panne intermittente est causée par un composant qui ne possède pas d'état stable. Elles peuvent être réparées en remplaçant le composant ou en changeant le design.

1.3.2 Les différentes techniques d'injection de pannes

L'injection de pannes peut être réalisée au niveau du circuit physique, par émulation et par simulation. Une panne matérielle est injectée physiquement, en perturbant le matériel, en modifiant le voltage, en bombardant d'ions, etc. L'injection de panne par émulation a été développée pour réduire le temps de l'injection par simulation en utilisant un FPGA. Ce qui permet au concepteur d'étudier le véritable comportement d'un circuit et de prendre en

compte les interactions en temps réel. L'injection de panne par simulation consiste à injecter des pannes dans des modèles à différents niveaux d'abstraction, ce qui permet de tester un circuit à différentes étapes du flot de conception. Les différents types d'injections sont résumés dans la Figure 1.9.

D'après (Vanhauwaert, 2008) les différents critères qui permettent de décrire une technique d'injection de panne sont :

- intrusion ;
- destructivité ;
- contrôlabilité spatiale : contrôle du lieu d'injection ;
- contrôlabilité temporelle : contrôle du temps/moment d'injection ;
- reproductibilité : il s'agit de la capacité à reproduire des résultats identiques avec une certaine configuration ;
- accessibilité : définit la capacité à atteindre les points d'injection souhaités ;
- flexibilité : correspond à la facilité de changer de circuit cible.

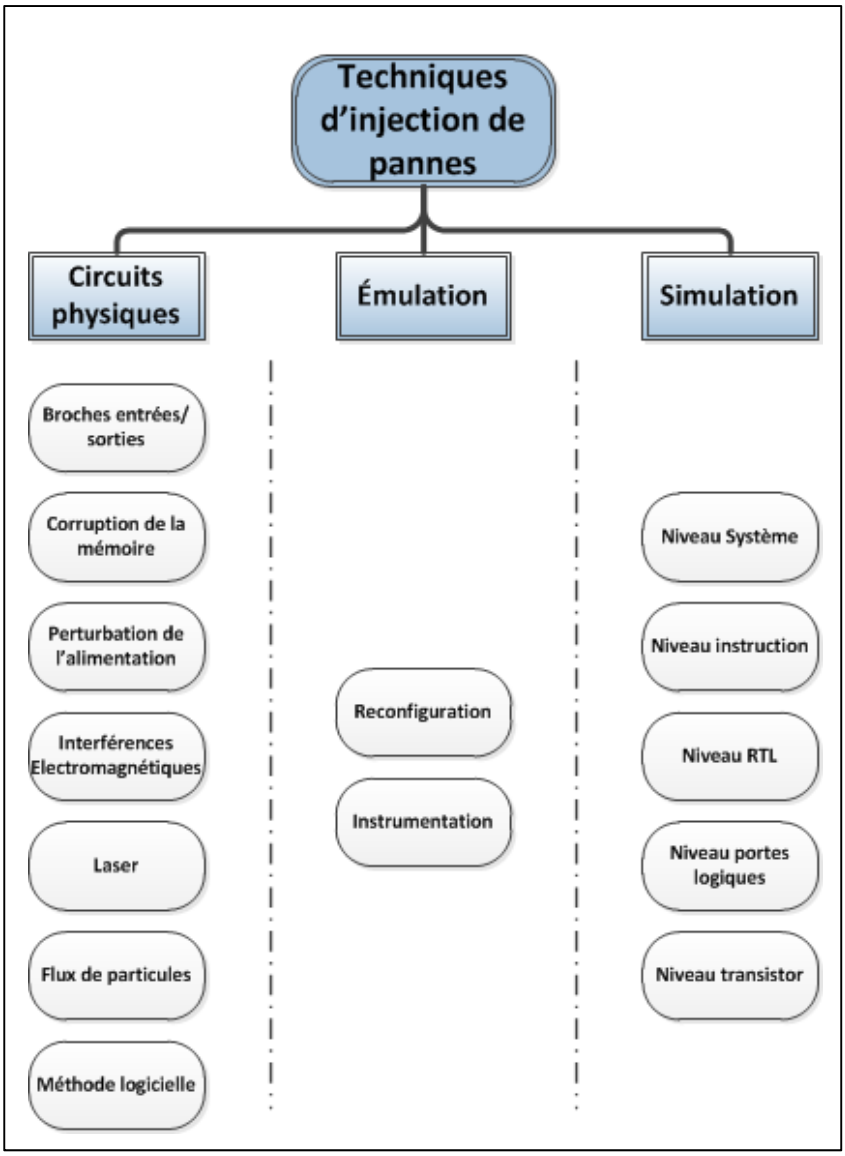

Figure 1.9 Classification des différents types d'injection de pannes

1.3.2.1 Injection de pannes dans des circuits physiques

Il existe plusieurs types d'approches d'injection utilisant des circuits physiques. Dans un premier temps, nous décrirons ces différents types et leur fonctionnement en proposant quelques exemples. Ensuite, nous détaillerons leurs avantages et inconvénients dans le Tableau 1.1.

Injection de pannes au niveau des broches d'entrée/sortie

Le premier outil à utiliser afin d'effectuer ce type d'injection est l'outil MESSALINE (Arlat, Crouzet et Laprie, 1989). Cette méthode consiste, par l'intermédiaire de sondes, à injecter des pannes sur les entrées/sorties d'un circuit.

Injection de pannes par corruption de la mémoire

Cette méthode consiste à injecter des pannes permanentes ou transitoires, simples ou multiples, dans la mémoire d'un processeur. Elle a pour but de valider des mécanismes de détection d'erreurs. Un exemple d'équipement permettant ce type d'injection est l'appareil commercial DEFI (Michel *et al.*, 1994).

Injection de pannes par perturbation de l'alimentation

Il existe divers méthodologies permettant l'injection de pannes par perturbation de l'alimentation. Un exemple de ce type d'injection est l'approche proposée dans (Karlsson *et al.*, 1991) qui utilise l'ajout d'un transistor MOS entre le port Vcc du circuit et l'alimentation. En modifiant la tension à la grille de ce transistor, on peut venir perturber l'alimentation.

Injection de pannes par interférences électromagnétiques (ou EMI)

Cette méthode consiste à générer un champ magnétique d'une certaine amplitude, fréquence et puissance. Elle est souvent utilisée pour certifier la robustesse d'un circuit dans des conditions critiques.

Injection de pannes par laser

Ce type d'injection permet d'injecter des pannes similaires à des erreurs passagères. En traversant le substrat, le laser va générer des paires électrons/trous. Si le faisceau laser est proche d'une jonction pn, un pic de tension aura lieu. Une injection précise, nécessite un spot laser le plus fin possible.

Injection de pannes par un flux de particules

Il s'agit de l'approche qui permet d'injecter des pannes les plus similaires aux effets des radiations cosmiques. Cela permet l'analyse du taux de pannes accéléré d'un système (ou Accelerated SER). L'énergie et le type de particule sont contrôlables. Cette méthode est la plus utilisée par les industriels pour caractériser la sensibilité aux radiations de leurs circuits. L'accès à des installations permettant ce type d'approche est limité. Chaque campagne d'injection nécessite une phase de préparation importante et des coûts substantiels.

Injection de pannes par des méthodes logicielles (SWIFI)

À cause du coût élevé en temps et en argent, différentes approches d'injection par méthodes logicielles (SoftWare Implemented Fault Injection) ont été développées. Par exemple, l'outil FERRARI (Karlsson *et al.*, 1991) permet la modification des instructions au moment de leur exécution. Quant à l'outil Xception, il utilise les exceptions et le débug des processeurs modernes pour injecter des pannes. Il est aussi possible d'inverser des bits dans la mémoire d'un processeur.

Les principaux inconvénients de ces techniques d'injection sont qu'elles peuvent être destructives, leur accessibilité, contrôlabilité et observabilité sont souvent limitées. Les types de pannes sont limités et spécifiques à chaque méthodologie. De plus, un équipement est nécessaire pour réaliser ce type de campagne d'injection, ce qui peut être contraignant. Les différents avantages et inconvénients ainsi que des exemples d'outils sont détaillés dans le Tableau 1.1.

Tableau 1.1 Comparatif des différents types d'injection sur des circuits physiques

Type d'approche	Avantages	Inconvénients	Exemples d'outils
Broches entrées/sorties	• non intrusif ; • bonne contrôlabilité spatiale et temporelle ; • bonne reproductibilité ; • faible coût en temps et argent.	• mauvaise accessibilité ; • définition difficile des modèles représentatifs de pannes internes ; • bruit induit par les sondes.	MESSALINE (Arlat, Crouzet et Laprie, 1989), RIFLE (Madeira *et al.*, 1994)
Corruption de la mémoire	• injection de pannes permanentes ou transitoires, simples ou multiples ; • faibles coûts.	• mauvaise flexibilité.	DEFI (Michel *et al.*, 1994)
Perturbation de l'alimentation	• bonne flexibilité ; • injection de pannes transitoires similaires à des chutes de tension ; • faibles coûts.	• mauvaise contrôlabilité spatiale et temporelle ; • mauvaise reproductibilité ; • effets difficilement estimables.	(Karlsson *et al.*, 1991)
Interférences Electromagnétiques	• bonne flexibilité ; • rafales de pannes ; • représentatif de charges inductives lors de la commutation.	• mauvaise contrôlabilité spatiale et temporelle ; • mauvaise reproductibilité ; • mauvaise accessibilité.	(Vargas *et al.*, 2005)
Laser	• non intrusif ; • bonne contrôlabilité temporelle ; • bonne flexibilité ; • bonne reproductibilité ; • pannes représentatives des erreurs passagères.	• fréquence d'opération et précision limitées ; • pulsations de fuite ; • taille du spot laser.	(John R. Samson, Moreno et Falquez, 1997)

Type d'approche	Avantages	Inconvénients	Exemples d'outils
Flux de particules	• accessibilité ; • taux de pannes accéléré ; • représentatif des SEU dus aux radiations.	• mauvaise contrôlabilité spatiale et temporelle ; • accès aux installations limité ; • coût élevé.	TRIUMF
Méthodes logicielles (SWIFI)	• bonne reproductibilité ; • faible cout en temps et argent.	• mauvaise flexibilité ; • interfère avec l'app.	FERRARI (Kanawati, Kanawati et Abraham, 1995), Xception(Carreira, Madeira et Silva, 1998)

1.3.2.2 Injection de pannes par émulation

L'émulation matérielle consiste à implémenter le circuit sur un FPGA pour y injecter des pannes. Cette approche est beaucoup plus rapide que par simulation. On distingue deux techniques d'injection, par instrumentation et par reconfiguration.

Injection par instrumentation

Afin d'injecter des pannes par instrumentation, il est nécessaire de modifier le circuit original pour permettre cette injection, le synthétiser et l'émuler sur un FPGA.

L'outil FIFA (Fault Injection by means of FPGA) (Civera *et al.*, 2001) utilisant ce type d'injection de pannes a été développé par l'institut Politecnico de Torino. Ce dernier permet de générer une liste de pannes à partir de la liste d'interconnexions du circuit (ou netlist), et instrumente le circuit dans le but de pouvoir y injecter des pannes. L'outil dispose aussi de modules de contrôle et d'analyse de l'injection de pannes. Il s'agit de l'instrumentation à

base de registres masques. L'instrumentation est effectuée par un registre et de la logique combinatoire, dans le but de permettre d'injecter des pannes et d'observer leurs effets. Cet outil utilise des saboteurs venant contrôler les blocs qui forment le circuit.

Cette méthodologie est intrusive, mais propose une bonne contrôlabilité. L'ajout d'instrumentation peut aussi rajouter des délais sur les lignes. L'accessibilité est limitée par l'endroit où l'on a placé les injecteurs. Cette méthodologie nécessite un travail préalable de modification du circuit avant de l'implémenter sur le FPGA.

Injection par reconfiguration

Les FPGA à base de SRAM utilisent un fichier de configuration qui contient toutes les données nécessaires à la configuration du FPGA. Ces données sont transmises de l'ordinateur vers le FPGA lorsque l'on souhaite le configurer. On peut ensuite les relire grâce à la fonction de relecture (ou « readback ») du FPGA. Il est aussi possible de reconfigurer un FPGA de manière statique ou dynamique.

Une configuration statique (Compile Time Reconfiguration ou CTR) consiste à recompiler et synthétiser afin d'appliquer les modifications réalisées sur le code, pour générer de nouveau le système complet et reconfigurer tout le FPGA.

La configuration dynamique (Real Time Reconfiguration ou RTR) permet de modifier la configuration pendant le déroulement d'une application. On peut choisir entre reconfigurer le FPGA partiellement (Local RTR) ou entièrement (Global RTR). La reconfiguration dynamique permet d'injecter des collages à 1 ou 0 permanents ou transitoires dans les blocs combinatoires du circuit. Il est possible aussi d'injecter des SEUs dans les éléments de mémoire. Cette méthode s'avère jusqu'à 30 fois plus rapide que par simulation. Cependant, si de nombreuses reconfigurations sont prédéfinies, cela va consommer de la mémoire. Il est difficile de faire la correspondance entre le fichier de configuration et les ressources physiques. Ce type d'injection de panne est maintenant disponible à tous les utilisateurs grâce à l'outil SEU Controller fourni par Xilinx pour les familles Virtex 5 ou supérieur.

SEU Controller

SEU Controller est un module créé par Xilinx, pouvant être inclus dans n'importe quel design utilisant la gamme de FPGA Virtex 5 ainsi que pour les familles plus récentes. Cet outil permet notamment de détecter et de corriger un SEU, mais aussi d'injecter des pannes sur les bits de configuration.

Ce module qui est basé sur un microcontrôleur de type PicoBlaze, est encapsulé dans deux primitives fournies par Xilinx : ICAP_VIRTEX5 et FRAM_ECC_VIRTEX5. Pour communiquer, il utilise un port RS232 (port série). La première primitive, ICAP_VIRTEX5, permet d'accéder à la configuration du FPGA. De plus, une reconfiguration partielle de cette dernière est réalisable. FRAME_ECC_VIRTEX5, permet quant à lui la détection et l'identification d'erreurs simples ainsi que la détection d'erreurs doubles. Pour cela, chaque trame de 1300 bits configurant le FPGA contient 12 bits supplémentaires, soit 1312 bits. 11 de ces bits sont des bits de correction de type Hamming, le $12^{ième}$ est un bit de parité.

SEU Controller n'utilise que très peu de ressources du FPGA, à savoir 350 LUTs, 330 registres et une BRAM. Cela correspond à moins de 1 % des ressources d'un FPGA Virtex 5 VLX50T.

Comme détaillé à la Figure 1.10, le fonctionnement des primitives composant SEU Controller est très simple, la primitive ICAP lit une trame de configuration de 1312 bits, et la stocke dans une BRAM. Ensuite, la primitive FRAME_ECC lit cette trame,

- si aucune erreur n'est détectée alors la primitive ICAP va lire la trame à l'adresse suivante et le processus va recommencer ;
- si plusieurs erreurs sont détectées par FRAME_ECC alors le système ne peut la corriger, il ne fonctionne plus ;
- si une erreur simple est détectée, elle est alors corrigée, et réécrite dans le BRAM par FRAME_ECC, ICAP va ensuite écrire cette trame dans la configuration du FPGA et ainsi la réparer. La trame à l'adresse suivante va être lue par ICAP et le processus recommence.

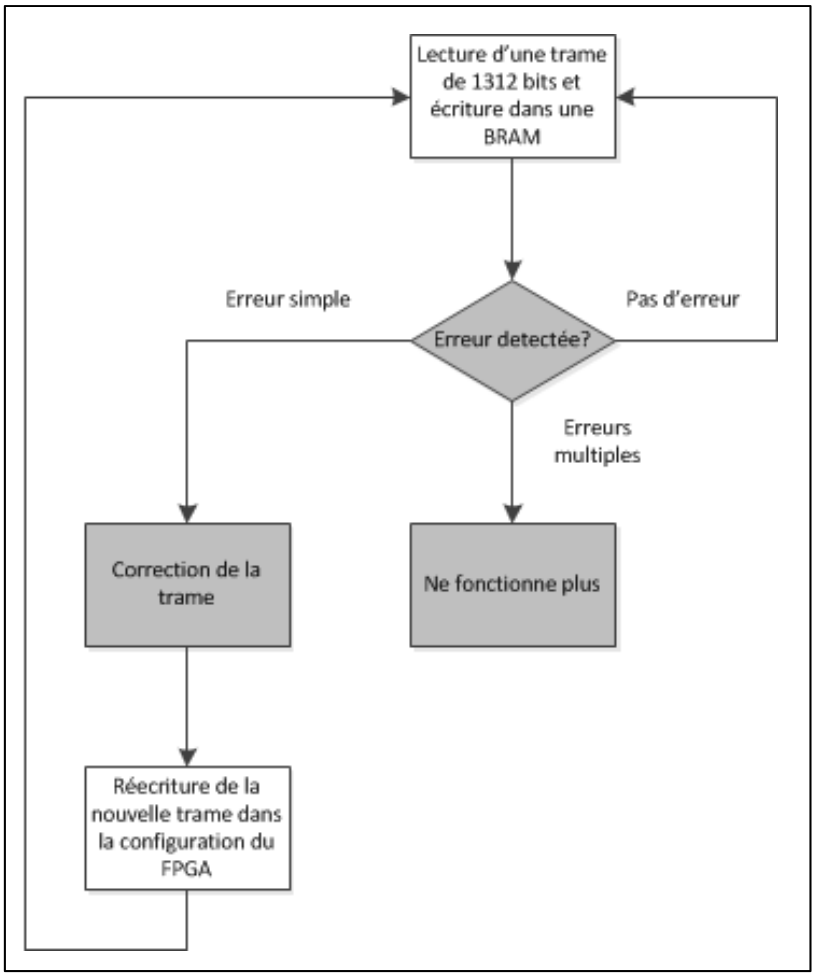

Figure 1.10 Correction d'erreur avec SEU Controller (blanc : réalisé par le module ICAP, gris : réalisé par le module FRAME_ECC)

Deux modes de fonctionnement sont disponibles. Le mode ACM (par défaut), qui corrige automatiquement les erreurs rencontrées. Le temps nécessaire pour corriger une erreur dans ce mode dépend de la taille du FPGA. Le temps nécessaire à vérifier et corriger toutes les trames est compris entre 3.05 ms (XC5VLX20T) et 44.15 ms (XC5VLX330T). Le temps moyen de correction d'une erreur étant la moitié de cette valeur.

Le second mode disponible est le mode DOM (Detection Only Mode). Comme son nom l'indique, il permet de lire et de scanner les trames sans les corriger. Il est utilisé pour la

simulation et l'injection de SEU. Via la communication série, il est possible d'utiliser les fonctions suivantes :

- injecter un SEU simple aléatoirement ;
- injection un MBU sur deux bits aléatoirement ;
- modifier un bit spécifique dans une trame définie ;
- obtenir le rapport sur le statut du CRC et le mode choisi (DOM ou ACM) ;
- afficher le contenu d'une trame ;
- demander l'adresse d'une trame.

Il est possible d'observer le statut du module et de le contrôler grâce à un logiciel de communication sériel HyperTerminal. L'injection d'un MBU ou de plusieurs SEU rend la correction impossible et arrête le fonctionnement des modules. SEU Controller peut aussi injecter une erreur sur sa propre configuration, ce qui rend l'outil erroné. La correspondance entre la relecture de la configuration et les ressources du FPGA n'est pas directe. Un traitement est nécessaire. SEU Controller accède aux bits avec deux paramètres, l'adresse de la trame à laquelle le bit se situe, et sa position dans cette trame. Une trame est composée de 1312 bits, son adresse peut être obtenue grâce à la commande « q » de SEU Controller. L'adressage n'est pas linéaire, car certaines trames peuvent ne pas exister physiquement sur le FPGA, leurs adresses seront alors décalées. Il est très facile de réaliser un tableau de correspondance pour convertir un numéro de trame en son adresse.

1.3.2.3 Injection de pannes par simulation

Dans cette partie nous détaillerons les différentes méthodes d'injection de pannes par simulation. Les techniques d'injection de pannes par simulation peuvent être utilisées à différentes étapes du flot de conception. À haut niveau d'abstraction (HDL), on peut vérifier la partie fonctionnelle du système, avant qu'il soit synthétisé. Au niveau portes logiques, on peut vérifier aussi les délais et timings du circuit étudié. Le temps de simulation et la charge de calcul peuvent varier d'un facteur supérieur à dix entre les deux niveaux énoncés précédemment (Vanhauwaert, 2008).

Niveau système

Des langages comme SystemC, POLIS ou encore Esterel permettent de décrire des circuits au niveau système. Mathworks Simulink est un outil qui permet la simulation à un niveau système, dans ce cas particulier on parle de niveau bloc. L'injection de pannes à ce niveau a pour but d'effectuer une analyse de sureté très tôt dans le flot de conception. Les systèmes sur puces sont les principales cibles de cette approche. À ce niveau d'abstraction les modèles de pannes peuvent manquer de précision. À notre connaissance, il n'existe à ce jour aucune méthodologie permettant de simuler l'effet des SEU à haut niveau d'abstraction avec précision.

Dans ce projet, nous utiliserons l'outil de simulation Simulink, car il est populaire et fortement utilisé afin de réaliser des simulations à haut niveau d'abstraction. Comme énoncé précédemment, cet outil utilise une représentation bloc d'un système, ce qui correspond à la représentation algorithmique d'un système. Un modèle est composé de différents blocs représentants des fonctions, des décisions conditionnelles, des constantes, etc. Lorsque connectés ensemble, ces blocs peuvent représenter des fonctions complexes. Au niveau bloc il existe deux classes de pannes pouvant être injectées : des saboteurs et des mutants. Un saboteur change la valeur d'un signal entre deux blocs. Un mutant, quant à lui, change la fonctionnalité d'un bloc. Il est important de noter que le niveau d'abstraction étant élevé, des détails d'implémentation sont abstraits. L'injection de pannes sous Simulink est donc limitée. Les modèles de panne tendent à manquer de précision, surtout lorsque des blocs complexes sont utilisés, car uniquement les entrées/sorties sont accessibles.

À notre connaissance, le seul outil d'injection de pannes développé pour Simulink est MODIFI (Svenningsson, Vinter et Törngren, 2010). Cet outil contrôle l'exécution et ajoute des mécanismes d'injection à travers le design. Son principal avantage est sa capacité à générer sont les groupes minimaux de pannes qui violent un requis de fiabilité (ou MCS pour Minimal Cut Sets). Cet outil peut aussi importer des modèles de panne depuis des fichiers XML et les adapter pour Simulink en les convertissant en code Matlab. L'utilisateur peut spécifier le signal qui sera affecté ainsi que le modèle de panne en choisissant le type de

panne et son timing. Le but de cet outil est de réaliser une estimation de la robustesse d'un circuit tôt dans son design, et de créer des cas de tests pour l'injection de pannes sur le système réel. Il a été prouvé qu'injecter des pannes en utilisant des modèles de pannes spécifiques avec MODIFI produit des résultats similaires à l'injection matérielle. Cependant, les pannes ont été injectées et des lieux d'injection spécifiques. En utilisant les modèles de pannes de MODIFI il est impossible de complètement répliquer l'injection matérielle, car certains lieux d'injection sont inaccessibles.

Niveau instruction

Au niveau instruction, il existe deux approches possibles : ISS (Instruction Set Simulation) et ISA (Instruction Set Architecture). La modélisation ISS consiste à simuler un processus en exécutant une instruction à la fois, ce qui permet d'observer les différents états des registres et autres signaux internes. L'approche ISA quant à elle prend en compte les transferts de registre, elle est donc plus précise que l'approche ISS qui ne considère que les transactions.
Ce type de simulations est couteux en temps. L'injection de pannes à ce niveau ne sera pas utilisée dans ce projet.

Niveau RTL

Au niveau RTL il existe deux façons d'injecter des pannes, soit en forçant la valeur de nœuds internes via des commandes du simulateur, soit en utilisant une description du circuit que l'on a préalablement modifiée afin d'injecter des pannes. Un exemple d'outil utilisant les commandes du simulateur est MEPHISTO (Multi level Error/Fault Injection Simulation Tool) (Jenn *et al.*, 1994). Ce dernier utilise aussi la manipulation de signaux ainsi que celle des variables afin d'injecter des pannes. L'agence spatiale Européene a de son côté développé un outil permettant l'injection de pannes au niveau RTL en utilisant la commande « force » de ModelSim, ainsi que les langages TCL et Perl.

Les outils d'injection par simulation au niveau RTL fonctionnent de la manière suivante : la simulation se déroule normalement jusqu'à un point d'arrêt, au moment où l'injection doit débuter, les processus sont alors arrêtés. Les signaux et variables peuvent alors être modifiés

de manière permanente ou transitoire. Cette méthodologie nécessite un outil de simulation puissant, permettant d'arrêter la simulation et de forcer des signaux. L'accessibilité est limitée due au niveau d'abstraction.

Dans le cadre d'injection de pannes au niveau RTL on parle de simulation avec description instrumentée lorsqu'on ajoute des saboteurs, des mutants ou des sondes. Un saboteur permet de rajouter un délai sur une ligne ou d'en modifier sa valeur. Un mutant correspond à une version modifiée d'un bloc/composant donnant un résultat erroné correspondant à un seul modèle de panne. Généralement, les modèles de pannes utilisés par des mutants sont plus complexes que ceux utilisés par les saboteurs. Ils peuvent être statiques si leur configuration reste la même durant toute la simulation. Ils peuvent aussi être dynamiques, en fonction de la valeur d'un signal le bloc mutant sera activé ou non. De cette manière, il est possible d'injecter des pannes permanentes et transitoires. Il faut noter que cette méthodologie augmente le temps de simulation. L'utilisation de mutants ralentit la simulation de manière plus significative que l'utilisation de saboteurs. L'accessibilité de cette méthodologie est aussi limitée. L'injection de pannes à ce niveau ne sera pas utilisée dans ce projet.

Niveau portes logiques

À notre connaissance, cinq outils ont été développés pour injecter des pannes au niveau portes logiques FAST, VERIFY, VFIT, ROBAN, et LIFTING.

Le premier outil qui est apparu dans la littérature, **FAST** (Hungse *et al.*, 1996), est un environnement de simulation d'injection de pannes transitoires au niveau porte. Il se démarque des autres par le fait qu'il utilise des modèles réalistes de pannes. L'environnement de simulation utilise un simulateur de pannes de timing au niveau porte ainsi qu'un simulateur de pannes parallèles.

La première partie de l'outil FAST, notée TIFAS permet de modéliser et de caractériser la durée de l'impulsion permettant de simuler une panne, et le comportement d'une bascule face à une erreur de ce type (fenêtre de latch-up).

La principale caractéristique du module de simulation est l'optimisation de l'algorithme de simulation. L'outil Fast utilise le fait qu'une panne transitoire pouvant être stockée dans une bascule doit arriver tard dans le cycle d'horloge, la valeur obtenue combinatoirement sera donc déjà obtenue. Il suffit alors de simuler les transitions dues aux pannes et non à la logique combinatoire. Comme présenté dans la Figure 1.11, la simulation s'effectue en deux parties, dans un premier temps l'outil détermine quelles pannes seront propagées jusqu'à la sortie d'une bascule. Ensuite, le simulateur va injecter des pannes à la sortie de ces bascules dans une nouvelle simulation. Cet outil permet uniquement d'injecter des pannes transitoires.

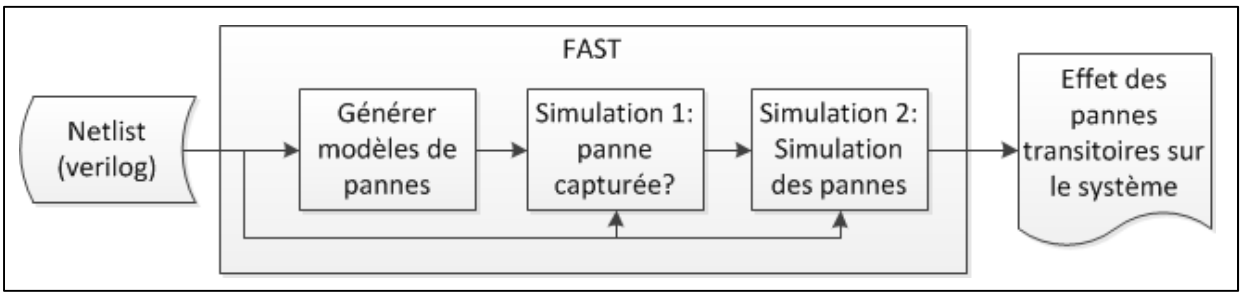

Figure 1.11 Architecture de l'outil FAST

Le second outil, **VERIFY** (Volkmar, Oliver et Frank, 1997) permet l'injection de pannes au niveau porte logique et RTL. Comme dans l'exemple de la Figure 1.12, un type de signal interne est ajouté aux types déjà existants en VHDL, il s'agit d'une extension du langage. Ce type contient le temps moyen entre chaque panne et la durée moyenne. Chaque panne est associée à un composant, ceci est donc transparent par rapport aux autres composants. L'interface du simulateur permet d'activer les pannes dans les différents composants. VERIFY permet l'injection d'inversion de bits et de collages à 1 ou 0.

```
ENTITY not_gate is
    port (
            input   : in    bit;
            output  : out   bit
            );
end not_gate;

architecture behav of not_gate is
    signal i_sa1:   boolean interval 20000 h duration 10ns;
    signal i_sa0:   boolean interval 10000 h duration 10ns;
    signal o_sa1:   boolean interval 15000 h duration 10ns;
    signal o_sa0:   boolean interval 30000 h duration 10ns;

begin
    process (input, i_sa1, i_sa0, o_sa1, o_sa0)
    begin

        if i_sa1 or o_sa0 then
            output <= '0';
        elsif i_sa0 or o_sa1 then
            output <= '1';
        else
            output <= NOT input after 10 ns;
        end if;
    end process;
end behav;
```

Figure 1.12 Exemple d'un code d'une porte NON pouvant être utilisé par VERIFY

L'outil nécessite un compilateur et un simulateur dédié pour comprendre et utiliser les nouveaux types permettant de décrire les pannes. De plus, cela nécessite une modification du code qui peut être très contraignante.

Ensuite, **VFIT** (Baraza *et al.*, 2000) est un outil d'injection de pannes fonctionnant sous Windows. Il a été réalisé autour du simulateur commercial ModelSim.

Cet outil est capable d'injecter des pannes dans des modèles VHDL au niveau portes logiques, registre et puce. Les modèles de pannes pouvant être injectés sont des collages et des inversions de bits, ces dernières peuvent être permanentes, transitoires ou intermittentes. L'outil peut utiliser trois techniques d'injection, par commande du simulateur, saboteur ou mutants. Une fois l'injection et la simulation réalisées, VFIT peut effectuer deux types d'analyse. La première analyse et classifie les pannes et les erreurs ainsi que leurs incidences et délais de propagation. La deuxième analyse vérifie si le système est tolérant aux pannes, la détection et les mécanismes de récupération sont validés.

Cet outil est actuellement en cours d'être retravaillé. En effet, pour le moment VFIT ne fonctionne uniquement que sur Windows XP avec une ancienne version de ModelSim. L'outil ne gère pas sa mémoire de manière optimale, il est donc impossible de réaliser une injection sur un modèle complexe.

L'outil suivant, **ROBAN** (Alexandrescu, Anghel et Nicolaidis, 2004), est un outil qui permet l'injection et la simulation rapide de pannes transitoires. On s'intéresse ici aux pannes ayant lieu dans la logique combinatoire d'un circuit et non dans la mémoire. Un outil d'analyse statistique a aussi été implémenté afin d'analyser les résultats obtenus. Le but étant d'évaluer la probabilité qu'une panne transitoire se propage jusqu'à une bascule. Cette analyse permet aussi de connaitre l'évolution du SER en fonction de la fréquence d'horloge ou des caractéristiques de la panne injectée.

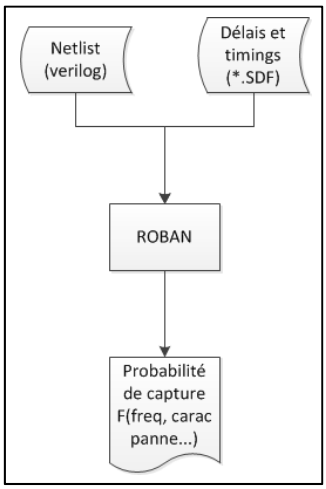

Figure 1.13 Architecture de l'outil ROBAN

Afin d'estimer la propagation et de simuler l'impact d'une panne, on utilise la liste d'interconnexions d'un circuit ainsi qu'un simulateur. Les timings sont donnés dans un fichier SDF (Standard Delay Format) donnant les informations sur les délais les plus élevés, les plus faibles et les délais nominaux. Chaque scénario sera utilisé lors de la simulation. Le défi majeur relevé par le projet est la difficulté rencontrée afin d'obtenir une probabilité réaliste. Pour cela, un grand nombre de vecteurs de test en entrée est nécessaire, ce qui implique un long temps de simulation. La solution principale consiste à injecter plusieurs pannes pour le même vecteur de test en découpant les différentes parties de logique combinatoire du circuit. Le temps de simulation est réduit de manière conséquente ce qui permet à l'outil d'être performant. Il est donc possible de simuler un grand nombre de pannes avec beaucoup de vecteurs de test, ce qui permet d'obtenir une probabilité précise et réaliste.

Le dernier outil, **LIFTING** (Bosio et Di Natale, 2008), est un outil disponible gratuitement permettant de réaliser des simulations logiques simples, ainsi que des simulations de pannes. Il permet aussi d'analyser des résultats de la simulation.

Comme présenté dans la Figure 1.14, l'outil prend en entrée la description en Verilog de la liste d'interconnexions d'un circuit numérique ainsi que la liste des vecteurs à injecter en entrée. Il est possible de spécifier la liste de pannes à injecter, dans le cas échéant l'outil considérera tous les lieux d'injections. Les types de pannes disponibles sont les collages simples/multiples et SEU (bit flip dans un élément de stockage comme les flip-flops, les bascules et les cellules mémoires). Ce simulateur possède la particularité d'utiliser une approche orientée objet. L'outil est réalisé de manière très flexible afin de pouvoir être modifié aisément pour convenir à diverses utilisations. Les rapports d'injection générés par LIFTING sont détaillés. Pour une séquence de test donnée, l'outil va générer la séquence de sortie parfaite correspondante ainsi que chaque séquence de sortie pour chaque panne injectée.

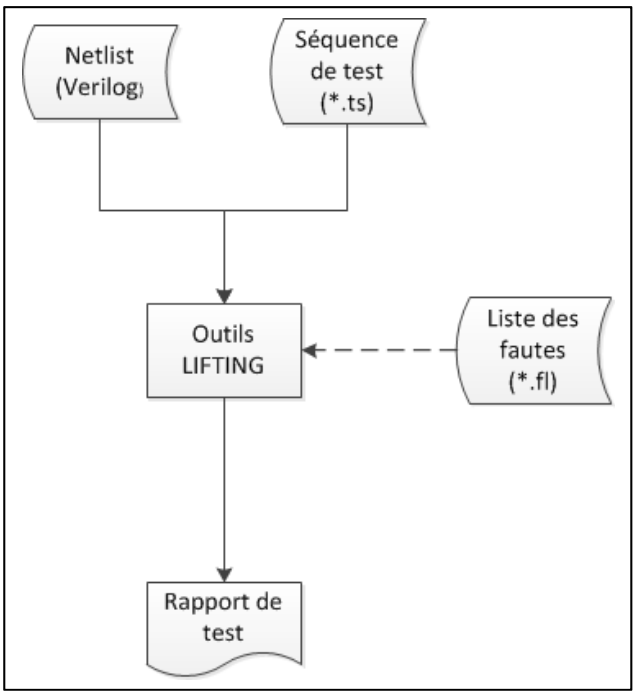

Figure 1.14 Architecture de l'outil LIFTING

1.3.3 Bilan des différentes méthodologies d'injection de pannes

Le Tableau 1.2 présente le résumé des avantages et des inconvénients des différentes techniques d'injection présentées dans cette section.

Tableau 1.2 Récapitulatif des différents avantages et inconvénients des différentes techniques d'injection de pannes

Techniques	Avantages	Inconvénients
Circuit Physique	• bons modèles de pannes bas niveau ; • non intrusif ; • rapide d'exécution.	• peut-être destructif ; • accessibilité peut être limitée par divers facteurs ; • faible observabilité ; • types de pannes limités ; • nécessite un équipement.
Émulation	• plus rapide que par simulation ; • possibilité d'implémenter les vecteurs d'entrée sur le FPGA pour encore augmenter la vitesse.	• nécessite un code VHDL/Verilog synthétisable et optimisé ; • coût d'un FPGA ; • analyse seulement les conséquences fonctionnelles (non temporelles) ; • nombre d'entrées/sorties limitées par le FPGA ; • connexion entre ordinateur et carte de développement.
Simulation	• peut supporter de nombreux niveaux d'abstraction ; • bonne contrôlabilité ; • faible coût ; • bonne observabilité ; • modèles de pannes diversifiés.	• travail de développement ; • temps de simulation ; • création des modèles de pannes ; • la précision du résultat dépend de la précision du modèle.

Par émulation, il est possible d'injecter des pannes en utilisant le prototypage grâce à un FPGA. L'injection de pannes est plus rapide que par simulation. La reconfiguration permet de ne pas devoir instrumenter le circuit. Le temps nécessaire à la préparation de l'injection est donc plus faible. Globalement, les performances par émulation dépendent des caractéristiques du FPGA et de la rapidité de la connexion entre l'ordinateur et le FPGA.

Avec l'approche par instrumentation, la communication entre l'utilisateur et le FPGA nécessite le transfert de moins de données, ce qui peut améliorer la performance. La taille du FPGA et le nombre d'entrées/sorties reste cependant un facteur limitant. Par simulation, l'injection de pannes au niveau instruction et systèmes utilisent des modèles de pannes grossiers. De plus, le temps nécessaire à la simulation au niveau instruction est déjà conséquent. L'injection de panne par simulation au niveau RTL peut s'effectuer par instrumentation ou grâce aux commandes du simulateur. Dans ce dernier cas, l'injection est dépendante des capacités du simulateur. L'instrumentation, plus difficile à mettre en œuvre, permet l'utilisation de modèles de pannes plus précis. Cependant, il faut modifier la description originale du circuit. Au niveau porte logique, la simulation permet l'injection de pannes utilisant des modèles encore plus précis. De plus, l'analyse est meilleure, car la structure exacte du circuit est connue.

Nous avons vu les différentes techniques d'injection de pannes à ce jour. L'injection de pannes peut se faire à différents niveaux d'abstraction et à différents moments du flot de conception. Chaque méthodologie possède ses propres avantages et inconvénients. Typiquement, plus le niveau d'abstraction est bas plus les modèles de pannes seront précis. En contrepartie, le temps nécessaire pour l'injection à bas niveau est normalement élevé.

Il existe très peu d'outils permettant l'injection par simulation de pannes permanentes au niveau portes logiques. À notre connaissance, seul LIFTING et VFIT peuvent effectuer ce genre de simulation. Or VFIT est dépassé, la complexité d'un modèle au niveau portes logiques et la mauvaise gestion de la mémoire de l'outil font qu'il est impossible d'utiliser la liste d'interconnexions comme modèle pour y injecter des pannes. L'outil est actuellement en refonte sur des plateformes plus récentes afin de supporter cela.

Pour ce projet, un des objectifs est d'injecter des pannes à haut niveau d'abstraction pour simuler l'effet des rayons cosmiques. Pour cela, l'outil de MathWorks, Matlab/Simulink sera utilisé. Au niveau système beaucoup de détails d'implémentations sont abstraits, l'accessibilité est limitée aux entrées/sorties des blocs. À notre connaissance, MODIFI est le

seul un outil d'injection de pannes qui existe sur cette plateforme. Afin de collecter de l'information à bas niveau d'abstraction dans le but de caractériser avec précision le comportement fautif d'un circuit. Nous opterons donc pour de l'injection de pannes par simulation au niveau portes logiques. L'outil le plus adapté à cette application est LIFTING. Il permet l'injection automatisée de pannes variées sur tous les nœuds du circuit ainsi que de générer des rapports précis. De plus, il s'agit d'un logiciel libre. Si besoin est, il est possible de modifier l'outil afin de l'adapter aux objectifs ainsi qu'à son utilisation. Nous allons aussi proposer une seconde méthodologie permettant d'injecter des pannes ciblées sur certaines parties d'un circuit implémenter sur un FPGA grâce à l'outil SEU Controller.

CHAPITRE 2

MÉTHODOLOGIE PROPOSÉE DU PROJET

Dans ce chapitre nous présentons la méthodologie proposée dans ce projet afin de créer une bibliothèque de composants, à haut niveau d'abstraction, émulant le comportement fautif d'un circuit dû aux radiations. Nous souhaitons apprendre le comportement fautif d'un circuit et l'adapter à plus haut niveau d'abstraction afin d'injecter des pannes similaires à des rayons cosmiques. Avec des composants émulant le comportement fautif d'un circuit, il sera possible d'effectuer une analyse de fiabilité d'un modèle haut niveau, par exemple sous Simulink.

Dans un premier temps, nous présenterons la méthodologie de manière globale. Ensuite, nous décrirons les différentes étapes, les outils utilisés ou développés, les difficultés et limitations ainsi que les métriques qui nous permettront d'analyser la performance et les résultats obtenus.

2.1 Flux de conception et niveaux d'abstractions

Afin de comprendre la problématique de ce projet, il est important de comprendre les différentes étapes du flux de conception d'un circuit ainsi que les niveaux d'abstractions liés à cela. Comme détaillé à la Figure 2.1, un concepteur commence par valider l'algorithme de son circuit, puis son architecture, avant de réaliser son circuit physique. Chacune de ces étapes peut être réalisée avec différents outils, à différents niveaux d'abstraction. Pour une même étape, différents niveaux d'abstraction existent. Le modèle devient de plus en plus précis et détaillé lorsque le niveau d'abstraction diminue. À haut niveau, les détails sont abstraits, les modèles sont théoriques. À bas niveau d'abstraction, les modèles deviennent plus concrets et détaillés.

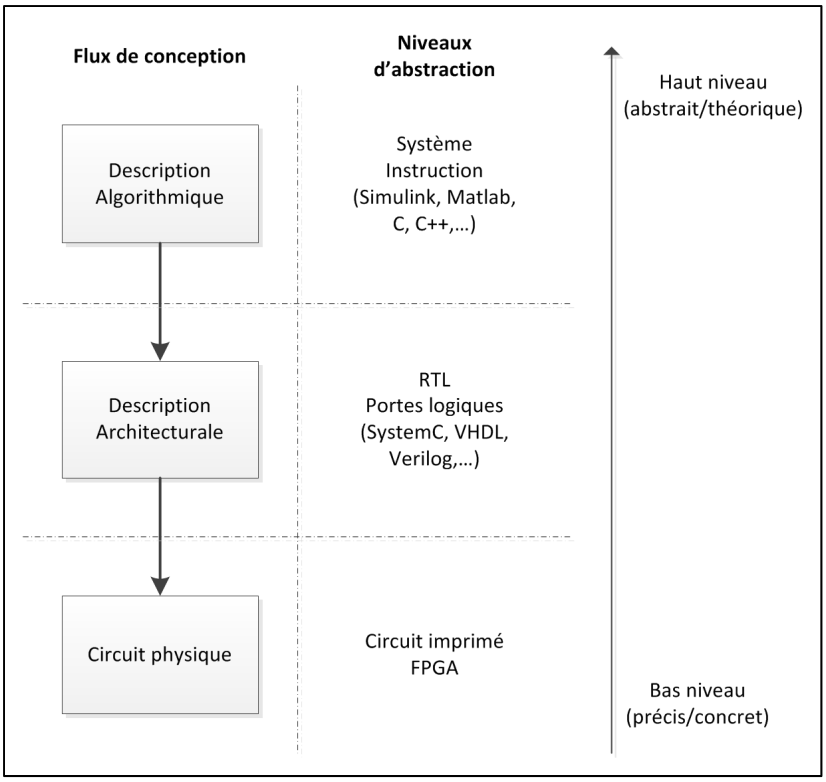

Figure 2.1 Diagramme du flux de conception d'un circuit ainsi que des niveaux d'abstractions liés aux différentes étapes

Dans ce projet, un des objectifs est de permettre d'injecter des pannes similaires à celles causées par des rayons cosmiques dans des modèles à hauts niveaux d'abstraction. Or, beaucoup de détails manquent, et il est impossible d'avoir la même précision et accessibilité qu'à bas niveau d'abstraction. Une méthodologie permettant de faire le lien entre le comportement fautif appris à bas niveau d'abstraction et un modèle haut niveau est proposée.

Un des objectifs de ce travail est de proposer des outils permettant d'effectuer une analyse de sureté tôt dans le flux de conception d'un circuit. Cette analyse, normalement effectuée plus tard, pouvait obliger les concepteurs à revenir dans le flux de conception afin d'adapter l'architecture ou l'algorithme pour rendre le circuit plus robuste.

2.2 Méthodologie de création de composants fautifs

Afin de mener à bien cette réalisation et de répondre aux objectifs précédemment définis, les trois étapes suivantes sont proposées : 1) la génération des rapports d'injection de pannes, 2) la génération de Signatures, et 3) la création de composants fautifs sous Simulink. L'objectif final est de proposer une méthodologie complète pouvant être répétée afin de générer ses propres composants Simulink ainsi que de présenter les limites et les caractéristiques des différents outils et étapes de ce projet.

La Figure 2.2 représente le schéma du procédé en trois étapes construit afin de répondre aux objectifs. Les différentes entrées/sorties et les outils y sont résumés. La méthodologie débute à partir d'un modèle bas niveau (portes logiques), et aboutis à la génération d'un composant fautif à haut niveau d'abstraction.

Afin de construire la première étape, un état de l'art des différentes techniques d'injection de pannes, et tout particulièrement des outils d'injection de pannes par simulation, ainsi que des outils Xilinx a été effectué. Cet état de l'art a abouti sur un choix d'un outil d'injection par simulation : LIFTING, ainsi que sur la définition d'une approche secondaire d'injection par émulation. Nous avons expérimenté avec ces outils, afin de déterminer et de comparer leurs performances et leurs limites. Pour cela, différents circuits ont été utilisés, en faisant varier les paramètres disponibles des outils. Le procédé ainsi que les choix effectués et leurs conséquences sur le résultat obtenu seront présentés dans le CHAPITRE 3 avec plus de précision.

La seconde étape consiste à transformer et adapter l'information contenue dans les rapports d'injection de pannes afin d'être applicable à plus haut niveau d'abstraction. Afin d'effectuer le « saut » entre ces niveaux, le nouveau concept de Signature du comportement fautif d'un circuit est proposé. Une Signature permet de capturer le comportement fautif observé à bas niveau d'abstraction et de l'appliquer à plus haut niveau d'abstraction, au niveau système. L'outil OGAS (Outil de Génération Automatisé de Signatures) a été créé afin de transformer

les rapports d'injection de pannes en Signature. Différents formats de Signature ont été définis en expérimentant avec des outils à haut niveau d'abstraction, comme Simulink, afin de déterminer comment formater l'information. La précision et la compression des différents formats en fonction des différents paramètres disponibles ont été étudiées. Dans cette étape les résultats générés suite à l'injection de panne de la première étape sont réutilisés. Dans le CHAPITRE 4, l'étude de l'effet des différents paramètres disponibles sur la précision des Signatures du comportement fautif d'un circuit sera présentée.

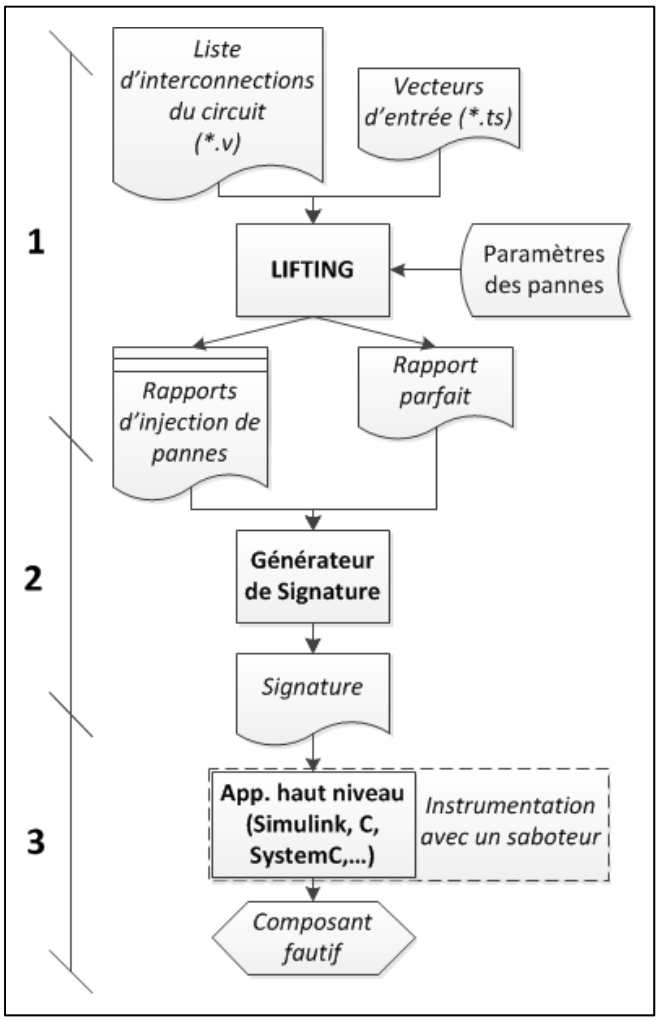

Figure 2.2 Schéma de procédé détaillé de la méthodologie proposée

La dernière étape consiste à instrumenter un modèle haut niveau, ici sous Simulink, afin d'injecter des pannes similaires à des rayons cosmiques grâce aux Signatures qui ont été générées. L'objectif est de reproduire à haut niveau le comportement observé à bas niveau d'abstraction grâce à l'outil LIFTING. Pour cela, il est possible de créer un bloc saboteur permettant la lecture modulaire de différentes Signatures. Les paramètres critiques sont la précision de l'injection ainsi que les performances de la simulation. Des composants fautifs ont été réalisés sous forme de blocs grâce aux Signatures qui ont été générées dans les étapes précédentes. Un concepteur peut alors remplacer sa fonction par un bloc erroné, simulant le comportement fautif du circuit. Les résultats des expérimentations seront détaillés dans le CHAPITRE 5.

2.2.1 Étape 1 : Génération des rapports d'injection de pannes

Afin de capturer le comportement fautif d'un circuit électronique, il faut l'étudier à un bas niveau d'abstraction. Pour cela, une approche principale a été choisie : l'injection de pannes par simulation à partir de la liste d'interconnexion d'un circuit. Une approche secondaire, utilisant l'injection par émulation, est aussi présentée. Bien que les performances des deux approches seront étudiées et comparées, uniquement l'approche principale sera retenue pour la suite de cette étude.

L'étape 1, présentée à la Figure 2.2, consiste à injecter des pannes similaires à l'effet de SEU sur un circuit par simulation. Un FPGA est majoritairement composé de cellules mémoires permettant de stocker sa configuration, il s'agit donc de la préoccupation principale lorsque l'on étudie la fiabilité en considérant les SEEs. Une modification du contenu d'une cellule mémoire sera permanente jusqu'à ce qu'elle soit reprogrammée. Un SEU sur un bit critique de la configuration modifie le routage ou la fonctionnalité du design. Afin de modéliser ce comportement, le modèle de pannes choisi est le collage à 1 ou 0. Ce modèle consiste à forcer la valeur d'un nœud lors de la simulation indépendamment de la valeur théorique. Comme observé dans la Figure 2.3, en fonction des entrées du modèle, la panne peut se propager ou non vers la sortie, on parle de masquage logique. Si la panne se propage, on

obtient une erreur observable. Il est possible qu'un collage n'ait aucun effet s'il est équivalent à la valeur parfaite du nœud où a lieu l'injection. Dans le premier exemple la panne est masquée, dans le second elle se propage et devient observable, et dans le dernier la panne n'a aucun effet sur la logique.

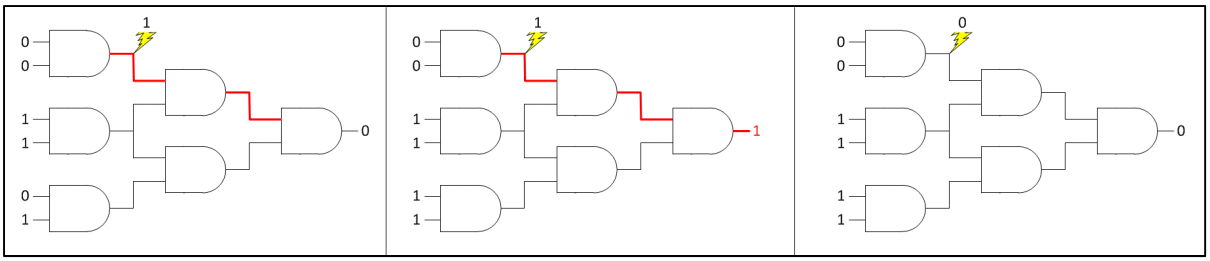

Figure 2.3 Propagation, masquage logique et observabilité d'une panne de type collage

Afin d'injecter des collages à 0 ou 1 par simulation, l'outil LIFTING a été choisi. Il permet d'injecter ce type de pannes à partir de la liste d'interconnexions d'un circuit. Il s'agit de la méthodologie principale. Comme présenté dans la Figure 2.2, cet outil nécessite la liste d'interconnexion d'un circuit, la liste des vecteurs d'entrées et les paramètres des pannes (lieux et type). Les rapports d'injection de pannes sont ainsi générés. Ils permettent de caractériser le comportement fautif d'un circuit à bas niveau d'abstraction. Pour cette approche, le modèle de panne est une approximation de l'effet d'un SEU sur un FPGA. De plus, la simulation d'injection de pannes dans un circuit de taille conséquente peut être contraignante au niveau du temps de simulation ainsi que dans la taille des rapports d'injection. Afin de caractériser et de critiquer les résultats obtenus, les paramètres suivants ont été étudiés :

- le temps de simulation ;
- la taille de la liste de vecteur d'entrée ;
- la taille des rapports générés.

Une approche secondaire, utilisant l'émulation, est aussi proposée. Elle permet d'avoir un modèle de panne plus précis, ainsi qu'un temps d'exécution plus court en implémentant le circuit sur un FPGA. Pour cela, l'outil SEU Controller est utilisé. Il permet d'injecter des

pannes dans la configuration du FPGA afin de simuler un SEU. Cette méthode est beaucoup plus précise que la précédente, le modèle de pannes est très proche de la réalité. Cependant, cela nécessite d'avantage de préparation, il faut avoir un modèle qui peut être implémenté sur un FPGA, établir une connexion avec l'ordinateur afin de récupérer l'information, et utiliser des outils complémentaires pour cibler une partie du FPGA pour y injecter des pannes. Cette approche est donc plus précise et rapide d'exécution, mais demande d'utiliser des outils plus complexes à mettre en œuvre et nécessite plus de temps de préparation.

2.2.2 Étape 2 : Génération des Signatures

Afin de transformer l'information obtenue dans les rapports d'injection de pannes, un nouveau concept permettant de capturer le comportement fautif d'un composant et de l'appliquer aisément à plus haut niveau est proposé. Ce concept a été nommé Signature du comportement fautif d'un circuit, ou simplement Signature. Ce terme sera défini dans la suite de ce mémoire dans le CHAPITRE 4. Les objectifs sont de proposer un nouveau format, permettant de transformer une information apprise à bas niveau jusqu'à un niveau d'abstraction supérieur de manière compacte, facilement lisible et précise. Chaque Signature correspond à une compression du comportement fautif d'un circuit. Comme spécifié dans la Figure 2.2, les Signatures sont générées à partir des rapports d'injection ainsi que du rapport parfait générés par LIFTING. Deux formats sont proposés, ils s'appliquent à différents niveaux d'abstraction, sous Simulink et pour des applications génériques haut niveau. Les circuits combinatoires seront la cible de l'étude effectuée dans ce travail de recherche. Le comportement fautif d'un circuit séquentiel est beaucoup plus difficile à modéliser, car il faut prendre en compte une séquence de vecteurs d'entrées. Le modèle de Signature deviendrait alors beaucoup plus complexe.

Pour cela, l'outil OGAS a été réalisé. Il permet la transformation des rapports d'injection de pannes afin de créer des Signatures pour différentes applications. En analysant les rapports générés par LIFTING, l'outil compresse et caractérise le comportement fautif pour l'adapter dans un format lisible à plus haut niveau d'abstraction. Les paramètres de cet outil et leurs

effets seront détaillés dans le CHAPITRE 4 dédié à ce sujet. Cette partie traitera de la précision et la compression des Signatures ainsi générées en fonction des rapports d'injection ainsi que des paramètres de génération des Signatures.

2.2.3 Étape 3 : Instrumentation sous Simulink

Enfin, après avoir généré des Signatures, il est nécessaire de créer une structure sous Simulink permettant de les lire et ainsi créer une bibliothèque de composants fautifs. Le but est d'obtenir une bibliothèque de composants standards pour les utilisateurs désirant juger de l'impact des rayons cosmiques sur leurs modèles hauts niveaux. Une approche permettant de générer sa propre bibliothèque en fonction des besoins de son projet est proposée.

Afin de lire et d'appliquer les Signatures à plus haut niveau, un saboteur sera ajouté comme présenté dans la Figure 2.4. Nous rappelons qu'un saboteur est un bloc qui perturbe le signal reliant deux blocs. L'objectif de cette étape est de permettre d'appliquer à haut niveau d'abstraction le comportement fautif d'un circuit appris au niveau porte logique. L'objectif ici est d'obtenir un outil d'injection compacte et modulaire qui ne ralentit pas considérablement la simulation.

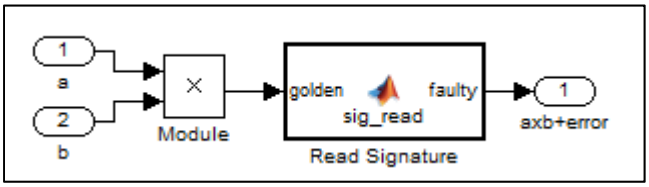

Figure 2.4 Multiplicateur suivi d'un saboteur utilisant une Signature, sous Simulink

L'objectif est de réaliser une injection de pannes à haut niveau, donnant des résultats proches de ce que l'on pourrait faire à bas niveau. À haut niveau d'abstraction, seules les entrées et sorties primaires des blocs sont accessibles à l'injection, la lecture d'une Signature apprise à bas niveau permettrait de pallier à ce manque d'accessibilité. Le saboteur va injecter des pannes de manière pseudo-aléatoire en respectant le comportement fautif décrit dans la Signature. Les expérimentations sont caractérisées en étudiant le temps de simulation ainsi

qu'en comparant les résultats d'injection obtenus sous Simulink avec le comportement fautif exact attendu observé à bas niveau d'abstraction. Les résultats seront présentés dans le CHAPITRE 5.

2.2.4 Bilan de la méthodologie

Dans ce chapitre, une méthodologie en trois étapes permettant de générer des composants fautifs à haut niveau d'abstraction simulant l'effet des rayons cosmiques a été présentée. La première étape consiste à générer des rapports d'injection de pannes à bas niveau d'abstraction. Pour cela, une approche principale utilisant l'outil LIFTING a été proposée. Elle permet d'injecter des pannes de type collage à 1 ou 0 par simulation. Une approche secondaire utilisant l'outil SEU Controller pour injecter des pannes par émulation a été proposée. Bien que les deux approches soient présentées, seule la principale sera utilisée dans le cadre des expérimentations de ce projet. La deuxième étape, utilise les rapports d'injections ainsi générés et permet de les compresser et adapter pour être réutilisés à plus haut niveau d'abstraction. Un nouveau concept de Signature a été introduit. Ces Signatures permettent de capturer le comportement fautif d'un circuit à bas niveau. Elles sont réutilisées dans la troisième étape sous Simulink pour générer des composants fautifs simulant le comportement observé à bas niveau d'abstraction grâce à LIFTING.

Cette méthodologie, si répétée pour différents circuits, permet de générer une bibliothèque de composants fautifs. Ces blocs permettront à un concepteur d'effectuer une analyse de sureté tôt dans le flux de conception d'un circuit. Une méthode permettant d'injecter des pannes similaires à des rayons cosmiques malgré le haut niveau d'abstraction limitant les détails des différents blocs utilisés a été proposée.

CHAPITRE 3

GÉNÉRATION DES RAPPORTS D'INJECTION DE PANNES

La première étape de la méthodologie proposée consiste à injecter des pannes dans le modèle d'un circuit. Ces pannes doivent correspondre à l'effet que produirait une perturbation de type SEU. L'objectif ici est de caractériser l'effet des rayons cosmiques sur la sortie primaire d'un modèle d'un circuit électronique combinatoire.

Pour cela, une approche principale est utilisée. Elle consiste à utiliser la liste d'interconnexions d'un circuit en y injectant des pannes par simulation grâce à l'outil LIFTING. Une discussion présente l'utilisation de cet outil, les paramètres utilisés, les résultats obtenus ainsi qu'une critique sur les performances et les limitations de l'outil à travers trois études de cas.

Ensuite, l'approche secondaire utilisant l'injection par émulation sera présentée. Les avantages et les inconvénients de ces deux méthodologies seront comparés. Aucune étude de cas ne sera proposée pour cette deuxième méthodologie, il s'agit uniquement d'une ouverture potentielle pour des projets futurs afin de proposer une méthodologie alternative.

3.1 Présentation de l'approche d'injection de pannes au niveau portes logiques par simulation avec l'outil LIFTING

Afin d'effectuer l'injection de panne par simulation, trois éléments sont nécessaires : la liste d'interconnexion du circuit, une liste de pannes, ainsi qu'une liste de vecteurs d'entrée à injecter. La version 1.1 de l'outil LIFTING a été utilisée. Les différents paramètres disponibles sont représentés dans la Figure 3.1. La version de LIFTING utilisée a été modifiée pour les besoins de ce projet. Les rapports d'injection ont été simplifiés. Ils sont initialement au format HTML avec les bits erronés surlignés rouge. Les rapports sont maintenant générés au format texte sans le code couleur pour les bits erronés. Comme on

peut le voir à la Figure 3.1, les options de LIFTING sont nombreuses. Les paramètres spécifiés et utilisés pour l'injection de panne sont résumés dans le Tableau 3.1. La bibliothèque AMS3.70 est utilisée pour générer des listes d'interconnexions (netlist). Les listes de vecteurs d'entrée sont générées préalablement au format spécifique à LIFTING (.TS). Le rapport parfait, ainsi que des rapports par pannes (collages à 1 et 0) sont générés. Pour chaque panne, un rapport est généré. Ces rapports contiennent le résultat de la simulation pour tous les vecteurs d'entrées spécifiés. Il est nécessaire d'utiliser l'algorithme séquentiel pour générer ce type d'information. La liste de pannes n'est pas spécifiée afin que l'outil considère la liste complète de pannes.

```
Usage: ./lifting [options]
        -l [or --library] { c35 core9gp }    Tech library: c35=AMS_3.70   core9gp=ST_CORE9GP   Default=C35
        -n [or --netlist] <filename>         File name of the verilog netlist. MANDATORY
        -i [or --input] <filename>           File name of the input patterns. MADNATORY
        -o [or --write_output <filename>     Name of the file where the simulator writes the output of the simulation.
                                             If not specified, the output of the simulation is not written.
        -fs [or --fault_simulation {stuckat|bitflip}
                                             Type of the fault simulation. If not specified, the fault simulation is not executed.
                                             Default=no fault simulation.
        -pfs [or --print_fault_simulation]   Default=not asserted. If asserted, it prints the output of the simulation for each fault
                                             defined in the faultlist.
        -pfsd [or --print_fault_simulation_dir] <dirname>
                                             Base directory for "print fault simulation" output. Default=./
        -fsa [or --force_sequential_algorithm]
                                             If set, the simulator uses the algorithm for sequential circuit.
                                             Fault statistic reports are different based on the executed algorithm.
                                             It can be useful to collect more information (for combinational circuits).
        -saf [or --saf_observe_last_input_only]
                                             If set, it observes the output of the circuit only after the last input pattern.
        -wfs [or --write_fault_statistic] <filename>
                                             If set, it writes a file(s) with the statistics of covered and not covered faults.
                                             <filename> is the name of the output file for the statistics (no extension).
        -fd [or --full_dictionary]
                                             If set, it stores the full dictionary. By default only the pass/fail dictionary is created .
        -wtft [or --write_truth_fault_table] <filename>
                                             If set it writes a file(s) with the truth table of the good machine and all the faulty machines.
                                             <filename> is the name of the output file (it can be big!!).
                                             WARNING: Sequential algorithm must be used.
        -fl [or --faultlist_filename] <filename>
                                             Name of the external filelist (tetramax format).
        -bitflip_experiments <e>             Total number of bitflips.
        -bitflip_mintime <m>                 For bit_flip fault model, the fault simulation is performed by injecting a bit flip
        -load_memory <memory_name><filename> Name of the memory instance and file storing the memory content
                                             starting from a particular time. This parameter specifies the lower bound.

NOTE: Combinational algorithm uses "cumulative" fault model. It means that when a fault has been detected
      by a pattern, it won't be simulated using the following input patterns.
      On the other hand, the sequential algorithm simulates each fault using each pattern.
```

Figure 3.1 Capture d’écran des options de l'outil LIFTING v1.1

Tableau 3.1 Paramètres de simulation utilisés dans ce projet

Option	Valeur	Effet/Justification
-n	<<liste d'interconnexions>>	Liste d'interconnexions (netlist) du circuit désiré en verilog et utilisant la bibliothèque AMS3.70.
-i	<<liste des vecteurs d'entrée>>	Liste des vecteurs d'entrée dans le format spécifique à LIFTING (.TS).
-o	<<nom du rapport parfait>>	Rapport de simulation parfait.
-fs	stuckat	Choix du modèle de panne.
-pfs		Imprime la sortie de la simulation pour chaque panne.
-pfsd	<<dossier des rapports d'inj.>>	De nombreuses pannes sont injectées, il est préférable de classer les rapports dans un dossier spécifique.
-fsa		Utilisation de l'algorithme séquentiel afin de simuler chaque panne pour chaque vecteur d'entrée.

Une fois ces paramètres spécifiés, une simulation pour un circuit spécifique est effectuée en utilisant la liste de vecteurs d'entrées données. La liste de pannes n'est pas spécifiée, l'outil va alors, pour chaque panne, étudier leur effet sur le résultat de la simulation pour chaque vecteur d'entrée spécifié dans la liste fournie dans les options. La simulation se termine lorsque toutes les pannes ont été simulées pour tous les vecteurs d'entrée comme spécifié à la Figure 3.2.

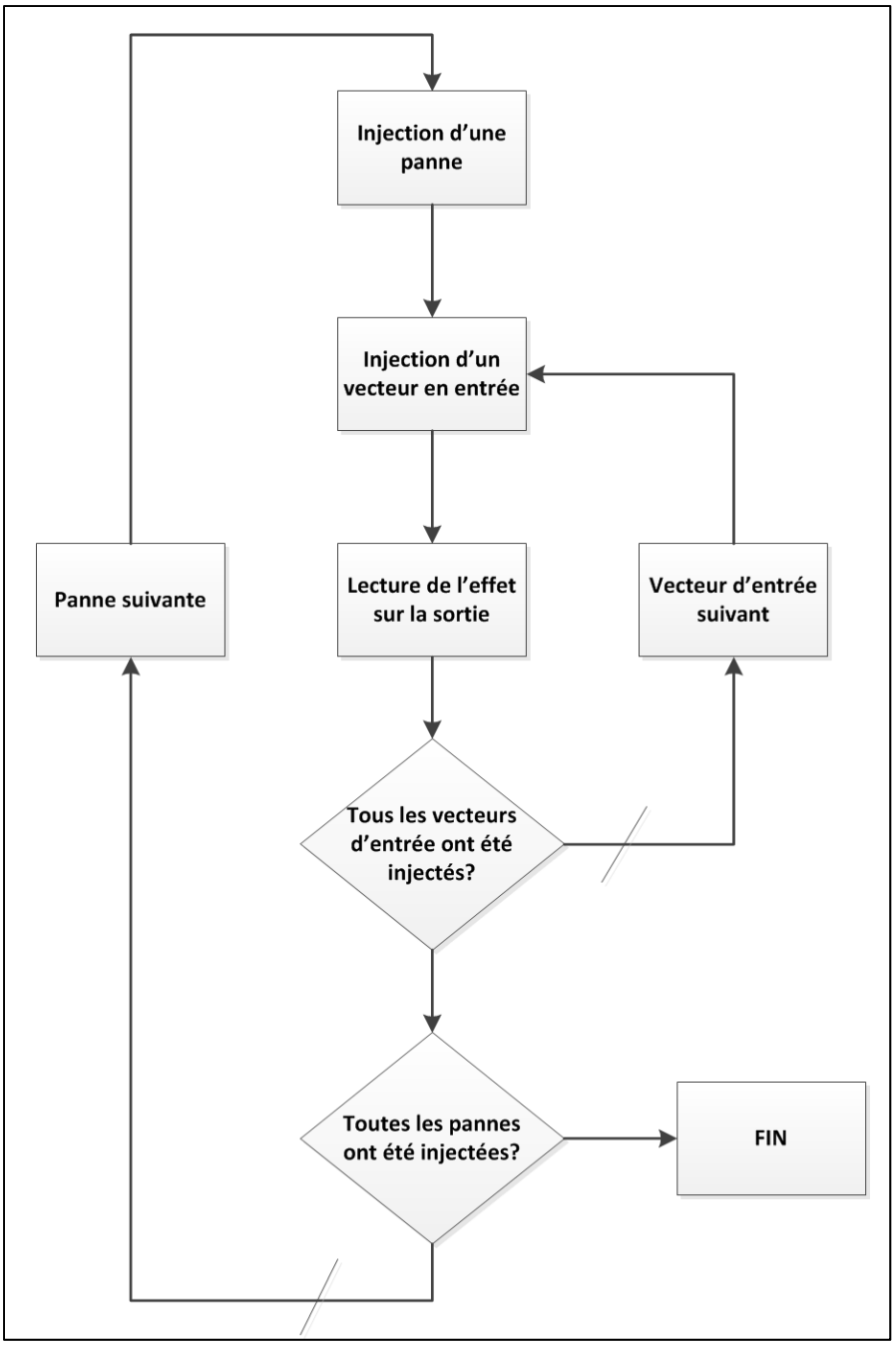

Figure 3.2 Procédé d'injection de pannes par simulation avec LIFTING

Ainsi, pour chaque panne contenue dans la liste de pannes, les vecteurs d'entrée spécifiés dans la liste de vecteur d'entrées sont injectés un par un en prenant soin de mesurer l'impact de la panne sur la sortie.

Ce cycle peut être très long si le circuit est de grande taille. Plus le circuit comporte un grand nombre de portes logiques, plus le nombre de points d'injections sera élevé et donc plus la liste de pannes sera grande. Ceci a pour conséquence d'augmenter considérablement le temps d'injection.

La Figure 3.3 montre un exemple de quelques rapports d'injection générés par LIFTING ainsi que leur contenu. Chaque nom de fichier comporte deux parties, la première désigne le lieu de l'injection, la seconde le type de panne. Par exemple, U45_S_SA0 signifie qu'une panne de type collage à 0 sera injectée sur le signal S dans le module U45. Le fichier contient la valeur la sortie primaire du circuit en présence de la panne pour chaque entrée spécifiée dans la liste de vecteurs d'entrée. Dans cet exemple, la sortie primaire du circuit utilisé possède 16 bits.

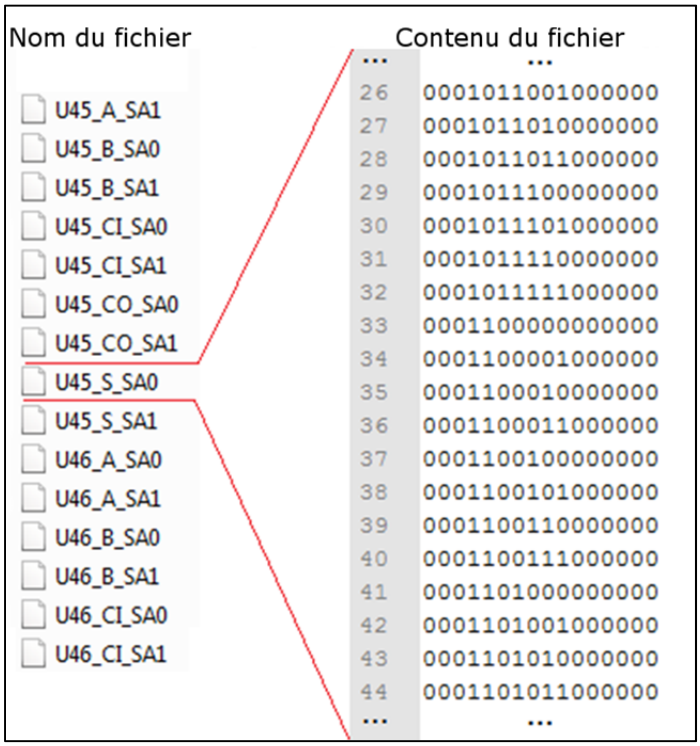

Figure 3.3 Exemple de rapports d'injection générés par LIFTING

Afin d'obtenir des résultats pertinents tout en limitant le temps de simulation, il faut injecter en entrée un nombre suffisant de vecteurs différents. Plus le circuit possède des entrées de

grande taille plus le nombre de vecteurs nécessaires est élevé. Il s'agit d'une autre limitation. Si la liste complète des vecteurs en entrée est utilisée, le temps de simulation sera maximisé, mais les résultats seront complets. Si une sous-partie de la liste complète est utilisée, de l'information sera perdue dans les rapports d'injection générés par LIFTING. En contrepartie, réduire le nombre de vecteurs peut réduire le temps d'injection.

3.2 Présentation de l'approche secondaire d'injection de pannes par émulation avec SEU Controller

Dans cette partie l'approche secondaire d'injection de panne par émulation est présentée. Comme détaillé dans la section 1.3.2.2 de la revue de littérature, une injection de panne par émulation est réalisée en implémentant le circuit ciblé sur un FPGA. Les pannes sont alors injectées par reconfiguration ou par instrumentation. La principale caractéristique de cette méthode est sa rapidité, ce qui est l'avantage principal de cette approche comparé à l'injection par simulation. En revanche, par émulation, le modèle de pannes injecter peut être extrêmement réaliste. C'est pourquoi ici, une ouverture potentielle est proposée afin de réduire considérablement le temps de simulation tout en utilisant un modèle de pannes très proche de la réalité.

SEU Controller permet d'injecter des pannes similaires à des SEU dans un FPGA en cours en fonctionnement en modifiant sa configuration. Le modèle de panne est réaliste. Grâce aux résultats obtenus, ceux obtenus à partir de l'injection de pannes au niveau portes logiques seront critiqués.

La structure détaillée proposée est présentée dans la Figure 3.4 afin d'injecter des pannes dans un modèle. Chaque module est implémenté sur le FPGA, l'entrée est générée par un Générateur pseudo aléatoire. Il existera deux versions implémentées sur le FPGA, une permettant d'obtenir la sortie de référence et une seconde qui sujette à l'injection de pannes. L'information peut alors être traitée et compressée dans le FPGA pour être ensuite transmise à l'ordinateur

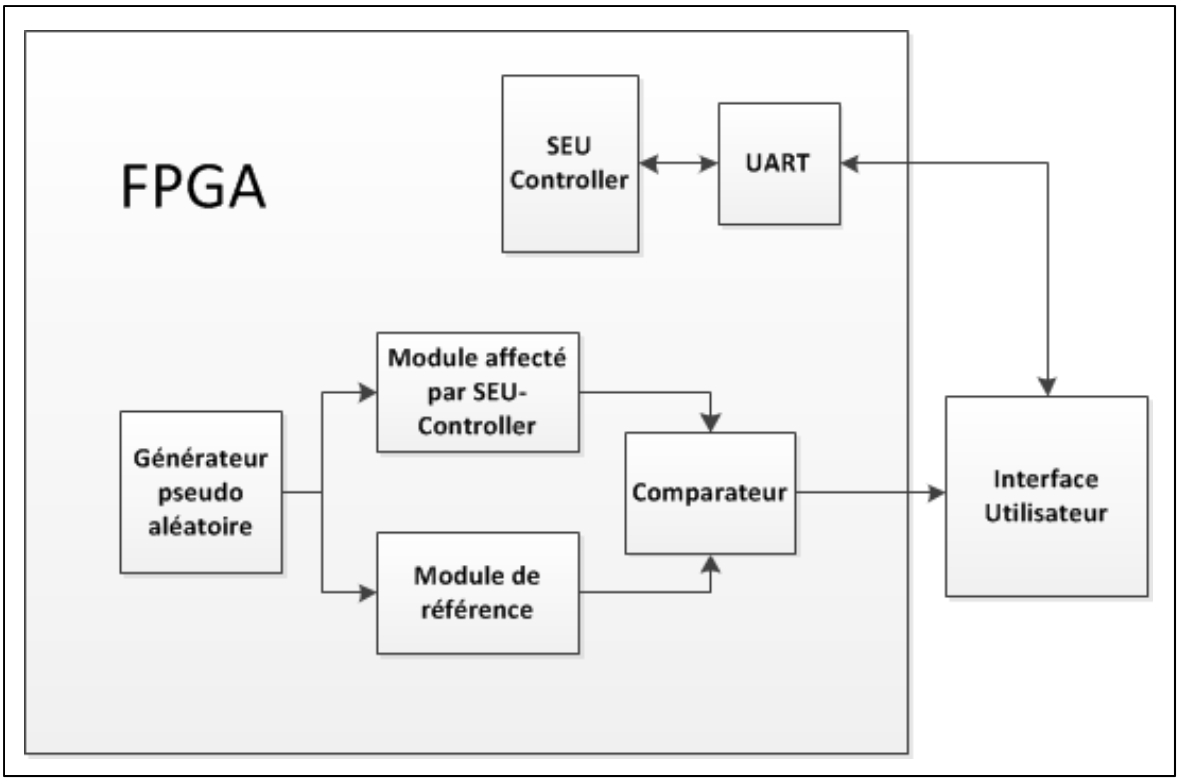

Figure 3.4 Implémentation de la structure de test avec SEU Controller

Le nombre de bits nécessaires au design implémenté dans le FPGA correspond à une faible proportion du nombre total en fonction de la taille du circuit implémenté (Le, 2012). De plus, une partie de ces bits ne sont pas critiques et leur modification n'entrainera aucun changement majeur dans le circuit. Xilinx propose des outils permettant d'estimer quels sont les bits essentiels du design.

Afin d'injecter des pannes susceptibles de causer une erreur dans le design, il est possible d'utiliser la commande BitGEN qui permet de générer les données de configurations. En spécifiant l'option BitGEN –g EssentialBits :Yes, deux fichiers sont alors générés. Le premier, le fichier EBC, contient la donnée de configuration telle qu'elle serait lue par la fonction de relecture de SEU Controller (readback). Il est important de noter que ce fichier est différent du BIT file, il s'agit uniquement du contenu des trames de configuration telle que présentes sur le FPGA. Le second fichier généré par BitGEN, le fichier EBD, contient le masque indiquant quels bits du fichier EBC sont essentiels au bon fonctionnement du design.

Le fichier EBD est de la même taille que le fichier EBC ; les bits correspondent un à un entre ces deux fichiers. Dans le fichier EBD, un bit du masque est à 0 lorsque BitGEN le définit comme étant non essentiel. Cette catégorie de bit correspond aux bits qui ne sont pas associés avec les éléments composant le design. Si un bit du masque est à 1 alors il est considéré comme étant essentiel et nécessaire à la définition des éléments composant le design (Le, 2012).

Il existe une fonction complémentaire de BitGEN, permettant de cibler ou d'exclure une partie du circuit afin de générer les fichiers EBC et EBD. Avant de définir cette méthode, il est important de définir ce qu'est une zone de contrainte (ou area constraint). Il s'agit, d'une zone physique sur le FPGA, correspondant à des ressources spécifiques, dans laquelle il est possible de contraindre l'implémentation d'une instance du design. Afin d'utiliser la fonction de filtrage de BitGen, quatre étapes sont nécessaires :

1. Créer une ou plusieurs zones de contrainte pour les parties ciblées comme dans l'exemple suivant ;

```
## Zone de contrainte sur une région hiérarchique du projet :
INST "example/example_adder/*" AREA_GROUP = "PARTIE_CIBLE" ;
INST "example/example_adder2/*" AREA_GROUP = "PARTIE_CIBLE" ;
AREA_GROUP "PARTIE_CIBLE" RANGE = SLICE_X49Y110:SLICE_X47Y132 ;
## Interdire l'addition de logique non incluse dans ce groupe
AREA_GROUP "PARTIE_CIBLE" GROUP = CLOSED ;
## Autoriser le placement de composants non inclus dans ce groupe dans la zone
AREA_GROUP "ESSENTIAL_LOGIC" PLACE = OPEN ;
```

2. Implémenter le design ;

3. Ouvrir l'implémentation avec FPGA editor et réaliser les commandes suivantes afin de générer le fichier log portant le nom du circuit suivi de *_routedfpga_editor* ;

```
clear
select -k comp *example/example_adder/*
select -k comp *example/example_adder2/*
select -k net *example/example_adder/*
select -k net * example/example_adder2/*
list
clear
exit
```

4. Spécifier les options BitGen.

```
-g essentialbits :yes
-g essentialbitsfilter :(none|mask|enable)
-g essentialbitsfilterfile :XXX_routedfpga_editor.log
```

Dans cette dernière étape il existe deux types de filtrage ; mask et enable. Le premier permet de générer les fichiers EBC et EBD de tout le circuit en excluant la partie définie. Le filtrage enable, quant à lui, permet de générer ces deux fichiers uniquement pour les entités du design spécifiées. De cette manière, il est possible d'isoler une partie du design, de générer la liste des bits essentiels correspondant et donc de les cibler avec SEU Controller. L'utilisation de SEU Controller est détaillée à travers un exemple en ANNEXE I.

Afin de capturer l'effet des pannes ainsi injectées, la formule permettant de calculer l'erreur peut être implémentée sur le FPGA pour simplifier le calcul ainsi que le transfert de données. Différentes options s'offrent à l'utilisateur à cette étape. Il est possible d'implémenter un module permettant la génération des Signatures et leur enregistrement dans des cellules mémoires pour ensuite transférer la Signature complète et finalisée vers l'utilisateur. Une autre solution serait de capturer le résultat de l'erreur pour chaque entrée grâce à un analyseur logique ou autre interface, pour ensuite traiter l'information et calculer la Signature grâce à un outil sur l'ordinateur de l'utilisateur. Cette solution semble plus modulaire, car les

paramètres de génération de la Signature peuvent être gérés par l'utilisateur de manière dynamique une fois toutes les données nécessaires capturées.

3.3 Études de cas

Afin de comprendre et d'analyser l'approche principale proposée pour l'injection de pannes par simulation, trois études de cas ont été réalisées. Ces expérimentations ont pour but de caractériser les performances l'outil de simulation utilisé ainsi que ses limitations. Le premier circuit utilisé est un additionneur 8 bits sans retenue. Il s'agit d'un circuit arithmétique, combinatoire, et simple. Il est composé d'un nombre restreint de portes logiques et possède une sortie primaire de 8 bits. Afin de tester le fonctionnement de LIFTING pour un circuit plus complexe, un multiplicateur 8 bits a été sélectionné. Ce circuit possède plus de portes logiques ainsi qu'une sortie primaire de 16 bits. Pour finir, le troisième circuit proposé est le circuit ISCAS'85 c432. Cette fois-ci, il s'agit d'un circuit logique de taille conséquente.

Les expérimentations sont réalisées sous Linux un serveur de type ProLiant DL380 G5 (XEON E5440, 2,83 GHz, 20 Go de RAM).

3.3.1 Injection de pannes par simulation dans un additionneur 8 bits

Pour cette première expérimentation, un additionneur sans retenue avec deux entrées primaires de 8 bits et une sortie primaire 8 bits a été choisi. Une fois décrit en VHDL, le code est ensuite synthétisé sous forme de liste d'interconnexion au niveau porte logique en Verilog. Pour cela, il faut utiliser un outil de synthèse tel que Design Vision de Synopsys. Afin que la liste d'interconnexion soit lisible par LIFTING il faut utiliser la bibliothèque AMS3.70 ou CORE9GP. Pour cette recherche, la première bibliothèque sera utilisée.

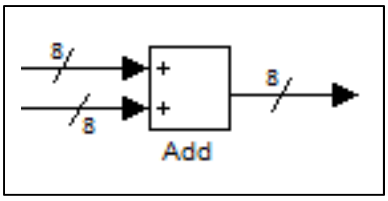

Figure 3.5 Additionneur 8 bits sans retenue

Une campagne d'injection de pannes de type collage à 0 et à 1 est lancée avec LIFTING. Suite à la simulation, le rapport présenté à la Figure 3.6 présente un sommaire de la campagne d'injection effectuée. On y retrouve entre autres le nombre de pannes injectées ainsi que la couverture. Dans ce cas-ci, 138 pannes différentes ont été injectées. Il existe donc 69 points d'injection en considérant que deux types de pannes (0/1) sont injectés par point d'injection.

```
SUMMARY:
  Library: AMS C35
  Netlist filename: add8_gate.v
  Input patterns filename: input.ts
  Simulation output filename: output.txt
  Fault model: Stuck-At
  Write fault statistics: no

Reading netlist....add8_gate.v done.
Reading input patterns....done.

summary..
 - # PI                          : 16
 - # PO                          : 8
 - # non_scan FFs                : 0
 - # scan FFs                    : 0
 - # memories                    : 0
 - # faults                      : 138

Running simulation of 65536 patterns.. done!
Allocating memory for fault list (138 Bytes)...done.

Starting fault simulation for sequential circuits...100% (138 faults)...done
Fault coverage=138 on 138 (100.0000)
```

Figure 3.6 Résumé de l'injection par simulation avec LIFTING pour un additionneur 8 bits avec la liste complète de vecteurs d'entrée

Cinq différentes campagnes d'injection ont été effectuées en utilisant un nombre différent de vecteurs d'entrée, à savoir 65 536, 50 000, 40 000, 30 000, 20 000. Nous étudierons, dans cette première étape, l'effet de la diminution du nombre de vecteurs lors de l'injection de panne sur les tailles des fichiers, et le temps d'injection en fonction du nombre de vecteurs d'entrée utilisés. Les résultats obtenus sont présentés dans le Tableau 3.2. En diminuant le nombre de vecteurs, plusieurs combinaisons de vecteurs sont possibles.

Tableau 3.2 Performances de la simulation d'injection de pannes pour un additionneur 8 bits en fonction du nombre de vecteurs d'entrée

Nombre de vecteurs d'entrée utilisés	**Taille de l'entrée (Ko)**	**Temps d'injection (s)**	**Taille des rapports (Mo)**
65 536	1280	30	77.6
50 000	977	23	59.1
40 000	782	18	47.4
30 000	586	14	35.5
20 000	391	9	23.7

Ce circuit de petite taille ne pose aucun problème de simulation. Le temps de simulation est court, et la taille des fichiers est réduite. Ceci se justifie par le faible nombre de lieu d'injection et donc de rapports. Les différents paramètres varient linéairement, ce qui est prévisible. Réduire le nombre de vecteurs en entrée est une solution viable pour diminuer le temps d'injection et la taille des rapports d'injection, bien que cela ne soit pas nécessaire dans ce cas. En revanche, une diminution du nombre de vecteurs utilisés engendrera une perte d'information inévitable. Il est impossible de quantifier cela à ce stade de la méthodologie.

3.3.2 Injection de pannes par simulation dans un multiplicateur 8 bits

Pour cette deuxième expérimentation, nous avons choisi d'utiliser un multiplicateur avec deux entrées 8 bits et une sortie 16 bits.

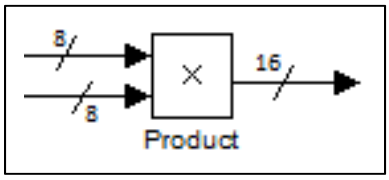

Figure 3.7 Multiplicateur 8bits

Cet exemple est plus complexe que le précédent. Cette fois-ci, comme observé dans le rapport d'injection présenté dans la Figure 3.8, le circuit possède 16 entrées et 16 sorties primaires. Il existe une liste de 1152 pannes différentes, soit presque dix fois plus que pour l'additionneur 8 bits.

Encore une fois, pour cet exemple cinq différentes campagnes d'injection ont été réalisées afin d'étudier l'effet de la réduction du nombre de vecteurs d'entrée. Le Tableau 3.3 spécifie : la taille du fichier d'entrée ainsi généré, le temps d'injection ainsi que la taille totale des rapports d'injection générés. Le but de cette étape est de comparer les résultats obtenus avec les précédents afin de mettre en avant l'augmentation de la complexité, d'estimer les limites de l'outil, et aussi de valider les conclusions tirées dans la partie précédente.

```
SUMMARY:
  Library: AMS C35
  Netlist filename: mult8_gate_ams.v
  Input patterns filename: input.ts
  Simulation output filename: output.txt
  Fault model: Stuck-At
  Write fault statistics: no

Reading netlist....mult8_gate_ams.v done.
Reading input patterns....done.

summary..
 - # PI                          : 16
 - # PO                          : 16
 - # non_scan FFs                : 0
 - # scan FFs                    : 0
 - # memories                    : 0
 - # faults                      : 1152

Running simulation of 65536 patterns.. done!
Allocating memory for fault list (1152 Bytes)...done.

Starting fault simulation for sequential circuits...100% (1152 faults)...done.
Fault coverage=1152 on 1152 (100.0000)
```

Figure 3.8 Résumé de l'injection par simulation avec LIFTING pour un multiplicateur 8 bits avec la liste complète de vecteurs d'entrée

Tableau 3.3 Performances de la simulation d'injection de pannes pour un multiplicateur 8 bits en fonction du nombre de vecteurs d'entrée

Nombre de vecteurs d'entrée utilisés	**Taille du fichier (Ko)**	**Temps d'injection (min)**	**Taille de la sortie (Mo)**
65 536	1280	22	1267
50 000	977	16,5	964
40 000	782	13,5	769
30 000	586	10	575
20 000	391	6,5	383

Le Tableau 3.3 détaille les résultats de simulation pour un multiplicateur 8 bits, à savoir le temps nécessaire à la simulation ainsi que la taille des fichiers d'entrée et de sortie. Plus le nombre de vecteurs d'entrée est grand plus la taille du fichier est grande et plus le temps

nécessaire à le générer est élevé. Encore une fois, le temps de simulation et la taille de la sortie restent proportionnels au nombre de vecteurs d'entrées utilisés. Ici, aucun problème n'est rencontré, car ce circuit possède uniquement 16 bits d'entrée. Il est important de remarquer que la liste complète des vecteurs d'entrée sera de plus en plus difficile à générer. Par exemple, la liste complète pour un circuit possédant des entrées primaires de 24 bits possèdera 16 777 216 lignes et la taille du fichier d'entrée sera de 459 Mo.

Le temps d'injection est un point critique de l'injection de panne par simulation. En effet, le circuit utilisé est simple et ne possède pas un très grand nombre de portes logiques. Si l'on utilise la liste complète de vecteurs d'entrées alors la simulation dure plus de vingt minutes sur cette plateforme, ce qui n'est pas négligeable pour un circuit non complexe.

Les rapports d'injection sont environ seize fois plus lourds que ceux de l'additionneur 8 bits présentés dans le Tableau 3.2. Ceci se justifie par l'augmentation du nombre de lieux d'injection ainsi que l'augmentation de la taille de la sortie. On multiplie par huit le nombre de pannes injectées et par deux la taille de la sortie, d'où le facteur seize précédemment calculé.

3.3.3 Injection de pannes par simulation dans le circuit ISCAS'85 c432

Comme dernier exemple, nous avons choisi le circuit c432 tiré du benchmark ISCAS 85. Il s'agit d'un contrôleur de canal d'interruption comme détaillé dans la Figure 3.9. Ce circuit comporte 4 entrées primaires 9 bits chacune et des sorties primaires de 7 bits. Afin de simplifier la simulation et de réduire le temps de simulation, uniquement les 4 premiers bits de chaque canal d'entrée seront utilisés. Le rapport d'injection présenté dans la Figure 3.10, indique que 728 pannes sont injectées et qu'uniquement 470 sont couvertes. Cela est dû au fait qu'une faible partie de tous les vecteurs d'entrées possibles est utilisée. La Signature générée à partir de ces rapports possèdera donc une fréquence d'erreur nulle élevée.

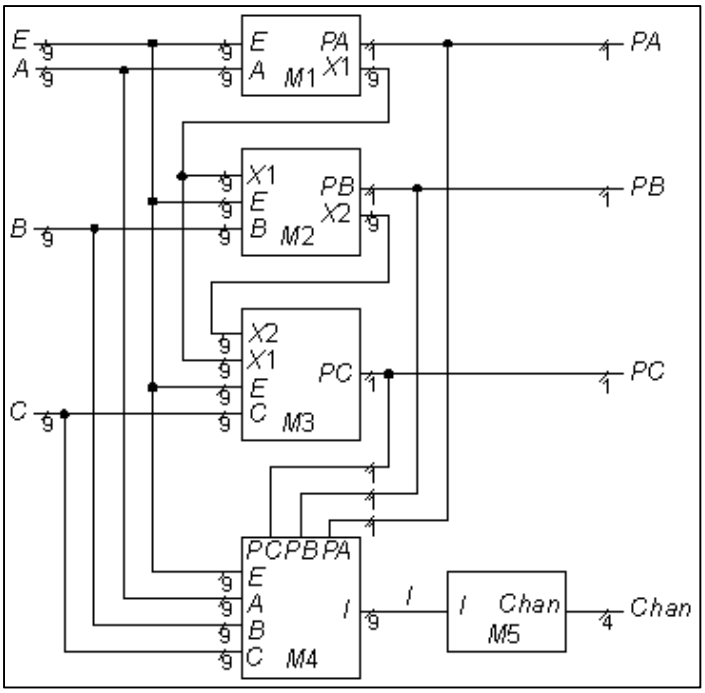

Figure 3.9 Schéma du circuit ISCAS85 C432

```
SUMMARY:
  Library: AMS C35
  Netlist filename: c432.v
  Input patterns filename: input_spec.ts
  Simulation output filename: output.txt
  Fault model: Stuck-At
  Write fault statistics: no

Reading netlist....c432.v done.
Reading input patterns....done.

summary..
 - # PI                          : 36
 - # PO                          : 7
 - # non_scan FFs                : 0
 - # scan FFs                    : 0
 - # memories                    : 0
 - # faults                      : 728

Running simulation of 65536 patterns.. done!
Allocating memory for fault list (728 Bytes)...done.

Starting fault simulation for sequential circuits...100% (728 faults)...done.
Fault coverage=470 on 728 (64.5604)
```

Figure 3.10 Résumé de l'injection par simulation avec LIFTING pour le circuit ISCAS85 c432 avec la liste complète de vecteurs d'entrées pour les canaux partiels

Pour cet exemple, cinq différents ensembles de vecteurs d'entrée ont été utilisés afin de pouvoir comparer les résultats obtenus avec les précédents. La principale différence avec les études de cas précédentes étant la taille élevée des entrées primaires du circuit. Même si une sous-partie faible de la totalité des vecteurs d'entrée est utilisée, la distance entre les vecteurs est plus élevée. Or le simulateur LIFTING réutilise entre chaque simulation les résultats similaires afin de réduire le temps de simulation. On peut donc s'attendre à un temps de simulation nettement supérieur aux résultats obtenus précédents. La taille des sorties primaires et le nombre de pannes étaient inférieurs à celles du multiplicateur, la taille des rapports générés doit être inférieure.

Le Tableau 3.4 résume les caractéristiques de la simulation de l'injection de pannes pour le circuit c432. Les résultats présentés sont ceux auxquels nous nous attendions. Le temps de simulation est significativement augmenté par rapport aux deux expérimentations précédentes, et les rapports de sortie ont une taille inférieure à ceux générés pour le multiplicateur, mais supérieure à l'additionneur. Encore une fois, on note que le nombre de vecteurs d'entrées utilisés est proportionnel aux trois paramètres étudiés.

Tableau 3.4 Performances de la simulation d'injection de pannes pour le circuit ISAC85 c432 en fonction du nombre de vecteurs d'entrée

Nombre de vecteurs d'entrée utilisés	**Taille du fichier (Ko)**	**Temps d'injection (min)**	**Taille de la sortie (Mo)**
65 536	2500	60	364
50 000	1900	45	275
40 000	1520	36	220
30 000	1140	27	165
20 000	760	18	110

Il est fortement déconseillé d'utiliser la liste complète de vecteurs d'entrées pour les 4 canaux de 9 bits. La liste de vecteurs d'entrées possèdera plus de 68 milliards de vecteurs. La limite du simulateur est largement dépassée, le fichier d'entrées sera bien trop gros (2,621 Go), et le temps de simulation sera incommensurable.

3.4 Bilan de la génération des rapports d'injection de pannes

Nous avons présenté le fonctionnement détaillé de l'outil LIFTING et expliqué les paramètres spécifiés afin de répondre aux objectifs de cette étape. Pour résumer, des pannes de type collages à 1/0 sont injectées sur tous les nœuds du circuit et générons un rapport par panne injectée contenant le résultat de la simulation pour chacun des vecteurs spécifiés dans la liste de vecteur d'entrées donnée à l'outil. La limitation principale de l'outil est son temps de simulation. Il s'agit d'un outil bas niveau, qui utilise une liste d'interconnexion d'un circuit pouvant être complexe. Le temps d'injection peut rapidement exploser en fonction de la taille des entrées primaires du circuit et du nombre de portes combinatoires qui le compose. Afin d'optimiser le temps d'injection, plusieurs simulations avec un nombre de vecteurs d'entrée différents ont été effectuées, pour pouvoir estimer plus tard l'effet sur la précision de la Signature. Dans les expérimentations effectuées nous détaillerons les bénéfices de la diminution du nombre de vecteurs sur le temps de simulation et la taille des rapports d'injection. Une seconde limitation de cet outil est le modèle de panne utilisé. Effectivement, le modèle de collage à 1/0 est une approximation de l'effet d'un SEU sur un FPGA. Le circuit n'est pas implémenté sur un FPGA, il est donc impossible d'avoir un modèle de pannes totalement réaliste.

Une approche secondaire a aussi été proposée. Elle permet d'injecter des pannes similaires à des SEU sur une partie d'un design implémenté sur un FPGA. Cette méthode est rapide et précise. Cependant elle demande plus de temps de préparation ; il faut réaliser le design de la Figure 3.4 et l'implémenter sur le FPGA. Suite à cette méthode, il est possible de définir un nouveau modèle de pannes applicable à haut niveau grâce à un nouveau type de Signatures. Il est fort probable que le modèle de pannes soit plus complexe que celui utilisé pour l'injection de pannes par simulation. Nous ne présenterons pas d'étude de cas utilisant l'injection de pannes par émulation, car cela dépasse le cadre de ce mémoire. Des travaux futurs pourront porter sur la génération de Signature à partir de cette méthode d'injection.

Ensuite, trois différentes études de cas ont été présentées. Elles possèdent chacune un nombre d'entrées primaires, de sorties primaires, et de portes différent. Pour chaque circuit, une injection de pannes a été effectuée avec la liste complète de pannes pour cinq tailles de listes de vecteurs d'entrée différentes. Le nombre de vecteurs d'entrée peut devenir un handicap si la taille de l'entrée primaire est trop élevée. Le temps de simulation est quant à lui influencé par le nombre de vecteurs d'entrée, la distance entre ces vecteurs, ainsi que le nombre de pannes différentes à injecter. La taille des rapports de sortie dépend du nombre et de la taille des sorties primaires ainsi que du nombre de pannes différentes injectées. Ces trois paramètres varient linéairement en fonction du nombre de vecteurs d'entrées. Diminuer le nombre de vecteurs utilisés lors de l'injection permet donc de réduire le temps de simulation et la taille des vecteurs d'entrée ainsi que des rapports de sortie. Cependant, cela engendre une perte d'information. Cela peut avoir pour conséquence de diminuer la précision des Signatures générées à partir de ces rapports de simulation.

CHAPITRE 4

GÉNÉRATION DE SIGNATURES

Pour permettre de propager à plus haut niveau d'abstraction les résultats de l'injection de pannes par simulation, un nouveau concept de Signature du comportement fautif d'un circuit est proposé. Ces Signatures compressent la distribution de probabilité du comportement fautif d'un circuit. La définition de Signature utilisée ici diffère fortement de la définition classique utilisée dans le demain du test post fabrication effectué afin de valider qu'un circuit soit fiable (Agrawal, 2000). La différence principale entre les deux définitions est le niveau d'abstraction auquel les Signatures sont utilisées.

Ce chapitre présente le concept de Signature ainsi que l'analyse de l'impact du filtrage et du nombre de vecteurs d'entrées utilisés lors de l'injection de panne sur la précision. Les résultats générés précédemment sont réutilisés dans les trois études de cas effectuées. L'impact du type de Signature en fonction du type de circuit est aussi présenté.

4.1 Le concept de Signature

Les Signatures sont générées à partir de l'information collectée dans les rapports générés par un outil d'injection de pannes. Leur but est de compresser cette information afin de la réutiliser à plus haut niveau d'abstraction. Lors de la définition du format d'une Signature, les objectifs étaient qu'il s'agisse d'un format compact qui puisse être lu et utilisé aisément à plus haut niveau d'abstraction. On peut noter que, plus les rapports d'injection de pannes sont détaillés, plus la Signature pourra être précise. Plus le niveau d'abstraction est bas, plus les nombres de détails et d'informations sont élevés, ce qui généralement implique plus de précision. C'est pourquoi, dans ce projet, les Signatures sont générées à partir de rapports d'injection de pannes par simulation au niveau porte logique avec l'outil LIFTING.

Afin de réduire la complexité et la taille des Signatures au maximum, le nombre de paramètres utilisés dans les Signatures a été minimisé. Chaque Signature contient l'erreur observée en sortie, ainsi que la probabilité d'occurrence correspondante. Nous avons choisi de nous focaliser uniquement sur la sortie et non sur l'entrée. Certes, de l'information est perdue, mais la taille de la Signature en est significativement réduite. L'erreur sur la sortie primaire est calculée avec différentes formules. De cette manière, la Signature n'est pas reliée à une sortie ou à une entrée, elle est indépendante et contient uniquement les données concernant les probabilités d'erreurs sur la sortie primaire. Plus il y a de paramètres, plus l'information est précise, mais cela a pour conséquence d'augmenter de manière considérable la taille de la Signature. Cela rend ainsi l'application à haut niveau plus difficile, et peut aussi potentiellement ralentir fortement la simulation, ce qui n'est pas souhaitable.

Afin de générer une Signature, chaque rapport d'injection de LIFTING correspondant à une panne est lu, les erreurs sur la sortie primaire du circuit sont calculées pour toutes les entrées, le nombre d'occurrences de chaque erreur est compté afin de retranscrire leur fréquence d'apparition. Le nombre d'occurrences étant proportionnel à la probabilité de l'erreur; il n'est donc pas nécessaire de diviser par le nombre total d'occurrences, comme présenté au Tableau 4.1.Une Signature consiste donc en un tableau d'erreurs et de leurs nombres d'occurrences correspondantes généralisées pour toutes les entrées.

Tableau 4.1 Exemple de Signature du comportement fautif d'un circuit

Erreur	**Nombre d'occurrences**
…	…
Ei	4514
Ei+1	132
…	
En	7412

Une Signature aura un format différent en fonction du niveau d'abstraction auquel l'utilisateur souhaite la lire, l'erreur ne sera pas calculée de la même façon, et l'information sera plus ou moins compressée. L'équation utilisée pour calculer l'erreur peut changer, et l'information sera alors compressée de différente manière.

Pour ce projet, deux types de Signatures sont présentés. Une première Signature arithmétique, et une seconde Signature logique seront détaillées dans la suite de ce mémoire.

4.1.1 Signatures arithmétiques

La première formule que proposée permet de calculer l'erreur arithmétiquement grâce à une soustraction. Ce choix permet d'obtenir une erreur décimale caractérisant la différence entre la sortie erronée et la sortie parfaite. Pour générer une Signature arithmétique, l'équation suivante est utilisée :

$$E_{A\,(10)} = Sortie_{erronée\,(10)} - Sortie_{parfaite\,(10)} \tag{4.1}$$

L'erreur est calculée en soustrayant la sortie parfaite à la sortie erronée, l'information est conservée sous format décimal. Ce premier format ainsi défini est par exemple adapté pour l'outil MathWorks Simulink. Cet outil, très utilisé par un grand nombre de professionnels et de chercheurs, utilise pour ses simulations, des données sous formes décimales. À ce haut niveau d'abstraction, l'information sur les bits n'existe pas, la Signature devra donc être décimale afin de s'adapter à Simulink.

Le fait d'effectuer le calcul (4.1) compresse l'information. Initialement, les rapports générés suite à l'injection de panne contiennent l'information sur l'effet de chaque panne sur chaque entrée spécifiée dans la liste de vecteurs d'entrée. Si cette liste est complète, on peut quantifier le nombre de données différentes contenues dans ce rapport comme étant le produit du nombre de vecteurs d'entrée par le nombre de pannes injectés. Soit $N = 2^A \times P$ avec A le nombre de bits composant les entrées primaires du circuit et P le nombre de pannes

différentes injectées. Une fois la formule et la généralisation appliquées, le nombre maximum de données différentes peut être calculé comme étant $M = 2(2^B - 1)$, avec B le nombre de bits composants les sorties primaires du circuit. Il est important de noter que cette valeur est une surestimation conséquente du nombre de pannes différentes. De nombreuses pannes produiront une erreur E_A similaire et certaines pannes ne causeront aucune erreur visible. Le facteur de vulnérabilité architectural (AVF) diminue considérablement le nombre maximal d'erreurs différentes observables en sortie. Le nombre maximum d'informations contenues dans une Signature arithmétique est donc $M' = AVF \times 2(2^B - 1)$. La compression peut se calculer par le rapport $R = \frac{N}{M'} = \frac{2^A \times P}{AVF \times 2(2^B - 1)} = \frac{2^A}{2^B - 1} \times \frac{P}{2 \times AVF}$.

- Si le nombre B de bits composants les sorties primaires du circuit est faible, le dénominateur sera faible et le numérateur sera supérieur, même avec A faible, car multiplié par P. Le rapport R sera donc élevé, ce qui signifie une forte compression de l'information ;
- Si B est élevé, il faut différencier deux cas. Soit B inférieur à A, ce qui est le cas pour la plupart des circuits, alors le rapport R sera grand et l'information fortement compressée. Le cas ou B est supérieur à A est beaucoup plus rare pour un circuit et la grandeur de R est plus difficilement estimable. Si B est légèrement supérieur à A, le facteur de vulnérabilité architecturale (AVF) étant inférieur à 1, et le nombre de lieux d'injection font que le rapport R sera tout de même conséquent et qu'il y aura compression de l'information. Dans le cas extrême, où B est très supérieur à A, il est possible que R soit faible et que la compression soit mauvaise. Toutefois, ce cas est très rare.

Cette compression engendre une perte d'information. L'erreur est indépendante de l'entrée, et est caractérisée par une différence. Il est impossible de connaitre la position, et donc le poids des bits erronés. La notion de changement de bit simple ou multiple n'est pas apparente. À cause de cela, il est très important qu'une Signature arithmétique soit générée pour un circuit dont tous les bits des sorties primaires sont liés logiquement. En effet, une erreur simple peut avoir la même valeur arithmétique qu'une erreur multiple sur les bits de sortie. Ceci se justifie par le fait que chaque puissance de deux peux se décomposer en une différence de deux puissances de 2, car $2^N = 2^{N+1} - 2^N$. Donc chaque puissance de 2 peut

se décomposer en une infinité de différences de puissances de 2. Ceci a pour conséquence que l'information sur la corrélation entre les positions des bits affectés en sortie est perdue. Afin d'obtenir cette information, il est nécessaire de proposer un nouveau format contenant l'information sur la position des bits erronés. Pour cela, un second type de Signature, appelé Signature logique, est proposé.

4.1.2 Signatures logiques

Le nouveau format de Signature logique est proposé. Ce dernier est complémentaire à la Signature arithmétique, et permet de conserver l'information sur la position des bits erronés. Cette fois-ci, l'équation suivante est utilisée afin de calculer l'erreur :

$$E_{L\,(2)} = Sortie_{erronée\,(2)} \oplus Sortie_{parfaite\,(2)} \tag{4.2}$$

Ce calcul permet de conserver l'information sur quels bits sont modifiés ainsi que la corrélation entre les différents bits de sortie. Cependant, l'information sur la transition effectuée est perdue, on s'intéresse uniquement au changement d'état.

Une Signature logique est plus générique qu'une Signature arithmétique, car elle est applicable sur un plus grand nombre d'outils et de plateformes. Ce type de Signature s'applique à un niveau d'abstraction inférieur, car cette fois-ci l'information est binaire.

Il est possible de quantifier la compression réalisée en utilisant la formule (4.2). Comme précédemment, si la liste des vecteurs d'entrée est complète, on peut quantifier le nombre de données différentes contenues dans le rapport d'injection comme étant le produit du nombre de vecteurs d'entrée par le nombre de pannes injectés. Soit $N = 2^A \times P$ avec A le nombre de bits composant les entrées primaires du circuit et P le nombre de pannes différentes injectées. Une fois la formule et la généralisation appliquées, le nombre maximum de données différentes peut être calculé comme étant $M = 2^B$, avec B le nombre de bits composants les sorties primaires du circuit. En appliquant le facteur de vulnérabilité architectural, l'équation

devient $M = AVF \times 2^B$. De plus, de nombreuses pannes produiront une erreur E_L similaire, M' est donc une surestimation. La compression peut se calculer par le rapport $R = \frac{N}{M'} = \frac{2^A \times P}{AVF \times 2^B} = 2^{A-B} \times \frac{P}{AVF}$. Le numérateur étant similaire au précédent et le dénominateur inférieur, les conclusions tirées dans la partie 4.1.1 s'appliquent toujours. Le taux de compression sera environ deux fois supérieur à celui d'une Signature arithmétique. Si le nombre B de bits composants les sorties primaires du circuit est faible, le rapport R sera donc élevé, ce qui signifie une forte compression de l'information. Si B est élevé R sera aussi élevé à moins que B soit très supérieur à A, ce qui est un cas critique très rare et négligeable.

Il est évident que cette compression engendre une perte d'information. Nous rappelons que l'erreur est indépendante de l'entrée. Il est aussi important de noter que l'erreur est caractérisée par le changement d'état d'un ou plusieurs bits et non par une transition 1/0 ou 0/1.

4.1.3 Filtrage des Signatures

En généralisant la Signature pour toutes les entrées, de nombreuses erreurs calculées auront une probabilité d'occurrence très faible comparativement aux erreurs les plus récurrentes. Une Signature de grande taille sera plus difficilement lisible à haut niveau d'abstraction et le temps de simulation sera plus conséquent. Afin de pallier à ce problème et réduire la taille des Signatures, différentes méthodes de filtrage seront utilisées. La première méthode consiste à définir un seuil minimal de probabilité. Si une erreur est en dessous de ce seuil, elle n'apparaitra pas dans la Signature. Ceci permet à l'utilisateur, en fonction de la précision qu'il souhaite, d'adapter la taille de la Signature. Il est aussi possible de définir un nombre de bits modifiés simultanément maximum. L'utilisateur pourra alors simplifier sa Signature et par exemple ne considérer que les erreurs simples ou doubles sur la sortie. Cette deuxième méthode de filtrage est basée sur l'assomption que les erreurs les plus simples, n'affectant qu'un nombre limité de bits sur la sortie, sont les plus probables.

Nous étudierons plus tard l'effet du filtrage sur la précision des Signatures en comparant les différentes probabilités d'erreurs à la Signature non filtrée pour différents taux de filtrage. Supprimer les erreurs les moins probables permet de garder uniquement les erreurs principales, communes à la plupart des entrées. L'effet du filtrage peut avoir un effet positif, en limitant le nombre de pannes impossible pour certaines entrées particulières et en se focalisant sur les erreurs principales observées pour la majorité des entrées primaires.

4.2 Présentation de l'outil de génération automatisée de Signatures (OGAS)

L'outil OGAS a été créé. Il permet de traiter les résultats de simulation obtenus suite à l'injection de pannes avec l'outil LIFTING et de les transformer en Signature. Une capture d'écran de l'interface est présentée dans la Figure 4.1. OGAS lit les rapports générés par LIFTING, et génère une Signature en fonction des spécifications de l'utilisateur. Les paramètres sont le type de Signature (arithmétique ou logique), ainsi que la méthode de filtrage. L'utilisateur peut choisir entre donner le nombre de « bit flips » maximum mesuré pour la sortie erronée, ou la probabilité minimum d'une erreur pour qu'elle soit prise en compte.

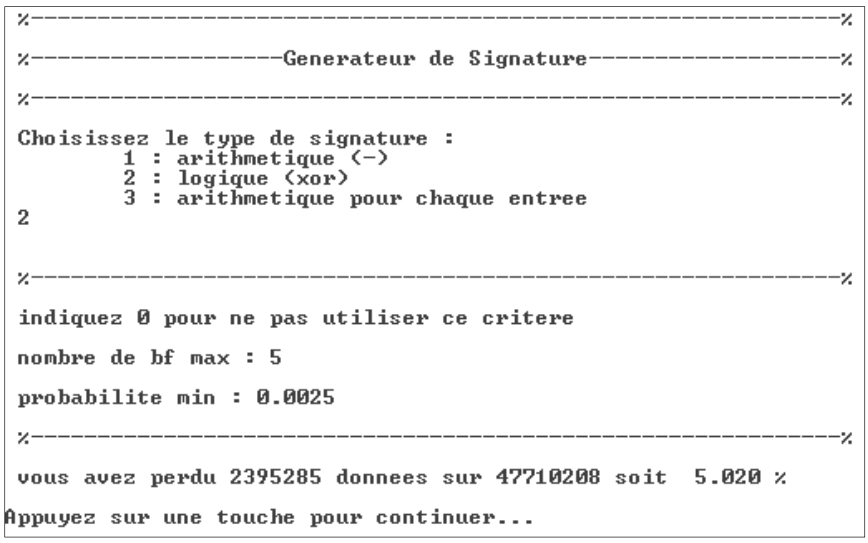

Figure 4.1 Capture d'écran de l'interface de l'outil OGAS permettant la génération de Signatures

Le fonctionnement de l'outil réalisé est le suivant :

1. L'utilisateur spécifie les paramètres de la Signature qu'il souhaite générer, à savoir le type de Signature, ainsi que les paramètres de filtrage ;
2. Le rapport de simulation parfait est lu et écrit dans la mémoire sous forme d'un tableau. Un tableau vierge est aussi créé afin de stocker la Signature ;
3. Les rapports d'injection sont lus un par un, chaque ligne est comparée avec la formule correspondant au type de Signature choisie. Une fois l'erreur calculée, la ligne correspondante de la Signature est incrémentée afin de compter le nombre d'occurrences de cette erreur ;
4. Lorsque tous les rapports d'injection ont été parcourus, le tableau de Signature est alors écrit sous forme d'un fichier contenant deux colonnes correspondant à l'erreur et à son nombre d'occurrences. Les occurrences nulles ou inférieures au seuil de filtrage défini par l'utilisateur sont omises de ce fichier, car ces données n'apportent rien et ne feraient qu'augmenter inutilement la taille de la Signature ;
5. L'exécution du programme se termine en indiquant le pourcentage d'erreurs qui ont été supprimées à cause du filtrage comme observé dans la Figure 4.1.

La précision des Signatures générées sera estimée en calculant l'erreur relative entre la probabilité de la Signature de référence et imparfaite. L'équation utilisée est la suivante :

$$E_{relative} = \frac{\left|P_{référence} - P_{imparfaite}\right|}{P_{référence}} \tag{4.3}$$

La Signature parfaite est générée à partir des résultats de simulation utilisant tous les vecteurs d'entrées possibles sans qu'aucun filtrage ne soit appliqué. Il s'agit du plus haut niveau de précision atteignable avec ce format.

Dans le cadre des expérimentations, des Signatures logiques et arithmétiques sont générées à partir de différents résultats d'injection de pannes. Puis elles sont filtrées en utilisant plusieurs seuils de probabilité minimum. Le filtrage en fonction du nombre de bit flips ne

sera pas appliqué, car avant de l'utiliser il faut valider le fait que les erreurs avec peu de bit flip sont plus probables.

Pour chaque circuit, l'erreur relative maximale observée sera comparée dans chaque Signature en fonction du nombre de vecteurs utilisés lors de l'injection de pannes et du taux de filtrage choisi. Le but de cette étape étant d'estimer la précision ainsi obtenue pour optimiser le nombre de vecteurs utilisés lors de l'injection de pannes, ainsi que le taux de filtrage qui permet de diminuer la taille de la Signature.

4.3 Études de cas

Les résultats obtenus dans la première étape ont été réutilisés afin de générer des Signatures pour les circuits suivant : un additionneur 8 bits, un multiplicateur 8 bits et le circuit ISCAS85 c432. Nous distinguons deux types de circuits : arithmétiques et logiques. Ces circuits possèdent aussi une complexité différente, ce qui permet de valider la performance de l'outil OGAS pour des cas différents.

4.3.1 Génération de la Signature d'un additionneur 8 bits

En reprenant les résultats obtenus suite à l'injection de pannes dans un additionneur 8 bits sans retenue avec l'outil LIFTING plusieurs types de Signatures ont été générés. Comme expliqué dans la méthodologie, l'erreur obtenue sur la Signature en fonction du nombre de vecteurs utilisés pour générer la liste de pannes ainsi que du filtrage de la Signature a été quantifiée.

Pour cela, les résultats de campagnes d'injection utilisant 65 536, 50 000, 40 000, 30 000, et 20 000 vecteurs d'entrées ont été analysés. Pour chacun de ces quatre derniers cas, nous avons utilisé trois listes de vecteurs d'entrées aléatoirement générées. Chaque liste de pannes est alors compressée sous forme de Signature, à la fois logique et arithmétique, avec différents taux de filtrage. Pour un même taux de filtrage et un même nombre de vecteurs

d'entrée utilisés, la moyenne entre les résultats obtenus est effectuée. L'erreur relative est calculée par rapport à la probabilité parfaite obtenue si la liste complète de tous les vecteurs d'entrée est utilisée et qu'aucun filtrage n'est appliqué. Une partie du résultat obtenu est détaillé dans le Tableau 4.2. La probabilité d'occurrence est utilisée ici au lieu du nombre d'occurrences afin de pouvoir comparer les différentes Signatures qui ne possèdent pas le même nombre total d'occurrences. De cette manière les résultats sont plus clairs et plus simplement analysables.

Au Tableau 4.2, les Signatures sont filtrées avec un seuil de probabilité de 1,5 % de manière à ce que les erreurs dont les probabilités d'occurrences sont inférieures à ce seuil n'apparaissent pas. Ce résultat étant intermédiaire, il ne sera pas présenté dans son intégralité dans le corps de ce mémoire. Pour les résultats complets, veuillez vous référer à l'ANNEXE II. Les erreurs les plus probables sont les erreurs qui affectent un nombre faible de bits sur la sortie qui se présentent sous la forme 2^n pour les Signatures arithmétiques.

L'erreur relative pour la Signature utilisant la liste de panne générée à partir de la liste complète de vecteur est constante. Ceci s'explique par le fait que les données sont filtrées, chaque erreur possède le même nombre d'occurrences, uniquement le nombre d'occurrences totales est modifié. En dehors de ce cas-ci, l'erreur fluctue; on remarque ainsi que, la plupart du temps, plus la probabilité de l'erreur est faible plus l'erreur relative calculée est élevée, ce qui était prévisible. Il est très difficile d'analyser ces résultats et de les comparer. L'erreur relative maximale sera analysée en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes, les résultats obtenus sont exposés à la Figure 4.2 et à la Figure 4.3.

Tableau 4.2 Résultats partiels de génération de Signatures logiques et de calcul d'erreurs relatives pour un additionneur 8 bits

	Parfait	**Filtré (65 536 vect.)**		**Filtré (50 000 vect.)**		**Filtré (40 000 vect.)**	
Erreur	P(erreur)	P(erreur)	Erreur r.	P(erreur)	Erreur r.	P(erreur)	Erreur r.
0	51,99 %	59,30 %	14,05 %	59,30 %	14,05 %	59,29 %	14,04 %
1	4,53 %	5,16 %	14,05 %	5,17 %	14,16 %	5,17 %	14,078 %
10	3,53 %	4,03 %	14,05 %	4,03 %	14,04 %	4,033%	14,19 %
11	1,54 %	1,76 %	14,05 %	1,76 %	14,18 %	1,75 %	13,95 %
100	3,62 %	4,13 %	14,05 %	4,13 %	14,14 %	4,12 %	13,81 %
1000	3,62 %	4,13 %	14,05 %	4,13 %	14,012 %	4,14 %	14,24 %
10000	3,62 %	4,13 %	14,05 %	4,13 %	13,96 %	4,13 %	14,15 %
100000	3,62 %	4,13 %	14,05 %	4,12 %	13,87 %	4,13 %	14,07 %
1000000	3,62 %	4,13 %	14,05 %	4,13 %	14,04 %	4,13 %	14,00 %
10000000	5,80 %	6,61 %	14,05 %	6,61 %	14,05 %	6,61 %	14,04 %
11000000	2,17 %	2,48 %	14,05 %	2,47 %	14,067 %	2,48 %	14,10 %

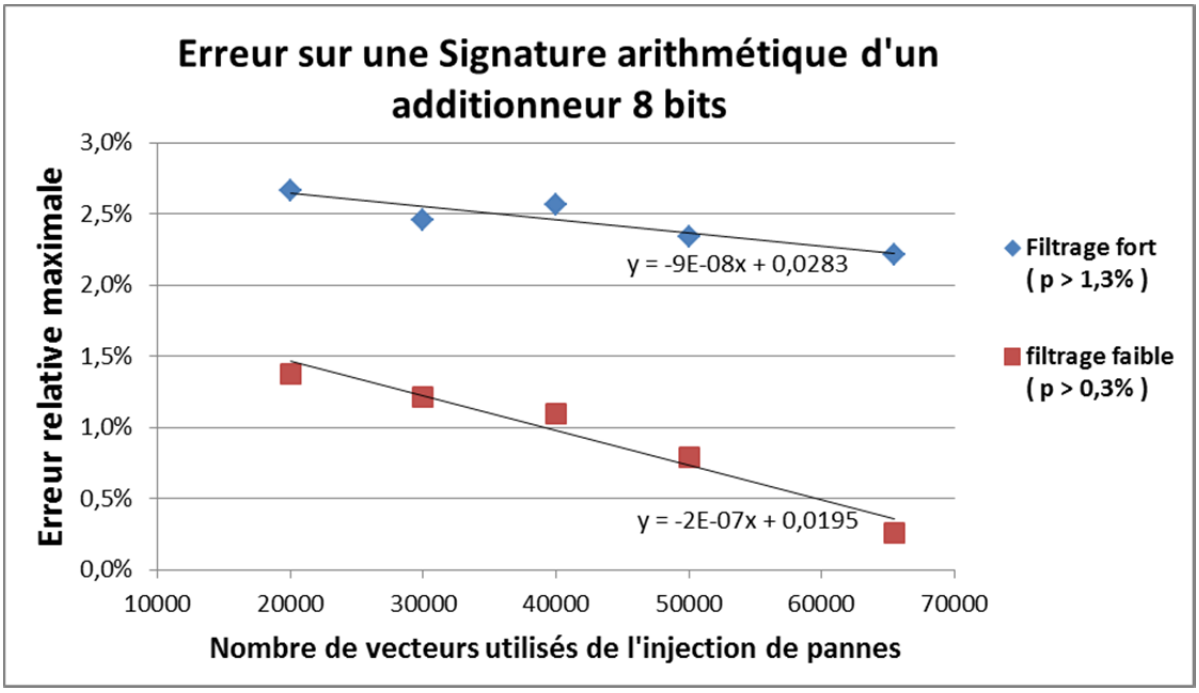

Figure 4.2 Erreur relative maximale en fonction du filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un additionneur 8 bits (Signatures arithmétiques)

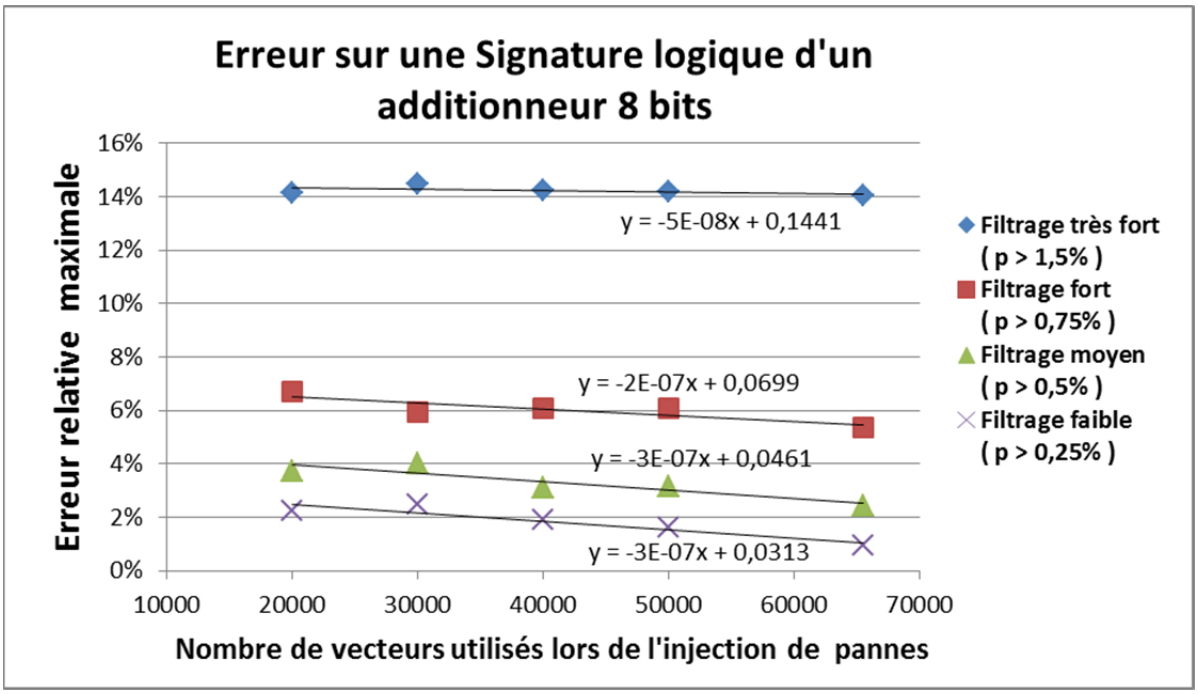

Figure 4.3 Erreur relative maximale en fonction du filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un additionneur 8 bits (Signatures logiques)

Afin de choisir les seuils de filtrage, la distribution de probabilités des erreurs a été étudiée. Cette distribution n'est pas uniformément répartie. En effet, les probabilités d'erreurs sont concentrées autour de certaines valeurs. Les différents seuils de filtrage seront donc fixés aux valeurs supérieures de ces concentrations de probabilités.

Pour les Signatures arithmétiques, deux taux de filtrage plus ou moins « forts » ont été choisis, filtrant chacun les probabilités supérieures à 1.3 % et 0.3 %. Les tailles respectives des Signatures sont 17 et 23 lignes. Seulement deux taux de filtrage ont été choisis car les erreurs relatives obtenues sont très faibles même avec un seuil de filtrage bas.

La Figure 4.2 représente l'erreur relative maximale calculée sur la Signature arithmétique en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un additionneur 8 bits. Dans le cas où tous les vecteurs d'entrées sont utilisés lors de l'injection de pannes, on obtient des erreurs relatives de 2,21 % et 0,26 %, ce qui est très faible. Ici, l'erreur relative varie encore plus faiblement lorsque l'on diminue le nombre de vecteurs d'entrées utilisés lors de l'injection de pannes. On mesure des variations pour le taux de filtrage fort et faible valant respectivement 0,45 % et 1,1 %, ce qui est très faible. Le fait que la variation soit plus élevée pour le taux de filtrage faible est surement dû au fait que plus le taux de filtrage est faible plus le nombre de probabilités faibles augmente. Ces probabilités faibles sont plus susceptibles de générer des erreurs relatives élevées.

Pour les Signatures logiques, quatre taux de filtrage plus ou moins « forts » ont été utilisés, filtrant chacun les probabilités supérieures à 1,5 %, 0,75 %, 0,5 %, et 0,25 %. Les tailles respectives des Signatures sont 11, 18, 23 et 28 lignes. Pour les taux de filtrage plus faibles. Dans certains cas, il a fallu supprimer une valeur extrême causée par une probabilité faible et qui engendrait une erreur relative très supérieure au reste et qui donc n'était pas représentative.

La Figure 4.3 représente l'erreur relative maximale calculée sur la Signature logique en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes

dans un additionneur 8 bits. En l'analysant, on remarque que l'erreur relative maximale dépend principalement du taux de filtrage choisi et varie faiblement en fonction du nombre de vecteurs utilisés lors de l'injection de pannes. Comme la logique le suggère, plus le taux de filtrage est faible, plus l'erreur relative diminue. Lorsque l'on utilise tous les vecteurs possibles en entrées lors de l'injection, les différents taux de filtrage choisis permettent d'obtenir une Signature logique dont l'erreur relative maximale se situe entre 14 % et 0,9 %. Pour les trois taux de filtrage les plus faibles, nous observons une variation de 1,3 % de l'erreur relative entre les valeurs obtenues avec 65 536 et 20 000 vecteurs d'entrée. Pour le filtrage très fort la variation est beaucoup plus faible et n'atteint que 0,1 %.

4.3.2 Génération de la Signature d'un multiplicateur 8 bits

En reprenant les résultats obtenus suite à l'injection de pannes dans un multiplicateur 8 bits sans retenue avec l'outil LIFTING, plusieurs types de Signatures ont été générés. Encore une fois, nous allons étudier l'erreur obtenue sur la Signature en fonction du nombre de vecteurs utilisés pour générer la liste de pannes ainsi que du filtrage de la Signature.

Pour ce cas aussi, les résultats de campagnes d'injections utilisant 65 536, 50 000, 40 000, 30 000, 20 000 vecteurs d'entrées ont été utilisés. Pour chacun de ces quatre derniers cas, trois listes de vecteurs d'entrées aléatoirement générées ont été utilisées.

Chaque liste de pannes est alors compressée sous forme de Signature, à la fois logique et arithmétique avec différents taux de filtrage. Pour un même taux de filtrage et un même nombre de vecteurs d'entrée utilisés, la moyenne entre les résultats obtenus est effectuée. Ensuite, l'erreur relative est calculée par rapport à la probabilité parfaite obtenue généré avec la liste complète de tous les vecteurs d'entrée sans que l'information soit filtrée. Une partie du résultat obtenu est détaillé dans le Tableau 4.3.

Tableau 4.3 Résultats partiels de génération de Signatures logiques et de calcul d'erreurs relatives pour un multiplicateur 8 bits

	Parfait	**Filtré(65 536 vect.)**		**Filtré(50 000 vect.)**		**Filtré(40 000 vect.)**	
Erreur	P(erreur)	P(erreur)	Erreur r.	P(erreur)	Erreur r.	P(erreur)	Erreur r.
0	60,18%	72,89%	21,12%	72,88%	21,10%	72,88%	21,09%
100	0,91%	1,10%	21,12%	1,10%	20,96%	1,10%	21,01%
1000	1,17%	1,42%	21,12%	1,41%	20,87%	1,42%	21,12%
1000	1,51%	1,830	21,12%	1,83%	21,21%	1,83%	21,24%
100000	1,723%	2,09%	21,12%	2,09%	21,00%	2,10%	21,30%
1000000	2,01%	2,44%	21,12%	2,44%	21,23%	2,44%	21,43%
10000000	2,30%	2,78%	21,12%	2,78%	20,90%	2,79%	21,48%
100000000	2,26%	2,73%	21,12%	2,74%	21,45%	2,75%	21,65%
110000000	1,03%	1,24%	21,12%	1,25%	21,98%	1,25%	21,39%
1000000000	2,05%	2,49%	21,12%	2,49%	21,25%	2,48%	20,75%
1100000000	1,00%	1,22%	21,12%	1,22%	21,19%	1,21%	20,70%
10000000000	1,78%	2,15%	21,12%	2,16%	21,15%	2,15%	20,69%
11000000000	0,91%	1,10%	21,12%	1,10%	20,82%	1,10%	21,27%
100000000000	1,52%	1,84%	21,12%	1,84%	21,31%	1,84%	20,81%
1000000000000	1,25%	1,51%	21,12%	1,50%	20,72%	1,51%	21,26%
10000000000000	0,95%	1,16%	21,12%	1,16%	21,28%	1,16%	21,31%

Dans ce tableau les Signatures sont filtrées avec un seuil de probabilité de 0,9 %, les erreurs moins probables que cela n'apparaissent pas. Ce résultat état intermédiaire, il ne sera pas présenté dans son intégralité dans le corps de ce mémoire. Pour les résultats complets, veuillez vous référer à l'ANNEXE III. Nous avons aussi observé l'erreur relative maximale en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes, les résultats obtenus sont exposés dans la Figure 4.4 et la Figure 4.5.

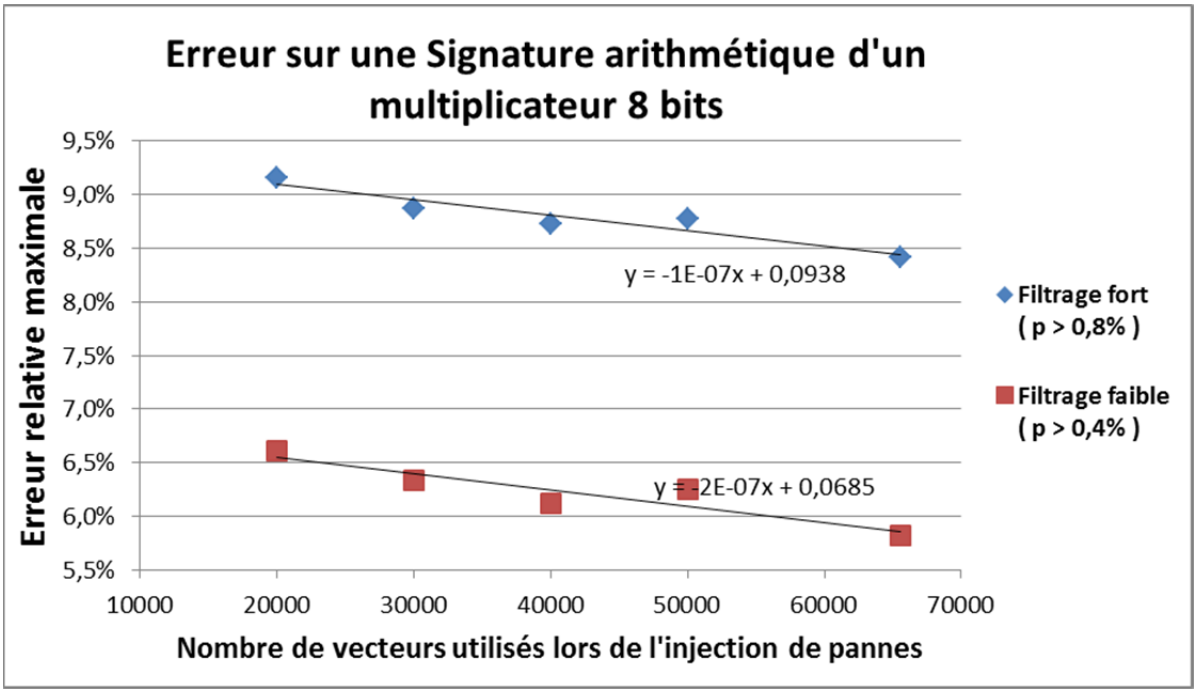

Figure 4.4 Erreur relative maximale en fonction du filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un multiplicateur 8 bits (Signatures arithmétiques)

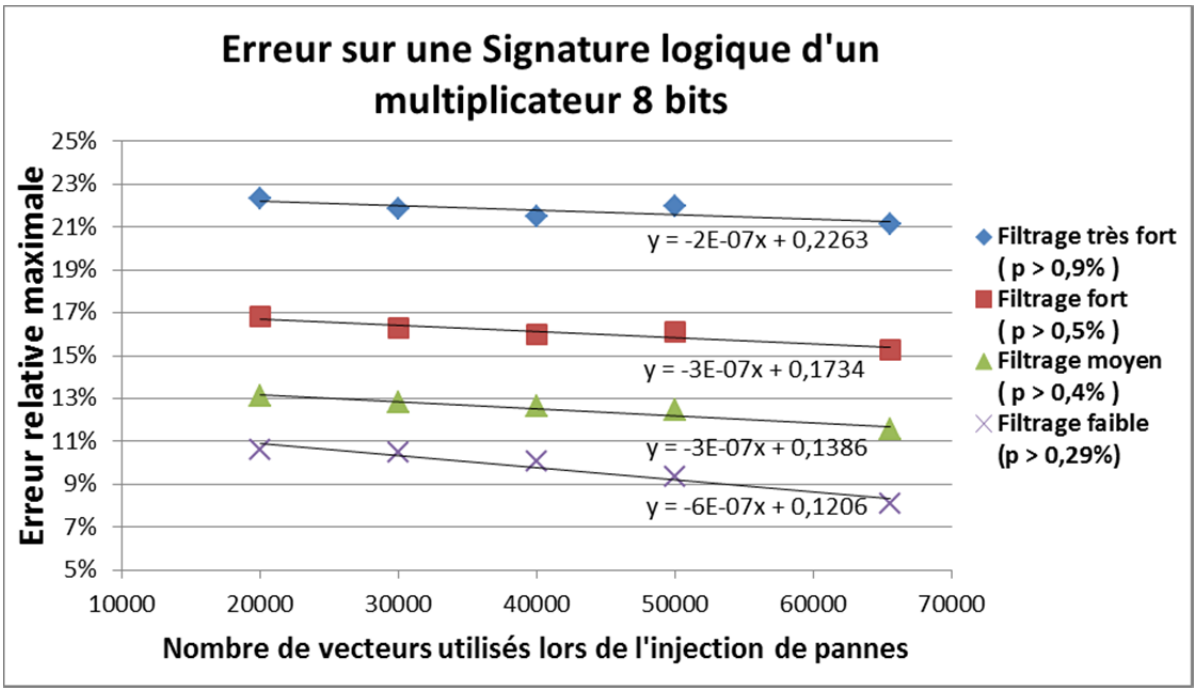

Figure 4.5 Erreur relative maximale en fonction du filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un multiplicateur 8 bits (Signatures logiques)

Pour les Signatures arithmétiques, deux taux de filtrages ont été choisis, avec des seuils de 0,8 % et 0,4 %. Les tailles des Signatures correspondantes étant de 17 et de 23 lignes. L'erreur relative maximum observée étant faible nous n'avons pas utilisé de taux de filtrage moins fort.

La Figure 4.4 représente l'erreur relative maximale calculée sur la Signature arithmétique en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un multiplicateur 8 bits. En utilisant la liste complète de vecteurs d'entrées, nous obtenons une erreur relative de 8,4 % et 5,8 %. En diminuant le nombre de vecteurs d'entrées utilisés lors de l'injection, un comportement linéaire est observé. L'erreur augmente respectivement pour ces deux courbes à 9,3 % et 7 % soit une faible augmentation proche de 1 %. L'erreur est majoritairement causée par le filtrage et non par la diminution des vecteurs d'entrée. Ces résultats sont similaires à l'expérience précédente avec l'additionneur 8 bits.

Dans le cas des Signatures logiques, les quatre différents seuils de filtrage ont été utilisés : 0,9 %, 0,5 %, 0,4 % et 0,29 %. Les Signatures correspondantes comportent respectivement 16, 22, 28, et 37 lignes.

La Figure 4.5 représente l'erreur relative maximale calculée sur la Signature logique en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes dans un multiplicateur 8 bits. Pour le taux de filtrage le plus faible et le plus fort, avec le nombre complet de vecteurs d'entrée utilisés lors de l'injection, les erreurs relatives maximales obtenues sont respectivement 8 % et 21 %. Encore une fois l'erreur relative maximale est proportionnelle au nombre de vecteurs d'entrée. Lorsque l'on diminue le nombre de vecteurs utilisés de 65 536 à 20 000 l'erreur relative maximale augmente d'un peu plus de 1 % ce qui est faible comparé à l'erreur initiale.

Pour ce circuit plus complexe, les Signatures générées restent précises malgré le filtrage et la diminution du nombre de vecteurs utilisés lors de l'injection. Cependant, on remarque que l'erreur relative maximale observée est supérieure à celle observée pour l'additionneur. Cela

se justifie par l'augmentation de la complexité du circuit et donc de la Signature. Un plus grand nombre de pannes sont injectées et la taille de la sortie primaire est supérieure. Le comportement fautif du circuit sera donc plus diversifié et sur une plage de valeurs plus grande.

Plus la liste de vecteurs utilisés lors de l'injection utilise un faible nombre de vecteurs, plus le nombre de combinaisons possibles est élevé. Afin d'étudier la variation de l'erreur en fonction du choix des vecteurs utilisés lors de l'injection, quinze listes différentes de 20 000 vecteurs ont étés générées aléatoirement et utilisés lors d'une campagne d'injection LIFTING avec le multiplicateur 8 bits. Grâce à OGAS, quinze Signatures ont été ensuite générées et filtrées avec un seuil de probabilité de 0,4%. La Figure 4.6 présente l'erreur relative maximale observée pour ces quinze Signatures différentes.

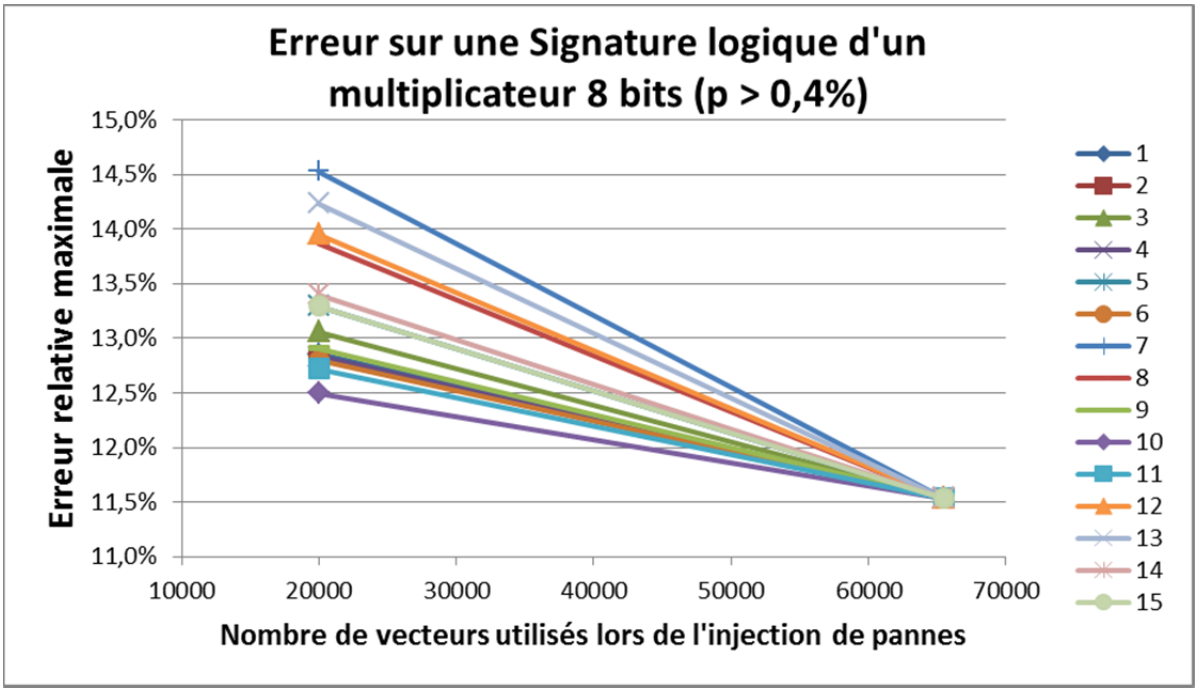

Figure 4.6 Erreur relative maximale pour quinze Signatures logiques différentes générées à partir de rapport d'injection LIFTING utilisant differentes listes de 20 000 vecteurs

L'erreur relative maximale varie entre 12,5% et 14,5%. Diminuer le nombre de vecteurs utilisés lors de l'injection engendrera donc une précision variable. Huit Signatures sur quinze

se situent entre 12,5% et 13,0%, on observe donc un comportement moyen qui correspond à la linéarité observée dans la Figure 4.5. On peut s'attendre à une plus grand dispersion lorsque le nombre de vecteurs est encore plus faible. Il semble aussi logique que la variance augmente et tende vers 100% lorsque le nombre de vecteurs diminue. C'est pourquoi il n'est pas recommandé d'utiliser une trop faible partie du nombre total de vecteurs possibles.

4.3.3 Génération de Signatures du circuit c432 ISCAS'85

À partir des résultats générés suite à l'injection de pannes dans le circuit c432 avec l'outil LIFTING plusieurs types de Signatures ont été générés. L'erreur obtenue sur la Signature sera étudiée en fonction du nombre de vecteurs utilisés pour générer la liste de pannes ainsi que du filtrage de la Signature.

Pour ce cas aussi, les résultats de campagnes d'injections utilisés utilisent 65 536, 50 000, 40 000, 30 000, 20 000 vecteurs d'entrées. Pour chacun de ces quatre derniers cas, trois listes de vecteurs d'entrées aléatoirement générées ont été utilisées.

Chaque liste de pannes est alors compressée sous forme de Signature, à la fois logique et arithmétique avec différents taux de filtrage. Pour un même taux de filtrage et un même nombre de vecteurs d'entrée utilisés, la moyenne entre les résultats obtenus est effectuée. Ensuite l'erreur relative est calculée par rapport à la probabilité parfaite obtenue en utilisant la liste complète de tous les vecteurs d'entrée sans appliquer de filtrage. Une partie du résultat obtenu est détaillé au Tableau 4.4.

Tableau 4.4 Résultats partiels de génération de Signatures logiques et de calcul d'erreurs relatives pour le circuit c432

	Parfait	**Filtré(65 536 vect.)**		**Filtré(50 000 vect.)**		**Filtré(40 000 vect.)**	
Erreur	P(erreur)	P(erreur)	Erreur r.	P(erreur)	Erreur r.	P(erreur)	Erreur r.
0	79,64 %	85,55 %	7,42 %	85,57 %	7,44 %	85,56 %	7,43 %
1	1,55 %	1,66 %	7,42 %	1,66 %	7,35 %	1,66 %	7,36 %
10	1,49 %	1,60 %	7,42 %	1,60 %	7,69 %	1,59 %	7,05 %
100	1,64 %	1,76 %	7,42 %	1,75 %	7,06 %	1,75 %	7,06 %
101	0,49 %	0,53 %	7,42 %	0,53 %	7,50 %	0,53 %	8,00 %
110	0,87 %	0,94 %	7,42 %	0,94 %	7,54 %	0,94 %	7,57 %
111	0,64 %	0,68 %	7,42 %	0,68 %	7,28 %	0,68 %	7,30 %
1000	1,11 %	1,19 %	7,42 %	1,18 %	7,20 %	1,19 %	7,50 %
1010	0,47 %	0,50 %	7,42 %	0,50 %	6,91 %	0,50 %	7,98 %
10000	1,25 %	1,34 %	7,42 %	1,34 %	6,99 %	1,35 %	7,79 %
11000	0,53 %	0,57 %	7,42 %	0,57 %	7,37 %	0,57 %	6,67 %
11001	0,57 %	0,61 %	7,42 %	0,62 %	7,86 %	0,60 %	5,75 %
11010	0,62 %	0,67 %	7,42 %	0,67 %	6,87 %	0,67 %	7,59 %
11011	0,66 %	0,71 %	7,42 %	0,71 %	7,82 %	0,71 %	7,88 %
100000	0,88 %	0,94 %	7,42 %	0,94 %	7,45 %	0,94 %	7,65 %
1000000	0,70 %	0,75 %	7,42 %	0,74 %	6,87 %	0,75 %	7,66 %

Le Tableau 4.4 correspond aux résultats partiels de génération de Signatures logiques pour le circuit c432. Les probabilités inférieures à 0,45 % sont filtrées. Ici n'apparaissent que les probabilités des erreurs présentées pour la Signature parfaite non filtrée ainsi que les Signatures filtrées et générées à partir de rapports d'injection utilisant 65 536, 50 000, et 40 000 vecteurs d'entrée. Les résultats complets sont présentés dans l'ANNEXE IV. L'erreur relative maximale a été observée en fonction du taux de filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes. Les résultats obtenus sont présentés dans la Figure 4.7 et la Figure 4.8.

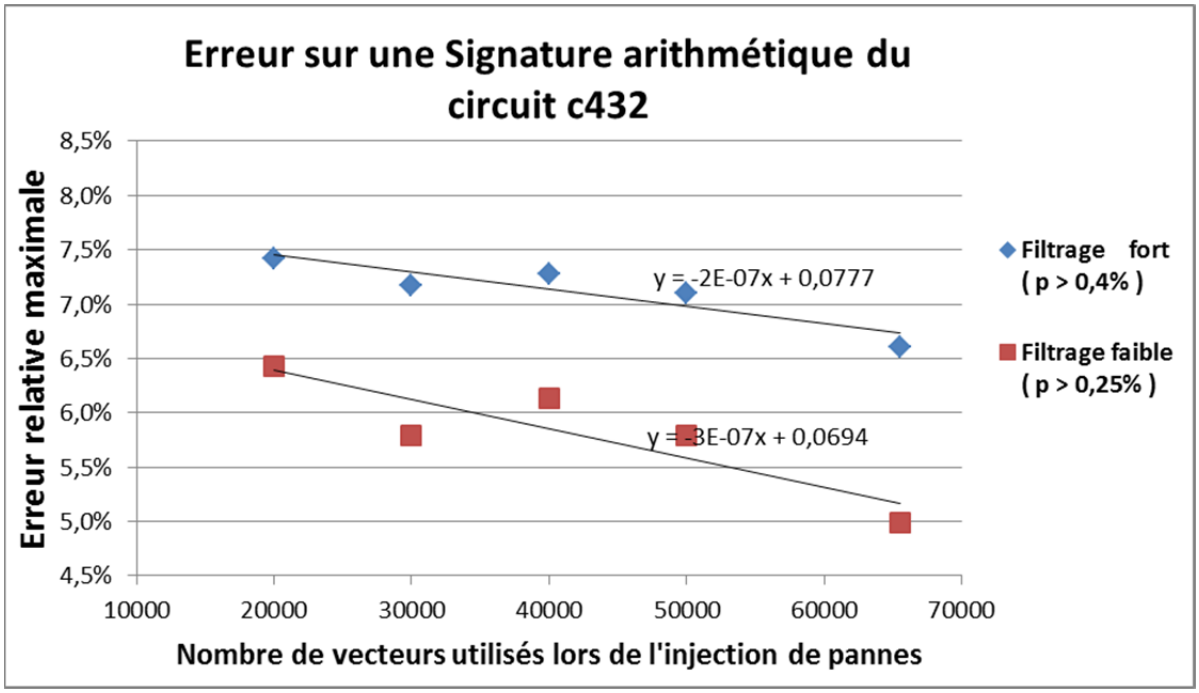

Figure 4.7 Erreur relative maximale en fonction du filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes pour le circuit c432 (Signatures arithmétiques)

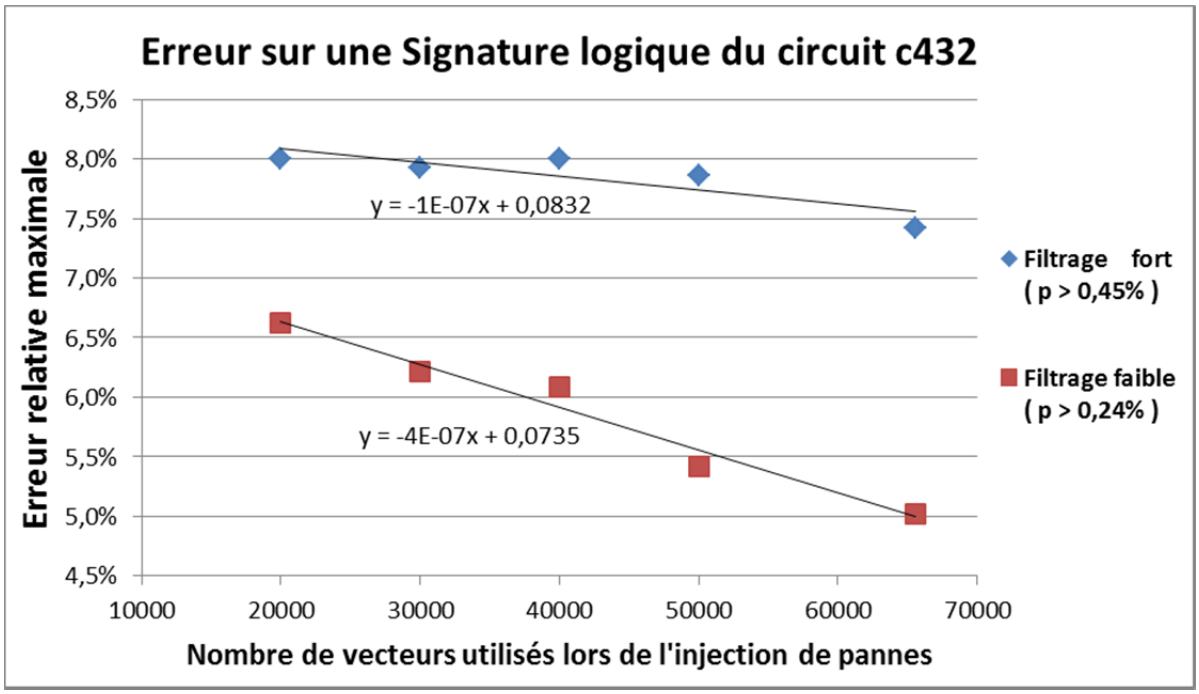

Figure 4.8 Erreur relative maximale en fonction du filtrage et du nombre de vecteurs utilisés lors de l'injection de pannes pour le circuit c432 (Signatures logiques)

Pour les Signatures arithmétiques, deux différents seuils de filtrages ont été choisis, à savoir 0,4 % et 0,25 %. Les tailles respectives des Signatures ainsi générées sont de 18 et 23 lignes. Ceci correspond aux tailles moyennes des Signatures générées précédemment, et l'erreur relative obtenue étant faible, aucun filtrage moins contraignant n'a été proposé.

La Figure 4.7, présente l'erreur relative maximale observée dans plusieurs Signatures arithmétiques, en fonction du taux de filtrage choisi ainsi que du nombre de vecteurs utilisés lors de l'injection de pannes dans un circuit c432. Pour les deux taux de filtrages choisis, les erreurs obtenues sont de 5 % et 6,6 % lorsque la liste complète de vecteurs d'entrée est utilisée. En réduisant le nombre de vecteurs utilisés à 20 000, ces probabilités augmentent respectivement de 1,4 % et 0,8 %. Encore une fois, il ressort que le filtrage influence plus fortement l'erreur relative calculée que le nombre de vecteurs utilisés. Le comportement de l'erreur relative maximale n'apparait pas aussi linéaire que ce qui a été observé précédemment. Cela peut se justifier par le fait que les probabilités d'erreurs sont plus faibles, donc influence plus fortement l'erreur relative observée.

Les taux de filtrages utilisés pour les Signatures logiques sont 0,45 % et 0,24 %. La taille des Signatures ainsi générées est respectivement 16 et 24 lignes. L'erreur relative maximale observée dans plusieurs Signatures logiques, en fonction du taux de filtrage et du nombre de vecteurs d'entrées utilisés avec l'outil LIFTING est présentée dans la Figure 4.8. Pour les deux seuils de filtrages choisis, lorsque la liste complète de vecteurs d'entrée est utilisée, l'erreur relative maximale est de 7,4 % et 5 %. En diminuant le nombre de vecteurs d'entrées à 20 000, ces deux erreurs augmentent respectivement de 0,6 % et 1,6 %, ce qui est faible comparé à l'erreur causée par le filtrage.

Ensuite, pour ce circuit logique, les résultats obtenus pour les Signatures logiques sont comparables, pour une taille similaire, à ceux obtenus avec des Signatures arithmétiques. Ce qui n'était pas le cas pour les deux exemples de circuits arithmétiques précédents. Pour l'additionneur, les Signatures arithmétiques et logiques composées de 23 lignes possèdent une erreur maximale relative est respectivement de 0,26 % et 2,4 %. Pour le multiplicateur,

les Signatures arithmétiques et logiques composées de 23 et 22 lignes possèdent une erreur maximale relative étant respectivement de 5,8 % et 15 %. Dans le cas du circuit c432, les Signatures arithmétiques et logiques de 23 et 24 lignes présentent toutes les deux une erreur relative maximale de 5 %. Par conséquent, une Signature logique peut être handicapante lorsque l'on utilise un circuit arithmétique, mais cela n'est pas le cas pour un circuit logique.

Pour ce circuit logique, il est préférable d'utiliser une Signature logique, car l'information est faite pour être interprétée de manière binaire et non décimale, même à haut niveau d'abstraction. Pour le circuit c432, il est important de connaitre la position des bits erronés, car il existe plusieurs bus de sortie. Il est impossible d'utiliser une Signature arithmétique. Si une Signature arithmétique est générée par bus alors la corrélation entre les différentes sorties ne sera pas apparente.

4.4 Bilan de la génération de Signatures

Nous avons présenté les résultats générés pour des Signatures logiques et arithmétiques pour trois circuits différents (deux circuits arithmétiques, et un logique). Des Signatures partielles ont été présentées, tandis que les résultats complets se situent en ANNEXE II, III et IV. Pour chaque circuit, différentes résultats de campagnes d'injection utilisant un nombre diffèrent de vecteurs d'entrée ont été analysés. À partir de ces résultats de simulation, des Signatures ont été générées avec l'outil OGAS en utilisant différents seuils de filtrage. Afin d'analyser les résultats obtenus, l'erreur relative maximale observée a été étudiée dans chaque Signature et comparée à la probabilité d'occurrence obtenue pour une Signature non filtrée générée à partir de rapports d'injection utilisant la liste complète de vecteurs d'entrée. Suite à l'analyse des différentes Signatures générées pour les trois différents circuits il est possible de tirer les conclusions suivantes :

- les erreurs dont les probabilités d'occurrences sont les plus élevées sont les erreurs les plus « simples » qui se traduisent par un faible nombre de bits modifiés sur les sorties primaires du circuit ;

- l'erreur relative maximale sur les probabilités d'occurrences est inversement proportionnelle au nombre de vecteurs utilisés lors de l'injection de pannes ;
- l'erreur relative maximale sur les probabilités d'occurrences est principalement due au filtrage. Le fait de diminuer le nombre de vecteurs lors de l'injection avec LIFTING afin de diminuer le temps d'injection est donc bénéfique tout en ayant un faible impact sur la précision de la Signature ;
- plus le circuit possède un grand nombre de lieux d'injection et une grande sortie primaire, moins la Signature sera précise ;
- pour les circuits arithmétiques testés, à tailles égales, les Signatures logiques sont moins précises que leur équivalent arithmétique. Pour le circuit logique testé, des Signatures logiques et arithmétiques de tailles égales sont autant précises l'une que l'autre. Il est donc possible de conjecturer que les Signatures arithmétiques et logiques sont respectivement mieux adaptées pour des circuits arithmétiques et logiques.

Les erreurs relatives obtenues sont de l'ordre de 10 % ou inférieur. Ces résultats sont satisfaisants, car il le fait d'utiliser une Signature est déjà une approximation. Le choix du taux de filtrage devra aussi se faire en fonction du temps de simulation nécessaire sous Simulink pour lire et appliquer la Signature. La méthodologie utilisée pour générer une Signature est basée sur la supposition qu'il existe un comportement moyen au circuit lorsque l'on généralise pour tous vecteurs d'entrées et que l'on considère l'effet d'une panne parmi une liste de pannes. Il est possible de définir le concept de Signature comme étant l'invariance du comportement d'un circuit en présence d'une panne parmi une liste de pannes indépendamment de l'entrée. Dans le chapitre suivant, les Signatures arithmétiques obtenues seront appliquées sous Simulink, la précision de l'injection de pannes sera estimée, et les performances quantifiées.

CHAPITRE 5

INSTRUMENTATION SOUS SIMULINK

Dans ce chapitre nous détaillerons la méthodologie nécessaire afin de créer un composant émulant le comportement fautif dû aux radiations. Nous montrerons quelques exemples et applications. L'injection effectuée à haut niveau sera comparée avec celle à bas niveau afin de quantifier sa précision et ses performances.

5.1 Création de composants fautifs sous Simulink

L'objectif est de proposer une méthodologie permettant de réaliser une nouvelle bibliothèque de composants fautifs sous Simulink. Pour cela, une structure d'instrumentation a été définie dans le but de lire les Signatures et de générer une erreur appliquée à la sortie parfaite afin de générer la sortie erronée du module. Étant donné que l'outil de simulation Simulink est utilisé, des Signatures arithmétiques produisant une information décimale seront appliquées. La Signature contient la différence entre la valeur parfaite et la valeur erronée, il faut donc, une fois la valeur obtenue par la Signature, l'additionner à la valeur parfaite pour obtenir la valeur erronée.

Les critères fixés pour cette méthodologie sont les suivants :

- changer aisément la Signature à utiliser ;
- changer aisément le module à utiliser ;
- être le plus simple possible pour ne pas ralentir de manière significative le temps d'exécution ;
- éviter le débordement de la valeur.

Afin de réaliser cela, la structure présentée à la Figure 5.1 a été proposée.

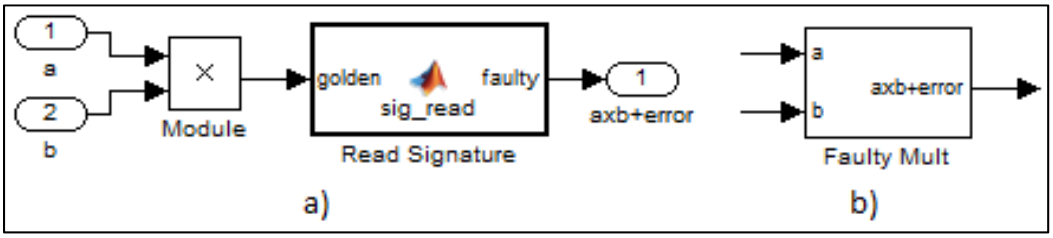

Figure 5.1 Structure Simulink permettant de lire et d'appliquer les Signatures ; (a)Instrumentation d'un modèle avec un saboteur afin d'appliquer une Signature à un multiplicateur ; (b) Block multiplicateur fautif sous Simulink

Comme observé sur la Figure 5.1 a), un saboteur a été ajouté en sortie du module ciblé, ici un multiplicateur. Le saboteur est réalisé par une fonction Matlab qui contient la Signature sous la forme d'un tableau. Il prend comme entrée la sortie parfaite du circuit. Sa sortie correspond à la sortie erronée, soit $Sortie_{parfaite} + Erreur = Sortie_{éronnée}$. Cette dernière est choisie de manière pseudo-aléatoire en respectant la distribution de probabilités d'erreur décrite dans la Signature. Le saboteur prend aussi en compte que certaines erreurs sont impossibles. Les deux blocs, module cible et saboteur, sont encapsulés dans un bloc représentant le module fautif (Figure 5.1 b)). Les entrées correspondent aux entrées du module souhaité. La sortie du bloc ainsi réalisé correspond à la valeur erronée, soit la valeur parfaite additionnée à l'erreur générée à partir de la distribution de probabilité d'erreur décrite dans la Signature.

La lecture de la Signature est la fonction clé de l'injection de pannes, elle est réalisée par le bloc « Read Signature » présenté dans la Figure 5.1. La Signature est lue, et l'erreur à un instant t est déterminée de manière pseudo aléatoire.

Nous rappelons que la Signature est sous la forme présentée au Tableau 5.1.

Tableau 5.1 Format d'une Signature

Erreur	**Nb d'occurrences**
…	…
…	…

La première étape consiste à enlever les valeurs interdites et à créer une nouvelle Signature temporaire. Une valeur interdite, correspond à un débordement, par exemple, si un bus 8 bits est utilisé, et que la valeur est non signée, alors si la sortie parfaite vaut 234, il est impossible d'avoir une erreur supérieure à +21 (255-234).

Pour les descriptions suivantes, nous considérons que la Signature temporaire prend la forme présentée au Tableau 5.2.

Tableau 5.2 Exemple de Signature

Erreur	**Nb d'occurrences**
X1	N1
X2	N2
X3	N3
...	...
Xn	Nn

Ensuite, afin de la lire et de respecter les probabilités indiquées par le nombre d'occurrences, les étapes suivantes sont effectuées :

1. Un nombre R est aléatoirement choisi, R étant compris entre 1 et le nombre total d'occurrences de la Signature ($\sum_{i=1}^{n} Ni$);

2. Afin de lier le nombre obtenu à une erreur, il suffit de considérer que la valeur Xi correspond à l'intervalle]Ni-1 ;Ni], par exemple si N2 < R < N3 alors la valeur X3 sera retournée comme erreur. Il est évident que l'erreur X1 correspond à l'intervalle]0 ;N_1] ;
3. La dernière étape consiste à additionner l'erreur avec la valeur parfaite pour obtenir la valeur erronée. De cette manière, pour des vecteurs d'entrée aléatoire, la distribution de probabilité de l'erreur tendra vers celle décrite dans la Signature lorsque le temps de simulation est suffisamment grand. On se rapproche ainsi du comportement fautif dû aux radiations observé à bas niveau.

C'est en prévision de cette lecture sous Simulink que le format de la Signature a été défini. Effectivement, nous avons opté pour conserver le nombre d'occurrences d'une erreur et non sa probabilité en pourcentage. Comme expliqué précédemment, ceci facilite l'injection de pannes sous Simulink. De plus, en utilisant les probabilités d'occurrences, il serait plus difficile de supprimer les erreurs impossibles, car la sommes des probabilités de vaudrait plus 100%. Chaque probabilité devrait alors être recalculée. Tandis qu'en utilisant le nombre d'occurrences, aucun calcul n'est nécessaire.

5.2 Étude de la précision de l'injection de pannes à haut niveau d'abstraction

Afin de quantifier la précision de la campagne d'injection à haut niveau, sous Simulink, les distributions de probabilités seront comparées avec celles obtenues à bas niveau pour une entrée spécifique suite à l'injection de chaque panne un à une avec l'outil LIFTING. Il est évident que la distribution de probabilité sous Simulink va différer de celle observée à bas niveau. Tout d'abord, à haut niveau, la distribution est indépendante de l'entrée, à cause de la généralisation de la Signature. Cette dernière est aussi filtrée, ce qui augmentera les disparités entre les deux distributions. Afin de comparer les deux distributions, le concept de Signature spécifique à une entrée sera utilisé. Ceci correspond à la distribution spécifique à une entrée. À partir des rapports d'injection générés par LIFTING, une Signature parfaite a été générée, spécifiquement à chaque entrée, et sans appliquer de filtrage. Cette Signature spécifique correspond donc à la distribution exacte des erreurs observées en sortie lorsque

chaque panne de la liste de pannes est injectée une à une pour une entrée définie. Il s'agit de l'étalon, de la distribution référence qu'il est souhaitable d'approcher au maximum grâce à une Signature. À partir d'un certain temps de simulation suffisamment élevé, la distribution d'erreur observée en sortie va tendre vers celle contenue dans la Signature. Si le temps de simulation est trop court, il est impossible de caractériser de manière précise la distribution de l'erreur. Le phénomène est alors aléatoire.

Les différences observées ont été classifiées en trois catégories (ou types) :

1. Certaines erreurs apparaissent dans la Signature, mais n'apparaissent pas dans les distributions spécifiques à certaines entrées ;
2. Certaines erreurs, présentes dans certaines distributions spécifiques à certaines entrées, n'apparaissent pas dans la Signature. Ceci est dû au fait que la Signature est filtrée contrairement à la distribution spécifique ;
3. Des erreurs sont communes entre les deux distributions, mais possèdent des probabilités d'occurrence différentes.

Pour les deux premiers types de différences, la somme des probabilités d'occurrence des erreurs appartenant à ces catégories est calculée pour chaque Signature spécifique. Ensuite, une moyenne est effectuée sur l'ensemble des erreurs observées pour chaque entrée. Pour le dernier type de différence, l'erreur absolue moyenne est calculée entre les probabilités d'occurrence correspondantes sur toute la distribution. L'erreur absolue est calculée ici grâce à la valeur absolue de la différence entre les deux valeurs. Afin de comparer la distribution de la Signature à la distribution spécifique, les coefficients de corrélation sont calculés entre la distribution de la Signature et la distribution spécifique à une entrée pour la distribution complète (qui contient toutes les erreurs et les probabilités de chacune des deux distributions) et partielle (qui ne contient que les erreurs communes aux deux distributions). Ces critères sont calculés pour chaque entrée, pour ensuite calculer la moyenne globale. Pour des raisons d'interprétation, des probabilités d'occurrences seront présentées au lieu des nombres d'occurrences.

Afin d'illustrer cette méthode de quantification d'erreur, l'exemple présenté au Tableau 5.3 sera utilisé.

Tableau 5.3 (a) Signature spécifique à une entrée (b) Signature généralisée

Erreur	Probabilité
-1024	2 %
-512	9 %
-256	12 %
-128	7 %
0	50 %
+4	4 %
+1024	11 %
+2048	5 %

(a)

Erreur	Probabilité
-2048	5 %
-512	7 %
-256	13 %
-128	5 %
0	53 %
+256	5 %
+512	4 %
+1024	15 %
+2048	3 %

(b)

Les différences de catégorie 1 sont surlignées en bleu dans le Tableau 5.3 (b), celles de type 2 sont en rouge dans le Tableau 5.3 (a). Dans ce cas fictif, les différences se quantifient de la manière suivante :

- 14 % de probabilités d'occurrences cumulées, présentes dans la Signature, n'apparaissent pas dans la Signature spécifique à une entrée ;
- 6 % de probabilités d'occurrences cumulées, observées dans une Signature spécifique à une entrée, ne se trouvent pas dans la Signature ;
- pour les erreurs présentes dans les deux distributions, on peut calculer l'imprécision de leurs probabilités avec la formule suivante :

$$E = \frac{1}{6} \times (|7-9| + |13-12| + |5-7| + |53-50| + |15-11| + |3-5|)$$
$$= 3{,}1\%;$$

- le coefficient de corrélation obtenue est de 97,45 % si l'on considère les deux tableaux au complet. Le coefficient de corrélation vaut 99,28 % lorsque l'on considère uniquement les erreurs communes dans les deux distributions.

Afin d'appliquer cette analyse aux exemples précédents, à savoir l'additionneur et le multiplicateur 8 bits, OGAS a été utilisé afin de créer les différentes Signatures spécifiques pour chaque entrée. Un script Matlab permet ensuite de comparer ces distributions à celles d'une Signature. Les résultats obtenus sont les présentés au Tableau 5.4.

Tableau 5.4 Quantification de la précision de l'injection de panne sous Simulink avec les Signatures d'un multiplicateur (X) et d'un additionneur (+) générés précédemment

	Signature	**Différence Type 1**	**Différence Type 2**	**Différence Type 3**	**Coeff. Corrél. complet**	**Coeff. Corrél. Partiel**
+	p > 1,3 %	7,2 %	2,1 %	1,1 %	98,37 %	99,59 %
	p > 0,3 %	8,1 %	0,3 %	1,1 %	98,53 %	99,56 %
X	p > 0,8 %	1,2 %	7,7 %	0,64 %	99,80 %	99,94 %
	p > 0,4 %	1,7 %	5,5 %	0,52 %	99,86 %	99,93 %

Le Tableau 5.4 contient les résultats de l'analyse décrite ci-dessus pour les Signatures générées dans la partie 4.3.1 (Génération de la Signature d'un additionneur 8 bits) et 4.3.2 (Génération de la Signature d'un multiplicateur 8 bits). Nous rappelons que la différence de type 1 correspond à une erreur présente dans la Signature ou non dans la Signature spécifique. Une différence de type 2 correspond à une erreur présente dans la Signature spécifique et non dans la Signature. De plus, une différence de type 3 correspond aux erreurs présentes dans les deux distributions avec des probabilités d'occurrence différentes.

Les probabilités d'erreurs cumulées obtenues sont toutes inférieures ou égales à 8 % pour les différences de type 1 et 2. Cette valeur est très faible quand l'on considère la généralisation et le filtrage appliqué à la Signature. L'erreur moyenne absolue observée sur les probabilités d'occurrences des erreurs correspondantes est elle aussi faible. Pour finir, le coefficient de corrélation est quant à lui très élevé et avoisine les 99 % dans les quatre exemples présentés ce qui est très satisfaisant.

5.3 Étude du temps de simulation de l'injection de pannes à haut niveau

Afin de caractériser l'injection à haut niveau effectuée sous Simulink, pour différents nombres d'échantillons, le temps nécessaire pour simuler le modèle utilisant un saboteur afin d'appliquer la Signature a été mesuré. Suite aux expérimentations précédentes deux exemples ont été choisis : l'additionneur 8bits sans retenue utilisant une Signature avec le seuil de filtrage 0,3 % ainsi que le multiplicateur 8bits utilisant une Signature avec le seuil de filtrage de 0,4 %. Il s'agit en effet des cas les plus performants pour ces exemples, et aussi des Signatures les plus longues pour chaque circuit.

Comme observé sur la Figure 5.2, pour 500 000 échantillons, le temps de simulation nécessaire pour le composant fautif est 10 fois supérieur à celui du composant unique. Les résultats sont similaires pour les deux circuits, le temps de Simulation reste court, même pour un grand nombre d'échantillons simulés. Le temps de simulation du circuit fautif est nettement plus élevé que le circuit simple « parfait », mais il faut aussi le comparer au temps de simulation à bas niveau. Le gain en temps de simulation se fait particulièrement ressentir pour les circuits plus complexes dont le temps de simulation se compte en minutes et non en secondes.

Le temps supplémentaire nécessaire à l'exécution du saboteur se calcule de la manière suivante pour les Signatures de taille similaires à celles utilisées dans les divers exemples :

$$Tsup = 1{,}47 \times 10^{-5} \times t + 0{,}11\ s \tag{5.1}$$

Ce temps vient s'additionner au temps nécessaire à simuler le modèle parfait, sans saboteur. La formule 5.1 indique que le temps additionnel causé par le saboteur est d'environ 1 seconde tous les 60 500 échantillons simulés. Plus le modèle représenté sera complexe, plus le temps supplémentaire nécessaire à l'injection sera faible comparé au temps de simulation du modèle simple.

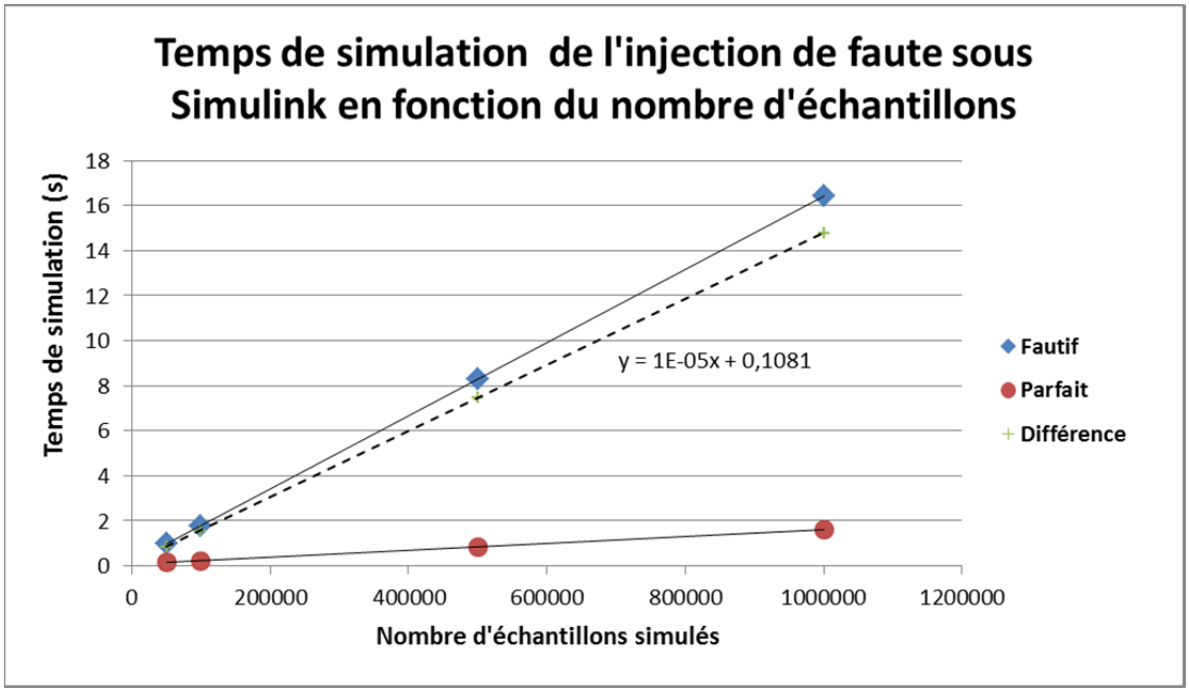

Figure 5.2 Temps de simulation sous Simulink d'un composant fautif et un composant parfait en fonction du nombre d'échantillons simulés

5.4 Bilan de l'instrumentation sous Simulink

Nous avons présenté la méthode utilisée afin d'instrumenter un modèle à haut niveau grâce à un saboteur permettant d'injecter des pannes à partir d'une Signature. Une simple fonction Matlab permet d'injecter des pannes pseudo aléatoires en respectant une distribution de probabilité définie. Le format proposé est modulable et peut être aisément réappliqué à différents modèles. La méthodologie utilisée permet d'injecter des pannes similaires à une injection à bas niveau. Les distributions de probabilités obtenues à haut niveau, lors de la simulation avec Simulink, ont été comparées avec les distributions de probabilités observées à bas niveau avec l'outil LIFTING. Bien que les Signatures soient généralisées puis filtrées, un coefficient de corrélation de 99 % ou supérieur est obtenu pour les quatre exemples présentés. Le comportant invariant d'un circuit face à une panne parmi une liste de pannes généralisée pour toutes les entrées a donc été capturé avec précision.

L’instrumentation d’un modèle par une fonction Matlab augmente le temps de simulation. La lecture d’une Signature composée d’une vingtaine d’erreurs augmente le temps de simulation d’une seconde tous les 60 500 échantillons simulés. Mais le temps de simulation reste très faible comparé au temps nécessaire à bas niveau qui lui se compte en minutes. La différence est encore plus marquée pour les circuits les plus complexes.

CONCLUSION

Le premier objectif de recherche de ce projet était d'investiguer et de mettre en place une méthodologie de conception permettant l'injection de pannes dans un modèle se rapprochant de l'effet des radiations cosmiques sur un FPGA. Ensuite, le second objectif est de proposer un format permettant de capturer le comportement fautif d'un circuit et d'appliquer cette information à plus haut niveau d'abstraction. Pour finir, le dernier objectif est de proposer et de caractériser une méthodologie d'injection de pannes sous Simulink émulant le comportement fautif d'un circuit dû aux radiations.

Afin de répondre au premier objectif, nous avons investigué et présenté cinq outils d'injection de pannes par simulation. LIFTING a été celui retenu pour ses fonctionnalités, sa flexibilité et ses rapports d'injection détaillés. Les outils d'injection de pannes par simulation pouvant être laborieux pour les circuits complexes, une seconde méthodologie d'injection de pannes avec l'outil SEU Controller a été proposée. Ce dernier permet d'injecter des pannes en émulant, sur un FPGA, l'effet des radiations cosmiques sur la configuration. Ce deuxième type d'injection est plus précis, mais plus difficile à mettre en place, car il nécessite l'implémentation sur un FPGA et l'importation des données. De plus, la campagne d'injection est plus complexe, car la liste de pannes est considérablement plus grande. Dans ce projet, l'injection par simulation a été choisie. Le modèle de pannes est plus simple, un outil analysant l'effet des pannes est déjà à disposition et ne requiert pas d'effort de développement supplémentaire. Afin de compenser l'explosion de de la complexité limitant l'injection par simulation, nous avons validé que diminuer le nombre de vecteurs d'entrée utilisés lors de la campagne d'injection permet de diminuer le temps de simulation ainsi que la taille des rapports générés.

Ensuite, afin de satisfaire le second objectif de ce projet, le nouveau concept de Signature du comportement fautif d'un circuit a été proposé. Ce format, généré à partir de différents rapports d'injection générés à bas niveau, constitue une distribution de probabilité de l'erreur observée en sortie d'un circuit en présence d'une panne parmi une liste de pannes,

généralisée pour toutes les entrées. Une Signature permet de compresser et d'adapter l'information pour être lu efficacement à haut niveau d'abstraction. Deux exemples de formats de Signature correspondant à des formules de calcul d'erreur différentes ont été présentés. Une Signature arithmétique utilise une soustraction et est en format décimal. Une signature logique, quant à elle, utilise un ou exclusif et est donc binaire. Ces deux Signatures sont complémentaires et compressent différemment l'information. Le format arithmétique s'adapte à l'utilisation sous Simulink grâce à son format décimal. Le format logique permet de garder l'information sur la position des bits en sorties qui sont modifiés ainsi que de connaitre la corrélation entre les différents bits. L'erreur introduite par les différents facteurs menant à la génération d'une Signature a été estimée pour un additionneur 8 bits, un multiplicateur sans retenue 8bits et le circuit ISCAS85 c432. Réduire le nombre de vecteurs utilisés lors de l'injection de pannes augmente de manière linéaire l'erreur observée dans les Signatures. Toutefois, l'erreur que mesurée semble principalement causée par le filtrage effectué lors de la génération de la Signature. Il est donc conseillé de fixer un taux de filtrage faible. Si le circuit est trop complexe, il est possible de diminuer le temps de simulation en réduisant le nombre de vecteurs d'entrée utilisé et tout en gardant une bonne précision sur la Signature. De plus, les erreurs les plus probables sont les erreurs « simples » qui ne modifient qu'un faible nombre de bits sur la sortie. Une Signature logique semble aussi plus adaptée pour un circuit logique. Cette affirmation est similaire pour les Signatures et les circuits arithmétiques.

Pour finir, un modèle Simulink a été instrumenté avec un saboteur réalisé par une fonction Matlab contenant une Signature. Ceci permet d'injecter des erreurs de manière pseudo-aléatoire en respectant la distribution de probabilité définie dans la Signature. Il s'agit d'un format simple et modulaire, qui peut aisément s'appliquer à d'autre modèle que ceux présentés. Afin de caractériser l'injection ainsi réalisée, la distribution de probabilité observée à haut niveau a été comparée à celle observée à bas niveau pour chaque entrée. Le concept de distribution spécifique à une entrée a été introduit, ceci correspond à la distribution de l'erreur observée en sortie en présence d'une panne parmi une liste de pannes pour une entrée spécifique. Plusieurs métriques ont permis de valider la précision des

Signatures. Malgré le fait que les Signatures soient généralisées pour toutes les entrées et ensuite filtrées, il est possible d'obtenir une corrélation de coefficient moyen d'environ 99 % ou supérieur pour les quatre Signatures présentées (multiplicateur et additionneur). Le temps supplémentaire nécessaire à l'injection de pannes utilisant une Signature sous Simulink a aussi été quantifié. Avec les tailles de Signature utilisées, légèrement supérieures à 20 lignes, un temps supplémentaire de simulation d'une seconde tous les 60 500 échantillons simulés a été mesuré. Ce temps est négligeable comparé au temps d'injection nécessaire à bas niveau afin d'injecter des pannes similaires. Plus le circuit est complexe, plus ce temps additionnel sera négligeable.

Pour résumer, dans ce projet, une nouvelle méthodologie permettant de générer des composants émulant le comportement fautif dû aux radiations avec précision a été présentée. À partir d'un modèle représenté à bas niveau, sous forme de liste d'interconnexions, le comportement invariant d'un circuit a été capturé face à une panne parmi une liste de pannes généralisé pour toutes les entrées. L'injection à haut niveau est évidemment couteuse en temps de simulation, mais cela reste considérablement plus faible comparé à l'utilisation d'un outil d'injection par simulation.

Les résultats de ce projet ont fait l'objet d'une publication lors de la conférence NEWCAS 2013 (Robache *et al.*, 2013). Cet article introduit le nouveau concept de Signature. La méthodologie de génération proposée ainsi que les expérimentations avec un multiplicateur 8 bits y sont présentées.

RECOMMANDATIONS

Les ouvertures à ce projet sont nombreuses. L'utilisation de l'outil SEU Controller permettrait d'avoir des modèles de pannes plus précis, mais nécessiterait la définition d'un nouveau type de Signature adapté. L'injection par émulation permettrait aussi de traiter des circuits plus complexes. Nous pouvons aussi imaginer un format de Signature permettant d'injecter des pannes de type transitoires de façon pseudo-aléatoire avec différentes durées. Beaucoup d'autres formats de Signatures peuvent être définis, apportant d'autres informations nécessaires dans certaines simulations (nouvelle formule, nouveau modèle de panne, etc.). D'autre part, les Signatures actuelles ne prennent pas en compte l'entrée du circuit. Afin de profiter d'une injection plus précise, il pourrait être intéressant de proposer une nouvelle méthodologie permettant cela. Le travail présenté dans ce mémoire est aussi limité par le type de circuit dont le comportement fautif peut être capturé. Actuellement, seuls les circuits combinatoires ou sans mémoire sont utilisés. Il serait potentiellement intéressant de moduler la Signature en fonction de la séquence de vecteurs d'entrées.

ANNEXE I

EXEMPLE D'UTILISATION DE SEU CONTROLLER

À travers différentes captures de l'interface disponible via hyperterminal les fonctionnalités de SEU Controller seront détaillées.

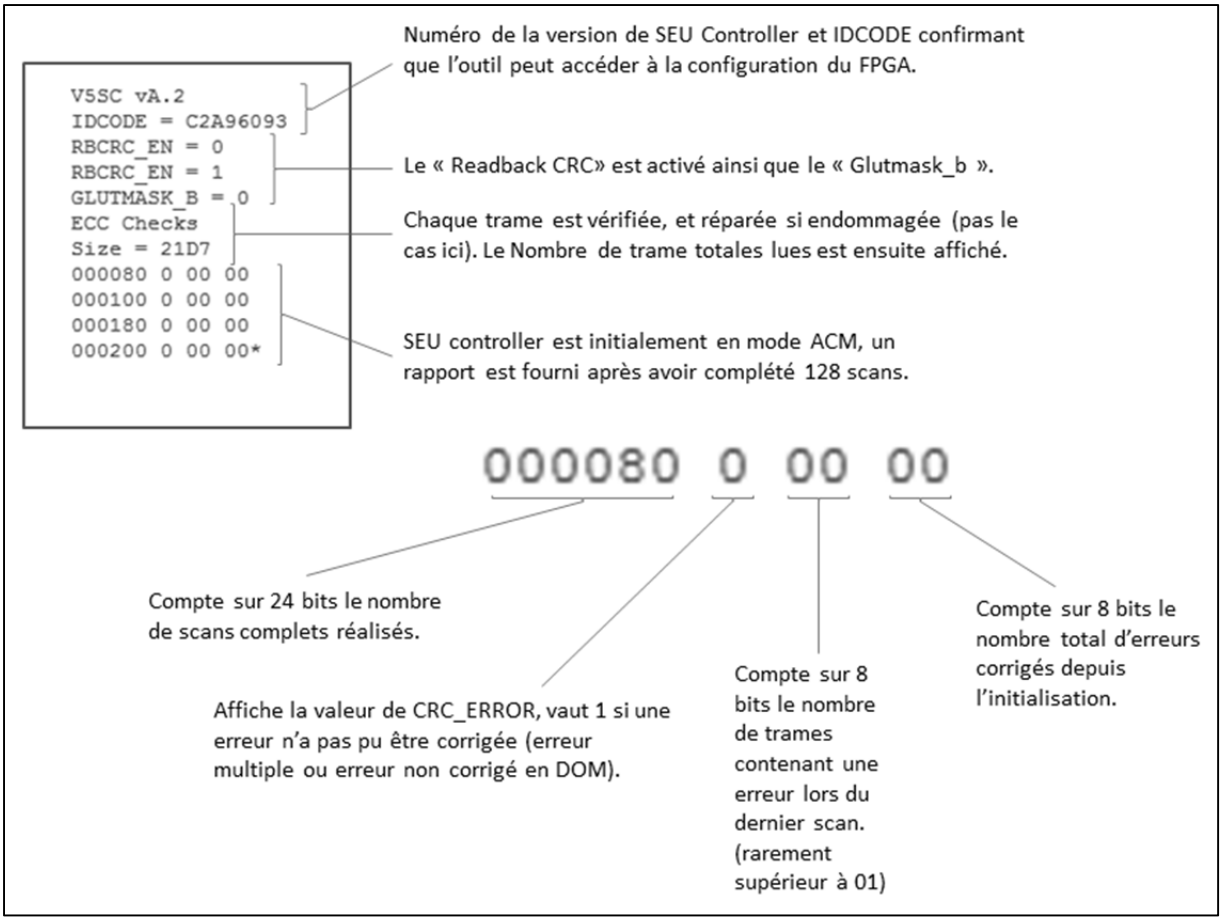

Figure-A I-1 Exemple d'utilisation de SEU Controller (initialisation et rapports)

Cette étape correspond à l'initialisation de SEU Controller. On y voit aussi les premiers rapports de scan.

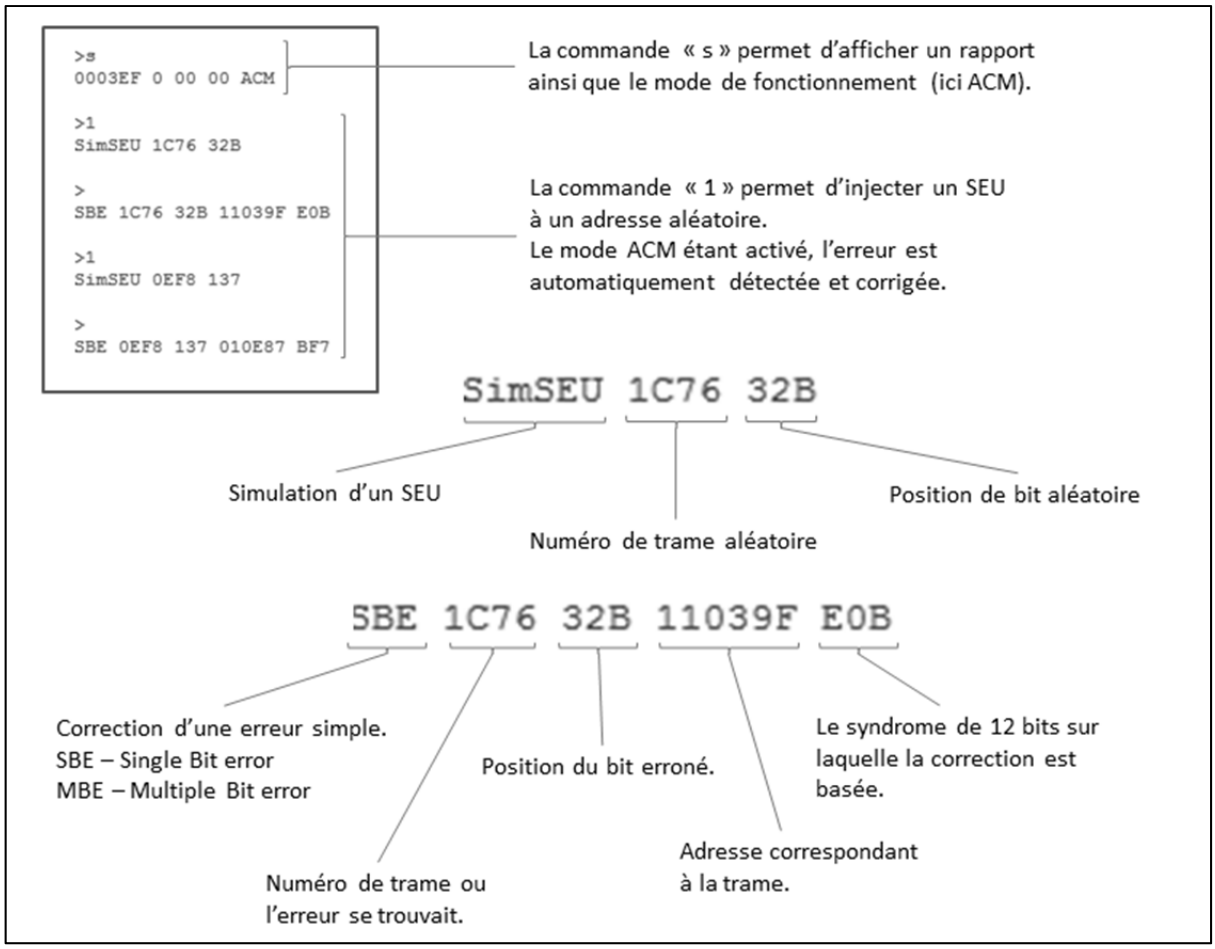

Figure-A I-2 Exemple d'utilisation de SEU Controller
(injection de SEU aléatoirement placés)

Cette figure présente les commandes permettant d'injecter des SEU de manière aléatoire ainsi que les rapports générés suite à leur correction en mode ACM.

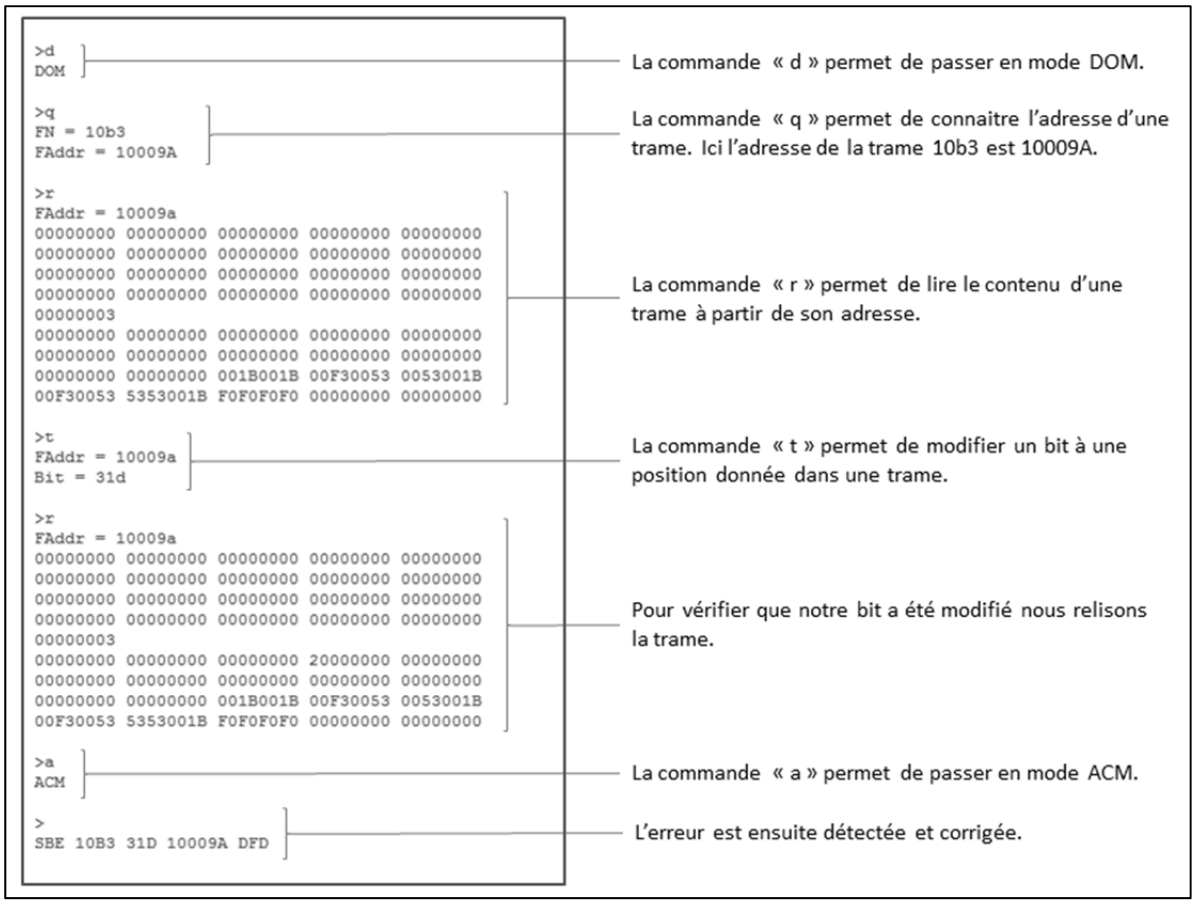

Figure-A I-3 Exemple d'utilisation de SEU Controller (modification d'un bit précis)

Ici, la fonction « t » est détaillée. Elle permet de modifier un bit dans une trame à une position donnée. Afin de vérifier l'effet de ce changement, le mode DOM est appliqué en début de test afin qu'aucune correction ne soit effectuée.

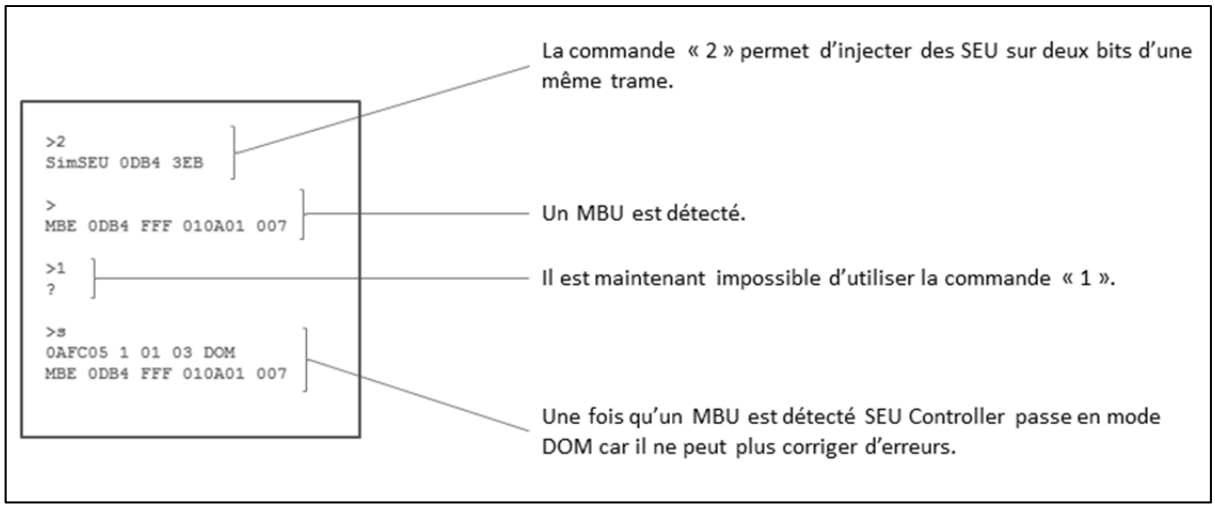

Figure-A I-4 Exemple d'utilisation de SEU Controller
(Injection de MBU aléatoirement placés)

Dans cette figure la fonction permettant d'injecter un MBU aléatoirement placé est détailée. Une fois un MBU injecté, il est impossible de le corriger. L'outil passe alors en mode DOM automatiquement. Une fois dans cette situation il devient impossible d'injecter un SEU ou MBU de manière aléatoire avec les commandes « 1 » et « 2 ». Mais il est possible d'injecter une erreur grâce à la commande « t », en fournissant l'adresse de la trame et la position du bit à modifier.

ANNEXE II

PRÉCISION DE SIGNATURES D'UN ADDITIONNEUR 8 BITS

Tableau-A II-1 Signatures arithmétiques d'un additionneur 8 bits et leurs erreurs relatives en fonction du nombre de vecteurs utilisés lors de l'injection

Erreur	Parfait	Filtré (65536 vect.)		Filtré (50000 vect.)		Filtré (40000 vect.)		Filtré (30000 vect.)		Filtré (20000 vect.)	
	P(Erreur)	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.
-240	0,1387%	0,1391%	0,2611%	0,1379%	0,5626%	0,1410%	1,6794%	0,1391%	0,3009%	0,1380%	0,5195%
-224	0,2746%	0,2753%	0,2611%	0,2801%	2,0008%	0,2776%	1,0958%	0,2774%	1,0408%	0,2714%	1,1697%
-192	0,5463%	0,5477%	0,2611%	0,5475%	0,2100%	0,5512%	0,8970%	0,5529%	1,2156%	0,5418%	0,8263%
-128	2,8986%	2,9061%	0,2611%	2,9055%	0,2391%	2,9145%	0,5490%	2,9045%	0,2068%	2,8928%	0,1994%
-64	2,3466%	2,3527%	0,2611%	2,3480%	0,0584%	2,3607%	0,6012%	2,3518%	0,2233%	2,3559%	0,3956%
-32	2,6070%	2,6138%	0,2611%	2,6125%	0,2124%	2,6157%	0,3346%	2,6031%	0,1498%	2,6176%	0,4080%
-16	2,7202%	2,7273%	0,2611%	2,7296%	0,3444%	2,7288%	0,3162%	2,7236%	0,1245%	2,7160%	0,1536%
-8	2,7429%	2,7500%	0,2611%	2,7512%	0,3056%	2,7479%	0,1820%	2,7477%	0,1750%	2,7453%	0,0903%
-4	2,6863%	2,6933%	0,2611%	2,6912%	0,1845%	2,6960%	0,3630%	2,6987%	0,4625%	2,7002%	0,5176%
-2	2,3423%	2,3485%	0,2611%	2,3499%	0,3223%	2,3420%	0,0131%	2,3505%	0,3485%	2,3553%	0,5554%
-1	3,7944%	3,8043%	0,2611%	3,7993%	0,1291%	3,8002%	0,1514%	3,8058%	0,3001%	3,8087%	0,3749%
0	51,9928%	52,1285%	0,2611%	52,1259%	0,2561%	52,1324%	0,2686%	52,1343%	0,2723%	52,1258%	0,2559%
1	3,7902%	3,8001%	0,2611%	3,8039%	0,3607%	3,8025%	0,3240%	3,7984%	0,2172%	3,7965%	0,1650%
2	3,2411%	3,2495%	0,2611%	3,2516%	0,3264%	3,2567%	0,4834%	3,2465%	0,1693%	3,2433%	0,0696%
4	3,0429%	3,0509%	0,2611%	3,0545%	0,3803%	3,0465%	0,1182%	3,0450%	0,0692%	3,0463%	0,1112%
8	2,9184%	2,9260%	0,2611%	2,9242%	0,2012%	2,9252%	0,2340%	2,9271%	0,2977%	2,9319%	0,4622%
16	2,8051%	2,8125%	0,2611%	2,8101%	0,1756%	2,8084%	0,1156%	2,8172%	0,4308%	2,8202%	0,5376%
32	2,6466%	2,6535%	0,2611%	2,6485%	0,0698%	2,6498%	0,1188%	2,6597%	0,4952%	2,6522%	0,2112%
64	2,3636%	2,3697%	0,2611%	2,3717%	0,3444%	2,3632%	0,0141%	2,3604%	0,1321%	2,3659%	0,1006%
128	2,8986%	2,9061%	0,2611%	2,9066%	0,2766%	2,8982%	0,0124%	2,9082%	0,3316%	2,9189%	0,7028%
192	0,5406%	0,5421%	0,2611%	0,5449%	0,7935%	0,5375%	0,5774%	0,5443%	0,6688%	0,5481%	1,3767%
224	0,2689%	0,2696%	0,2611%	0,2710%	0,7692%	0,2696%	0,2520%	0,2708%	0,7183%	0,2705%	0,5865%
240	0,1330%	0,1334%	0,2611%	0,1345%	1,0767%	0,1344%	1,0364%	0,1328%	0,2128%	0,1374%	3,3130%
		Moyenne	0,2611%		0,7935%		1,0958%		1,2156%		1,3767%
		Maximum	0,2611%		0,4174%		0,4234%		0,3723%		0,5697%

Tableau-A II-2 Signatures logiques d'un additionneur 8 bits et leurs erreurs relatives en fonction du nombre de vecteurs utilisés lors de l'injection

Erreur	Parfait	Filtré (65536 vect.)		Filtré (50000 vect.)		Filtré (40000 vect.)		Filtré (30000 vect.)		Filtré (20000 vect.)	
	P(Erreur)	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.
0	51,9928%	53,2591%	2,4356%	53,2543%	2,4265%	53,2570%	2,4316%	53,2530%	2,4239%	53,2634%	2,4439%
1	4,5290%	4,6393%	2,4356%	4,6433%	2,5244%	4,6407%	2,4668%	4,6504%	2,6807%	4,6333%	2,3041%
10	3,5326%	3,6186%	2,4356%	3,6180%	2,4183%	3,6233%	2,5683%	3,6326%	2,8304%	3,6203%	2,4830%
11	1,5399%	1,5774%	2,4356%	1,5790%	2,5397%	1,5761%	2,3518%	1,5766%	2,3862%	1,5646%	1,6088%
100	3,6232%	3,7114%	2,4356%	3,7142%	2,5107%	3,7040%	2,2312%	3,7073%	2,3217%	3,7075%	2,3276%
110	1,0417%	1,0670%	2,4356%	1,0695%	2,6729%	1,0613%	1,8815%	1,0570%	1,4700%	1,0583%	1,5927%
111	0,7699%	0,7887%	2,4356%	0,7866%	2,1706%	0,7876%	2,3011%	0,7795%	1,2395%	0,7988%	3,7491%
1000	3,6232%	3,7114%	2,4356%	3,7101%	2,3977%	3,7179%	2,6137%	3,7081%	2,3427%	3,7115%	2,4369%
1100	1,0870%	1,1134%	2,4356%	1,1131%	2,4009%	1,1210%	3,1279%	1,1169%	2,7545%	1,1209%	3,1224%
1110	0,5208%	0,5335%	2,4356%	0,5361%	2,9351%	0,5342%	2,5739%	0,5348%	2,6727%	0,5395%	3,5845%
10000	3,6232%	3,7114%	2,4356%	3,7082%	2,3452%	3,7148%	2,5290%	3,7095%	2,3809%	3,7074%	2,3235%
11000	1,0870%	1,1134%	2,4356%	1,1108%	2,1970%	1,1066%	1,8031%	1,1095%	2,0777%	1,1185%	2,9040%
11100	0,5435%	0,5567%	2,4356%	0,5550%	2,1223%	0,5587%	2,8001%	0,5565%	2,4025%	0,5596%	2,9585%
100000	3,6232%	3,7114%	2,4356%	3,7054%	2,2687%	3,7125%	2,4655%	3,7170%	2,5894%	3,7127%	2,4696%
110000	1,0870%	1,1134%	2,4356%	1,1150%	2,5776%	1,1132%	2,4177%	1,1195%	2,9973%	1,1149%	2,5715%
111000	0,5435%	0,5567%	2,4356%	0,5585%	2,7597%	0,5594%	2,9275%	0,5653%	4,0232%	0,5497%	1,1374%
1000000	3,6232%	3,7114%	2,4356%	3,7109%	2,4206%	3,7100%	2,3951%	3,7237%	2,7752%	3,7101%	2,3999%
1100000	1,0870%	1,1134%	2,4356%	1,1112%	2,2298%	1,1076%	1,9032%	1,1120%	2,3053%	1,1114%	2,2482%
1110000	0,5435%	0,5567%	2,4356%	0,5573%	2,5412%	0,5524%	1,6438%	0,5576%	2,5967%	0,5532%	1,7838%
10000000	5,7971%	5,9383%	2,4356%	5,9379%	2,4283%	5,9380%	2,4313%	5,9381%	2,4328%	5,9385%	2,4396%
11000000	2,1739%	2,2269%	2,4356%	2,2270%	2,4410%	2,2281%	2,4917%	2,2144%	1,8622%	2,2284%	2,5057%
11100000	1,0870%	1,1134%	2,4356%	1,1213%	3,1585%	1,1179%	2,8456%	1,1091%	2,0383%	1,1145%	2,5308%
11110000	0,5435%	0,5567%	2,4356%	0,5574%	2,5703%	0,5576%	2,5952%	0,5516%	1,4859%	0,5631%	3,6054%
		Moyenne	2,4356%		3,1585%		3,1279%		4,0232%		3,7491%
		Maximum	2,4356%		2,4807%		2,4259%		2,3952%		2,5013%

ANNEXE III

PRÉCISION DE SIGNATURES D'UN MULTIPLICATEUR 8 BITS

Tableau-A III-1 Signatures arithmétiques d'un multiplicateur 8 bits et leurs erreurs relatives en fonction du nombre de vecteurs utilisés lors de l'injection

Erreur	Parfait	Filtré (65536 vect.)		Filtré (50000 vect.)		Filtré (40000 vect.)		Filtré (30000 vect.)		Filtré (20000 vect.)	
	P(Erreur)	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.
-2048	0,8379%	0,9085%	8,4190%	0,9087%	8,4463%	0,9113%	8,7576%	0,9026%	7,7129%	0,9018%	7,6227%
-1024	1,0413%	1,1289%	8,4190%	1,1319%	8,7029%	1,1268%	8,2145%	1,1250%	8,0457%	1,1243%	7,9756%
-512	1,2220%	1,3249%	8,4190%	1,3288%	8,7365%	1,3259%	8,4990%	1,3210%	8,0957%	1,3225%	8,2195%
-256	1,3910%	1,5081%	8,4190%	1,5093%	8,5047%	1,5082%	8,4317%	1,5034%	8,0866%	1,5050%	8,1977%
-128	1,4483%	1,5702%	8,4190%	1,5675%	8,2274%	1,5630%	7,9163%	1,5643%	8,0112%	1,5666%	8,1641%
-64	1,2364%	1,3405%	8,4190%	1,3378%	8,2060%	1,3394%	8,3297%	1,3321%	7,7439%	1,3322%	7,7535%
-32	1,0004%	1,0846%	8,4190%	1,0815%	8,1054%	1,0810%	8,0582%	1,0795%	7,9052%	1,0794%	7,9013%
-16	0,8486%	0,9200%	8,4190%	0,9227%	8,7401%	0,9187%	8,2686%	0,9164%	7,9940%	0,9175%	8,1272%
0	60,1846%	65,2515%	8,4190%	65,2478%	8,4128%	65,2546%	8,4241%	65,2605%	8,4339%	65,2603%	8,4336%
4	0,8367%	0,9071%	8,4190%	0,9103%	8,7983%	0,9081%	8,5290%	0,9077%	8,4885%	0,9059%	8,2719%
8	1,2057%	1,3072%	8,4190%	1,3050%	8,2393%	1,3081%	8,4926%	1,3048%	8,2158%	1,3049%	8,2302%
16	1,7356%	1,8817%	8,4190%	1,8798%	8,3053%	1,8835%	8,5237%	1,8846%	8,5851%	1,8835%	8,5232%
32	2,1058%	2,2831%	8,4190%	2,2862%	8,5651%	2,2859%	8,5529%	2,2857%	8,5439%	2,2861%	8,5628%
64	2,4801%	2,6890%	8,4190%	2,6916%	8,5252%	2,6894%	8,4376%	2,6959%	8,6990%	2,6958%	8,6942%
128	2,8806%	3,1231%	8,4190%	3,1264%	8,5324%	3,1299%	8,6549%	3,1276%	8,5730%	3,1251%	8,4874%
256	2,8440%	3,0835%	8,4190%	3,0830%	8,4039%	3,0834%	8,4168%	3,0864%	8,5219%	3,0845%	8,4548%
512	2,5880%	2,8059%	8,4190%	2,8028%	8,2980%	2,8039%	8,3397%	2,8078%	8,4929%	2,8059%	8,4188%
1024	2,1938%	2,3785%	8,4190%	2,3754%	8,2781%	2,3802%	8,4988%	2,3805%	8,5137%	2,3810%	8,5329%
2048	1,7840%	1,9342%	8,4190%	1,9336%	8,3822%	1,9312%	8,2505%	1,9382%	8,6422%	1,9386%	8,6669%
4096	1,3857%	1,5024%	8,4190%	1,5013%	8,3452%	1,5012%	8,3332%	1,5062%	8,6963%	1,5084%	8,8578%
8192	0,9842%	1,0671%	8,4190%	1,0687%	8,5838%	1,0663%	8,3410%	1,0698%	8,6899%	1,0706%	8,7718%
		Moyenne	8,4190%		8,4447%		8,3938%		8,3186%		8,3271%
		Maximum	8,4190%		8,7983%		8,7576%		8,6990%		8,8578%

Tableau-A III-2 Signatures logiques d'un multiplicateur 8 bits et leurs erreurs relatives en fonction du nombre de vecteurs utilisés lors de l'injection

Erreur	Parfait	Filtré (65536 vect.)		Filtré (50000 vect.)		Filtré (40000 vect.)		Filtré (30000 vect.)		Filtré (20000 vect.)	
	P(Erreur)	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.
0	60,1846%	69,3772%	15,2741%	69,3501%	15,2290%	69,3905%	15,2961%	69,3738%	15,2683%	69,3692%	15,2607%
100	0,9113%	1,0505%	15,2741%	1,0523%	15,4694%	1,0517%	15,4055%	1,0511%	15,3399%	1,0491%	15,1245%
1000	1,1696%	1,3482%	15,2741%	1,3436%	14,8759%	1,3499%	15,4190%	1,3484%	15,2880%	1,3487%	15,3130%
10000	1,5107%	1,7415%	15,2741%	1,7389%	15,1040%	1,7443%	15,4578%	1,7469%	15,6320%	1,7458%	15,5580%
100000	1,7277%	1,9916%	15,2741%	1,9952%	15,4854%	1,9920%	15,3012%	1,9940%	15,4146%	1,9967%	15,5723%
110000	0,5434%	0,6264%	15,2741%	0,6310%	16,1227%	0,6240%	14,8287%	0,6161%	13,3734%	0,6180%	13,7248%
1000000	2,0110%	2,3182%	15,2741%	2,3185%	15,2923%	2,3169%	15,2123%	2,3286%	15,7911%	2,3274%	15,7329%
1100000	0,6970%	0,8035%	15,2741%	0,8038%	15,3177%	0,7964%	14,2566%	0,8065%	15,7145%	0,8060%	15,6370%
10000000	2,2964%	2,6472%	15,2741%	2,6510%	15,4401%	2,6512%	15,4479%	2,6514%	15,4580%	2,6494%	15,3713%
11000000	0,8615%	0,9931%	15,2741%	0,9986%	15,9086%	1,0007%	16,1539%	0,9860%	14,4535%	0,9858%	14,4292%
100000000	2,2582%	2,6031%	15,2741%	2,6046%	15,3419%	2,6040%	15,3119%	2,6150%	15,8011%	2,6181%	15,9379%
110000000	1,0266%	1,1834%	15,2741%	1,1921%	16,1270%	1,1628%	13,2670%	1,1874%	15,6689%	1,1937%	16,2768%
1000000000	2,0527%	2,3662%	15,2741%	2,3679%	15,3558%	2,3645%	15,1928%	2,3622%	15,0799%	2,3558%	14,7681%
1100000000	1,0048%	1,1583%	15,2741%	1,1586%	15,3047%	1,1604%	15,4869%	1,1499%	14,4448%	1,1417%	13,6226%
10000000000	1,7797%	2,0515%	15,2741%	2,0516%	15,2804%	2,0513%	15,2603%	2,0449%	14,9045%	2,0477%	15,0619%
11000000000	0,9110%	1,0502%	15,2741%	1,0472%	14,9438%	1,0492%	15,1699%	1,0521%	15,4883%	1,0622%	16,5896%
100000000000	1,5196%	1,7517%	15,2741%	1,7540%	15,4294%	1,7519%	15,2916%	1,7451%	14,8408%	1,7430%	14,7044%
110000000000	0,7759%	0,8944%	15,2741%	0,8981%	15,7452%	0,8918%	14,9302%	0,8933%	15,1287%	0,8897%	14,6646%
1000000000000	1,2461%	1,4364%	15,2741%	1,4388%	15,4641%	1,4395%	15,5252%	1,4423%	15,7464%	1,4444%	15,9152%
1100000000000	0,6175%	0,7118%	15,2741%	0,7101%	14,9973%	0,7110%	15,1366%	0,7071%	14,5189%	0,7086%	14,7536%
10000000000000	0,9555%	1,1014%	15,2741%	1,1002%	15,1462%	1,1017%	15,3001%	1,1029%	15,4333%	1,1046%	15,6077%
100000000000000	0,6890%	0,7943%	15,2741%	0,7938%	15,2102%	0,7945%	15,2977%	0,7948%	15,3523%	0,7944%	15,2926%
		Moyenne	15,2741%		15,3905%		15,1795%		15,1882%		15,2236%
		Maximum	15,2741%		16,1270%		16,1539%		15,8011%		16,5896%

ANNEXE IV

PRÉCISION DE SIGNATURES DU CIRCUIT ISCAS85 C432

Tableau-A IV-1 Signatures arithmétiques du circuit c432 et leurs erreurs relatives en fonction du nombre de vecteurs utilisés lors de l'injection

Erreur	Parfait	Filtré (65536 vect.)		Filtré (50000 vect.)		Filtré (40000 vect.)		Filtré (30000 vect.)		Filtré (20000 vect.)	
	P(Erreur)	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.
-64	0,4094%	0,4298%	4,9841%	0,4249%	3,8047%	0,4288%	4,7463%	0,4295%	4,9209%	0,4294%	4,8971%
-32	0,6072%	0,6375%	4,9841%	0,6389%	5,2176%	0,6373%	4,9550%	0,6354%	4,6344%	0,6379%	5,0412%
-16	0,7565%	0,7942%	4,9841%	0,7896%	4,3835%	0,7925%	4,7666%	0,7979%	5,4829%	0,8002%	5,7781%
-10	0,3997%	0,4196%	4,9841%	0,4175%	4,4590%	0,4211%	5,3455%	0,4207%	5,2492%	0,4192%	4,8888%
-8	1,0232%	1,0742%	4,9841%	1,0720%	4,7743%	1,0742%	4,9853%	1,0730%	4,8720%	1,0758%	5,1457%
-2	0,5201%	0,5460%	4,9841%	0,5482%	5,4089%	0,5511%	5,9648%	0,5484%	5,4535%	0,5447%	4,7434%
-1	0,7091%	0,7445%	4,9841%	0,7472%	5,3714%	0,7426%	4,7241%	0,7456%	5,1442%	0,7454%	5,1082%
0	79,6419%	83,6113%	4,9841%	83,6249%	5,0011%	83,6241%	5,0001%	83,6211%	4,9964%	83,6201%	4,9951%
1	1,1349%	1,1914%	4,9841%	1,1880%	4,6804%	1,1934%	5,1550%	1,1886%	4,7376%	1,1904%	4,8882%
2	1,5440%	1,6209%	4,9841%	1,6222%	5,0644%	1,6146%	4,5762%	1,6194%	4,8826%	1,6246%	5,2189%
3	0,4547%	0,4773%	4,9841%	0,4769%	4,8942%	0,4796%	5,4927%	0,4762%	4,7381%	0,4745%	4,3699%
4	1,6386%	1,7203%	4,9841%	1,7145%	4,6293%	1,7146%	4,6349%	1,7252%	5,2796%	1,7279%	5,4493%
5	1,3135%	1,3790%	4,9841%	1,3835%	5,3232%	1,3911%	5,9022%	1,3864%	5,5490%	1,3676%	4,1161%
6	1,0911%	1,1455%	4,9841%	1,1425%	4,7123%	1,1438%	4,8271%	1,1493%	5,3334%	1,1464%	5,0624%
7	0,8701%	0,9134%	4,9841%	0,9205%	5,7943%	0,8991%	3,3384%	0,9130%	4,9328%	0,9191%	5,6300%
8	0,7602%	0,7981%	4,9841%	0,7936%	4,3900%	0,7936%	4,3994%	0,7854%	3,3139%	0,7876%	3,6104%
16	0,4926%	0,5172%	4,9841%	0,5165%	4,8434%	0,5197%	5,5059%	0,5093%	3,3924%	0,5115%	3,8354%
21	0,4331%	0,4547%	4,9841%	0,4536%	4,7249%	0,4596%	6,1314%	0,4567%	5,4411%	0,4508%	4,1003%
22	0,3305%	0,3470%	4,9841%	0,3459%	4,6757%	0,3429%	3,7446%	0,3484%	5,4312%	0,3459%	4,6740%
23	0,3040%	0,3191%	4,9841%	0,3191%	4,9949%	0,3130%	2,9883%	0,3216%	5,7954%	0,3235%	6,4329%
24	0,2615%	0,2746%	4,9841%	0,2756%	5,3743%	0,2739%	4,7177%	0,2708%	3,5404%	0,2755%	5,3463%
32	0,2706%	0,2841%	4,9841%	0,2829%	4,5411%	0,2863%	5,8106%	0,2807%	3,7380%	0,2833%	4,6974%
64	0,2861%	0,3004%	4,9841%	0,3015%	5,3605%	0,3030%	5,9014%	0,2973%	3,9156%	0,2986%	4,3740%
		Moyenne	4,9841%		5,7943%		6,1314%		5,7954%		6,4329%
		Maximum	4,9841%		4,8880%		4,9397%		4,8163%		4,8871%

Tableau-A IV-2 Signatures logiques du circuit c432 et leurs erreurs relatives en fonction du nombre de vecteurs utilisés lors de l'injection

Erreur	Parfait	Filtré (65536 vect.)		Filtré (50000 vect.)		Filtré (40000 vect.)		Filtré (30000 vect.)		Filtré (20000 vect.)	
	P(Erreur)	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.	P(Erreur)	Erreur r.
0	79,6419%	83,6392%	5,0192%	83,6645%	5,0509%	83,6579%	5,0426%	83,6514%	5,0344%	83,6519%	5,0350%
1	1,5460%	1,6236%	5,0192%	1,6226%	4,9569%	1,6229%	4,9716%	1,6219%	4,9072%	1,6229%	4,9709%
10	1,4862%	1,5608%	5,0192%	1,5649%	5,2935%	1,5557%	4,6749%	1,5613%	5,0476%	1,5647%	5,2798%
11	0,2656%	0,2790%	5,0192%	0,2762%	3,9647%	0,2793%	5,1527%	0,2786%	4,8709%	0,2791%	5,0629%
100	1,6386%	1,7209%	5,0192%	1,7153%	4,6789%	1,7153%	4,6772%	1,7258%	5,3177%	1,7286%	5,4894%
101	0,4892%	0,5138%	5,0192%	0,5142%	5,1107%	0,5166%	5,5967%	0,5135%	4,9699%	0,5133%	4,9117%
110	0,8709%	0,9146%	5,0192%	0,9157%	5,1453%	0,9159%	5,1739%	0,9158%	5,1577%	0,9155%	5,1220%
111	0,6378%	0,6698%	5,0192%	0,6690%	4,8942%	0,6691%	4,9106%	0,6701%	5,0726%	0,6725%	5,4518%
1000	1,1058%	1,1613%	5,0192%	1,1591%	4,8165%	1,1624%	5,1145%	1,1584%	4,7518%	1,1624%	5,1168%
1010	0,4652%	0,4885%	5,0192%	0,4862%	4,5312%	0,4911%	5,5781%	0,4882%	4,9491%	0,4874%	4,7780%
1011	0,2859%	0,3002%	5,0192%	0,3008%	5,2173%	0,3033%	6,0836%	0,3008%	5,2179%	0,2968%	3,8201%
10000	1,2491%	1,3118%	5,0192%	1,3067%	4,6144%	1,3164%	5,3895%	1,3078%	4,6964%	1,3122%	5,0518%
11000	0,5331%	0,5599%	5,0192%	0,5597%	4,9779%	0,5560%	4,2941%	0,5495%	3,0672%	0,5492%	3,0154%
11001	0,5721%	0,6008%	5,0192%	0,6004%	4,9525%	0,5915%	3,3949%	0,5989%	4,6932%	0,6040%	5,5875%
11010	0,6231%	0,6544%	5,0192%	0,6511%	4,4890%	0,6555%	5,1995%	0,6574%	5,5063%	0,6553%	5,1722%
11011	0,6578%	0,6909%	5,0192%	0,6935%	5,4175%	0,6867%	4,3848%	0,6934%	5,4072%	0,6763%	2,8016%
100000	0,8778%	0,9219%	5,0192%	0,9222%	5,0588%	0,9240%	5,2609%	0,9164%	4,3964%	0,9215%	4,9750%
101000	0,2666%	0,2800%	5,0192%	0,2777%	4,1622%	0,2786%	4,5081%	0,2755%	3,3398%	0,2811%	5,4332%
101001	0,2639%	0,2771%	5,0192%	0,2770%	4,9535%	0,2722%	3,1449%	0,2795%	5,8965%	0,2814%	6,6272%
101010	0,2623%	0,2755%	5,0192%	0,2728%	4,0248%	0,2726%	3,9213%	0,2786%	6,2107%	0,2749%	4,8221%
101011	0,2853%	0,2996%	5,0192%	0,2997%	5,0454%	0,2989%	4,7500%	0,3005%	5,3229%	0,2976%	4,2973%
111010	0,2412%	0,2533%	5,0192%	0,2515%	4,2692%	0,2519%	4,4334%	0,2561%	6,1797%	0,2521%	4,5234%
111011	0,2597%	0,2728%	5,0192%	0,2724%	4,8736%	0,2741%	5,5456%	0,2737%	5,3674%	0,2711%	4,3697%
1000000	0,6955%	0,7304%	5,0192%	0,7267%	4,4942%	0,7321%	5,2641%	0,7271%	4,5452%	0,7283%	4,7216%
		Moyenne	5,0192%		5,4175%		6,0836%		6,2107%		6,6272%
		Maximum	5,0192%		4,7914%		4,8528%		4,9969%		4,8515%

BIBLIOGRAPHIE

Actel. Decembre 2002. « Effects of Neutrons in Programmable Logic ». < http://www.actel.com/documents/SERWP.pdf >. Consulté le 13 janvier 2013.

Agrawal, V.D. 2000. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer.

Alexandrescu, D., L. Anghel et M. Nicolaidis. 2004. « Simulating single event transients in VDSM ICs for ground level radiation ». *Journal of Electronic Testing-Theory and Applications,* vol. 20, n° 4, p. 413-421.

Allkofer, O.C., et P.K.F. Grieder. 1984. *Cosmic rays on earth*. Fachinformationszentrum Energie, Physik, Mathematik.

Arlat, J., Y. Crouzet et J. C. Laprie. 1989. « Fault injection for dependability validation of fault-tolerant computing systems ». In *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*. (21-23 juin 1989), p. 348-355.

Baraza, J. C., J. Gracia, D. Gil et P. J. Gil. 2000. « A prototype of a VHDL-based fault injection tool ». In *Defect and Fault Tolerance in VLSI Systems, 2000. Proceedings. IEEE International Symposium on*. (2000), p. 396-404.

Baumann, R. C. 2005. « Radiation-induced soft errors in advanced semiconductor technologies ». *Ieee Transactions on Device and Materials Reliability,* vol. 5, n° 3, p. 305-316.

Bolchini, Cristiana, et Chiara Sandionigi. 2010. « Fault Classification for SRAM-Based FPGAs in the Space Environment for Fault Mitigation ». *IEEE Embedded Systems Letters,* vol. 2, n° 4, p. 107-110.

Bosio, A., et G. Di Natale. 2008. « LIFTING: A Flexible Open-Source Fault Simulator ». In *ATS '08. 17th*. (24-27 Nov. 2008), p. 35-40.

Carreira, J., H. Madeira et J. G. Silva. 1998. « Xception: A technique for the experimental evaluation of dependability in modern computers ». *Ieee Transactions on Software Engineering,* vol. 24, n° 2, p. 125-136.

Civera, P., L. Macchiarulo, M. Rebaudengo, M. S. Reorda et M. Violante. 2001. « FPGA-based fault injection for microprocessor systems ». In *Test Symposium, 2001. Proceedings. 10th Asian*. (2001), p. 304-309.

Hazucha, P., et C. Svensson. 2000. « Impact of CMOS technology scaling on the atmospheric neutron soft error rate ». *Ieee Transactions on Nuclear Science,* vol. 47, n° 6, p. 2586-2594.

Hungse, Cha, E. M. Rudnick, J. H. Patel, R. K. Iyer et G. S. Choi. 1996. « A gate-level simulation environment for alpha-particle-induced transient faults ». *IEEE Transactions on Computers,* vol. 45, n° 11, p. 1248-1256.

JEDEC. 2006. *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*. < http://www.jedec.org/sites/default/files/docs/jesd89a.pdf >. Consulté le 14 avril 2013.

Jenn, E., J. Arlat, M. Rimen, J. Ohlsson et J. Karlsson. 1994. « Fault injection into VHDL models: the MEFISTO tool ». In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*. (15-17 June 1994), p. 66-75.

John R. Samson, Jr., Wilfrido A. Moreno et Fernando J. Falquez. 1997. « Validating fault tolerant designs using laser fault injection (LFI) ». In *1997 Workshop on Defect and Fault-Tolerance in VLSI Systems*. p. 175-185. 737521: IEEE Computer Society.

Kanawati, G. A., N. A. Kanawati et J. A. Abraham. 1995. « Ferrari - a Flexible Software-Based Fault and Error Injection System ». *Ieee Transactions on Computers,* vol. 44, n° 2, p. 248-260.

Karlsson, Johan, Ulf Gunneflo, Peter Liden et Jan Torin. 1991. « Two Fault Injection Techniques for Test of Fault Handling Mechanisms ». In *Proceedings of the IEEE International Test Conference on Test: Faster, Better, Sooner*. p. 140-149. 761071: IEEE Computer Society.

Le, Robert. 2012. « Soft Error Mitigation Using Prioritized Essential Bits ». In *XAPP538*. < http://www.xilinx.com/support/documentation/application_notes/xapp538-soft-error-mitigation-essential-bits.pdf >. Consulté le 5 février 2013.

Lesea, A., S. Drimer, J. J. Fabula, C. Carmichael et P. Alfke. 2005. « The Rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs ». *Ieee Transactions on Device and Materials Reliability,* vol. 5, n° 3, p. 317-328.

Madeira, Henrique, Mario Zenha Rela, Francisco Moreira et Joao Gabriel Silva. 1994. « RIFLE: A General Purpose Pin-level Fault Injector ». In *Proceedings of the First European Dependable Computing Conference on Dependable Computing*. p. 199-216. 649779: Springer-Verlag.

Michel, T., R. Leveugle, G. Saucier, R. Doucet et P. Chapier. 1994. « Taking advantage of ASICs to improve dependability with very low overheads [PLC] ». In *European*

Design and Test Conference, 1994. EDAC, The European Conference on Design Automation. ETC European Test Conference. EUROASIC, The European Event in ASIC Design, Proceedings. (28 Feb-3 Mar 1994), p. 14-18.

Mukherjee, S. 2008. *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers/Elsevier.

Normand, E. 1996. « Single-event effects in avionics ». *IEEE Transactions on Nuclear Science,* vol. 43, n° 2, p. 461-474.

Olsen, J., P. E. Becher, P. B. Fynbo, P. Raaby et J. Schultz. 1993. « Neutron-induced single event upsets in static RAMS observed a 10 km flight attitude ». *IEEE Transactions on Nuclear Science,* vol. 40, n° 2, p. 74-77.

Robache, R., J.-F. Boland, Y. Savaria et C. Thibeault. 2013. « A Methodology for System-level Fault Injection Based on Gate-level Faulty Behavior ». In *NEWCAS*. (Paris, 16-19 juin 2013).

Santarini, M. 2005. « Cosmic radiation comes to ASIC and SOC design ». *Edn,* vol. 50, n° 10.

Svenningsson, R., J. Vinter, H. Eriksson et M. Törngren. 2010. « MODIFI: A model-implemented fault injection tool ». In *Proceedings of the 29th SAFECOMP 2010*. (September).

Taber, A., et E. Normand. 1993. « Single Event Upset in Avionics ». *IEEE Transactions on Nuclear Science,* vol. 40, n° 2, p. 120-126.

Vanhauwaert, P. 2008. « Analyse de sûreté par injection de fautes dans un environnement de prototypage à base de FPGA ». Institut National Polytechnique de Grenoble - INPG.

Vargas, F., D. L. Cavalcante, E. Gatti, D. Prestes et D. Lupi. 2005. « On the Proposition of an EMI-Based Fault Injection Approach ». In *Proceedings of the 11th IEEE International On-Line Testing Symposium*. p. 207-208. 1084339: IEEE Computer Society.

Volkmar, Sieh, Tschche Oliver et Balbach Frank. 1997. « Comparing Different Fault Models Using VERIFY ». In *Proc. 6th Conf. on Dependable Computing for critical Applications*. (Grainau, Germany, Mars), p. 59-76.

Ziade, H., R. Ayoubi et R. Velazco. 2004. « A survey on fault injection techniques ». *International Arab Journal of Information Technology,* vol. Vol. 1, No. 2, July, p. 171-186.