Analyse statique de l'effet des erreurs de configuration dans des FGPA configurés par SRAM et amélioration de robustesse

Jean-Baptiste Ferron

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Nanoélectronique et Nanotechnologies**

Arrêté ministériel : 7 août 2006

Présentée par

Jean-Baptiste Ferron

Thèse dirigée par **Régis Leveugle** et
codirigée par **Lorena Anghel**

préparée au sein du **Laboratoire TIMA**
dans **l'École Doctorale EEATS**

Analyse statique de l'effet des erreurs de configuration dans des FPGA configurés par SRAM et amélioration de robustesse

Thèse soutenue publiquement le **« 26 mars 2012 »**,
devant le jury composé de :

M., Frédéric, Rousseau
Professeur, Université de Grenoble, Grenoble-INP, Président
M., Habib, Mehrez
Professeur, Université Pierre et Marie Curie, UPMC, Paris 6, Rapporteur
M., Fabrice, Monteiro
Professeur, Université de Lorraine, Rapporteur
M. Régis Leveugle
Professeur, Université de Grenoble, Grenoble-INP, Directeur de thèse
Mme. Lorena Anghel
Professeur, Université de Grenoble, Grenoble-INP, Codirecteur de thèse
M., Antonin, Bougerol
Docteur, EADS France, Examinateur

A mon père, André Ferron, qui ne me quittera jamais.

Remerciements

Ces travaux de thèse ont été réalisés grâce aux financements mis à ma disposition par le ministère de la recherche et par l'association pour le développement et la recherche de Grenoble. Ils se sont déroulés au sein du laboratoire TIMA de Grenoble, dont je tiens en particulier à remercier la directrice, Dominique Borrione (qui m'avait auparavant initié, avec Stéphane Mancini, aux joies du VHDL), ainsi que tout le personnel. Je pense notamment à Corrine Durand-Viel, Anne-Laure Fourneret-Itié, Julie Correard, Sophie Martineau, Lucie Torella et Laurence Ben Tito.

Je veux remercier ici les membres de mon jury de soutenance, en commençant par Frédéric Rousseau, en sa qualité de président. J'en profite pour remercier le regretté Paul Amblard qui travaillait avec Frédéric, qui fut un professeur fascinant en master recherche, quelqu'un qui créait des vocations. Je tiens également à remercier Habib Mehrez et Fabrice Monteiro, qui ont accepté de participer à ce jury en tant que rapporteurs. Un grand merci à Antonin Bougerol, qui fût mon examinateur lors de la soutenance.

Je tiens aussi à adresser mes plus sincères remerciements à mes directeurs de thèse, Régis Leveugle et Lorena Anghel, pour leur attention, leur énergie, et leur patience. Je n'y serais jamais arrivé sans eux.

Pour leur contribution à ces travaux, je tiens également à remercier chaleureusement Gilles Foucard (qui travailla sous la direction de Raoul Velazco), Gaëtan Canivet, Vincent Maingot et Alexandre Bocquillon.

J'ai également une pensée pour Alexandre Chagoya et Bernard Bosc, les maîtres du CIME, qui m'ont plus d'une fois donné le petit coup de pouce nécessaire à l'avancement des travaux. J'en profite pour remercier Katell Morin Allory, Jérémie Hamon, et Sophie Dumont, avec qui j'ai eu la chance d'animer certains TDs et projets pour les jeunes pousses de l'INPG.

Je veux également dire merci à tous ceux qui ont partagé ces quatre années au laboratoire (et tellement plus), Michele Portolan, Pierre Vanhauwaert, Paolo Maistri (la bise à Anna et aux bambini), Salma et Mohamed Ben Jrad (dire qu'on a vécu la révolution ensemble), Gilles Bizot, Hai Yu, Adrien Prost Boucle, Etienne Rudolff, Fabien Chaix, Vladimir Petru Pasca.

Je souhaite également remercier mes anciens colocataires, Jonah Luther et Matthieu Quinsat, et les membres de mon cercle grenoblois (pour la plupart anciens de l'ENSERG), Thomas Champseix (le T de JBTsoft, l'ancêtre de l'outil SEFEA-ProD), Daweon Suh, Emilie Ponthus, Nicolas "Roger" Royer, Alban Forichon, Julien Bobin, Robin Delcros, Bérenger Roelens, Younès Igrouche, Christophe Cafiero et Xavier Roche.

Mille mercis à Brice Parisot, mon ami depuis 20 ans, sa compagne Audrey Guerrazzi, qui m'ont supporté bon nombre de week-end ces dernières années et Alcibiade Lichterovicz, un ami de longue date.

Enfin, je veux remercier ma famille: ma mère (Jeannine Duclos Ferron), mes sœurs (Clémentine et Sandrine) et ma compagne depuis 10 ans, Marion Blandin, qui m'ont soutenu (supporté?) tout au long de ces années.

Table des matières

Index des figures

Index des tableaux

Introduction

Les FPGAs configurés par mémoire SRAM présentent de nombreux avantages dans le monde de la micro électronique d'aujourd'hui. D'un point de vue technique, leurs possibilités de reconfiguration en ligne et/ou de reconfiguration partielle peuvent être exploitées par les concepteurs. Ces composants sont également moins coûteux que leurs homologues configurables (fusibles, FLASH). C'est aussi le cas si on les compare aux circuits ASIC dans le cas des petites et moyennes séries. Ils offrent également de nouvelles possibilités pour le prototypage. En conséquence, leur utilisation est de plus en plus importante pour différents types d'applications, y compris celles demandant un haut niveau de robustesse (applications aéronautiques, militaires, etc...).

Ces circuits doivent donc être capables d'opérer dans des conditions de travail difficiles. Ceci inclut des environnements hostiles, comme des radiations ioniques si le circuit est dans l'espace, mais aussi des impacts de protons, de neutrons et d'autres particules dans l'atmosphère terrestre, jusqu'au niveau de la mer.

Les effets singuliers (Single Event Effects en anglais, ou SEEs) induits par les interactions de ces particules avec les circuits intégrés constituent une menace importante et identifiée pour l'ensemble des systèmes électroniques. Les circuits contenant une quantité importante de mémoire SRAM sont les premiers concernés, y compris au niveau de la mer, avec l'intégration croissante des nouvelles technologies. L'effet le plus courant à ce niveau est l'effet singulier d'inversion (Single Event Upset en anglais, ou SEU) [ALD05] qui consiste en l'inversion d'un point mémoire embarqué suite à un effet transitoire de type SEE. Les réseaux programmables configurés par SRAM entrent naturellement dans la catégorie des circuits très vulnérables. Les SEUs affectent à la fois les éléments mémoires utilisateurs embarqués (blocs de mémoire RAM embarqués et bascules utilisateurs) et la mémoire de configuration SRAM du FPGA. Les fautes dans la mémoire de configuration peuvent modifier de manière définitive (au moins jusqu'à la prochaine configuration [MOR05]) la définition de l'application réalisée par le circuit, induisant de forts risques de disfonctionnement. La détection et la correction des erreurs entraine généralement de forts coûts en termes de matériel, de performance, et de temps, et n'est dans certains cas pas directement réalisable, notamment si la probabilité d'occurrence d'erreurs multiples est non négligeable. Ceci inclut les phénomènes conduisant à des inversions de bits multiples (Multi Bit Upset en anglais, ou MBUs), ou à des inversions de bits multiples intercellulaires (Multi Cell Upset en anglais, ou MCUs).

Afin de pouvoir protéger efficacement un circuit implanté sur FPGA SRAM contre ces perturbations naturelles, il est fondamental de bien comprendre le type de fautes pouvant survenir et leurs conséquences au niveau fonctionnel. Sur la base de la modélisation de fautes obtenue, il est ensuite possible de proposer des techniques de protection adaptées, cherchant à atteindre de bons compromis entre surcoûts d'implantation et couverture de fautes (donc moins coûteuses que l'approche

classiquement proposée, à savoir la redondance modulaire triple, ou TMR). C'est dans cet esprit que s'inscrivent les travaux de cette thèse.

Les objectifs de ces travaux concernent en premier lieu la modélisation des fautes, et plus particulièrement l'analyse de la sensibilité des différents types de bits de configuration. Cette étude concerne les perturbations naturelles citées précédemment, mais aussi l'effet de perturbations volontaires malicieuses, dans le but d'attaquer un circuit sécurisé. La famille de FPGA Virtex II de Xilinx est utilisée comme premier cas pratique d'expérimentation, puis une comparaison est réalisée avec la famille AT40K de chez ATMEL. Ce travail sur la modélisation a permis de mieux comprendre l'impact réel de différentes sources de perturbations, et les motifs d'erreurs devant réellement être pris en compte pour améliorer la robustesse d'un circuit implanté sur ce type de technologie. Cette étude a nécessité le développement d'outils de conception spécifiques, permettant d'automatiser les analyses, compte tenu de la grande quantité de données à traiter.

Cette étape a également permis de développer une méthodologie innovante d'évaluation de la sensibilité de la mémoire de configuration aux SEUs. Une classification des bits de configuration en fonction des effets produits par leur inversion sur le fonctionnement normal de l'application est établie. Ceci permet de déterminer les zones les plus critiques, autorisant le développement de stratégies de protection sélectives de la mémoire de configuration.

Après un rapide historique sur les réseaux programmables et leurs technologies, le chapitre 1 présente une description de l'architecture type des FPGA SRAM et de la structure de leur mémoire de configuration. Les réseaux ciblés par cette étude sont ensuite plus précisément décrits en insistant sur les spécificités de leurs architectures et de leurs mémoires de configuration.

Le chapitre 2 traite de l'état de l'art. Il présente les différentes techniques d'analyse de criticité des réseaux programmables à mémoire SRAM, puis un aperçu des méthodologies de protection contre les fautes de configuration présentées dans la littérature à ce jour.

Le chapitre 3 présente l'étude réalisée sur les fautes de configuration pour les FPGAs Virtex II. Cette analyse a permis d'identifier le rôle des bits de configuration de ces circuits, et la structure de leur mémoire. Des méthodologies de caractérisation à la fois de la répartition des fautes et de leur effet sont ensuite présentées, et utilisées pour l'analyse de résultats de campagnes d'injection de fautes par laser. Enfin des travaux sur la caractérisation de la sensibilité des différentes zones de la mémoire en fonction de l'application sont expliqués et des résultats pratiques sont présentés.

Le chapitre 4 présente une analyse similaire pour les FPGAs AT40K.

Le chapitre 5 décrit enfin une nouvelle technique de durcissement des applications sur ce type de FPGAs, basée sur l'analyse des zones sensibles de la mémoire de configuration et la fiabilisation pragmatique et locale des parties les plus critiques.

1. Présentation des réseaux programmables SRAM

L'objectif de ce premier chapitre est d'introduire les différentes familles de réseaux programmables pour ensuite se focaliser sur les technologies SRAM, et plus particulièrement les FPGAs cibles de l'étude, le Xilinx Virtex II, et l'ATMEL AT40K.

1.1. Généralités sur les réseaux programmables

1.1.1. Rapide panorama des différents types de réseaux.

1.1.1.1. Rappel historique

L'apparition des composants à mémoire programmable date de la fin des années 50, sous la forme des mémoires PROMs (*Programmable Read Only Memory*). Ces mémoires furent développées pour répondre aux besoins de l'armée américaine, et étaient configurées de manière définitive en fabrication, à l'aide de masques. Elles ne permettaient cependant pas le développement de circuits logiques (elle ne peuvent être écrites qu'une fois) et furent remplacées par les EPROMs (*Erasable* PROM ou UV-PROM) et les EEPROMs (*Electrical Erasable* PROM) qui permettaient une reprogrammation, après effacement du contenu respectivement par ultraviolet, ou électriquement.

Le développement des mémoires permit à IBM de lancer le premier composant programmable par masques dans les années 70 (le TMS2000), créant ainsi la famille des PLA (*Programmable Logic Array*). Pour combler les lacunes en termes de performance et de coût de ces circuits, MMI lança à la fin de la même décennie les PAL (*Programmable Array Logic*) qui se composaient d'un plan de portes logiques « ET » programmables connectées à des portes logiques « OU » (Figure 1).

Lattice créa ensuite les circuits GALs (*Generic Array Logic*), qui, en plus d'intégrer des cellules « ET » en technologie EEPROM, offraient la possibilité d'utiliser des registres. Cependant, il fallut attendre la création des CPLD (*Complex Programmable Logic Devices*) pour pouvoir implémenter des circuits logiques plus évolués. Ces plateformes étaient constituées de plusieurs PAL reliés par des interconnexions programmables et pouvaient implémenter des architectures demandant jusque-là plusieurs centaines de milliers de portes logiques.

Finalement, Actel lança en 1985 une nouvelle génération de circuits, les FPGA (*Field Programmable Gate Array*). Il s'agissait de réseaux de portes connectées de manière programmables par des anti-fusibles. A la même époque, Xilinx proposa des circuits basés sur des cellules élémentaires plus complexes, et configurés par chargement d'une mémoire SRAM (Static Random Access Memory), donc reprogrammables. Ces circuits se démocratisèrent dans les années 90 et sont aujourd'hui utilisés dans de nombreux domaines, tout d'abord dans les télécommunications et les réseaux, puis pour des applications de plus en plus critiques, comme l'automobile, l'aéronautique, la sécurité, et le spatial. La Figure 2 présente une architecture générique de ce type de circuit, constituée de blocs d'entrée/sortie,

de blocs logiques implémentant la logique combinatoire et séquentielle de l'utilisateur, et d'un réseau d'interconnexion.

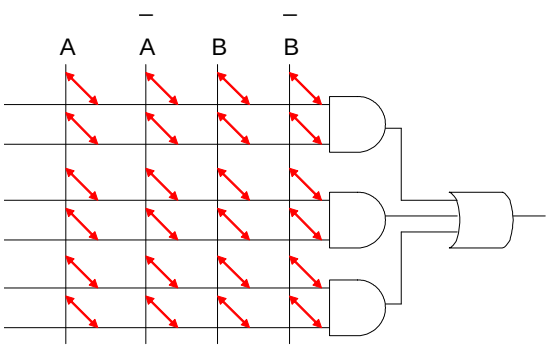

Figure 1: Architecture PAL.

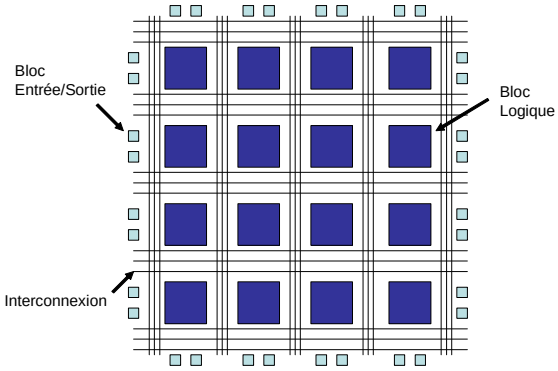

Figure 2: Architecture type d'un circuit FPGA.

Il existe aujourd'hui de nombreuses familles de FPGAs, qui se déclinent en fonction des critères suivants: le nombre de cellules équivalentes, les niveaux de tensions admissibles, les éléments additionnels (convertisseurs, mémoires, mais aussi DSPs (Digital Signal Processor), cœurs de processeurs, etc…), et finalement leur type de programmation. On trouve aujourd'hui trois grandes catégories de circuits sur le marché : les anti-fusibles, les technologies à mémoire FLASH, ainsi que les réseaux à mémoire SRAM.

1.1.1.2. Les réseaux configurables par anti-fusibles

Les réseaux programmables à anti-fusibles sont des composants configurables une seule fois.

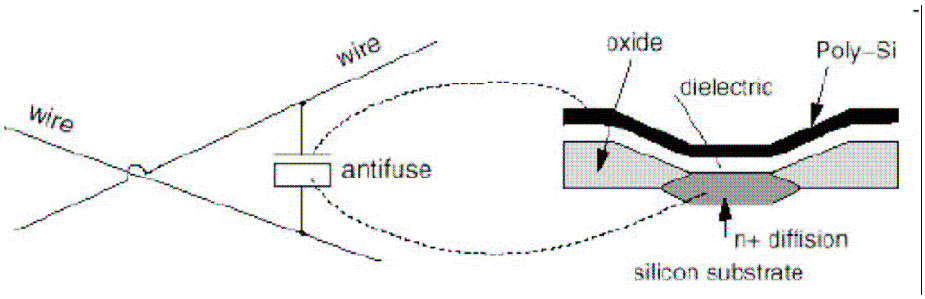

Figure 3: Programmation par anti-fusibles [CAN09].

Les anti-fusibles sont des composants généralement constitués de deux couches conductrices entourant un diélectrique. L'application d'une tension élevée fait claquer le diélectrique créant une liaison entre les deux couches conductrices du composant. Les anti-fusibles peuvent donc être considérés comme des circuits ouverts dans leur état initial, et des circuits fermés une fois programmés. La Figure 3 montre un anti-fusible utilisé pour configurer une connexion entre deux fils. L'avantage de cette technologie est sa faible taille, même si les transistors utilisés pour la configuration sont dimensionnés de manière à laisser passer de forts courants lors de la programmation et réaliser une bonne isolation entre les tensions élevées de configuration et les faibles tensions de fonctionnement, et sont de fait de taille imposante. Les autres points forts de ces composants sont leur haut niveau de confidentialité, due

aux difficultés de relecture de l'état du fusible, et leur faible sensibilité aux radiations et autres perturbations environnementales, induisant une bonne fiabilité. Ces caractéristiques impliquent cependant un manque de flexibilité, ainsi que des fortes tensions de programmation.

1.1.1.3. Les technologies à mémoire FLASH

Les réseaux programmables à mémoire FLASH ont remplacé progressivement les réseaux programmables par mémoires EPROM ou EEPROM. L'avantage des mémoires FLASH est de permettre une reconfiguration en ligne des fonctions, sans perte de configuration lors des coupures d'alimentation, et avec de meilleures caractéristiques que pour les technologies EEPROM. Ces réseaux sont peu sensibles aux particules ionisantes, et ils bénéficient en outre de temps de configuration relativement courts. Les réseaux à mémoire FLASH sont cependant plus coûteux à fabriquer que leurs homologues à mémoire SRAM et ils nécessitent une alimentation supplémentaire pour la configuration et l'effacement de la mémoire.

Deux architectures distinctes sont généralement utilisées pour la conception des bancs mémoires de ces réseaux: les NOR-EEPROMs, et les NAND-EEPROMs. La Figure 4 présente ces deux architectures.

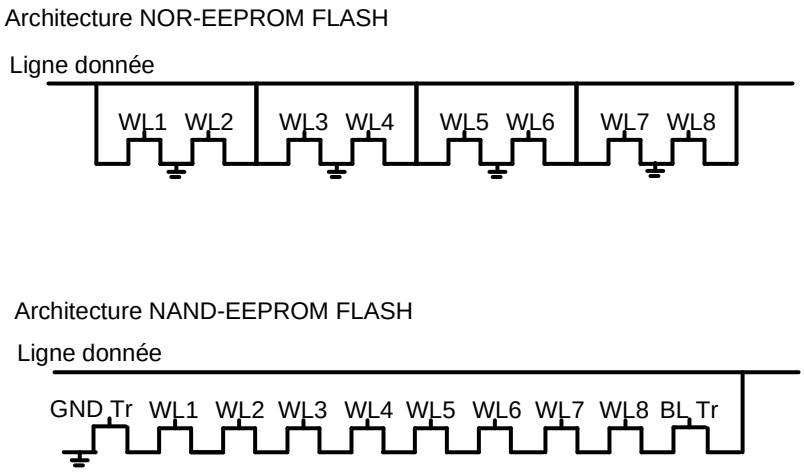

Figure 4: Architectures EEPROM FLASH.

En dépit du plus faible nombre de transistors de la cellule NOR, le nombre plus important de connexions nécessaire à l'implantation de l'architecture NOR-FLASH induit une densité d'intégration plus faible. Le temps d'accès et la taille globale du bloc mémoire rend également la technologie NAND plus attractive. Le seul inconvénient des NAND est une fiabilité moindre par rapport au bloc mémoire NOR, l'application de tensions sur toutes les grilles des transistors pendant les opérations de lecture pouvant inverser les valeurs stockées si la fréquence de lecture est très supérieure à celle des écritures.

1.1.1.4. Les réseaux programmables à mémoire SRAM

Les mémoires SRAM sont également utilisées pour la conception de réseaux programmables du fait de leurs performances (leur rapidité d'accès notamment) et de leur coût plus faible que pour les mémoires FLASH. On trouve deux grandes architectures pour ces points mémoires, les 6T-SRAM et 5T-SRAM, respectivement à six et cinq transistors. La cellule à six transistors est constituée de deux inverseurs montés tête-bêche et deux transistors d'accès (Figure 5). Elle est commandée par le bit « Word Line » '(WL, Figure 5.a) pour l'écriture et la lecture. On positionne des valeurs complémentaires sur BL et BL| tout en ayant WL à « 1 » pour écrire une donnée BL. On laisse les lignes BL et BL| flottantes et on positionne WL à « 1 » pour lire la donnée contenue par le point mémoire.

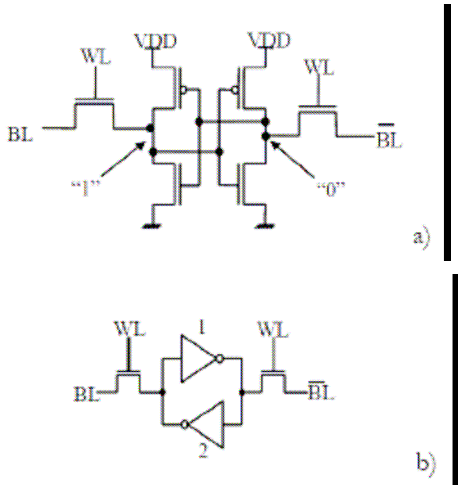

Figure 5: Cellule SRAM à six transistors [CAN09].

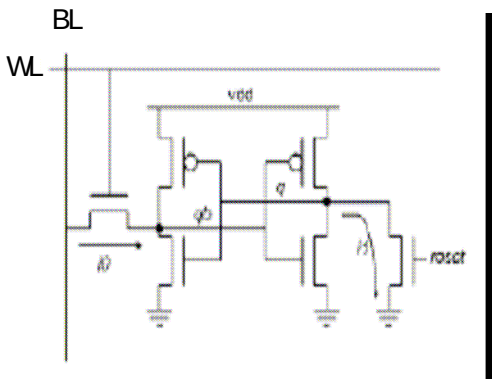

Figure 6: Cellule SRAM à cinq transistors [CAN09].

1.1.1.5. Le futur

Des travaux de recherche sont menés sur les mémoires non-volatiles afin de remplacer les mémoires FLASH et SRAM qui équipent actuellement les réseaux programmables. On peut notamment citer les travaux sur les mémoires magnétiques basées sur la charge de l'électron [ZHA07], ainsi que les travaux sur les nano-RAM (basées sur la position de nanotubes de carbone) [GUI08] ou les mémoires PC RAM (Phase-Change RAM) [POS08].

1.1.2. Organisation hiérarchique des réseaux programmables SRAM

Dans la suite du document, nous allons nous focaliser sur les réseaux configurables par SRAM.

Les réseaux programmables SRAM se composent d'un ensemble de ressources logiques contenues dans un tableau de cellules identiques, de blocs I/O, et d'interconnexions configurables. La configuration est stockée dans une mémoire SRAM qui présente l'avantage d'être relativement compacte et rapide, bien que volatile et sensible aux radiations naturelles. Dans un souci de simplicité, nous présentons en Figure 7 une vision hiérarchique de ces réseaux, en séparant les éléments

fonctionnels du circuit des éléments de configuration. On parlera respectivement de couche fonctionnelle ou opérative et d'une couche de configuration.

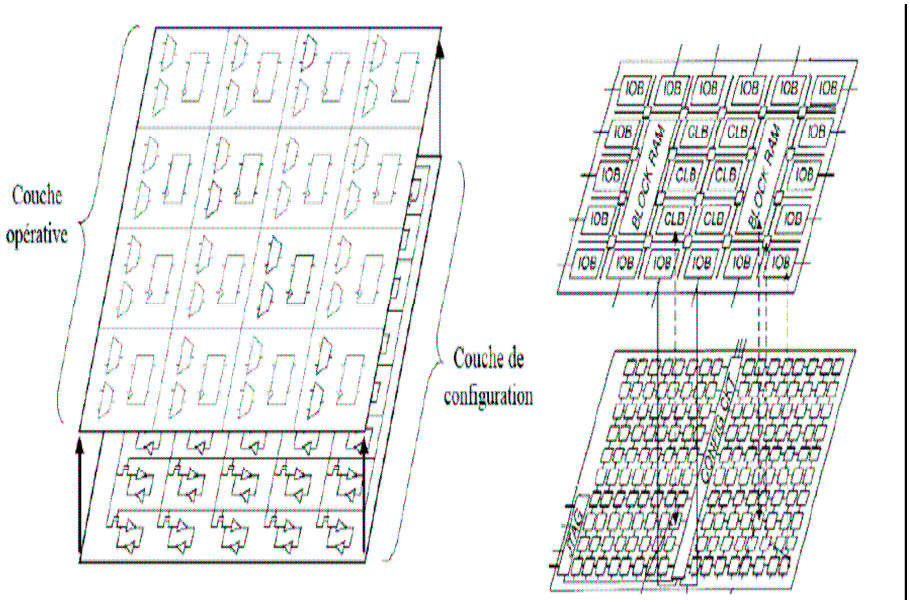

Figure 7: Vision hiérarchique d'un FPGA: couche fonctionnelle et couche de configuration [BOC09].

La partie suivante donne le détail des éléments contenus dans les couches fonctionnelles des FPGAs utilisés durant ces travaux, le Xilinx Virtex II et l'ATMEL AT40K. La partie 1.3 traitera de la couche de configuration de ces mêmes circuits.

1.2. Couche fonctionnelle, architecture des circuits ciblés

1.2.1. Xilinx Virtex II

1.2.1.1. Généralités

Les FPGA de la famille Virtex II se présentent sous la forme d'une matrice de « tuiles », dont le type, et donc la fonction, diffèrent selon leur emplacement dans le réseau (voir Figure 8). La configuration de chaque tuile est stockée dans une mémoire SRAM appelée matrice de configuration.

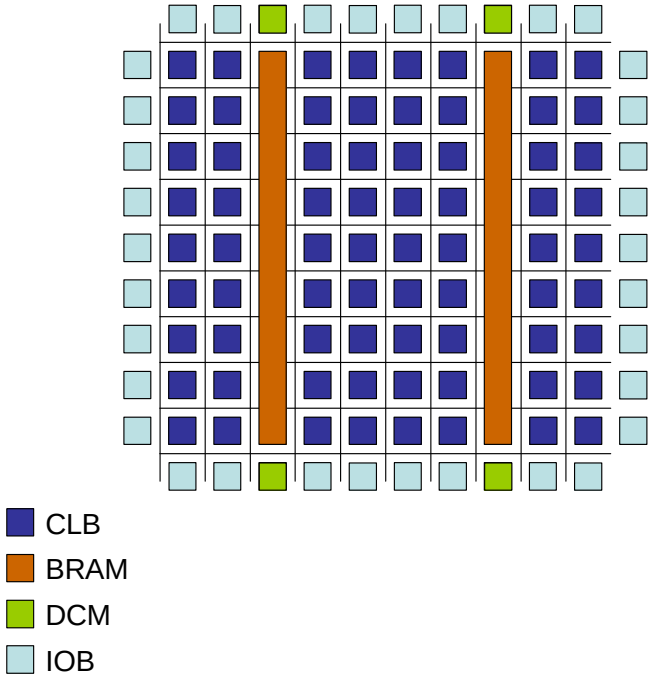

Figure 8: Matrice de CLBs

A la périphérie de la matrice se trouvent les IOB (*Input/Output Block*) et les DCM (*Digital Clock Manager*). Les IOB sont chargés de faire l'interface entre les broches du composant et les signaux internes. Les DCM sont chargés de répartir les signaux d'horloge dans le FPGA.

Au centre, on trouve les CLB (*Configurable Logic Block*) et les BRAM. Les CLB contiennent les ressources logiques combinatoires et séquentielles du FPGA, tandis que les BRAM (*Block RAM*) fournissent chacune 18Kbits de « selectRAM » et un multiplieur avec deux entrées de 18 bits.

La structure de ces composants ne varie pas selon le modèle dans la même famille, c'est la taille du tableau de CLB et donc le nombre de ressources disponibles qui change.

Les IOBs, les DCMs, les BRAMs, et les CLBs utilisent tous le même schéma d'interconnexion : à chaque bloc correspond une matrice d'interconnexions qui lui permet de communiquer avec l'extérieur. Chaque bloc a ainsi accès via la matrice à des ressources de connexions générales, fournissant un accès à n'importe quel autre bloc de la matrice, et à des connexions locales, qui lui permettent d'être relié aux blocs voisins. La Figure 9 montre l'organisation générale des ressources d'interconnexion des Virtex II. Des détails supplémentaires se trouvent dans la partie dédiée aux ressources de routage.

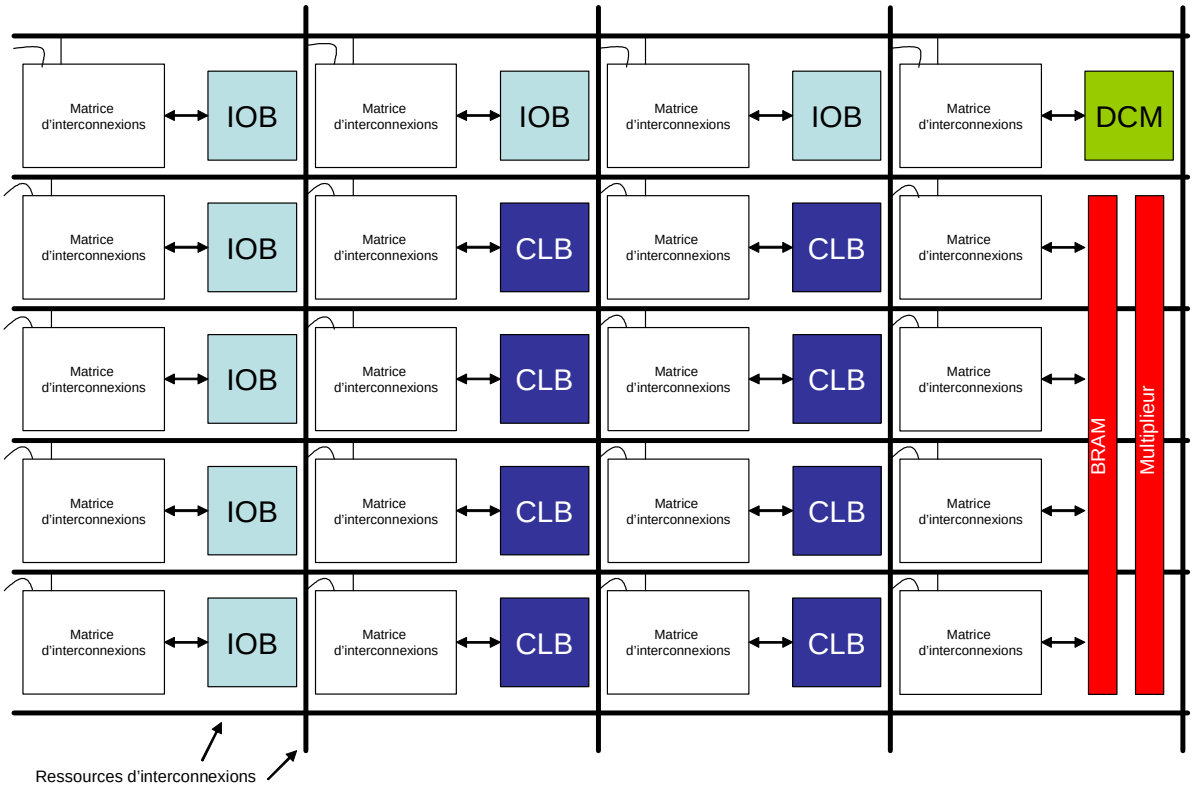

Figure 9: Organisation des éléments fonctionnels et des interconnexions dans la matrice Virtex II

1.2.1.2. Architecture des blocs programmables

Les ressources dans un CLB se répartissent principalement à l'intérieur de 4 modules élémentaires appelés « slice », auxquels s'ajoutent deux registres à 3 états, et un multiplexeur de sortie. La Figure 10 montre l'agencement de ces éléments ainsi que la matrice d'interconnexions associée à la tuile CLB. La matrice d'interconnexions ainsi que le multiplexeur de sortie sont entièrement configurables. La Figure 11 détaille les principales ressources de chaque slice : deux générateurs de fonctions 16 bits, deux registres paramétrables et des multiplexeurs.

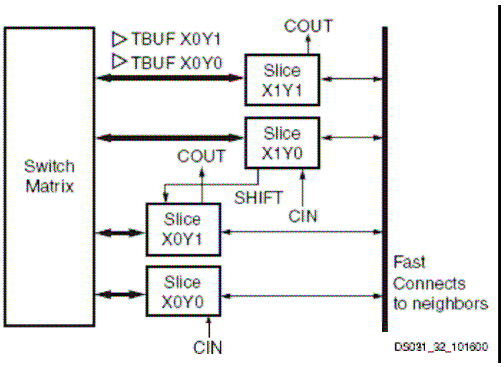

Figure 10: Vue interne de la tuile CLB et de sa matrice d'interconnexions [XIL05a].

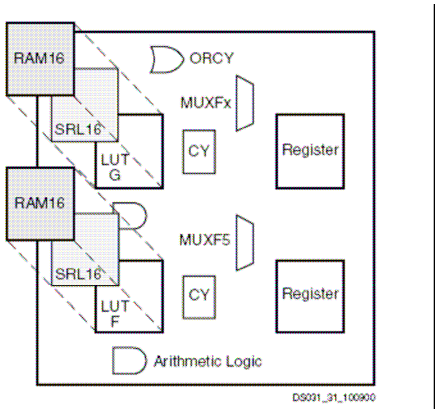

Figure 11: Architecture d'un slice [XIL05a].

1.2.1.3. Ressources de routage

Les ressources de routage des FPGAs Virtex II sont optimisées pour être rapides et prévisibles temporellement. La technologie utilise une matrice d'interconnexions configurable dont la programmation est aussi stockée dans la matrice de configuration. Elle se trouve aux croisements des bus interconnectables (voir Figure 9) et est aussi reliée aux entrées de la tuile concernée (les sorties de la tuile sont gérées par un multiplexeur). A chaque point d'intersection se trouve un PIP (*Programmable Interconnect Point*). Lorsqu'un (ou plusieurs) bit(s) de routage de la matrice de configuration est mis à 1, le PIP correspondant connecte les deux fils qu'il contrôle (voir Figure 12). Il est important de préciser que toutes les connexions ne sont pas possibles.

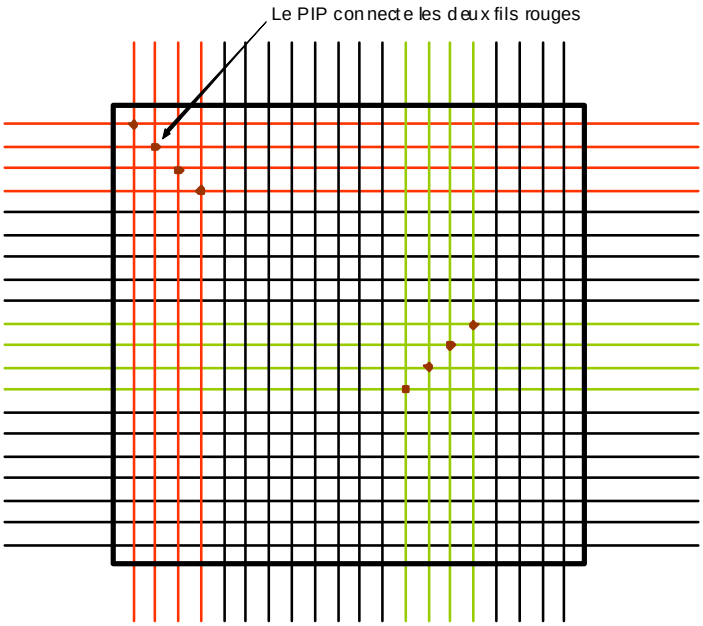

Figure 12: Matrice d'interconnexions programmables

Les ressources de routage connectées à la matrice d'interconnexions sont segmentées en plusieurs types, et partagées entre deux grands groupes :

- Les ressources de routage globales
- Les ressources de routage locales

La Figure 13 présente un récapitulatif des différents types de ressources accessibles par une tuile.

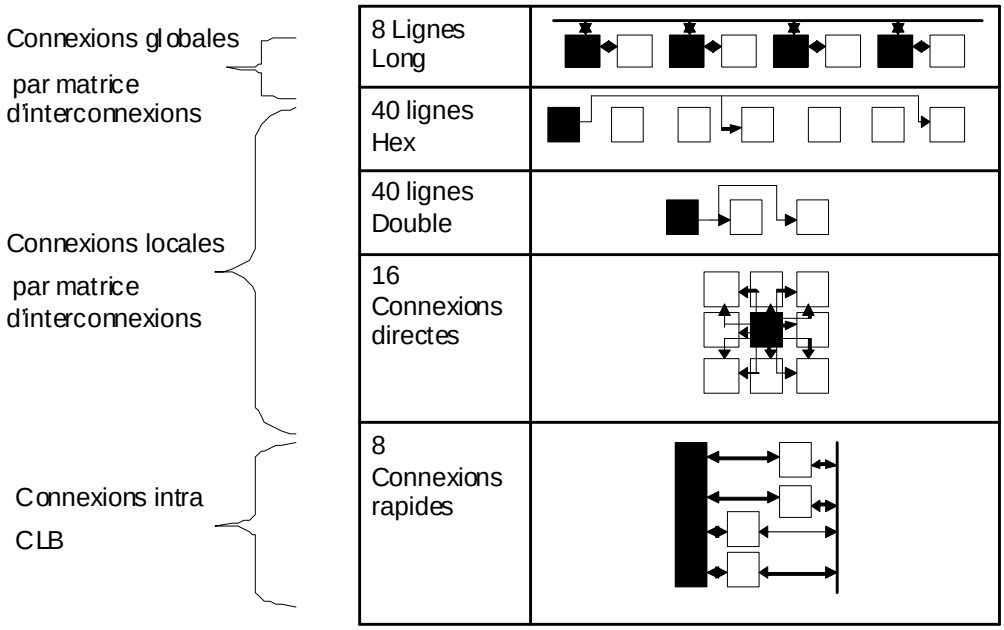

Figure 13: Types de connexions du Virtex II [XIL05a].

1.2.1.4. Bloc SelectRAM

Les blocs de mémoire « SelectRAM » sont des blocs de 18 Kbits en technologie double port. Ces blocs sont programmables avec des profondeurs et des largeurs de configuration différentes, allant de 16K x 1 bit à 512K x 36 bits. Chaque port est totalement synchrone et indépendant. Afin d'obtenir un élément de stockage plus important, il est possible de cascader les blocs. Enfin, à chaque bloc de mémoire est associé un bloc de multiplieur 29 x 18 bits.

1.2.2. ATMEL AT40K

1.2.2.1. Généralités

L'architecture du réseau programmable AT40K d'ATMEL s'articule autour d'un tableau symétrique de cellules de base. Le tableau (figure 14) est continu d'un bord à l'autre du FPGA, exception faite des lignes de répétiteurs horizontales et verticales présentes toutes les quatre cellules et qui assurent la continuité des signaux le long des bus.

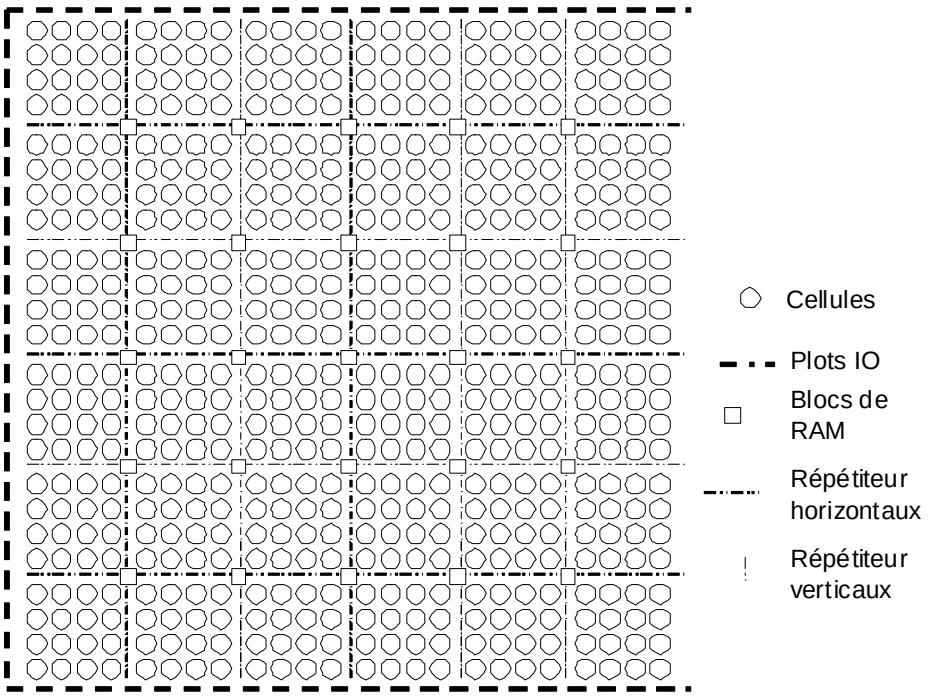

Figure 14: Tableau de cellule de l'AT40K [ATM03].

Des blocs RAM de 32 mots de quatre bits sont présents aux intersections de chaque ligne de répétiteurs.

1.2.2.2. Détail des blocs programmables

La Figure 15 montre la cellule de base de l'AT40K, également appelée PLB (*Programmable Logic Bloc*). Elle est constituée de deux LUTs (*Look Up Table*, table de correspondance), une bascule D, une porte ET, un registre trois états et d'un ensemble de multiplexeurs et de commutateurs permettant d'alimenter les différents éléments logiques. On trouve notamment 4 multiplexeurs en entrée, alimentés par les liaisons venant des cellules voisines et par les bus interconnectant les différentes cellules du tableau. Les signaux sortent de la cellule via deux multiplexeurs X et Y reliant la cellule à ses 8 voisines immédiates, et par le bus L. Les LUTs peuvent être utilisées séparément pour implémenter des fonctions à trois entrées ou conjointement pour des opérateurs à quatre entrées.

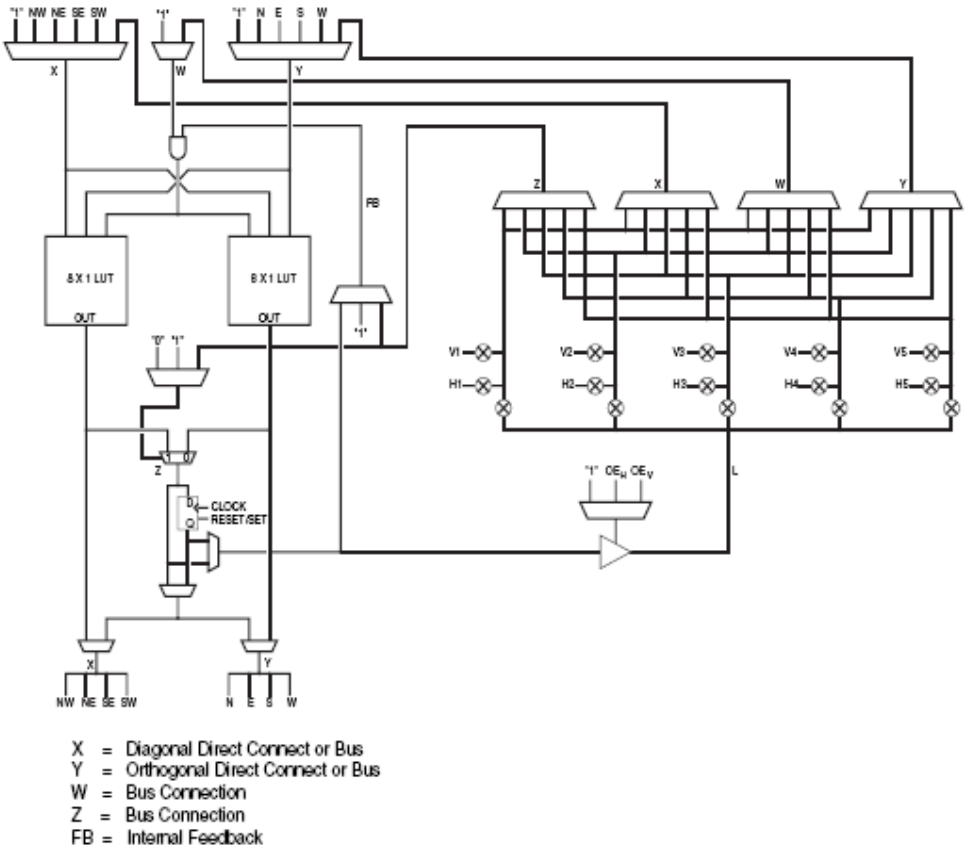

Figure 15: Architecture de la cellule de base ATMEL, le PLB [ATM03].

1.2.2.3. Réseaux d'interconnexions

Le réseau d'interconnexions est divisé en deux parties, un ensemble de bus traversant tout le FPGA permettant de relier n'importe quelles cellules, et des liaisons directes, permettant à chaque cellule d'échanger des signaux avec chacune des 8 cellules voisines.

La Figure 16 représente un des cinq plans de bus du réseau programmable ATMEL. Chacun se compose de trois bus, un bus « local » (celui du milieu) et deux bus « express ».Les bus sont connectés via les répétiteurs qui, en outre, régénèrent les signaux. Chaque répétiteur a un bus local et un bus « express » en entrée et en sortie (à noter que ces liaisons sont bilatérales). Toutes les connexions sont réalisables. On trouve un répétiteur toutes les quatre cellules pour un bus local, et un toutes les huit cellules pour les bus « express ». Les lignes de répétiteurs sont donc espacées de quatre cellules.

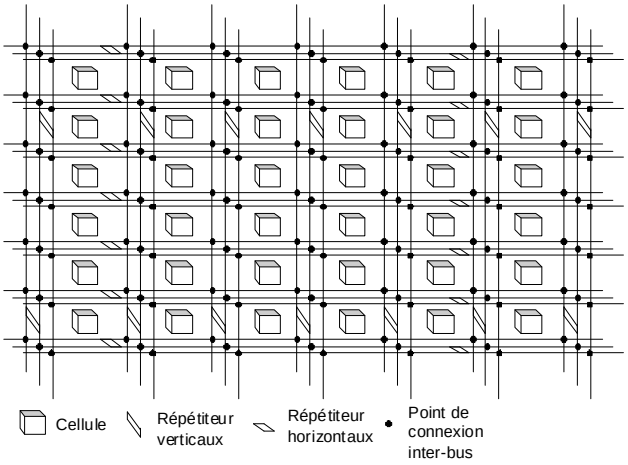

Figure 16: Plan de bus du système d'interconnexion de l'AT40K [ATM03].

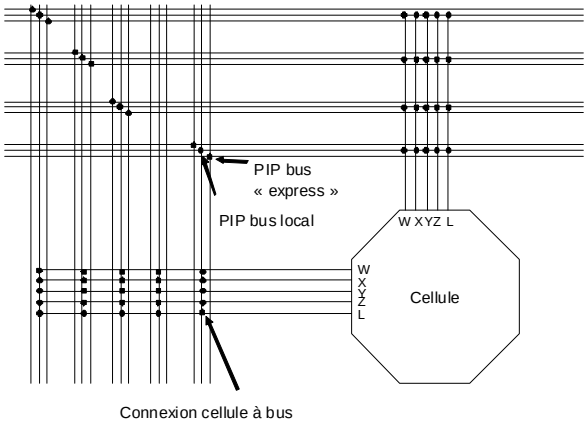

Figure 17: Connexions cellules/bus [ATM03].

Si la continuité et la connectivité des signaux le long de chaque axe du FPGA sont assurées par les répétiteurs, des « points de connexion programmables » (PIP) permettent de relier les bus horizontaux et verticaux entre eux, comme le montre la Figure 17.

Il faut noter que seuls les bus locaux sont accessibles aux entrées/sorties de la cellule directement, mais des liaisons directes permettent à chaque cellule d'accéder aux multiplexeurs voisines W, X, Y, Z. La sortie X du bus local alimente les « express » des multiplexeurs W de cellules situées à droite et à gauche, et d'autre part et d'autre du bloc RAM. Les quatre bits de données de lecture et d'écriture sont reliés aux bus locaux de quatre lignes de cellules consécutives, respectivement sur le plan 1 pour l'écriture, et sur le plan 2 pour les données de lecture. Enfin, des liaisons directes permettent à chaque multiplexeur d'entrée d'accéder aux sorties des répétiteurs Y des cellules voisines orthogonalement, tandis que la sortie X alimente le multiplexeur X de ses voisines diagonales.

1.2.2.4. Mémoire RAM utilisateur

La Figure 18 montre le positionnement des blocs de RAM destinés à l'utilisateur à l'intérieur du tableau de cellules. On retrouve un bloc de 32 mots de 4 bits à l'intersection de chaque ligne de répétiteurs horizontale et verticale. Les cinq bits d'adresse de lecture et d'écriture sont respectivement connectés à cinq « bus express » d'une même colonne de cellules (un bus « express » par plan) à droite et à gauche de part et d'autre du bloc RAM. Les quatre bits de données de lecture et d'écriture sont reliés aux bus locaux de quatre lignes de cellules consécutives, respectivement sur le plan 1 pour l'écriture, et sur le plan 2 pour les données de lecture.

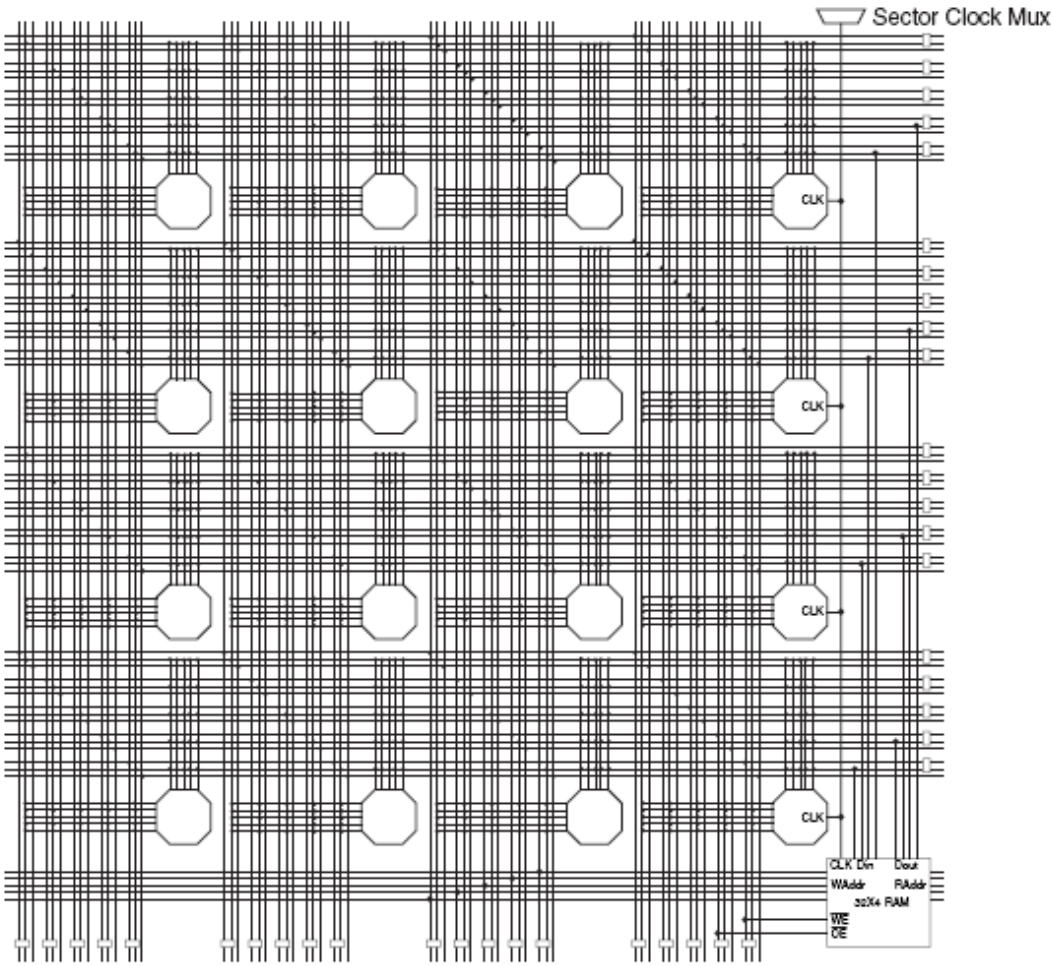

Figure 18: Connexion des blocs RAM au tableau de cellules [ATM03].

Il faut noter que les opérations de lecture et d'écriture sont complètement indépendantes, et que les lectures sont asynchrones.

1.2.2.5. Gestion de l'horloge

L'AT40K dispose de 8 bus d'horloge (GCK), avec 4 horloges spéciales destinées aux blocs d'entrée/sortie disposés de part et d'autre du FPGA. Chaque colonne de cellules a accès à une ligne d'horloge via le multiplexeur « Column Clock Mux » et des multiplexeurs de secteur desservent chaque zone délimitée par les rangées de répétiteurs horizontaux et verticaux. Un bus « express » du plan 4 est utilisé pour alimenter les différents secteurs du FPGA (voir Figure 19).

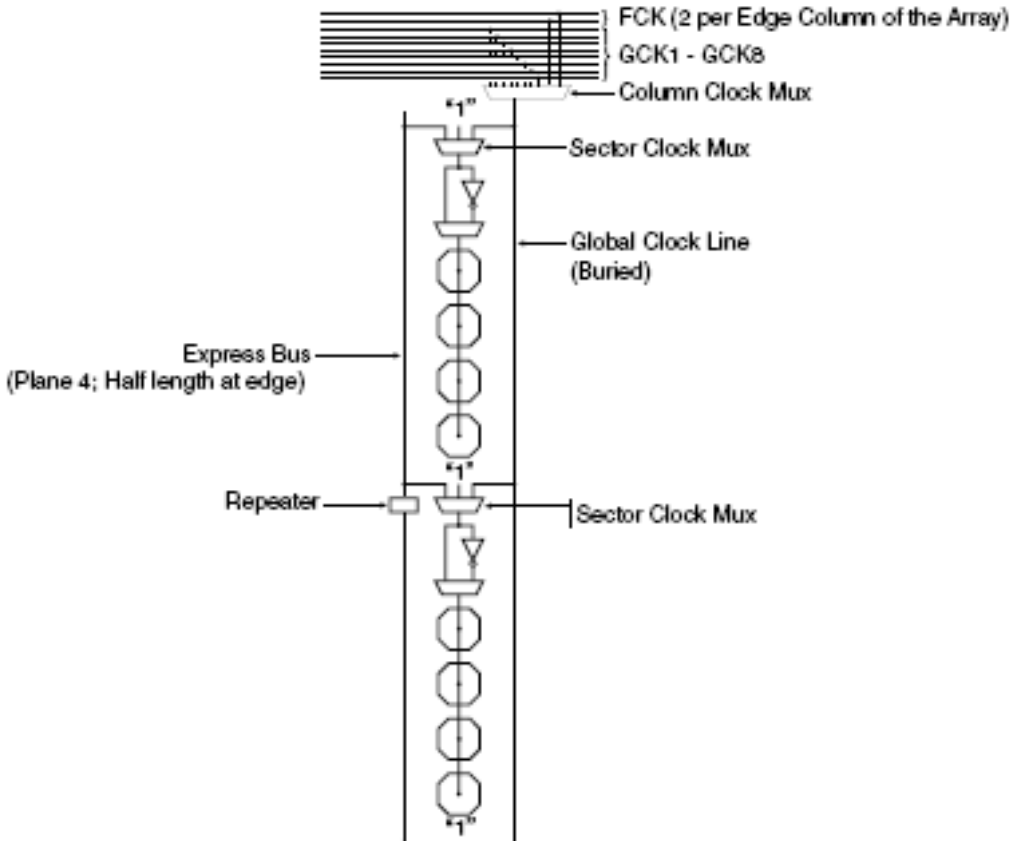

Figure 19: Distribution des horloges dans le tableau [ATM03].

1.3. Couche de configuration

Nous parlerons dans cette partie de la mémoire SRAM chargée de la programmation des éléments logiques et des éléments de routage des réseaux programmables, communément appelée mémoire de configuration. Nous nous limitons ici aux éléments fournis par les constructeurs des plateformes Virtex II et AT40K.

1.3.1. Organisation de la mémoire de configuration sur Virtex II

1.3.1.1. Organisation physique de la couche de configuration

La Figure 20 montre un point mémoire SRAM 5T de la mémoire de configuration d'un réseau programmable SRAM. Les points mémoires sont répartis parmi les éléments fonctionnels du FPGA [BOC07] et sont directement connectés aux transistors de la couche opérative.

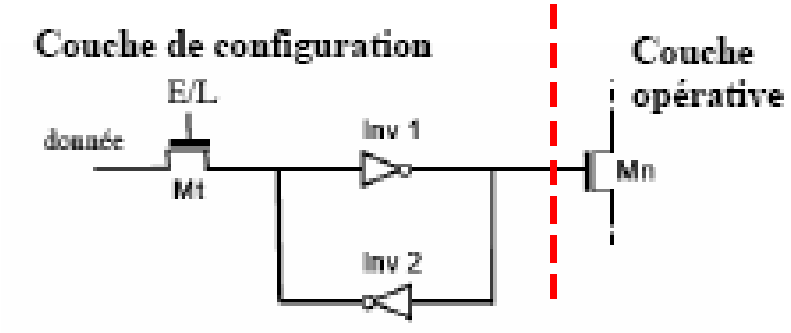

Figure 20: Lien entre couche opérative et couche de configuration [BOC09].

Néanmoins, on peut représenter cette couche de configuration comme une mémoire classique, avec des lignes de données et des lignes d'adresses. Cette mémoire est donc accessible en écriture et en lecture, respectivement lors des opérations de configuration et de relecture (« readback »). La taille des mots adressés correspond aux « frames », ou paquets élémentaires de bits, avec le Virtex II.

1.3.1.2. Structure du Bitstream

Le bitstream est composé de paquets de bits, chaque paquet ayant un en-tête spécifiant l'adresse du registre de configuration à utiliser pour la lecture ou l'écriture des données de configuration. Suivant la taille de paquet, deux types d'en-tête peuvent être utilisés

- Type 1, pour un paquet de taille inférieure à 2^11-1 mots
- Type 2, pour un paquet de taille variant entre 2^11-1 et 2^27-1 mots

Les deux types d'en-tête spécifient le mode lecture ou écriture ainsi que le nombre de mots contenus. Le reste du bitstream contient tous les frames nécessaires à la configuration du FPGA.

1.3.1.3. Organisation en frames

Une frame correspond à une colonne complète de 1 bit de large de la matrice de configuration. Les bits présents à l'intérieur d'une frame gèrent donc la configuration des ressources présentes dans des tuiles différentes (voir Figure 21). La taille du bitstream varie en fonction du circuit choisi dans la famille Virtex II.

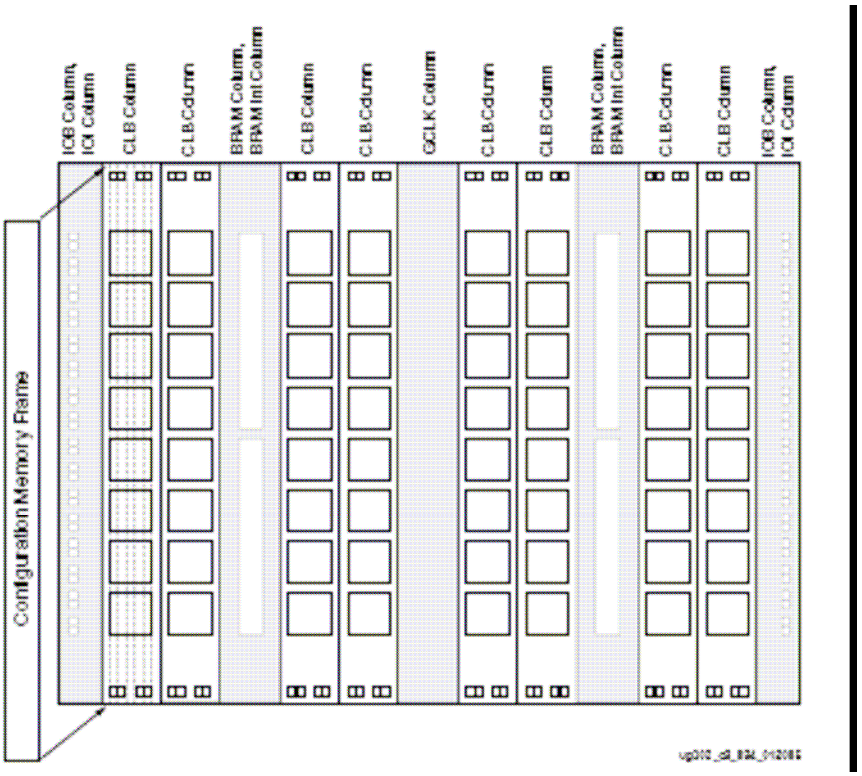

Figure 21: Répartition des ressources de configuration en frames [XIL05b].

Chaque frame a une adresse unique sur 32 bits, composée d'une adresse de bloc (BA), d'une adresse majeure (MJA), d'une adresse mineure (MNA), et d'un numéro d'octet (BN).

L'adresse majeure identifie une colonne à l'intérieur d'un bloc, tandis que l'adresse mineure identifie une frame à l'intérieur d'une colonne. Il faut noter qu'on ne peut pas adresser un octet séparément, l'accès en écriture sur la partie BN étant réservé pour la logique de configuration elle-même. La frame est bien le plus petit élément adressable chez Xilinx.

La Figure 22 présente la carte de la mémoire de configuration du point de vue de l'adressage. Les blocs 1 et 2 sont réservés aux blocs de RAM et à la gestion de leurs interconnexions, tandis que le bloc 0 contient la configuration des éléments de logique et de routage, de l'horloge, et des blocs d'entrée/sortie. Lors d'une configuration, les frames sont chargées par adresses (de blocs, majeures, mineures) croissantes, les frames de gestion d'horloge sont donc les premières écrites dans la mémoire.

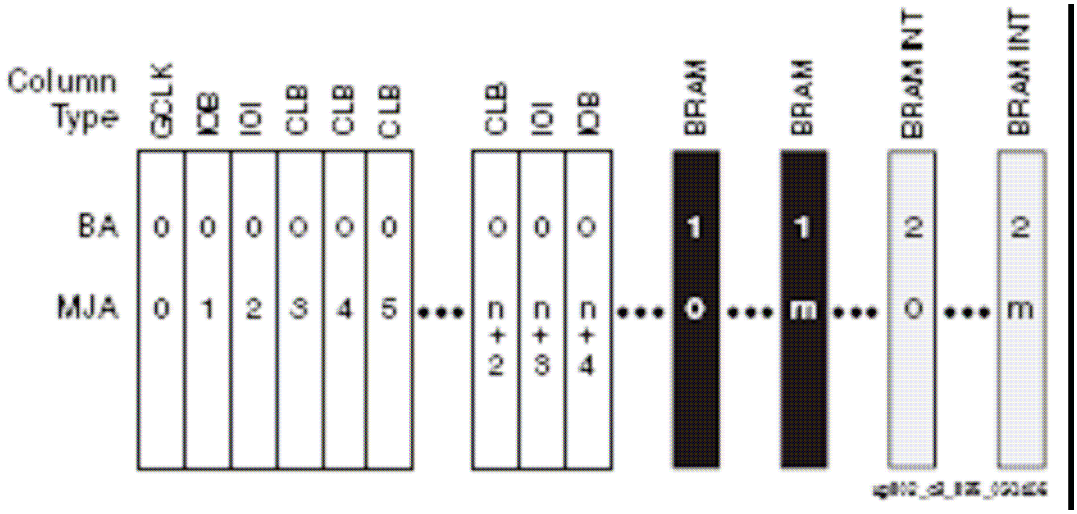

Figure 22: Carte de la mémoire de configuration, adressage par frame [XIL05b].

1.3.1.4. Modes de configuration

L'interface de configuration du Virtex II possède plusieurs modes : mode série, mode Select Map, ou JTAG [XIL05a]. Le mode série consiste à charger la mémoire bit par bit à partir d'une mémoire externe esclave. Le mode Select Map utilise un port parallèle et permet de charger les données octet par octet. Il est le plus rapide des trois. JTAG est un autre type de configuration série, qui utilise des entrées du FPGA initialement dédiées au test.

1.3.1.5. Logique de configuration

La logique de configuration à proprement dite fait le lien entre l'interface de configuration, et la mémoire elle-même. La mémoire de configuration est écrite à l'adresse contenue par le FAR (*Frame Address Register*) lorsque le FDRI (*Frame Data input Register*) est chargé. Le FB (*Frame Buffer*) étant pipeliné avec le FDRI, la première trame est écrite dans la mémoire lorsque la deuxième est chargée dans le FB. Ce mécanisme explique la présence d'une trame « blanche » à la fin de chaque bitstream pour pouvoir charger toutes les frames. Les opérations de readback sont similaires sauf que l'ordre des buffers dans le pipeline est inversé, le FDRO (*Frame Data Output Register*) alimentant le « Frame Buffer ». Une représentation est donnée en annexe.

1.3.1.6. Structure des fichiers associés aux configurations et aux readbacks

Les fichiers de configuration sont au format .bit tandis que les données issues du readback sont stockées dans un fichier format .rbd. Le fichier .rbd contient un en-tête sans rapport avec le reste des données, donc on associe aux fichiers .rbd un autre format contenant les mêmes données sans l'en-tête. La Figure 23 présente les différents formats des données de configuration.

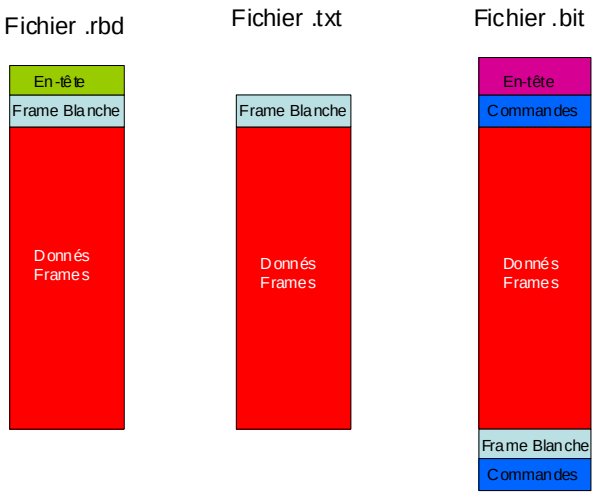

Figure 23: Format des fichiers manipulés.

En plus des données de configuration proprement dites, on trouve dans ces fichiers une « frame blanche », dont les données n'ont aucun sens du point de vue configuration, mais qui vient de la structure pipelinée de l'interface permettant de lire et écrire la mémoire de configuration du FPGA. Comme le fichier au format .bit est utilisé en lecture, la trame blanche se trouve après les données valides, alors que c'est l'inverse pour un readback. Il faut préciser que les données frames présentes dans les deux fichiers sont comparables, c'est à dire que la frame X du fichier .bit correspond aux mêmes ressources du FPGA que la frame X du fichier .rbd car les lectures comme les écritures de la mémoire de configuration se font par adresses croissantes. Davantage d'informations sont données en annexe.

1.3.2. Configuration de l'AT40K

Il existe trois modes de configuration pour l'AT40K :

- Mode Série maître :

Le FPGA se configure tout seul dans ce mode. Après un reset à la mise sous tension, il charge les données de configuration contenue dans une EEPROM externe, tout en lui fournissant l'horloge de configuration. Les données sont chargées en série.

- Mode Série esclave :

La configuration est initiée par l'interface externe de configuration, et non le FPGA. Les données sont écrites sur le front montant de l'horloge fournie par l'interface.

- Mode RAM synchrone :

On applique des mots de 32 bits au FPGA (24 bits d'adresse, 8 bits de donnée). Les données sont écrites sur le front montant de l'horloge fournie par l'interface de configuration. L'utilisateur a tous les droits d'écriture, c'est à dire qu'il n'y a pas d'ordonnancement prédéfini des données à écrire dans la mémoire.

Le tableau 1 récapitule les différents modes de configuration de l'AT40K.

Tableau 1: Modes de configuration de l'AT40K.

Mode	Description	Horloge	Type de donnée	Remarques
0	Liaison série, FPGA maître	Sortante	Série	Configuration automatique, EEPROM série
1	Liaison série, FPGA esclave, cascade	Entrante	Série	Microprocesseur, EEPROM série
4	RAM synchrone	Entrante	Mot de 8 bits	Port parallèle microprocesseur, 24 bit d'adressage en entrée

De plus amples informations sur la configuration de ce FPGA sont données dans la suite de ce manuscrit, notamment dans le chapitre 4.

1.4. Conclusion

Ce premier chapitre a permis de rappeler les caractéristiques générales des réseaux programmables, et de préciser celles de deux familles configurées par mémoire SRAM et qui seront les cibles de nos études : le Virtex II de chez Xilinx, et l'AT40K de chez Atmel. Dans les deux cas, nous avons résumé les informations importantes pour la compréhension de nos travaux, que ce soit au niveau architectural, ou du point de vue de leur mémoire de configuration. Mais pour mieux préciser dans quel contexte se situent nos études, le chapitre 2 va présenter l'état de l'art des méthodes d'analyse de criticité et d'amélioration de la robustesse pour les circuits implantés sur ce type de technologie.

2. Etat de l’art

Les réseaux programmables SRAM sont des circuits complexes et sensibles à de nombreuses perturbations, tant au niveau atmosphérique que dans l’espace, et utilisés pour des applications elles-aussi complexes et dans des contextes de plus en plus exigeants en terme de sûreté de fonctionnement. Nous aborderons donc dans ce chapitre les différentes techniques d’analyse du comportement de ces circuits en présence de SEUs, et les méthodes jusqu’ici développées afin de masquer ou de corriger les fautes du point de vue utilisateur.

2.1. Analyse de sensibilité de composants électroniques

Afin de concevoir des circuits robustes en utilisant des réseaux programmables à mémoire SRAM, il est d’abord nécessaire d'étudier leur comportement après occurrence d'une perturbation, et en particulier lors de fautes dans la mémoire de configuration. Plusieurs techniques ont été développées et utilisées pour comprendre le comportement des circuits en présence d’évènements singuliers, par exemple en présence de SEUs. Une première solution est de quantifier la sensibilité intrinsèque du composant, afin d’en obtenir sa section efficace. D’autres techniques s’attachent à étudier le comportement du système complet, carte et application, en adoptant des techniques diverses pour pouvoir injecter des fautes, allant des attaques par radiations aux attaques LASER. Enfin, plusieurs techniques de caractérisation « a priori » ont été développées, ne nécessitant pas de tests sur carte et pouvant être employées en phase de conception.

Après une rapide présentation des différents types de méthodologies d’analyse des réseaux programmables, nous résumerons plusieurs approches pour leur mise en œuvre pratique.

2.1.1. Méthodologies d’analyse

Les stratégies classiques d’analyse de robustesse des circuits implantés sur des composants SRAM peuvent être rangées dans deux grandes catégories : les stratégies statiques et dynamiques. Les stratégies statiques tiennent compte principalement du composant utilisé pour implanter l'application (et dans certains cas de la configuration du composant pour l'application). Les stratégies dynamiques tiennent compte aussi des ressources employées dans les différentes phases de l'exécution de cette application.

Les tests statiques permettent d’obtenir la sensibilité intrinsèque d’un composant, c'est à dire d’obtenir le nombre moyen de particules nécessaires pour induire une faute ou une erreur. On peut en extraire alors la section efficace pour un type de particule donné. On pourra par la suite déduire le taux de panne attendu dans un environnement particulier, dans un contexte pire cas qui ne tient pas compte du déroulement de l'application. Notons que ces tests statiques sont à différencier de l'analyse statique en

phase de conception, qui sera proposée dans les chapitres suivants, car ils nécessitent un composant physique.

Les tests dynamiques permettent une évaluation de la sensibilité pour une application précise, en tenant compte du déroulement temporel de celle-ci. On pourra déduire de cette évaluation des taux d'erreurs plus réalistes pour un environnement et une application donnés. Ces tests dynamiques sont à nouveau à différencier des analyses en phase de conception, qui peuvent être réalisées sur la base d'injections de fautes simulées ou émulées.

2.1.1.2. Tests statiques

Les tests statiques sont utilisés pour l'évaluation de la sensibilité intrinsèque d'un composant, exprimée par sa section efficace. Cette mesure est généralement obtenue à l'aide d'accélérateurs de particules, en exposant le DUT (*Device Under Test*) à un flux de particules. Les zones sensibles potentielles, comme les points mémoires accessibles par l'utilisateur, doivent être identifiées préalablement et si possible initialisées à une valeur connue avant le test. On procède alors à l'irradiation du DUT, lui-même sous tension et inactif (mode « idle »). Finalement, on relit toutes les zones sensibles du composant qui sont accessibles et on compare leur valeur avec leur contenu initial afin de détecter les SEUs potentiellement engendrés.

L'opération est répétée pour plusieurs types de particules et plusieurs énergies (LETs) afin d'obtenir une valeur de section efficace pour des environnements différents. On peut alors tracer la courbe de section efficace en fonction du LET et par particule. Cela permet aux développeurs d'estimer le nombre de SEUs susceptibles de se produire dans le DUT en fonction de l'environnement de fonctionnement.

Le test statique caractérise donc le composant sans toutefois prendre en compte l'application s'exécutant sur le composant. Il n'est donc pas suffisant pour obtenir une estimation correcte du taux d'erreurs pour l'application finale. Une manière d'affiner cela, dans le cas d'un composant de type FPGA à mémoire SRAM, est d'initialiser la mémoire de configuration avec la description du circuit applicatif final, ou une description proche. Dans ce cas, la sensibilité évaluée peut être plus proche de la sensibilité finale. Toutefois, un tel test ne prend en compte que la dimension spatiale, c'est-à-dire le nombre et la nature des ressources utilisées. La dimension temporelle ne peut pas intervenir dans un test statique du fait de sa nature même.

2.1.1.3. Tests dynamiques

Les tests dynamiques reprennent la méthodologie des tests statiques en y ajoutant la notion de «fenêtre temporelle de sensibilité». En effet, le contenu des cellules mémoires n'est généralement pas critique à chaque instant de l'exécution de l'application. La Figure 24 décrit les fenêtres de sensibilité pour deux points mémoire P1 et P2. Le contenu de ces cellules est critique entre une phase d'écriture et une phase de relecture. Une particule modifiant le contenu du point mémoire, alors qu'il n'est pas

initialisé, ou bien entre une lecture et une écriture, n'aura pas d'effet sur l'application qui viendra de toute manière écraser le contenu de la cellule par la suite.

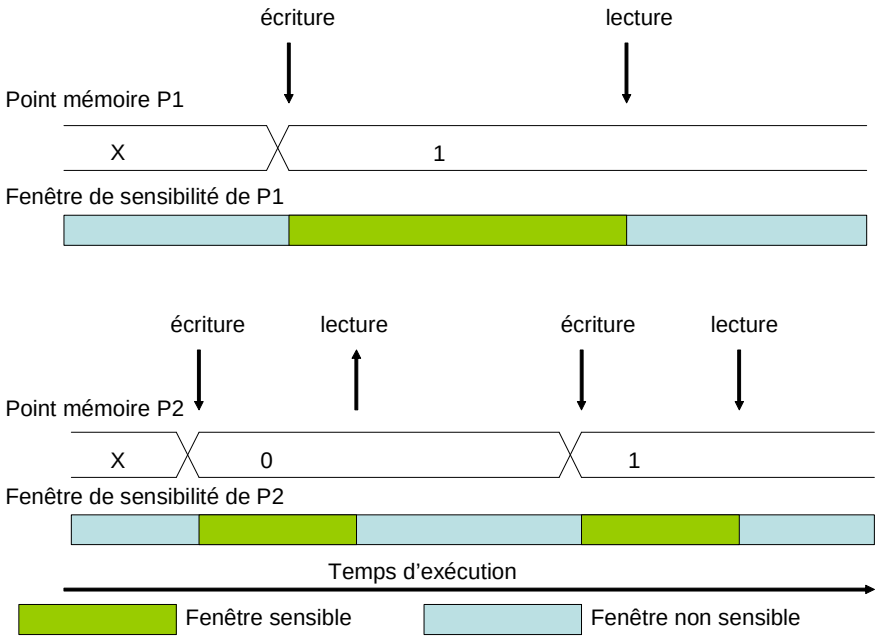

Figure 24: Fenêtre de sensibilité de deux points mémoires [FOU10].

Un test dynamique est donc effectué en enregistrant les erreurs au cours de l'exécution de l'application et en tenant compte de l'effet sur le déroulement de cette application. Ce type de test fournit des données statistiques sur le nombre moyen de particules nécessaires afin d'obtenir une erreur observable sur les sorties de l'application. Il est alors possible d'extrapoler ce nombre pour le système fonctionnant en environnement réel, et ainsi d'en déduire le nombre d'erreurs susceptibles d'apparaître par unité de temps (jour, mois, année, …). Bien que beaucoup plus proche de la réalité, ce type de test est extrêmement coûteux lorsqu'il est réalisé en accélérateur car il nécessite des temps d'exposition très longs ; c'est la raison pour laquelle les tests statiques sont souvent préférés.

2.1.2. Analyse par radiations

L'évaluation de la sensibilité d'un composant face aux évènements singuliers est très souvent réalisée sur la base de «tests accélérés», c'est-à-dire en mettant le composant dans un environnement où la fréquence des perturbations est beaucoup plus élevée que dans l'environnement naturel. Les deux principaux moyens employés sont les sources radioactives et les accélérateurs de particules.

- Sources radiatives

L'utilisation d'une source radioactive est une méthode peu coûteuse et qui fournit des informations préliminaires sur la sensibilité d'un composant [FOU10]. Le Californium 252, par exemple, émet des particules alpha, deux types d'ions lourds possédant respectivement des LETs de 45 et 46 MeV.cm²/mg ainsi que des neutrons en faible quantité. Une autre source couramment utilisée est l'Américium 242 qui produit des particules alpha ayant une énergie de 5 MeV (LET = 0,6

MeV/mg/cm²). Cependant, les particules générées par cette source ont une énergie bien plus faible que celles que l'on peut trouver en environnement spatial ou en accélérateur de particules. Ceci influence directement la distance de pénétration de la particule, qui se retrouve ainsi limitée à un parcours dans la matière d'environ 15 µm. Cette distance peut être insuffisante pour des composants possédant des couches superficielles importantes, malgré un amincissement. Cette méthode de test permet néanmoins, lorsque le composant l'autorise, de valider la plateforme matérielle et logicielle de test avant de pratiquer des essais en accélérateurs de particules qui sont plus coûteux et qui ne laissent guère de place à l'incertitude.

- Accélérateurs de particules

Différents types d'accélérateurs de particules (accélérateur linéaire, cyclotron, etc…) peuvent produire un large éventail de faisceaux de particules telles que des ions lourds, des protons et des neutrons. L'avantage de ces installations est la taille du faisceau (quelques centimètres) qui permet d'irradier toute la surface d'une puce en une seule fois. Cependant, pour la plupart des faisceaux, les essais aux ions lourds doivent se dérouler à l'intérieur d'une enceinte sous vide. Ceci impose des contraintes sur les connectiques vers l'extérieur de l'enceinte, qui doivent être prises en compte lors du développement de la plateforme. La reproduction exacte des espèces d'ions lourds présents dans l'environnement spatial n'est pas réalisable avec des équipements au sol à cause de la diversité des particules que l'on y trouve, de leur haute énergie et de leur incidence omnidirectionnelle. Des accélérateurs « faible » énergie sont donc utilisés afin de reproduire un dépôt de charges équivalentes en jouant sur le paramètre de LET des particules. Ces particules issues d'accélérateurs sont généralement peu pénétrantes, ce qui ne pose pas de problème pour les composants dont les zones actives sont accessibles par la face avant. Mais ceci n'est par exemple pas le cas avec le Virtex-II monté en boitier *flip-chip*. Les particules doivent traverser le substrat de la puce, il faut donc procéder à un amincissement de ce substrat [FOU10].

2.1.3. Analyse grandeur nature

Les tests en accélérateurs ont plusieurs limitations. Outre leur coût, ils demandent de disposer d'un accélérateur fournissant les particules souhaitées, aux énergies souhaitables. Ces équipements sont peu nombreux et il faut parfois attendre de nombreux mois pour pouvoir disposer d'un créneau temporel sous le faisceau. De plus, les résultats obtenus doivent être extrapolés pour les fluences réelles dans l'environnement final ; comme ces résultats sont généralement basés sur un nombre d'évènements très petit, la marge d'erreur lors de l'extrapolation peut devenir trop grande. Il peut alors être intéressant de réaliser des mesures en environnement réel, potentiellement sur de longues durées.

Les essais en conditions réelles consistent à exploiter le composant sous test dans un environnement naturel. Pour cela, ils peuvent être mis en œuvre dans un environnement spatial, en haute altitude ou bien au sol (voire même parfois sous le sol, pour l'étude de l'effet de certaines particules). Dans tous les cas, ces expériences requièrent l'utilisation d'un grand nombre de composants pour compenser la

densité de particules relativement faible de l'environnement naturel, comparativement aux flux que l'on trouve en accélérateurs de particules. Si ces tests permettent d'obtenir des résultats réalistes, ils comportent néanmoins aussi un certain nombre de contraintes et d'inconvénients.

- Essais en orbite

Il est implicite que des essais en orbite fournissent des résultats réalistes pour des applications du domaine spatial, étant donné que le composant est testé dans les conditions qui seront proches de celles de l'application finale. Des résultats de telles expériences sont disponibles dans la littérature [FAL94], [DUZ97]. Régulièrement les agences spatiales mettent en place des projets afin de tester en ambiance spatiale sévère des nouveaux composants et de nouvelles technologies.

- Essais en haute altitude

Les essais en haute altitude peuvent avoir lieu sur des durées plus ou moins courtes, par exemple avec des expériences embarquées à bord d'avions, ou grâce à des ballons stratosphériques. L'objet de ces expériences peut être la caractérisation du milieu pour établir ou valider des modèles [NOR01], ou bien encore pour étudier les effets sur les équipements et les personnels [CHE00] [GOL00] [SOH00].

- Essais au sol

Les essais au sol peuvent permettre de bien caractériser l'impact de l'environnement terrestre sur le fonctionnement de l'électronique employée dans les applications quotidiennes. L'expérience Rosetta [LES05] menée par Xilinx est un exemple de la prise de conscience des fabricants concernant le problème des effets des radiations sur les circuits intégrés. Cette expérience consistait à effectuer des tests statiques en ambiance naturelle sur un grand nombre de composants pour augmenter la surface sensible et ainsi compenser le faible flux de particules au sol. Des plateformes de test ont été installées sur plusieurs sites afin d'obtenir des résultats pour plusieurs altitudes et plusieurs latitudes étant donné que la densité de neutrons n'est pas uniforme [BAR97] sur toute la surface terrestre.

2.1.4. Analyse par attaques laser

Alors que les accélérateurs de particules apportent une approche globale à la caractérisation des composants vis-à-vis des SEEs, ils ne peuvent fournir aucune information sur la localisation des fautes détectées et sur les structures internes les plus sensibles. En revanche, le laser allie un faible diamètre de faisceau (de l'ordre du micron) et une précision de déplacement submicronique ; il est ainsi possible d'effectuer une cartographie précise de certaines zones sensibles. Malgré ses avantages, le test à l'aide d'un faisceau laser est soumis à deux contraintes :

- Le faisceau est réfléchi par les couches de métallisation ce qui pose un réel problème pour les composants complexes dotés de plusieurs niveaux de métallisation. Il est alors indispensable de procéder à des attaques par la face arrière.
- L'énergie du faisceau ne peut pas être corrélée avec l'énergie des ions lourds [POU01].

Par conséquent les tests à l'aide de faisceaux laser sont complémentaires aux mesures en accélérateurs de particules grâce à leur précision de positionnement du faisceau et la possibilité d'effectuer des tirs «coup par coup» mais ne permettent pas de remplacer les autres types de tests.

2.1.5. Analyse en phase de conception (injection de fautes)

Les tests mentionnés précédemment nécessitent de disposer d'un prototype physique du système, ou au moins d'un exemplaire du composant principal. Ils nécessitent de plus de développer un environnement de test qui peut être complexe. Dans tous les cas, ces tests ne peuvent pas être réalisés en cours de conception et lorsque le système démontre une sensibilité trop importante, une reprise de la conception est nécessaire, ce qui induit des coûts et des pertes de temps pouvant être inacceptables. Il est donc important de pouvoir évaluer, en phase de conception, et avant de mettre en place des tests lourds et coûteux, le niveau de robustesse d'une implantation. Ceci est réalisé grâce à diverses approches dites d'«injection de fautes» permettant d'analyser le comportement du système lorsqu'une erreur survient. Les tests statiques peuvent être utilisés (le plus souvent indépendamment de l'application) pour évaluer la probabilité d'erreurs en fonction de l'environnement et du composant visé. Les injections de fautes viennent en complément pour évaluer la probabilité qu'une erreur se traduise par un disfonctionnement de l'application. Ces injections de fautes sont donc réalisées selon une approche de test dynamique.

La plupart des approches visent à analyser l'effet d'erreurs dans les bascules utilisées par l'application. De nombreux exemples existent, qui ne seront pas détaillés ici. Quelques approches existent aussi pour analyser l'effet d'erreurs à la fois dans les bascules utilisateur et dans la mémoire de configuration pour des circuits implantés sur des FPGA à mémoire SRAM. Parmi ces approches, la plus couramment citée ces dernières années tire profit des capacités de reconfiguration dynamique de la plupart de ces réseaux.

On trouve plusieurs références dans la littérature, notamment des expériences réalisées sur des réseaux Xilinx. Dans la première approche [ANT01], les bitstreams sont modifiés avec JBits, un outil Java qui permet une lecture de la configuration du FPGA et sa reconfiguration. Les résultats obtenus démontrent la faisabilité de l'approche et montrent que celle-ci permet des économies de temps importantes dans certaines conditions. Une description plus complète de l'approche ainsi que des résultats obtenus est proposée dans [ANT03].

L'université de Séville et l'Agence Spatiale Européenne ont conjointement développé le système FT-UNSHADES [TOM04]. Celui-ci met en œuvre deux copies du circuit à analyser; une de référence et une pour l'injection. La comparaison des états des deux copies permet de détecter l'activation ou non de la faute injectée. L'observation de l'état des bascules est réalisée à l'aide des mécanismes de relecture de configuration et de "capture" (relecture des états des éléments mémoires utilisateurs) disponibles sur les FPGA Xilinx de la famille Virtex. Cette technique permet l'inversion de bits dans

les bascules. La connectique très rapide entre l'ordinateur hôte et le FPGA est sans doute une des forces de l'environnement présenté mais a nécessité le développement de cartes spécifiques. En ce qui concerne l'approche, l'utilisation de deux copies accélère la comparaison des résultats obtenus avec les résultats de référence mais réduit énormément l'espace disponible sur le FPGA.

Pour réduire les temps de reconfiguration induits par la reconfiguration partielle, il est proposé dans [KAF06] d'utiliser un processeur embarqué pour effectuer les reconfigurations. Les données nécessaires à la sélection des cibles d'injection (LUTs, multiplexeurs…) sont obtenues grâce à l'outil de placement/routage. Toutes les configurations nécessaires sont préparées avant la campagne et téléchargées au début de celle-ci. Durant la campagne, il n'y a aucun transfert de données entre l'hôte et le circuit programmable. Les résultats (classification des fautes) sont générés par un comparateur matériel et renvoyés par le FPGA à la fin de la campagne. Les expériences menées démontrent non seulement que l'approche est faisable, mais également qu'elle est approximativement 30 fois plus rapide qu'une approche à base de simulation. Elle peut cependant nécessiter beaucoup d'espace mémoire si un grand nombre de configurations prédéfinies doit être stocké. De plus, comme pour l'approche précédente, la mise en œuvre des deux copies du circuit analysé (référence et avec fautes) réduit de moitié la taille possible pour celui-ci.

D'autres approches ont été développées sur le même principe ces dernières années ; l'objectif n'est pas ici de les détailler. Toutefois, il faut noter que dans tous les cas les analyses dynamiques possibles grâce à ces approches conduisent à des temps expérimentaux pouvant devenir très élevés. C'est la motivation principale pour le développement de nouvelles méthodologies d'analyse visant à établir un diagnostique statique de criticité des bits de configuration. Ce sera le sujet des deux prochains chapitres. Il faut noter que ces travaux ont été initiés à TIMA en 2006. Des études assez proches ont été réalisées plus récemment dans d'autres groupes de recherche, notamment à Turin [BAT09].

2.2. Protections pour FPGA à mémoire SRAM

Nous nous concentrerons dans cette partie sur les techniques visant à fiabiliser les applications implantées sur FPGA à mémoire SRAM. On s'aperçoit dans la littérature qu'il y a essentiellement trois approches proposées, avec différentes variantes. La solution la plus simple consiste à utiliser les fonctions inhérentes au FPGA lui-même pour limiter, voire masquer, l'impact des fautes de type SEU. Une autre solution est de développer de nouvelles structures de cellules SRAM pour diminuer leur sensibilité aux évènements singuliers, et de nouvelles architectures de cellules de base pour limiter les erreurs fonctionnelles dues aux fautes de configuration. Enfin, les circuits implantés sur ces FPGA peuvent être conçus avec des architectures robustes (masquage de fautes, détection et éventuellement correction d'erreur) en utilisant plus ou moins de redondance, ou plus généralement des flots d'implémentation sur FPGAs orientés fiabilité dans le contexte des fautes de configuration.

2.2.1. Utilisation de mécanismes intrinsèques

La plupart des fabricants de réseaux programmables intègrent des mécanismes de relecture de configuration et de reconfiguration totale ou partielle, qui peuvent être utilisés pour fiabiliser les applications, via la détection et la correction des fautes.

2.2.1.1. Rafraîchissement périodique de la mémoire de configuration

Le rafraîchissement périodique de configuration (« configuration scrubbing ») est l'une des techniques les plus simples employées pour protéger les applications du problème représenté par les SEUs. Elle consiste à recharger de manière périodique la mémoire de configuration dans son intégralité à partir d'une seconde mémoire construite dans une technologie résistante aux SEUs. Elle permet d'éviter l'étape de relecture de configuration, et nécessite, en conséquence, peu de ressources, que ce soit en terme de matériel ou de temps. Cependant la mémoire de configuration se trouve en mode écriture pendant un temps nécessairement plus long que d'autres techniques, parce que la mémoire est rechargée entièrement et systématiquement. Ceci peut perturber le fonctionnement normal de l'application, particulièrement si les générateurs de fonctions sont utilisés pour implémenter des éléments mémoires utilisateurs (type RAM), car leur contenu est alors dynamique. Il est donc nécessaire d'estimer correctement le taux d'erreur attendu afin de mettre en œuvre un taux de rafraîchissement le plus faible possible et ainsi de limiter la perte de performances. L'implémentation de cette technique est relativement simple, et nécessite uniquement de contrôler les adresses de la mémoire à partir de laquelle est rechargée la mémoire de configuration, ainsi que les signaux d'interface avec le FPGA pour l'écriture de configuration. Un exemple d'implémentation pour le réseau XILINX Virtex II est présenté Figure 25 [XIL00].

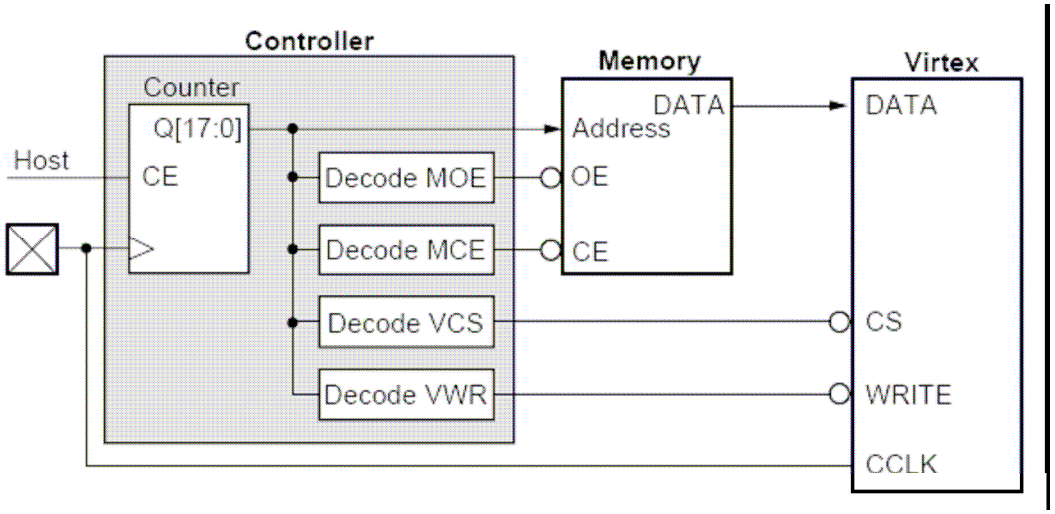

Figure 25: Circuit de contrôle pour le rafraîchissement périodique de configuration [XIL00].

2.2.1.2. Relecture de configuration

Le rafraîchissement périodique de configuration est une technique simple à mettre en œuvre mais implique que la mémoire est en mode écriture à intervalles réguliers. De plus le rafraîchissement est systématique, même si l'application fonctionne normalement. D'autres techniques ont été développées afin de s'affranchir de ces contraintes. Le principe est de détecter les fautes, et de reconfigurer uniquement les zones adressables fautées de la mémoire de configuration.

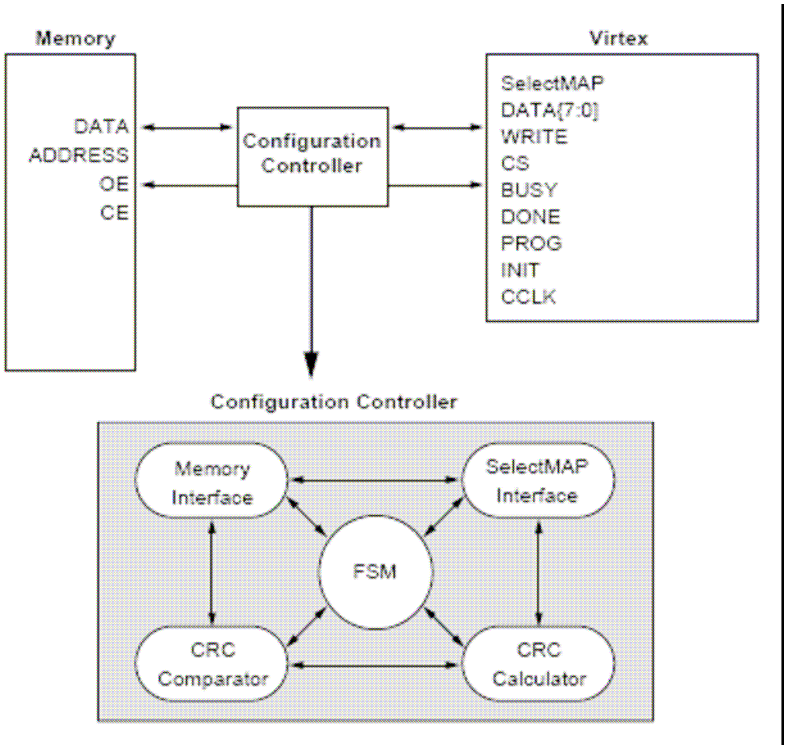

Figure 26: Exemple de mise en œuvre des techniques de détection et correction de SEUs [XIL00].

- Détection par comparaison bit à bit

Pour la détection, on peut procéder à une comparaison bit à bit des données relues à partir du FPGA, et du bitstream original utilisé pour configurer l'application. Cette technique nécessite généralement l'usage d'un fichier de masque pour séparer les bits mémoires utilisateurs des autres bits de configuration, fichier dont la taille est identique à celle du bitstream original. Cette technique a donc un coût mémoire important.

- Détection par code cyclique (CRC)

La détection peut aussi être obtenue en enregistrant une signature obtenue avec un code cyclique (CRC) pour chaque zone mémoire reconfigurable. Une nouvelle valeur est calculée lors de la relecture de configuration et comparée avec l'originale. Le circuit utilisé pour calculer les valeurs de CRC sur les FPGAs Virtex XILINX (sur 16 bits) est présenté en annexe. Une reconfiguration de la zone est déclenchée en cas de différence. La Figure 26 présente l'architecture d'un système de protection contre les SEUs à partir de reconfigurations partielles pour les FPGAs Xilinx [XIL00].

2.2.2. Techniques de construction de composants FPGAs SRAM plus robustes

2.2.2.1. Architectures robustes pour le point mémoire SRAM

Une autre méthodologie pour améliorer la fiabilité des réseaux programmables est de travailler sur les cellules SRAM afin de diminuer leur sensibilité aux SEUs. Différentes méthodes ont été utilisées jusqu'ici, par exemple en dupliquant la cellule initiale. La Figure 27 illustre le principe. On utilise deux verrous ("latchs") redondants afin de maintenir une donnée non corrompue après un SEU, en stockant une donnée identique dans les verrous L1 et L2. Une inversion de l'état stocké dans L1 inversera une des entrées différentielles du verrou L2 qui rétablira alors l'état de L1.

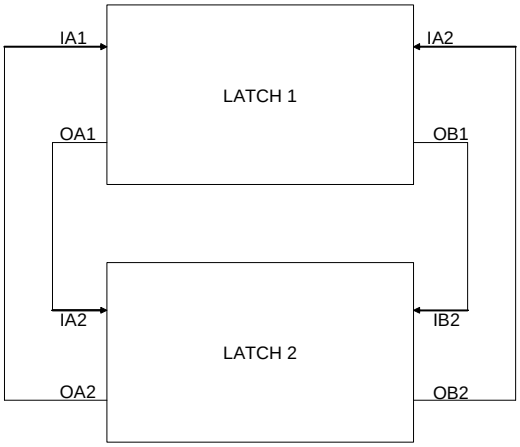

Figure 27: Durcissement par redondance des cellules mémoires [CAL96].

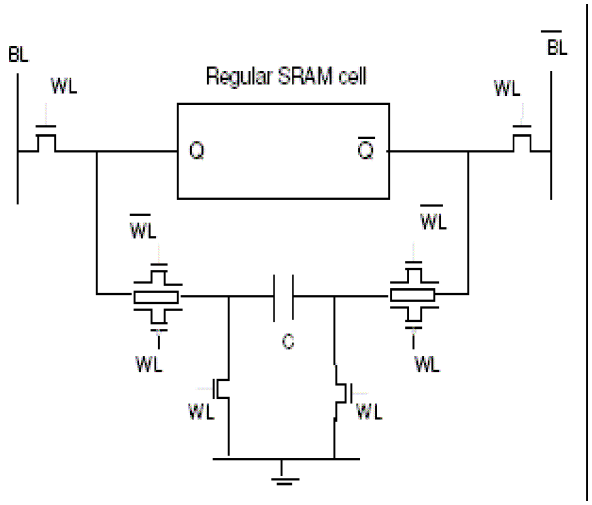

Figure 28: Exemple de protection d'une cellule SRAM avec des capacités [SHI08].

D'autres méthodes s'attachent à réparer le contenu erroné. Des moniteurs peuvent par exemple détecter les fautes et des codes correcteurs d'erreurs ou de la redondance sont utilisés pour les corriger [SHI08]. Ces méthodes fonctionnent correctement mais leurs performances diminuent avec la taille des grilles de transistors. De nouvelles techniques sont utilisées pour contourner ce problème. Elles utilisent le principe de masquage de fautes, insérant des capacités qui absorbent les charges excessives et empêchent ainsi l'inversion du point mémoire. La Figure 28 illustre un exemple d'implantation de ce principe [SHI08].

2.2.2.2. Nouvelles tuiles de bases

On trouve également dans la littérature des travaux proposant des techniques de fiabilisation aux SEUs au niveau de l'architecture des FPGAs, et plus particulièrement au niveau de la conception de nouvelles cellules de base. On peut, par exemple, citer les travaux de thèse présentés dans [BON07].

2.2.3. Architectures fiables

Pour éviter l’interruption du fonctionnement nominal d’une application lors d’opérations de (re)configuration, de nombreux travaux se sont efforcés de développer des techniques de fiabilisation innovante. La façon la plus simple est d’améliorer le durcissement de l’application, c’est également une des plus accessibles en terme de coût. La majorité des méthodes développées utilisent la redondance afin d’apporter une solution aux problèmes engendrés par les SEUs. On en distingue trois grandes déclinaisons: la redondance d’information, la redondance temporelle, et la redondance spatiale. Nous parlerons également à la fin de cette partie d’autres techniques de fiabilisation, comme par exemple les techniques de codage de machine à états pour la robustesse, ou encore des techniques de routage innovantes.

2.2.3.1. Redondance temporelle

La redondance temporelle permet de déterminer la valeur correcte à propager dans le cas de fautes dans les blocs combinatoires d'une application. Elle profite du caractère transitoire des fautes dans les blocs combinatoires pour comparer la valeur des signaux à des instants différents. Le principe est donc d'échantillonner la sortie du bloc combinatoire à trois instants différents. Comme indiqué sur la Figure 29, la deuxième bascule enregistre le signal avec un retard d, tandis que la troisième fait la même chose avec un retard de 2d par rapport à l'échantillonnage original. On utilise un "voteur" majoritaire pour déterminer la valeur correcte. Le surcoût surfacique est induit par les éléments mémoires additionnels, tandis que les performances sont diminuées par le fait que chaque cycle est augmenté de deux fois la durée typique d'une impulsion transitoire.

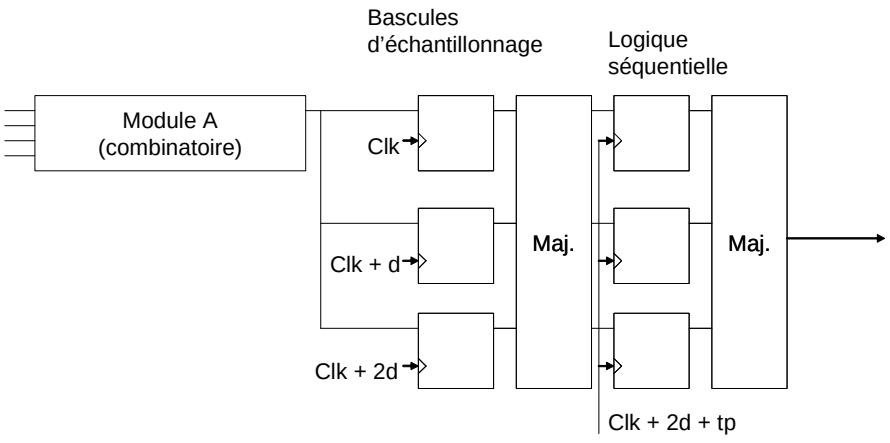

Figure 29: Modèle d'implantation de redondance temporelle [LIM03].

2.2.3.2. Redondance d’information

La redondance d'information consiste à utiliser des codes détecteurs d'erreurs et/ou des codes correcteurs d'erreurs afin d'améliorer la fiabilité sans altérer les performances (c'est à dire. sans surcoût temporel). Des données redondantes sont ajoutées au message original, afin de pouvoir en vérifier

l'intégrité ultérieurement. Cette technique est adaptée à la fois pour la protection des blocs combinatoires et pour celle des blocs séquentiels. A chaque fois, les bits de redondance sont générés deux fois, à partir des données protégées, et à partir des données originales. Un comparateur logique permet ensuite de détecter des différences et d'émettre un signal d'erreur si nécessaire.

La Figure 30 illustre l'implantation de redondance d'information sur un bloc séquentiel (registre). Les entrées sont utilisées pour calculer les bits de redondance à partir d'un module de codage. Ces bits sont stockés dans une extension du registre avec les données proprement dites. Un bloc de codage identique au précédent utilise les sorties du registre correspondant aux données d'entrée, et les bits de redondance obtenus avant et après mémorisation sont comparés afin d'émettre un signal d'erreur.

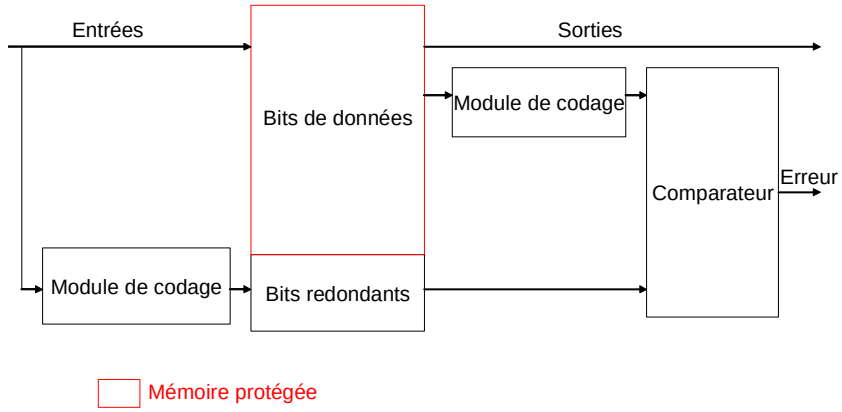

Figure 30: Modèle d'implantation de redondance d'information sur des éléments mémoires [MAI09].

La Figure 31 représente le cas d'un bloc combinatoire. Le bloc de prédiction est conçu afin de générer uniquement les bits de redondance directement à partir des entrées qui alimentent le bloc combinatoire protégé.

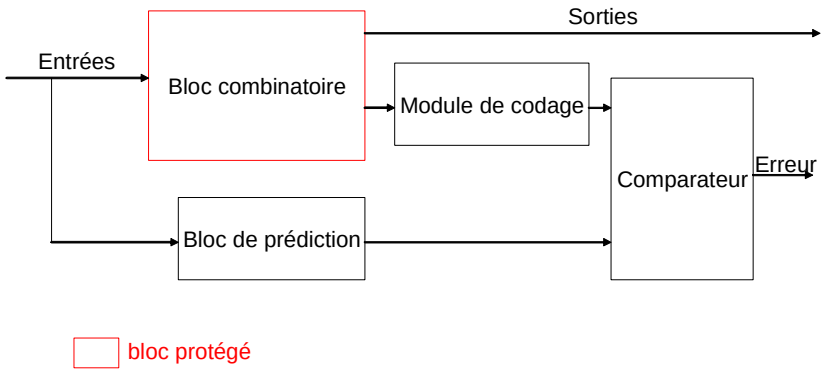

Figure 31: Modèle d'implantation de redondance d'information sur un bloc combinatoire [MAI09].

Les codes détecteurs d'erreurs peuvent être linéaires ou non linéaires. Le choix du code impacte grandement la valeur ajoutée en termes de fiabilité, ainsi que le surcoût surfacique. Les codes linéaires de taille n et de dimension k (k bits de données, n-k bits de redondance) sont des applications

vectorielles d'un espace de dimension n vers un espace de dimension k. Ils peuvent être représentés par une matrice de taille (n, n-k), nommée G pour référence. L'opération de codage (l'obtention du vecteur d'information codée C) consiste en la multiplication vectorielle des données à protéger (vecteur D, taille n-k) avec la matrice G (C= G*D). La vérification des données utilise une matrice H (matrice de contrôle) qui vérifie $G*H^T = 0$ (H^T est la matrice transposée de H). Pour tout vecteur X valide (calculé à partir de G*D), on a alors: H*X=0.

2.2.3.3. Différentes techniques de mise en œuvre de redondance matérielle

Les techniques utilisant la redondance matérielle sont également très utilisées pour fiabiliser les applications sur FPGAs SRAM. La plus connue et l'une des plus simples est l'architecture dite "Triple Modular Redundancy" [LIM03].

- Redondance Triple Modulaire (« Triple Modular Redundancy », TMR)

Le principe de l'architecture TMR est de répliquer deux fois les modules combinatoires et séquentiels d'une application et de faire un vote majoritaire sur les sorties, comme illustré Figure 32. Le surcoût surfacique de cette technique est de l'ordre de 200%, cependant les performances de l'application sont conservées (faiblement diminuées). En effet, seul le temps de propagation à travers le circuit "voteur" pénalise le chemin critique de l'application.

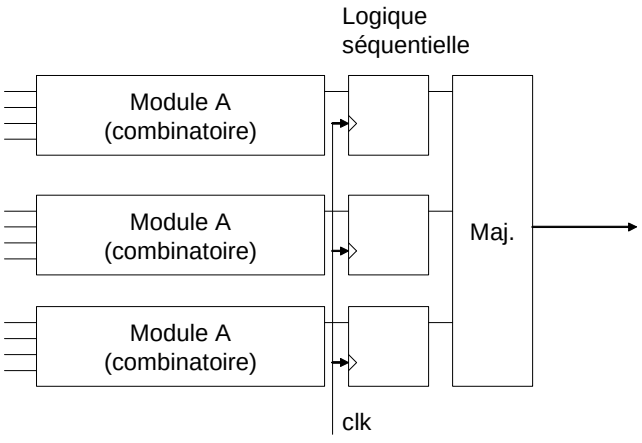

Figure 32: Architecture TMR.[LIM03].

- Redondance Triple Modulaire et rafraîchissement des éléments mémoires utilisateurs.

L'inconvénient majeur de l'architecture TMR, d'un point de vue fiabilité pure, est qu'une faute dans le "voteur" mettra en échec le système de protection. Il est, de fait, intéressant de tripliquer également les parties séquentielles et les "voteurs" comme le montre la Figure 33. Il y a en plus un risque d'accumulation d'erreurs. Le fait de disposer d'une valeur fiable en sortie de registre autorise un rafraîchissement des éléments mémoire si une faute est détectée. Cette technique est employée dans

l'application spécifique de la méthodologie TMR sur les architectures XILINX Virtex, l'architecture nommée XTMR.

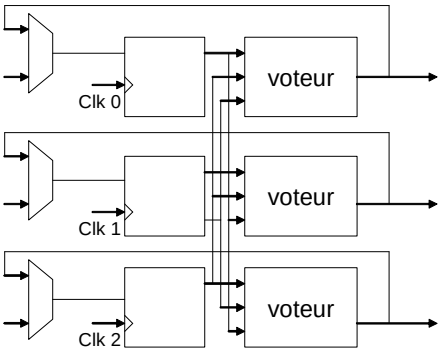

Figure 33: Architecture TMR pour les éléments mémoires [LIM05].

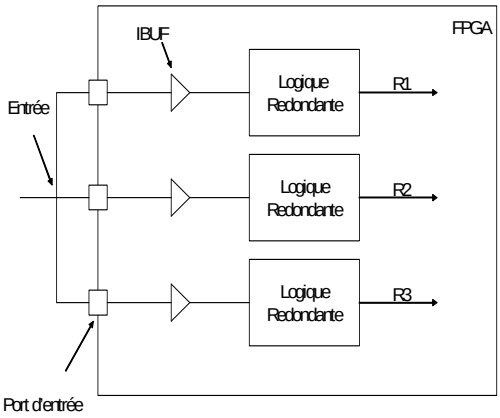

Figure 34: Implantation du schéma XTMR pour les entrées [BER04].

- Implantation spécifique de l'architecture TMR, « Xilinx Triple Modular Redundancy » (XTMR)

Outre le problème d'accumulation des erreurs mentionné précédemment,, chaque bloc répliqué dans une architecture TMR classique utilise un jeu d'entrées commun. Ainsi une faute dans la configuration d'un bloc d'entrée/sortie se propagera dans chaque réplique des blocs logiques et invalidera le système complet de protection.

C'est pourquoi il est important que chaque module répliqué ait son propre jeu d'entrées, et utilise des blocs d'entrées/sorties indépendants, comme l'illustre la Figure 34. Cette technique a été mise en place pour l'application TMR pour les FPGAs Xilinx Virtex [BER04].

- Redondance Modulaire Triple à temps partagé (« Time Shared Triple Modular Redundancy », TSTMR)

Des architectures combinant la redondance temporelle et matérielle ont été développées pour s'adapter spécifiquement aux problèmes liés aux applications arithmétiques. L'architecture TSTMR en est un bon exemple [ALA92]. Cette technique utilise de la redondance matérielle pour corriger les erreurs, en appliquant le même principe que l'architecture TMR. La redondance temporelle est, elle, employée afin d'obtenir le résultat final à partir des résultats obtenus sur chaque module répliqué. En effet, l'idée est de diviser chaque opérande et de leur appliquer trois fois le même traitement. La taille des blocs répliqués est ainsi trois fois plus petite que si l'on répliquait l'application entière. La Figure 35 illustre l'application de cette méthode sur un additionneur 12 bits séparé en trois additionneurs de 4 bits chacun. A chaque calcul partiel, la valeur correcte est déterminée par un "voteur" majoritaire. Ces calculs sont effectués à trois instants différents (t1, t2, t3), en utilisant une partition du jeu d'entrées de l'additionneur initial. Le principe de redondance temporelle est utilisé pour obtenir le résultat final. Sur

le même principe que l'architecture TMR, le système est sûr tant qu'au moins deux blocs fonctionnent correctement.

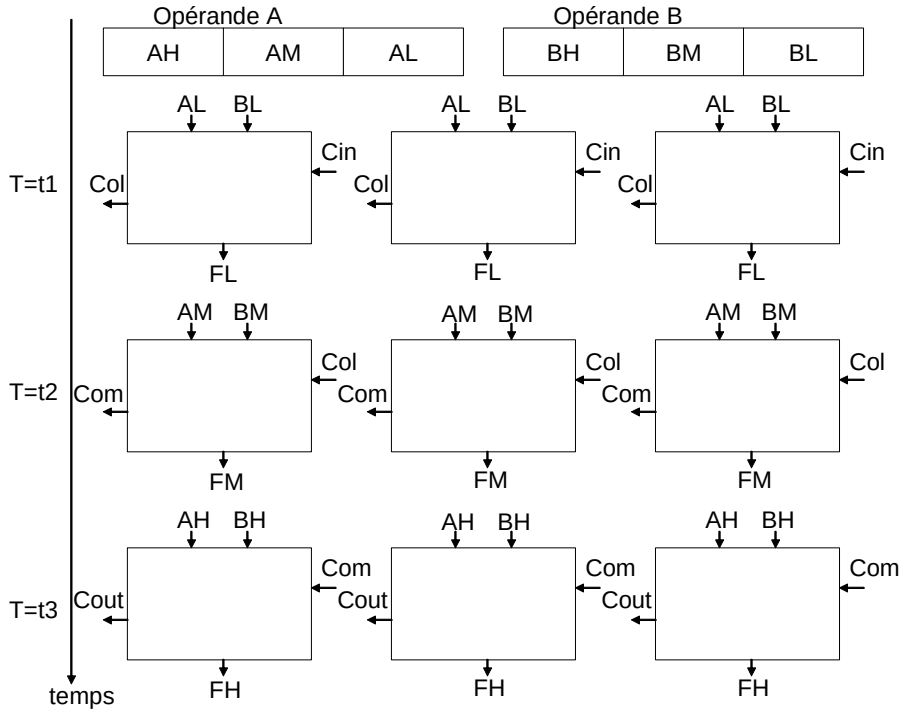

Figure 35: Architecture TSTMR pour l'implémentation d'une UAL résistante aux fautes [ALA92].

On peut citer d'autres travaux utilisant le même principe, par exemple [CSA03], avec l'architecture « Time-Triple Modular Redundancy ».

- Différents niveaux de granularité pour les architectures TMR

Des travaux ont été effectués afin d'améliorer l'efficacité des techniques TMR, techniques notamment très sensibles aux fautes induisant un court-circuit entre deux modules répliqués. Une telle faute compromet deux modules sur trois et entraîne une valeur incorrecte en sortie du module voteur majoritaire. L'idée est donc de partitionner les blocs redondants, en incluant des voteurs entre chaque partition des blocs. Ceci permet en effet (voir Figure 36) de s'affranchir de nombreux courts-circuits (par exemple les exemples a et b). Le seul cas non traité par cette technique est un court-circuit créé entre deux blocs d'une même partition.

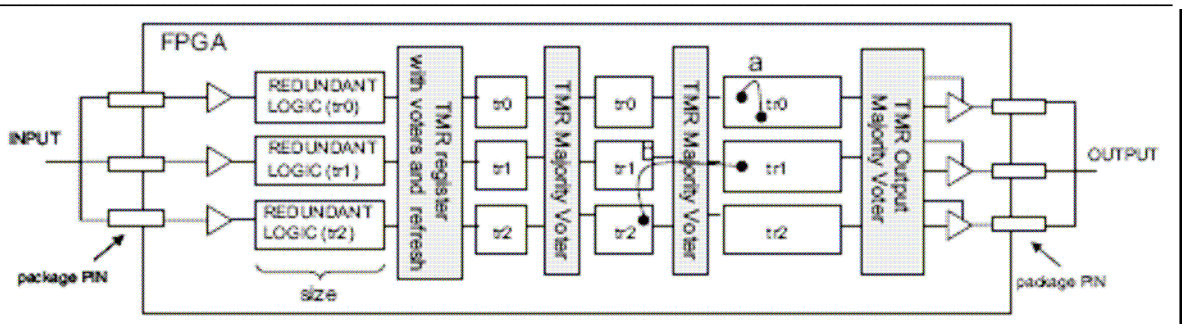

Figure 36: Architecture TMR partitionnée [LIM05].

Il y a donc un compromis à trouver entre la réduction de la taille des partitions logiques (afin de masquer un maximum de fautes possible) et le nombre de "voteurs" ajoutés. Ceux-ci représentent en effet le surcoût principal (en termes de surface et de chemin critique) de cette méthode par rapport à une implantation TMR classique.

- TMR et optimisation du processus de reconfiguration

D'autres techniques permettent d'améliorer encore le rendement et la couverture de fautes des architectures TMR. Ainsi, si l'on est capable de dire quelle réplique ou quel "voteur" propage une valeur fausse, il est possible d'utiliser les mécanismes de reconfiguration partielle (voir paragraphe 2.2.1.2.) de manière sélective et précise afin d'améliorer encore la robustesse des implantations de l'architecture TMR sur FPGAs à mémoire SRAM. De nombreux travaux ont été publiés dans ce sens, parmi lesquels [BOL07] (Figure 37).

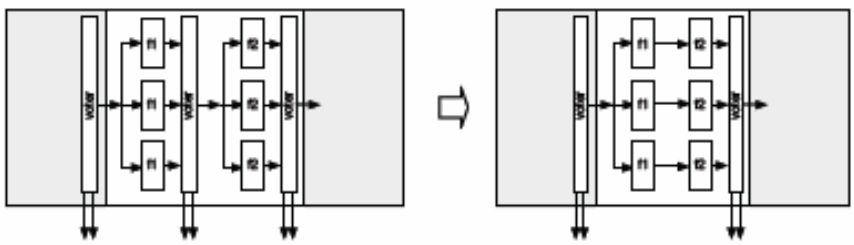

Figure 37: Répartitions des voteurs par « frames » (Xilinx) [BOL07].

A cette fin, l'application initiale doit être partitionnée en modules Mi afin de leur appliquer l'architecture TMR. Un FPGA à mémoire SRAM étant lui même partitionné en zones reconfigurables, il est possible d'affecter à chacune de ces zones Fi une ou plusieurs partitions Mi de l'application et leurs "voteurs". Quand une faute est détectée dans un module Mi ou un "voteur" Vi, la zone correspondante Fi du FPGA est reconfigurée à l'aide d'une portion de bitstream Bi. En jouant sur le placement des modules Mi et de leurs "voteurs" Vi parmi les zones reconfigurables Zi (c'est à dire. en jouant sur leur implantation et leur nombre dans chaque zone reconfigurable), il est possible d'améliorer sensiblement la robustesse de l'implantation globale de l'application.

Il est généralement inutile de placer plusieurs partitions Mi et "voteurs" Vi à l'intérieur d'une zone configurable, puisque celle-ci est la plus petite entité reconfigurable. Dans le cas contraire, des éléments sains et fonctionnels de l'application seraient reconfigurés inutilement.

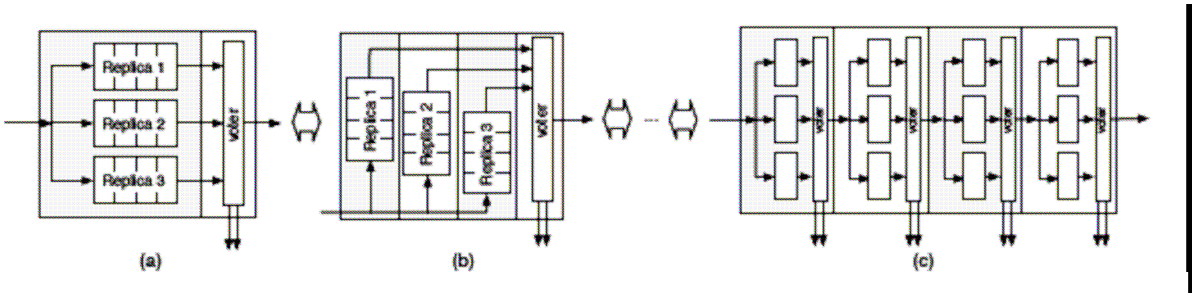

Figure 38: Différents niveaux de granularité dans les implantations TMR avec reconfiguration [BOL07].

Le fait d'augmenter la granularité du système implique des surcoûts supplémentaires pour localiser les partitions fautées du fait de l'augmentation du nombre de "voteurs". La diminuer entraîne une baisse de la robustesse de l'architecture comme expliqué auparavant.

La Figure 38 montre quelques possibilités de placement pour les FPGAs Xilinx Virtex. Les zones reconfigurables minimum pour ce type de réseau sont les « frames ». Chaque sous-système doit être contenu à l'intérieur de cet espace adressable de la matrice de configuration.

- Amélioration des architectures TMR à l'aide de chaînes de scan

La technique proposée dans [GER07] vise également à détecter les fautes dans la mémoire de configuration et à les corriger en utilisant la reconfiguration partielle. La méthodologie TMR est employée pour masquer les fautes tandis qu'une quatrième partition du FPGA implémente un contrôleur afin de détecter quel module est fauté et de piloter le processus de reconfiguration partielle en cas de faute(s). La localisation des fautes est réalisée à l'aide de chaînes de scan. Le schéma de principe est présenté Figure 39.

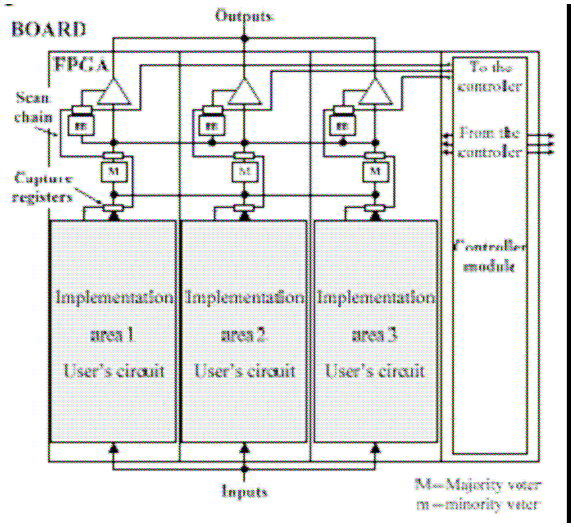

Figure 39: Utilisation de chaînes de scan pour l'optimisation de la reconfiguration dans une architecture TMR [GER07].

Le contrôleur implémenté dans la quatrième partition est représenté Figure 40. Il génère les signaux de contrôle [illegible] valeurs échantillo[illegible] on de la partition

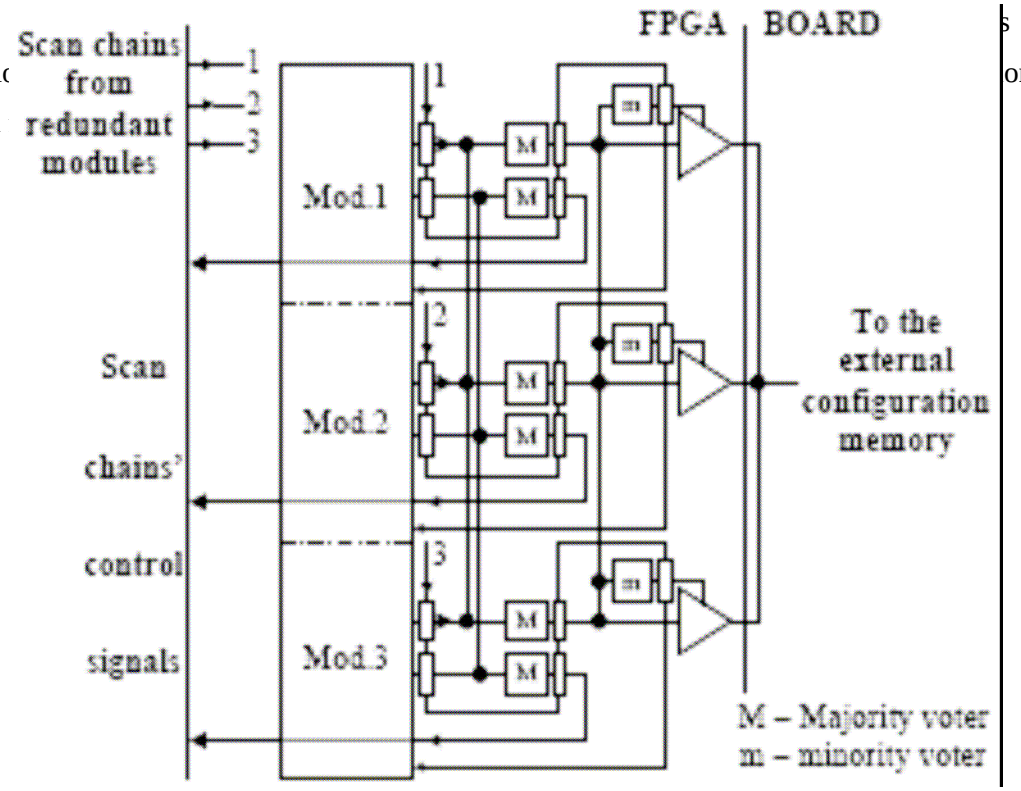

Figure 40: Module de contrôle des chaînes de scan [GER07].

- Duplication et Comparaison associé à une détection simultanée

Le principal inconvénient des variations de l'architecture TMR est son surcoût surfacique. Avec une augmentation de la surface occupée (donc des ressources logiques employées) incompressible de plus de 200%, l'architecture TMR est en effet très coûteuse. Ce coût se traduit aussi par une forte consommation du système. D'autres techniques de redondance ont ainsi été étudiées, n'utilisant qu'une seule réplique de la logique originale, plutôt que deux, mais en complétant cette duplication qui ne peut assurer que la détection au lieu de la tolérance des fautes. La technique de duplication et comparaison a été employée dans [LIM03]. Cette architecture (Figure 41) présente deux répliques de la logique combinatoire de l'application initiale, associées chacune à un module de détection de fautes qui doit en déterminer l'origine. La détection est effectuée à l'aide de modules de détection simultanée d'erreurs (CED, voir paragraphe suivant).

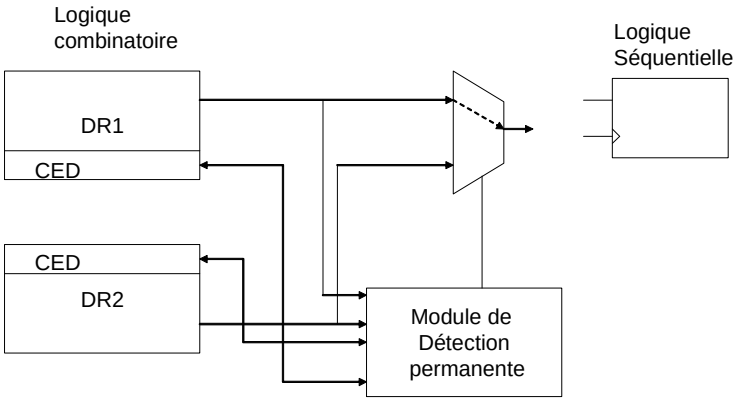

Figure 41: Modèle d'implantation du schéma détection par comparaison associé à une détection simultanée [LIM03].

- Routage redondant

Les FPGAs à mémoire SRAM utilisent plusieurs types de structures de mémoire de configuration. Leur point commun est que 90% de leurs ressources de configuration sont utilisées pour la programmation des ressources de routage du FPGA. En se basant sur l'analyse des effets produits par les SEUs dans cette partie de la mémoire de configuration, des méthodes de fiabilisation utilisant des chemins de données redondants entre modules logiques ont été développées [FIL05], l'objectif étant d'obtenir un routage préservant l'intégrité des signaux même en cas de faute. Les Figure 42 et Figure 43 présentent deux exemples d'implémentation d'un tel dispositif pour des FPGAs Xilinx Virtex II, respectivement dans le cas d'un court-circuit et d'un circuit ouvert.

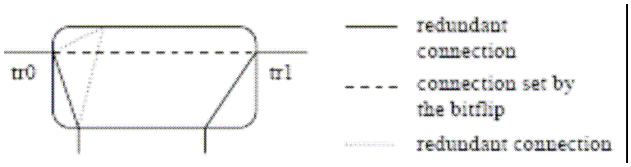

Figure 42: Exemple d'utilisation de routage redondant dans le cas d'un court-circuit [FIL05].

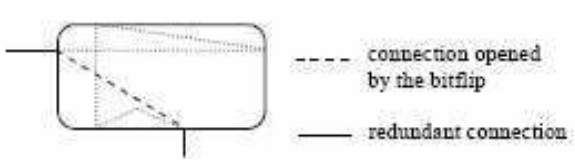

Figure 43: Exemple d'utilisation de routage redondant dans le cas d'un circuit ouvert [FIL05].

- Utilisation d'algorithmes de placement routage pour la fiabilisation du placement d'architectures TMR sur FPGAs à mémoire SRAM

D'autres techniques abordent le problème en amont de la phase de routage, en considérant la fiabilisation, non pas du routage des signaux de l'application, mais du placement intégral de l'application sur le réseau programmable. De nombreux algorithmes ont été développés dans cet esprit, nous parlons ici de [STE06] à titre d'exemple.

La première étape de ces méthodes est l'établissement d'un modèle générique des FPGAs à mémoire SRAM (Figure 44). Pour cela, on définit trois types de ressources typiques de ces plateformes : les

cellules (blocs) logiques, les segments (fils) de routage, et les matrices d'interconnexion. A partir de cette modélisation, un graphe de routage du FPGA (Figure 45) est créé afin de représenter l'utilisation particulière des ressources du réseau par une application donnée. Cette étape est fondamentale avant de procéder à une analyse de la robustesse de l'implémentation d'une application sur le composant.

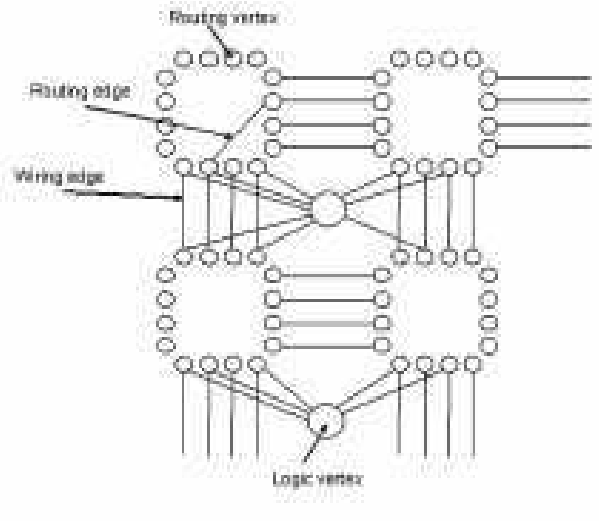

Figure 44: Modélisation du FPGA en arbre [STE06].

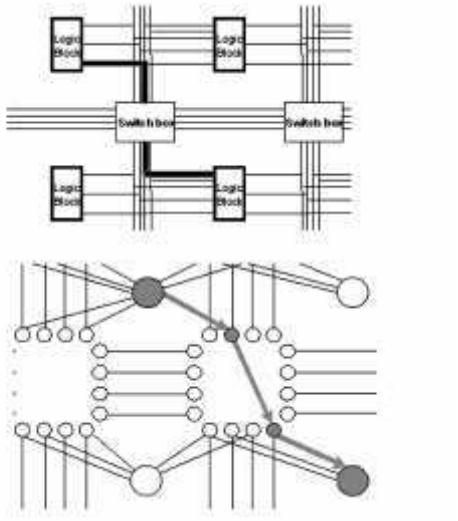

Figure 45: Graphe de routage du FPGA pour une application donnée [STE06].

Les cellules logiques sont représentées par des nœuds logiques tandis que les matrices d'interconnexion sont modélisées par les nœuds de routage et leurs liaisons sur le graphe. Ces liaisons correspondent à l'activation d'une connexion dans le circuit, par exemple à un PIP sur les Xilinx Virtex II. Le nombre de ressources varie en fonction de l'architecture de la famille de FPGA considérée et de leur taille.

A partir de cette modélisation de l'architecture du FPGA et de l'utilisation qui en est faite pour une application donnée, des contraintes peuvent être établies afin de garantir la fiabilité de l'architecture TMR que ce soit pour les SEUs ou les erreurs multiples. Elles doivent permettre d'éviter principalement deux types d'erreurs :

- ✓ Un court-circuit entre deux répliques,
- ✓ Un circuit-ouvert pour une connexion entre modules répliqués.

Les performances observées pour les implantations utilisant ces algorithmes sont légèrement moins bonnes que pour les architectures TMR classiques, mais sans surcoût surfacique. Leur couverture de fautes est aussi nettement améliorée.

- ➢ Influence du codage des machines à états

Des travaux [CAS06] ont été effectués afin d'étudier la sensibilité aux fautes intrinsèque des machines à états (FSMs) en fonction de leur codage. Ils mettent en évidence d'une part les performances relatives obtenues en fonction du codage, ainsi que leur robustesse à travers des expériences

d'injections de fautes. Le tableau 2 présente les caractéristiques des trois types de codage utilisés, le nombre de LUTs utilisés, le nombre de bascules utilisées, et enfin la fréquence maximale observée. Ces expériences ne mettaient pas en œuvre des techniques de fiabilisation.

Tableau 2: Caractéristiques des circuits en fonction du codage des FSMs.

Codage FSM	LUTs	Bascules	Fréquence maximale
Binaire	4	3	537.762 MHz
Gray	37	6	214.018 MHz
1 parmi N	10	8	366.703 MHz

Le tableau 3 rapporte les résultats des injections de fautes réalisées. Alors que le codage "1 parmi N" a la zone sensible la plus grande pour une application donnée (plus grand nombre de bascules), il présente également le plus faible nombre de réponses erronées, donc la meilleure robustesse aux fautes. Ceci s'explique par le plus grand nombre d'états inutilisés induits par ce codage, qui ne doivent pas stopper le déroulement normal de l'application dans une conception normale.

Tableau 3: Résultats des injections de fautes.

Codage FSM	Fautes injectées	Réponses correctes	Réponses erronées
Binaire	660	154	506 (76%)
Gray	1320	374	946 (71.6%)
1 parmi N	1760	1156	704 (40%)

2.3. Conclusion

L'état de l'art concernant l'analyse de sensibilité des composants électroniques, et plus particulièrement des FPGA SRAM, a mis en évidence dans la plupart des cas la nécessité de disposer d'un prototype physique du système. Les méthodes employées en phase de conception sont essentiellement fondées sur des injections de fautes, par simulation ou par émulation. Elles nécessitent une approche de test dynamique, ce qui induit des durées expérimentales pouvant être très longues. Il manquait une approche complémentaire, permettant de réaliser une analyse de robustesse peut être plus pessimiste que l'analyse dynamique, mais permettant d'avoir des résultats très rapides pour permettre au concepteur d'étudier l'impact de certains choix d'implantation. C'est le but des analyses statiques développées ces dernières années et présentées dans les deux chapitres suivant.

La fiabilisation (ou le durcissement) au niveau applicatif de circuits implantés sur des FPGAs à mémoire SRAM peut suivre plusieurs approches. Lorsque des FPGA du commerce doivent être employés, il n'est pas possible de chercher à durcir leur structure. On aura remarqué que la majeure

partie des techniques de protection se base encore sur l'architecture TMR et ses déclinaisons, entraînant toujours un surcoût important au niveau matériel et consommation. Le dernier chapitre présentera une nouvelle méthode de protection basée sur l'analyse de sensibilité de la mémoire de configuration pour un durcissement sélectif et à faible coût (voire à coût nul) des zones mémoires les plus critiques.

3. Etude prédictive de criticité sur la technologie XILINX

Nous avons abordé dans le chapitre précédent les techniques usuelles permettant d'analyser l'effet des perturbations dans les FPGAs à mémoire SRAM, en rappelant les grandes lignes des techniques de tests grandeur nature, mais aussi des tests réalisés par irradiation ou encore par injection de fautes. Nous allons ici présenter une nouvelle méthodologie d'analyse du comportement des réseaux programmables Virtex II de Xilinx face aux problèmes liés à des erreurs dans la mémoire de configuration, et plus particulièrement face à des SEUs. Les résultats présentés par la suite se basent sur l'analyse de campagnes d'injections de fautes réalisées au laboratoire IMS de Bordeaux d'une part, et au laboratoire LETI CEA de Grenoble d'autre part.

3.1. Définition des objectifs

La première étape de ce travail consiste à déchiffrer le rôle affecté à chaque bit dans la mémoire de configuration, afin de pouvoir comprendre les éventuelles modifications architecturales engendrées par une faute, ainsi que les conséquences sur le fonctionnement de l'application implémentée. Cette tâche a été rendue possible par l'utilisation des classes JAVA JBITS fournies par XILINX pour le FPGA Virtex II.

La deuxième partie de ce chapitre présentera les méthodologies pour la caractérisation de la sensibilité aux fautes de la mémoire de configuration, en se basant sur les données regroupées et analysées dans la première partie, et sur l'analyse d'injections de fautes laser.

3.2. Méthodes d'analyse du contenu de la mémoire de configuration XILINX

3.2.1. JBIT

3.2.1.1. Introduction

JBITS est un ensemble de classes Java créées spécialement par Xilinx pour accéder et modifier la configuration de certaines familles de leurs FPGAs. Ces classes permettent de faire des lectures et des écritures dans la configuration alors que le FPGA est en fonctionnement. La version 2.8 accepte les FPGAs des familles SPARTAN et Virtex et la version 3.0 accepte la famille Virtex II. Nous allons utiliser la version 3.0 pour effectuer la mise en correspondance des bits de la matrice de configuration avec les ressources qu'ils codent ce qui permettra de les classer selon les critères qui seront vus dans la partie architecture. Cette version 3.0 des classes va aussi nous servir à faire l'injection de fautes grâce à des reconfigurations partielles de la matrice. La version 3.0 est orientée architecture, c'est-à-dire que ses classes ne permettent pas de modifier directement un bit précis de configuration. On fournit aux classes la connexion ou la configuration de l'élément souhaité et celles-ci modifient les bits

correspondants. Cette méthode limite les possibilités de modification du bitstream mais empêche de créer des bitstreams détériorant la puce (impossible de créer des courts-circuits). Nous allons décrire la procédure utilisée pour l'analyse des bits de configuration et pour simuler l'injection d'une faute grâce à JBITS. Ensuite, nous ferons une description plus précise des classes utilisées dans le logiciel.

3.2.1.2. Flot de conception

JBITS peut être utilisé pour la conception de circuits sur les FPGAs Xilinx Virtex II (ou antérieurs), ou plus généralement pour modifier des parties pré-routées par les composants de la suite Xilinx ISE. Contrairement aux flots de conception usuels qui prennent en entrée un fichier VHDL ou Verilog contenant une description comportementale du circuit, ce flot travaille à partir d'un bitstream généré par ISE au format .bit. Ce fichier est traité à l'aide d'un programme Java basé sur les classes de la suite JBITS, qui commence par extraire toutes les données de configuration du FPGA. Les données sont ensuite modifiées à l'aide d'autres méthodes JBITS (détaillées plus loin) puis réécrites sous forme de bitstream qui pourra programmer une carte Virtex II.

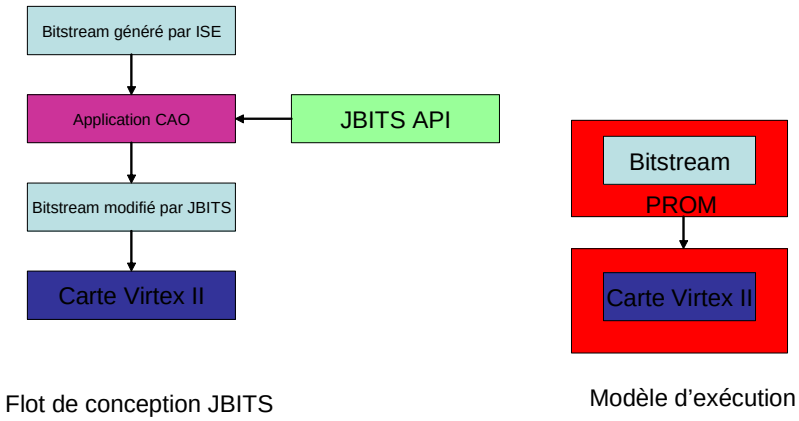

Figure 46: Flot de conception JBITS.

Le modèle d'exécution d'un tel flot est simple puisqu'il suffit de charger le bitstream ainsi généré sur la carte à partir d'une mémoire type EPROM ou PROM (voir Figure 46).

3.2.1.3. Flot de reconfiguration en ligne

JBITS peut également être utilisé dans un cadre dynamique, c'est-à-dire pour la reconfiguration en ligne du composant en cours de fonctionnement. On utilise alors une interface qui peut être implémentée avec les classes Java Xilinx XHWIF afin de télécharger le bitstream sur une carte comportant le composant visé. Le circuit initial est modifié à partir d'une application programmée à l'aide des classes JBITS comme dans l'exemple précédent.

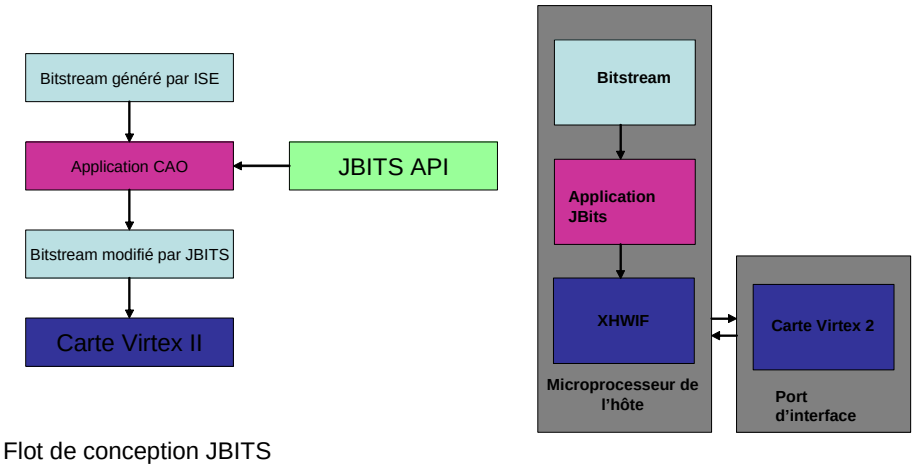

Figure 47: Reconfiguration en ligne avec JBITS.

Le modèle d'exécution est illustré sur la Figure 47. L'ordinateur hôte, qui configure l'application programmée à l'aide de JBITS, communique avec la carte Virtex II connectée à son port PCI grâce à l'interface XHWIF. Ce dispositif permet la reconfiguration en ligne de la carte.

3.2.1.4. Modèle de programmation

Pour utiliser les classes JBITS, il faut charger les informations relatives à la configuration du FPGA dans un objet JBITS. On doit lire un fichier de configuration bitstream .bit.

La procédure est alors toujours la même, comme le montre la Figure 48 :

- On instancie un objet JBITS en précisant le composant utilisé (Virtex II 40, 80, …,8000)
- On remplit les champs de cet objet grâce à la lecture d'un fichier .bit
- On peut ensuite utiliser tous les outils développés dans JBITS pour analyser ou modifier le bitstream
- Dans le cas où une modification a été effectuée, on peut sauvegarder l'objet JBITS dans un fichier .bit

La façon d'instancier les objets JBITS oblige à lire un fichier bitstream pour chaque nouvel objet. Pour que le logiciel effectue les tâches demandées dans un temps raisonnable, il faut limiter au maximum le nombre d'objets JBITS instanciés.

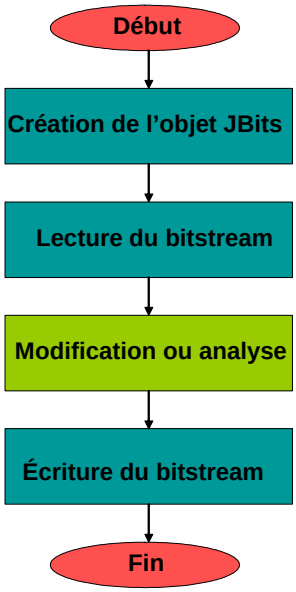

Figure 48: Modèle de programmation JBITS.

3.2.2. Protocole d'analyse

3.2.2.1. Lecture du bitstream

Comme indiqué précédemment, la lecture et le stockage des informations du bitstream se fait par l'intermédiaire de l'objet JBITS. Cette lecture et ce stockage sont totalement transparents pour l'utilisateur.

Voici le déroulement de la lecture :

- On instancie l'objet Device correspondant au modèle de Virtex II utilisé :

Device device = Device.getDevice(Device.XC2V2000) ;

- On instancie l'objet JBits correspondant au modèle de Virtex II utilisé en précisant le composant :

JBits jbits = new JBits(device) ;

- On remplit les champs de l'objet JBits grâce aux informations du bitstream :

jbits.read("infile.bit") ;

- On initialise la classe Wire qui va servir pour l'étude de l'architecture :

Wire.setJBits(jbits);

3.2.2.2. Lecture et modification de la configuration des ressources d'architectures

Cette partie donne un aperçu de l'emploi des méthodes des classes JBITS pour modifier l'architecture implémentée sur le FPGA. On parlera principalement des classes Wire, Logic, et JRoute.

- Méthodes de la classe Wire

Cette classe permet d'instancier un objet représentant n'importe quel fil du composant. L'objet contient les connexions possibles ainsi que les connexions actives dans une configuration donnée. Chaque objet Wire contient un champ CONFIG indiquant l'ID (un entier unique pour chaque fil) de la source connectée à ce fil et une liste de champs contenant les ID des sources connectables à ce fil.
Pour instancier, il faut préciser les coordonnées de la tuile du fil ainsi que son numéro d'index. Il existe une constante contenant le numéro d'index de chaque fil d'une tuile, les noms des constantes se trouvent en nomenclature (annexes) .Par exemple, la constante Wire.E6BEG4 est l'index du début du fil Hex Est n° 4.

Voici le code pour connecter un fil :

- on instancie l'objet Wire représentant le début du fil Hex Est 3
  - Wire wire = lookup.getWire(tileRow, tileCol, Wire.E6BEG3);
- on le connecte au milieu du fil Hex Sud 3, l'objet JBit est directement modifié
  - jbits2.setCLBBits(0,0,wire.getResource(), wire[i].S6MID3);

- Méthodes de la classe Logic

La classe Logic contient toutes les constantes permettant de lire et de modifier la configuration des éléments fonctionnels internes aux blocs CLB. De même que pour la classe wire, pour accéder à la configuration d'un de ces éléments il faut fournir les coordonnées du CLB et l'index de l'élément. Par contre, il n'est pas nécessaire d'instancier un objet pour modifier la configuration de ces éléments. Par exemple, la constante FXMUX.CONFIG[0] est le numéro d'index du multiplexeur d'entrée de la bascule X du slice 0.
La classe Logic contient aussi l'index de toutes les configurations possibles de chaque élément. La constante FXMUX.F5 est le numéro d'index à fournir pour relier l'entrée d'une bascule X au fil F5. La nomenclature de ces constantes est précisée dans l'API de JBits**.**

Voici les deux fonctions de base pour la manipulation de la configuration de ces éléments :

- On récupère le numéro d'index de la configuration de l'entrée de la bascule X
  - int config = jbits.getCLBBits(0, 0, FXMUX.CONFIG[0]);

- On configure le multiplexeur FX du slice 0 pour qu'il route en sortie le signal présent sur son entrée F5
  - jbits2.setCLBBits (0, 0, FXMUX.CONFIG[0], FXMUX.F5);

- L'utilisation de la classe JRoute

Cette classe permet d'obtenir le « chemin » que suit un signal dans le composant. Elle fournit la liste des ressources d'interconnexion par lesquelles le signal passe.
Cette classe permet d'émettre des rapports de ce type :

*IOB.I1[BOTTOM][20]
. TILE.N2BEG8[1][24]
. . [CLB.N2END8[1][20]]
. . . CLB.E2BEG9[1][20]
. . . . [CLB.E2MID9[1][21]]
. . . . . *CLB.F1_B2[1][21]
. . . . . . *CLB.CLB_CY0F_S2[1][21]

Chaque ligne représente un fil par lequel passe le signal (la nomenclature de leurs noms est donnée dans la partie suivante).
Les fils précédés d'une « * » se trouvent à l'intérieur d'un bloc (IOB, CLB ou BRAM).
Les fils entre « [] » sont reliés au fil précédent quelle que soit la configuration du FPGA (ces connexions ne sont pas contrôlées par des bits de configuration).
Explication détaillée :
Le signal "trace" rentre dans le circuit par le fil I1 de la tuile IOB de coordonnée tuile 1,24. Il passe ensuite dans le fil Double Nord n°8 jusqu'à la tuile de coordonnée CLB 1,20. Il passe ensuite dans le fil Double Est n°9. Dans le CLB 1,21 il rentre dans l'entrée n°1 du LUT F du slice 2 ainsi que dans le fil CYOF grâce à un multiplexeur interne.

Le code suivant permet d'afficher le rapport précédent une fois que la classe Wire a été initialisée :

- On instancie un objet Pin représentant le point I1 dans le bloc IOB bottom 20

```
Pin pin = new Pin(Pin.IOB,Device.BOTTOM,20, Wire.I1);
```

- On instancie l'objet RouteTracer qui va tracer le signal
  - Lookup lookup = Lookup.getLookup();
  - tracer = new RouteTracer(Lookup.getLookup());

- On affiche le rapport dans la console pour le point « pin »
  - System.out.println(tracer.trace(pin));

3.2.2.3. Décodage du bitstream

On a vu en 1.3.1.3. que la zone de configuration est divisée en plusieurs blocs, eux-mêmes divisés en plusieurs zones. La zone la plus importante est la zone de configuration des CLBs, située dans le bloc 0, à partir des adresses majeures 2.

Maintenant que l'on sait comment modifier les éléments architecturaux du Virtex II, nous allons détailler les méthodes permettant de déterminer la taille de la zone de configuration de chaque CLB, et de connaître l'agencement général de la zone de configuration CLB à l'intérieur du bitstream, ainsi que le rôle de chaque bit à l'intérieur de cette zone.

Les résultats et données extraits seront ensuite présentés dans la partie suivante.

- Transformation en fichier texte du bitstream

Une fois que l'objet JBITS a été instancié et initialisé, il est possible grâce à la fonction « generateFull() » de produire un tableau d'octets représentant les données du fichier bitstream. C'est cette fonction qui nous a permis de trouver la taille des bits d'une tuile CLB dans la matrice de configuration, ainsi que la position du premier bit de chaque tuile.

Pour obtenir la taille (en bits) d'une tuile CLB, la méthode consiste à changer la configuration du même composant sur deux CLB d'affilée sur une ligne et une colonne (voir figure 23). Pour cela, on utilise la méthode setCLBBits(). Les modifications effectuées apparaissent dans le tableau d'octet et donnent la taille du tableau de bit.

Une fois qu'un objet JBITS est initialisé avec un bitstream vide, le code suivant implémente la méthode précédente:

- On modifie la configuration du multiplexeur CEINV dans les CLB (55,0) ;(55,1) et (54,0)
  - jbits.setCLBBits(55, 0, CEINV.CONFIG[0], CEINV.CE);
  - jbits.setCLBBits(55, 1, CEINV.CONFIG[0], CEINV.CE);
  - jbits.setCLBBits(54, 0, CEINV.CONFIG[0], CEINV.CE);
- on instancie le tableau d'octets
  - Byte[] octettab= jbits.generateFull() ;
- On l'imprime dans un fichier en séparant l'entête et les différentes frames ; pour cela on insère un retour à la ligne à chaque frame (146*4 est le nombre d'octet dans une frame pour la Virtex 2000)
  - for (int i=0; i< octettab.length;i++)
    - if (i==146*4) out.println();

- On crée un tableau de huit 0 ou 1 avec chaque octet (i est l'index de l'octet) et on l'imprime dans le fichier

  - int []s = Util.IntToIntArray((int)(octettab [i]), 8);
  - for (int m=0;m<s.length;m++)
    - out.print(s[m]);

On obtient un résultat illustré par la Figure 49.

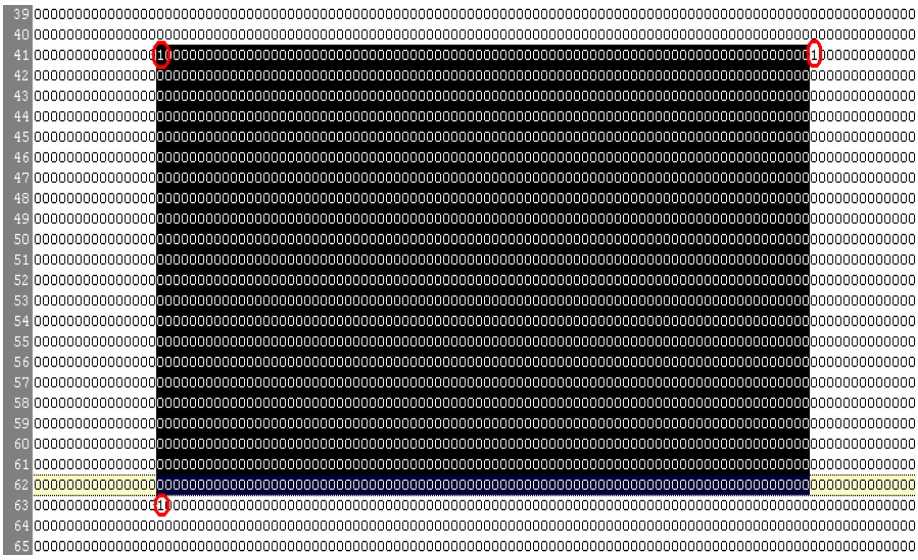

Figure 49: Résultat d'une modification du bitstream pour le décodage.

Les bits cerclés de rouge sont ceux qui ont été modifiés. Les bits surlignés correspondent à une tuile CLB. Le CLB fait donc 22 frames de large et 80 bits de hauteur.

La méthode pour trouver la position et la fonction du premier bit de chaque CLB est plus laborieuse. Il faut modifier la configuration de chaque élément de la tuile CLB pour trouver le bit qui a les coordonnées les plus faibles. Ces informations ont été obtenues par tâtonnement, il n'y a pas de code précis qui donne cela. Il s'avère que le premier bit de chaque CLB est le bit qui configure le multiplexeur SOPEXTSEL du slice 1 (champ SOPEXTSEL.CONFIG[1]).

- Algorithmes pour le décodage

Une fois que la taille et le premier bit du tableau de bits correspondant à un CLB ont été identifiés, il faut identifier le rôle de chaque bit de ce tableau. Pour cela, on instancie deux objets JBITS avec le même bitstream vide (toutes les tuiles en configuration par défaut). Grâce aux méthodes de la classe

Wire ou Logic (setCLBBits()), on crée une connexion ou l'on configure un élément sur l'un des deux objets JBITS. On convertit ensuite ces deux bitstreams en tableaux d'octets. On les compare bit à bit, les bits qui ont changé sont les bits qui contrôlent la connexion modifiée. On trouve ensuite ces coordonnées grâce aux informations dont on dispose sur la taille d'un CLB et son premier bit (voir Figure 50). Pour rendre cette méthode exhaustive, il faut avoir préalablement déterminé toutes les connexions possibles dans un CLB.

Pour chaque élément dont on veut trouver les bits de configuration, il faut ré-instancier les objets JBits puis lancer la comparaison. Cette procédure rend l'exécution de ce programme longue. Le détail du code se trouve dans le fichier « FinalFlash.java » reproduit en annexe.

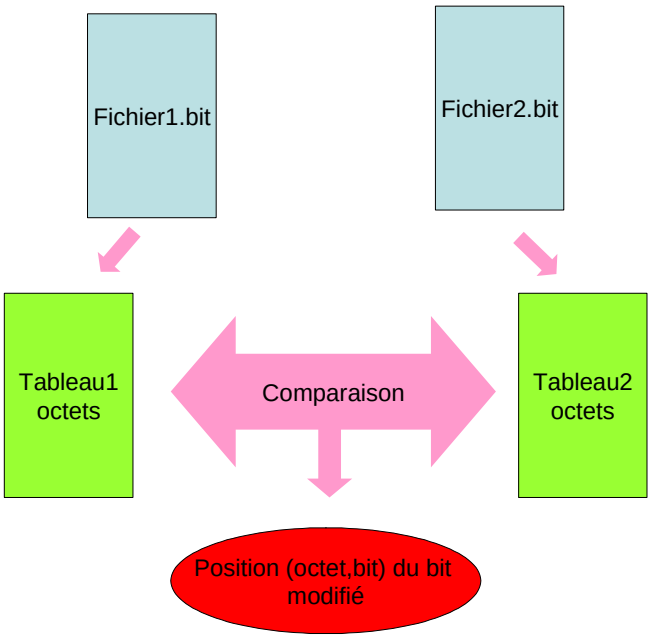

Figure 50: Mise en correspondance adressage/rôle des bits de configuration sur Virtex II.

3.2.3. Détails sur la configuration du Virtex II (point de vue Reverse Engineering)

3.2.3.1. Organisation générale du fichier de configuration

Nous avons vu, dans la partie sur la structure de la mémoire de configuration, que les frames sont lues et écrites dans le sens des adresses croissantes, ce qui veut dire que toutes les frames CLB se trouveront juxtaposées dans un bitstream au format .bit (car les emplacements mémoires correspondant sont jointifs). On va donc pouvoir définir une « zone CLB » dans le bitstream qui ne contient que des bits de configuration relatifs aux CLB (quelques bits pour les IOBs également). Ceci est représenté Figure 51.

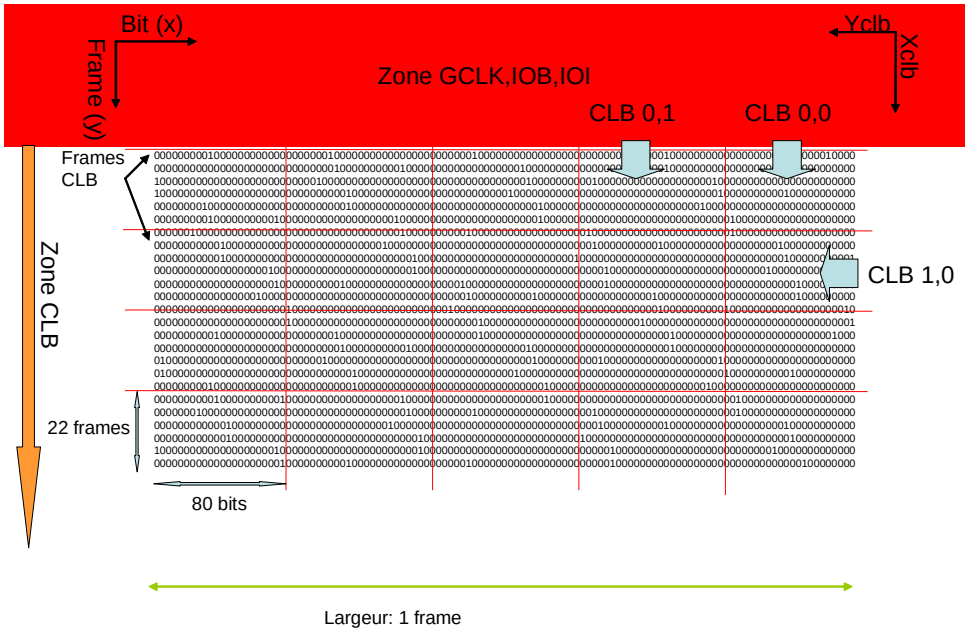

Figure 51: Format de la zone CLB du bitstream.

On peut découper le bitstream en paquets de bits configurant une tuile particulière. Pour la tuile CLB, ce paquet correspond à un rectangle de 22 bits de large (soit 22 frames) pour 80 bits de haut, soit 1760 bits de configuration par CLB.

3.2.3.1. Bits de configuration de la zone CLB

Nous résumons ici les fonctions de la majorité des bits de configuration de la zone CLB. Cette mise en correspondance a été effectuée à l'aide des méthodologies précédemment exposées. Nous les avons partagés entre différentes grandes catégories, qui sont rassemblées dans le tableau 4. On trouve deux grands types de bits de configuration, les bits dits d'interconnexion, et les bits programmant l'ensemble des éléments logiques du CLB. Les bits d'interconnexion configurent le routage des entrées et des sorties du CLB considéré, ainsi que l'accès aux différents bus de routage disponibles pour chaque cellule, les "Double Line", "les "Hex Line", et les "Long Line". Ils sont donc divisés en cinq sous-ensembles correspondant à chaque type de ressource. Les bits configurant les éléments fonctionnels sont eux scindés en trois sous-ensembles. Les bits de "LUT" configurent le contenu et le mode d'utilisation des huit générateurs de fonctions disponibles dans chaque CLB. Les bits "FF/Latch" configurent le mode et l'initialisation des huit éléments mémoires disponibles dans chaque cellule. Enfin, les bits "multiplexeurs internes" gèrent l'ensemble des multiplexeurs présents dans un CLB.

On a représenté sur la Figure 52 ces bits en les associant aux éléments qu'ils contrôlent pour chaque moitié de slice. Un CLB en présente 8 identiques. Le système d'interconnexions n'est lui pas représenté. On retrouve quatre grandes catégories : les bits configurant les éléments fonctionnels, c'est

à dire les bits de "LUT", de "FF/Latch", et de "multiplexeurs externes", et les bits configurant les entrées de slice.

Tableau 4: Bits de configuration et ressources associées.

Type de bits	Types de ressources associées		Détails sur le rôle des bits
Interconnexions	Entrées de slices		Bits de connexion avec les entrées de LUT (G1_B à G4_B et F1_B à F4_B) et BX, BY
	Sorties de slices		Bits de connexion avec X, XQ, XB, Y, YQ, YB pour les 4 slices
	Double line		Bits de connexion avec les Débuts de Double (E2BEG6 par exemple)
	Hex line		Bits de connexion avec les Débuts de Hex (E6BEG6 par exemple)
	Long line		Bits de connexion avec les Long (LH4 par exemple)
Eléments fonctionnels	LUT	Contenu	Les 16 bits de configuration du LUT ou de mémorisation de la ram (selon le mode)
		Configuration	Ces bits donnent le mode du LUT (LUT, RAM ou registre à décalage)
	Multiplexeurs internes		Bits configurant tous les multiplexeurs de routage à l'intérieur du slice
	FF/Latch	Configuration	Ces bits donnent le mode (bascule ou latch) ainsi qui l'état d'initialisation

Les multiplexeurs internes sont configurés par N bits tel que 2^N = M, où M est le nombre d'entrées (N = ln(M)/ln(2)). Deux bits configurent le mode d'utilisation et l'initialisation de chaque élément mémoire. Le contenu de chaque générateur de fonctions est programmé par seize bits, et deux bits configurent leur type d'utilisation (RAM, LUT, Registre à décalage). Les détails concernant la structure de configuration des interconnexions sont rassemblés dans la partie suivante.

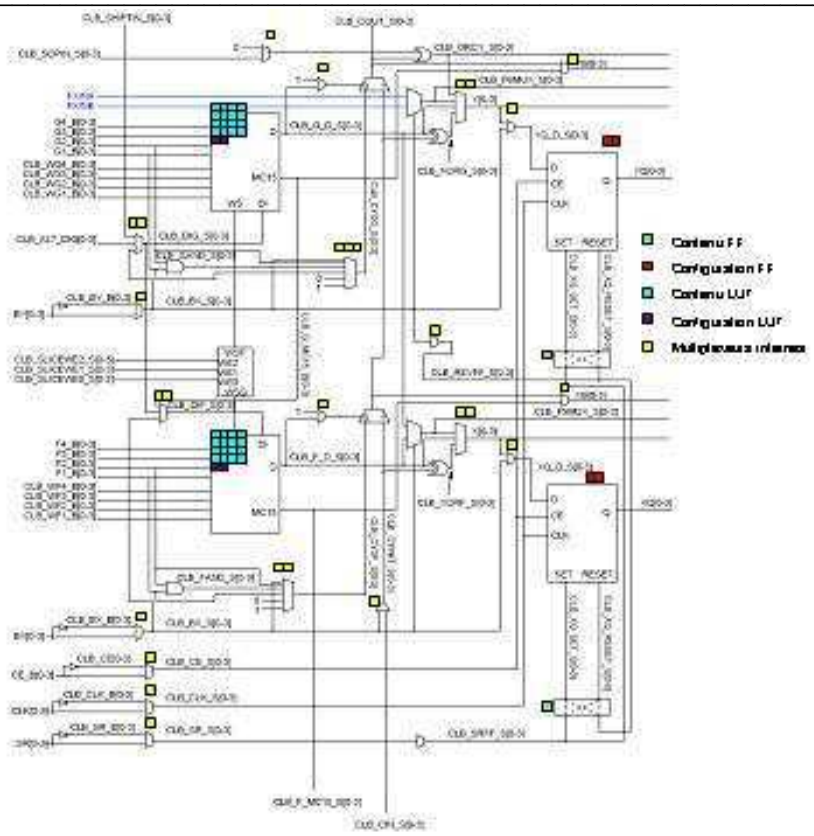

Figure 52: Mise en correspondance des bits de configuration de la logique CLB.

La Figure 53 représente toutes les catégories de bits CLB dans le bitstream. Elle est extraite de nos outils d'analyse pour les FPGA Xilinx Virtex II. On représente la configuration d'une tuile en particulier, qui comme nous l'avons montré précédemment, est comprise dans un paquet de 1760 bits, pouvant être représenté par un tableau de 80 colonnes et 22 lignes. Ces lignes correspondent aux frames du bitstream. La Figure 54 en constitue la légende. Elle est également extraite de nos outils d'analyse. On retrouve toutes les catégories introduites:

- FF/Lacth
  - Configuration de mode (*Config FF*)
  - Configuration de l'initialisation (*Contenu FF*)
- Multiplexeurs internes (*MUX interne*)
- Lut
  - Configuration (*Config LUT*)
  - Contenu (*Contenu LUT*)
- Long Line (*Long*)
- Hex Line (*Hex*)
- Double Line (*Double*)
- Sorties (*Sortie*)
- Entrées (*Entrée*)

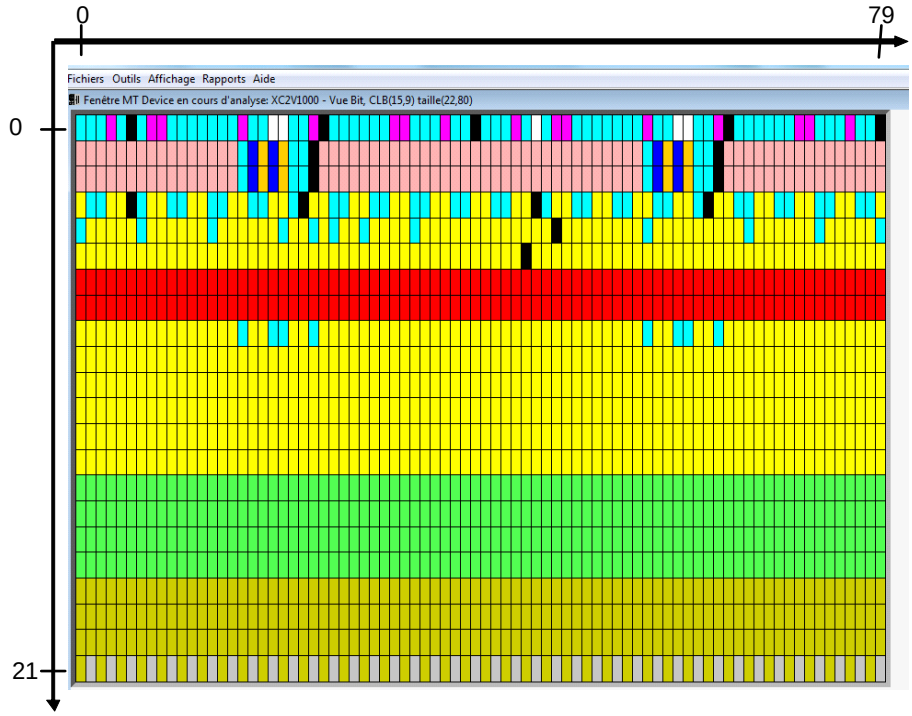

Figure 53: Représentation des bits configurant une tuile CLB dans le bitstream.

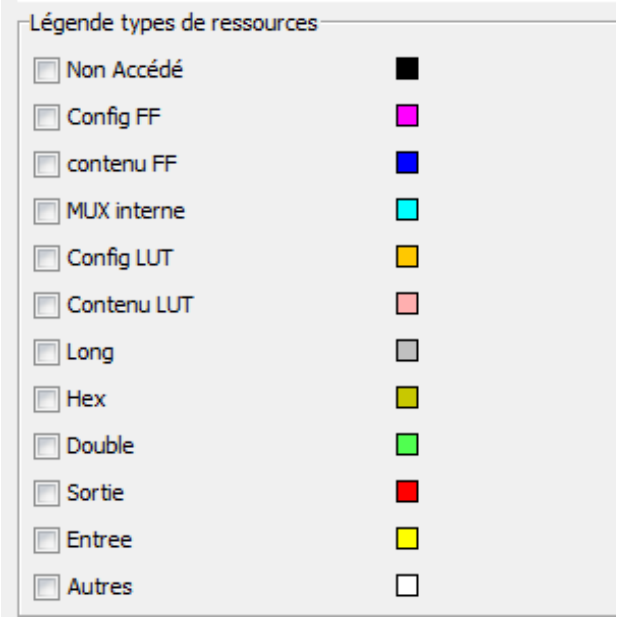

Figure 54: Catégories de bits d'une tuile CLB.

3.2.3.2. Configuration du réseau d'interconnexions

Afin de pouvoir détecter les motifs d'erreur créés par des fautes dans la matrice de configuration d'un FPGA, il convient de comprendre sur quel modèle les ressources sont configurées. Les classes JBITS nous ont permis d'élaborer une liste exhaustive des interconnexions possibles à l'intérieur d'une tuile CLB. Il apparaît tout d'abord que la structure de configuration n'est pas homogène. Le nombre de bits nécessaires pour configurer une interconnexion dépend de l'interconnexion considérée. La grande

majorité nécessite 2 bits (90.3%), tandis que d'autres n'ont besoin que d'un bit (9.5%) et que quelques-unes demandent 3 bits de configuration pour être (des)activées. Ces chiffres sont récapitulés dans le Tableau 5.

Tableau 5 : Types de configuration pour les interconnexions.

Nombre de bits pour la configuration	**1 bit**	**2 bits**	**3 bits**
Nombre d'interconnexions	322	3081	8
Pourcentage	9,5	90,3	0,5

Pour les connexions gérées par un seul bit de configuration, la situation est assez simple. Le bit de configuration agit comme un interrupteur. Il faut cependant préciser qu'un bit peut créer ou détruire plusieurs interconnexions à la fois. Pour les autres interconnexions, la situation est plus compliquée. Les bits ne sont pas affectés à des ressources de connexion (fils) mais à des listes de connexions possibles entre une ressource et plusieurs sources connectables. La Figure 55 présente un extrait de la structure de configuration de la ressource OMUX9, qui est une sortie de multiplexeur. La ressource n'est pas importante par elle-même, mais elle fait figure d'exemple.

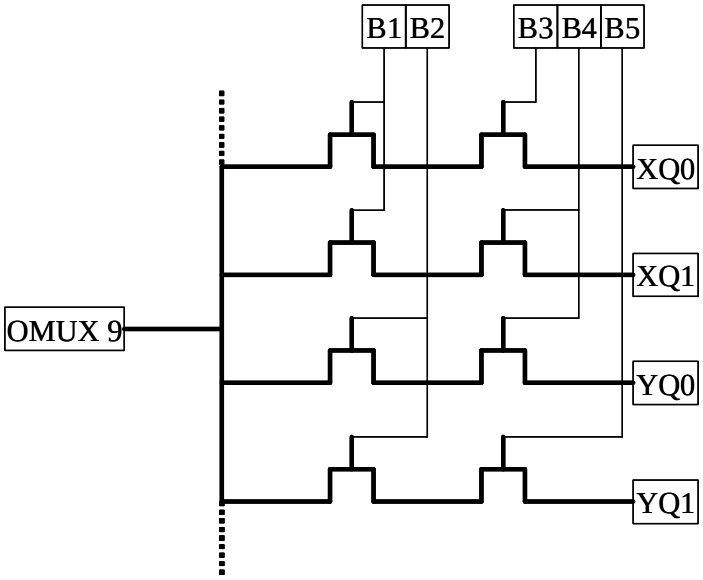

Figure 55: Structure de configuration d'une interconnexion programmée par 2 bits.

Dans ce cas, tous les bits sont associés à la ressource OMUX9, chaque bit de configuration définissant une liste d'interconnexions possibles. Ainsi, le bit B1 définit la liste {OMUX9 connecté à XQ0, OMUX9 connecté à XQ1}, le bit B4 définit la liste {OMUX9 connecté à XQ1, OMUX9 connecté à YQ0} etc… Une connexion entre la ressource OMUX9 et la source XQ1 mettra donc en jeu les bits de configuration B1 et B4. Ainsi, une connexion est définie par l'intersection de deux listes de connexions possibles.

En moyenne, le nombre de bits de configuration mis en jeu par ressource est assez élevé, de l'ordre de 9.

C'est la compréhension de ce modèle qui a permis l'automatisation de la détection de motifs d'erreurs dans les interconnexions d'un FPGA à la suite d'une injection de fautes dans la mémoire de configuration.

3.3. Caractérisation de la sensibilité aux fautes de la mémoire de configuration

3.3.1. Analyse de la répartition des fautes

La première chose à faire afin de caractériser l'effet des fautes de configuration est de les localiser. Pour ce faire nos outils permettent de comparer deux bitstreams et d'indiquer la position dans le bitstream des bits différents, ainsi que leur rôle dans la configuration. En comparant un bitstream "golden" (sain) avec un fichier issu d'une relecture de configuration, on peut ainsi obtenir l'ensemble des fautes, leur localisation dans le fichier de configuration, les types de tuile concernés, ainsi que les fonctions architecturales concernées à l'intérieur des tuiles CLB.

3.3.2. Analyse des effets des fautes

L'algorithme de détection de motifs d'erreurs utilise une base de données contenant le rôle de chaque bit dans la configuration. Tous les bits des tuiles CLB ont été identifiés sauf ceux de la dernière frame qui ne peut pas être modifiée. La Figure 56 montre les différentes étapes et les différents fichiers nécessaires à la détection de motifs d'erreurs architecturaux suite à une injection de fautes sur la matrice de configuration.

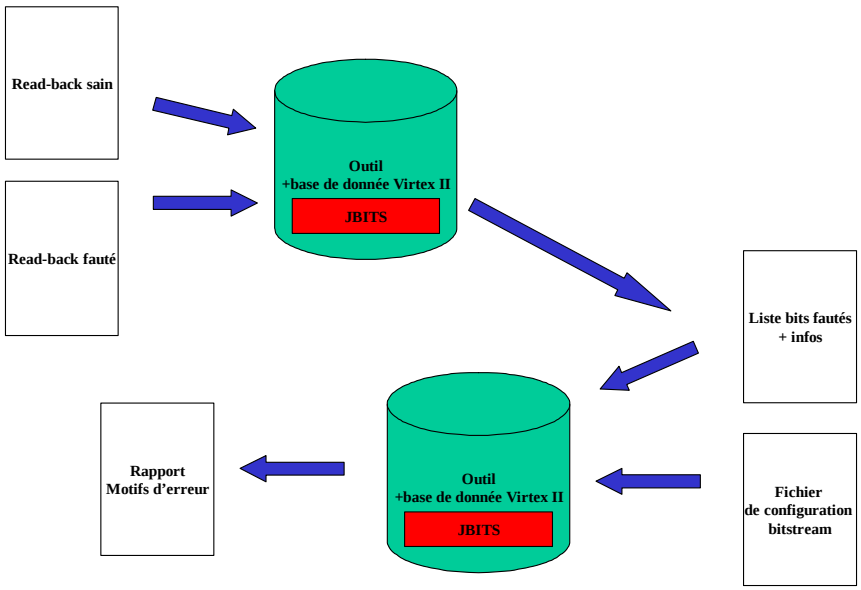

Figure 56: Processus de détection de motifs d'erreurs

Pour éditer le rapport de motif d'erreurs, l'outil a besoin d'un premier rapport contenant la liste des bits fautés ainsi que leur rôle de configuration, mais aussi du fichier de configuration initial qui va permettre de faire le lien avec l'architecture initialement implémentée sur le FPGA. Ce travail (création de la base de données - implémentation de l'outil - analyse des résultats) a été effectué sur la partie routage des tuiles CLB (on verra par la suite que la majorité des fautes se produisent sur ces ressources).

3.3.3. Méthodologie d'analyse statique de l'effet des fautes de configuration

Suite à l'analyse présentée précédemment, il est intéressant de fournir au concepteur une synthèse lui permettant d'évaluer la criticité des différents bits de la configuration, et d'identifier rapidement les bits les plus critiques pour son application. Le premier critère pour établir un tel classement est le rôle de chaque bit dans la matrice de configuration, mais nous pouvons compléter cette démarche en tenant compte du circuit implanté sur le FPGA, c'est-à-dire en prenant en compte les bits effectivement utilisés pour définir la configuration d'un circuit en particulier.

Une ressource est considérée comme utilisée si au moins une de ses entrées est connectée. On vérifie les entrées car, pour certaines configurations, les sorties sont routées par défaut (par exemple les sorties de slice dans une configuration en registre à décalage) tandis que les connexions amenant les données en entrée des slices sont toutes configurables par la matrice de configuration. Une tuile CLB sera donc considérée comme utilisée si au moins une des entrées de ses slices est connectée. Le classement de criticité proposé distingue trois catégories de bits :

- Les bits critiques : ceux qui peuvent provoquer une erreur en sortie de l'application s'ils sont modifiés, sous certaines conditions d'utilisation des ressources. Puisque l'analyse est faite de manière statique, les conditions temporelles permettant de décider de l'impact réel sur le déroulement de l'application ne sont pas prises en compte et la classification est donc faite selon une approche "pire cas".
- Les bits transparents : ceux dont la modification ne peut pas modifier le fonctionnement du circuit.
- Les bits suspects : ceux pour lesquels on ne peut pas décider du niveau exact de criticité.

Si un slice est inutilisé, tous les bits internes sont classés comme transparents. En revanche, s'il est utilisé, la criticité de chaque bit dépend du circuit implanté. En effet, de nombreux éléments sont dédiés à une fonction particulière. Par exemple, le multiplexeur à l'entrée DI du LUT ne sert que lorsque celui-ci est configuré en registre à décalage. Cela permet de classer les bits contrôlant de multiplexeur comme critiques lorsque la configuration en registre est utilisée et comme transparents lorsque le LUT est configuré autrement.

On peut étendre cet exemple à tous les bits du CLB grâce au détail des ressources.

- CLB utilisé, Slice utilisé

Lorsqu'un slice est utilisé, les LUT internes sont forcément utilisés ; donc une modification de leur configuration peut provoquer des erreurs. De même, les entrées sont obligatoirement utilisées donc les bits associés sont classés critiques.

Les multiplexeurs de sortie sont communs aux 4 slices. Les fautes sur les sorties ne correspondant pas aux slices utilisées sont sans conséquence. Les fautes sur les sorties utilisées provoquent des erreurs. Les bits correspondant à ces multiplexeurs sont donc suspects.

Les bascules ne sont pas forcément utilisées si le slice est utilisé donc les bits sont classés comme suspects.

- CLB inutilisé

Si le CLB est inutilisé, les sorties et les entrées sont déconnectées, donc une faute simple n'a aucune conséquence.

Tableau 6 : Classes de criticité des ressources logiques.

Type de ressource	**CLB utilisé**		**CLB inutilisé**
	Slice i utilisé	**Slice i inutilisé**	
OMUX	Suspect	Suspect	Transparent
Entré slice i	Critique	Transparent	Transparent
Configuration LUT	Critique	Transparent	Transparent
Contenu LUT	Critique	Transparent	Transparent
Configuration bascules	Suspect	Transparent	Transparent
Valeur par défaut bascules	Suspect	Transparent	Transparent

Les bits configurant les interconnexions ont une influence différente sur l'application en fonction du type de connexion qu'ils codent. Comme nous l'avons vu dans la partie architecture, il existe 3 types de connexions : les connexions internes aux CLB, les connexions locales de la matrice d'interconnexion (Double, Hex), et les connexions globales entre ce CLB et les CLB lointains (long). Voici les explications du classement.

- Si la connexion est inutilisée:

- Pour les Hex et les Double, les bits concernés configurent l'entrée de ces fils. Si un fil est connecté à cause d'une faute, sa sortie est déconnectée donc la faute n'a pas de conséquence. Les bits configurant ces entrées non utilisées sont donc transparents.
- Pour les long line, les connexions sont bidirectionnelles et ces fils parcourent tout le composant. Il est impossible de prévoir si une faute sur ces bits va provoquer une erreur.

- Si la connexion est utilisée:

Une dizaine de bits définissent la configuration de chaque connexion, mais seuls deux sont nécessaires pour établir une des liaisons possibles. Ces deux bits sont classés critiques car leur modification déconnectera le fil. Une modification des autres bits peut provoquer ou non une erreur.

Tableau 7 : Classes de criticité des ressources d'interconnexion.

Types de ressource	Connexion utilisée	Connexion inutilisée
Entrée de Double	- 2 bits de connexion critiques - Autres bits pilotant cette ressource suspects	Transparents
Entrée de Hex	- 2 bits de connexion critiques - Autres bits pilotant cette ressource suspects	Transparents
Long line	- 2 bits de connexion critiques - Autres bits pilotant cette ressource suspects	Suspects

Le Tableau 8 résume les résultats obtenus pour les benchmarks ISCAS c3540, c74181, et c74283. Ces résultats sont obtenus sans analyse des interdépendances entre ressources du système d'interconnexions. Le nombre de bits suspects est ainsi surestimé. Cependant, le pourcentage de bits critiques pour le comportement de chaque application est déjà assez bas. Une augmentation de la complexité de l'application (c3540) implique une hausse du nombre de bits classés critiques : le nombre de bits potentiellement dangereux est ainsi deux fois supérieur pour c3540 par rapport aux deux autres applications. Toutefois, le pourcentage de bits "non sûr" (donc critiques ou suspects) reste faible pour ces exemples.

Tableau 8: Résultats de criticité des circuits ISCAS sur Virtex II.

Applications \ Classes de criticité	Critique	Transparent	Suspect	Non sûr (Critique + Suspect)
c3540	1.6%	95.8%	3.6%	5.2%
c74181	0.1%	97.6%	2.3%	2.4%
c74283	0.0%	97.7%	2.3%	2.3%

3.3.4. Intégration dans l'outil SEFEA ProD

Pour permettre une analyse facile de l'influence fonctionnelle des SEUs sur les FPGAs à mémoire SRAM, il faut pouvoir visualiser leur effet directement sur une représentation de la matrice de

configuration. Nous avons donc développé un outil spécifique, en utilisant les classes java JBITS permettant de manipuler les bitstreams associés aux composants Virtex II.

3.3.4.1. Présentation

L'outil SEFEA-ProD ("Soft Error Functional Effect Analysis in Programmable Devices") que nous avons développé prend en entrée le fichier de configuration du FPGA (fichier bitstream au format .bit) ou le fichier issu d'une relecture de la configuration à partir de la carte (fichier read-back au format .rbd).

Plusieurs types d'analyses sont possibles à partir de ces fichiers. Une interface graphique a été développée pour visualiser le contenu de la configuration lorsqu'on charge un ou plusieurs fichiers .bit. Plusieurs vues de la matrice de configuration et de l'architecture implantée dans le FPGA sont disponibles. La Figure 57 présente une vue hiérarchique des possibilités offertes par le logiciel.

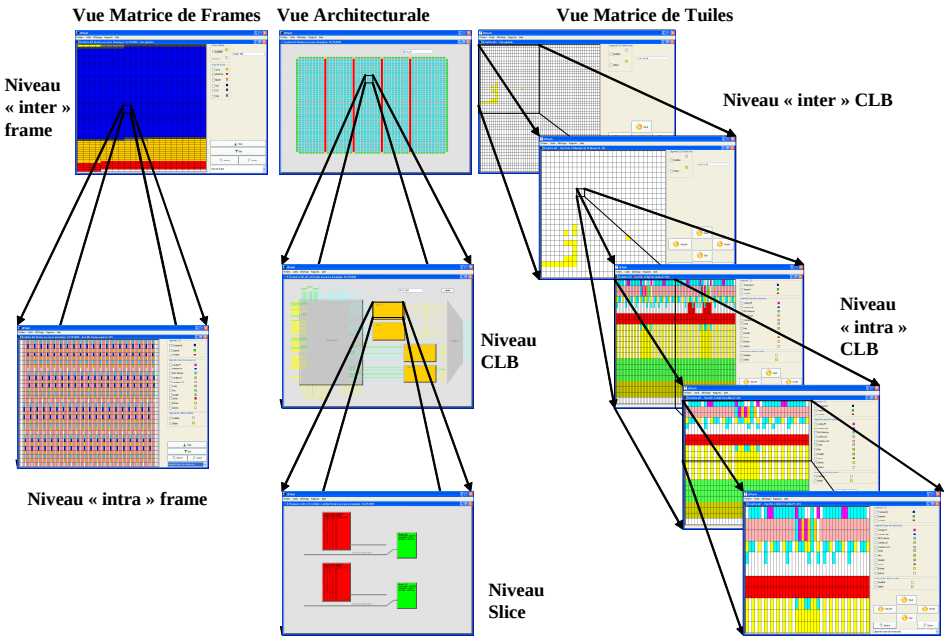

Figure 57: Interface graphique de l'outil d'analyse SEFEA-ProD

- Vue Matrice de tuiles :

La matrice de configuration est présentée comme un tableau de tuiles, les tuiles utilisées par le circuit sont mises en évidence. Pour chaque tuile CLB (utilisée ou non), on peut afficher les bits de configuration correspondant avec plusieurs niveaux d'affichage. Le logiciel affiche le rôle du bit dans la configuration et présente une estimation de sa criticité aux fautes en fonction de son utilisation dans le circuit ou de son rôle de configuration.

- Vue Matrice de frames :

La matrice de configuration est présentée ici comme un tableau de frames, les informations accessibles étant les mêmes que pour la vue matrice de tuiles (rôle des bits de configuration, utilisation, criticité).

- Vue Architecturale :

Elle représente l'architecture implantée sur le FPGA par le fichier en cours d'analyse. Pour chaque CLB utilisé, l'outil affiche toutes les connexions actives et permet d'accéder à la configuration (mode et contenu) des LUTs et éléments mémoires (verrou, bascule) de chaque slice.

Une autre partie de l'outil permet l'édition de rapports sur la comparaison de plusieurs fichiers read-back. On compare un read-back « sain » (fichier de référence) correspondant à une exécution de l'application sans injection de fautes à un read-back fauté. Le logiciel identifie les bits fautés et donne leur rôle dans la configuration du FPGA.

3.3.4.2. Fonctionnalités de l'outil pour les circuits Xilinx

L'outil développé aide à l'analyse en offrant les fonctionnalités suivantes : visualiser la matrice de configuration niveau CLB et niveau bit, afficher le taux d'utilisation et la disposition des éléments utilisés de la matrice de configuration, mettre en correspondance les bits de configuration avec leur ressource et donc obtenir une évaluation de la criticité de chaque bit en fonction de son rôle et du circuit implanté. Il permet également de découvrir la structure de la matrice de CLB et de l'intérieur d'un CLB du point de vue des interconnexions et des éléments fonctionnels, d'afficher les connexions utilisées dans les tuiles et de mettre en évidence les connexions possibles et les bits associés à cette connexion. Une option permet en outre de donner une version texte du bitstream.

SEFEA-ProD a également été utilisé pour l'injection en servant d'interface pour la gestion de la campagne de test. L'outil a en effet permis de sélectionner des bits pour l'injection selon de nombreux critères, de stocker les paramètres associés à chaque faute et de générer un script d'injection contenant toutes les informations précédentes. En fin d'injection, l'outil édite un rapport sur le rôle des bits modifiés ce qui permet de comparer un bitstream original et un bitstream fauté.

3.4. Etudes de cas

Nous avons utilisé cet outil SEFEA-ProD pour l'analyse de plusieurs campagnes d'injections de fautes laser, réalisées d'une part au laboratoire IMS (*Intégration du Matériau au Système*) de Bordeaux, et d'autre part au laboratoire CEA-LETI de Grenoble.

3.4.1. Présentation des bancs laser

3.4.1.1. Bancs du laboratoire IMS de Bordeaux

Trois campagnes ont été réalisées à Bordeaux sur le banc de test PLS (Photoelectric Laser Stimulation) de la plateforme ATLAS du laboratoire IMS. Une première campagne a été nécessaire pour calibrer l'ensemble de l'environnement d'injection (longueur d'onde optimale pour la pénétration dans le composant). Ainsi, nous ne détaillerons dans la suite que l'analyse des résultats des deux autres. Ces campagnes mettent en évidence le rôle de l'énergie du faisceau laser dans la création de SEUs et leur répartition, ainsi que l'influence de la configuration initiale du FPGA. Ces campagnes ont également permis d'étudier les motifs de modifications architecturales créés.

La source laser du banc de test PLS est un oscillateur TI Sapphire qui délivre des impulsions laser de 1 ps à une fréquence de 80 MHz. La meilleure pénétration dans le silicium en attaques face arrière a été obtenue avec une longueur d'onde de 980 nm (longueur d'onde ajustable de 750 nm à 1 µm). Un sélecteur d'impulsion placé en sortie de la cavité du laser permet de réduire la fréquence laser pour générer des tirs mono-coup. Le faisceau est focalisé sur le circuit sous test (DUT) grâce à différents objectifs, permettant d'obtenir des tailles de 1, 2, et 5 µm. Sa position est contrôlée avec une résolution de 100 nm sur tous les axes.

3.4.1.2. Bancs du laboratoire CESTI – LETI de Grenoble

Nous avons également analysé les résultats de campagnes d'injections de fautes laser réalisées au laboratoire CESTI-LETI de Grenoble. Ceci a permis de valider les premiers résultats obtenus, et d'affiner les résultats concernant la répartition des fautes et leurs effets, en fonction des paramètres du laser utilisé (par exemple en faisant varier la taille du spot).

Les bancs laser du CESTI-LETI du CEA Grenoble permettent d'obtenir des tailles de spot laser différentes allant de 4 µm à environ 100 µm, soit à partir d'une focalisation réalisée en interne soit à partir d'objectifs de microscope. Le banc microscope permet d'obtenir plusieurs tailles de spot puisqu'il possède plusieurs grossissements. Avec les grossissements de X100, X50, X20 et X10, il est possible d'obtenir respectivement des tailles de spot d'environ 4 µm, 8 µm, 20 µm et 40 µm. Un spot plus large peut être obtenu en utilisant une fibre optique.

Par ailleurs, selon l'effet désiré dans les zones actives du composant, il est possible de modifier plusieurs autres paramètres des lasers tels que la longueur d'onde de la diode laser ou la valeur de l'énergie émise. Tous les changements faits sur le banc conduisent à de nouvelles conditions expérimentales.

Au sein du laboratoire plusieurs diodes laser sont disponibles, possédant des longueurs d'onde différentes dans l'infrarouge comprises entre 745 nm et 1000 nm.

3.4.2. Préparation des FPGAs et cartes de tests

3.4.2.1. Amincissement des composants

Le FPGA étudié est un composant Virtex-II de Xilinx, fabriqué dans une technologie CMOS 0,15 µm avec 8 couches de métal. Il est mis dans un boîtier flip-chip fine-pitch avec 896 broches. Puisque le composant est flip-chip, et pour obtenir une bonne pénétration du faisceau lumineux dans les zones actives, une préparation de l'échantillon est nécessaire. Les préparations sont communes lors d'attaques laser sur des composants tels que les cartes à puces ou les FPGAs. Deux types de préparations peuvent être effectués sur le composant : une préparation chimique et une préparation mécanique. La préparation chimique est réalisée, par exemple, pour obtenir une photographie des différents éléments du circuit et/ou sera utilisée pour le positionnement géographique des différentes campagnes. Ce type de préparation est destructif pour le composant et est réalisé par des personnes spécialisées en chimie. La préparation mécanique consiste à amincir le silicium soit avec une micro-fraiseuse, soit par un usinage mécanique afin que le faisceau lumineux puisse toucher les zones actives en perdant le moins d'énergie possible. Cette préparation est très importante pour les attaques. La puce du FPGA étudié possède une surface de l'ordre du centimètre carré (10,6 mm de largeur et 9,7 mm de longueur) et la solution utilisant l'usinage mécanique a été retenue. Initialement la puce possède une épaisseur de 790 µm et nous avons choisi de l'amincir à 30 µm car, pour cette épaisseur résiduelle, nous avons une très bonne pénétration du faisceau laser.

3.4.2.2. Carte THESIC pour les tests laser

Le testeur THESIC développé à TIMA a été utilisé pour les campagnes de tests réalisées au laboratoire IMS de Bordeaux. Il a été développé afin de faciliter toutes les procédures de tests, grâce à sa capacité à s'adapter à un grand nombre de composants numériques à tester (mémoires, processeurs, FPGAs), et à sa facilité d'adaptation aux différents bancs de test.

Il se compose principalement de deux FPGAs, l'un embarquant un processeur LEON permettant la communication et la commande du système par un ordinateur, et le second chargé de l'interface entre les mémoires du testeur et le composant à tester. De plus amples informations peuvent être trouvées dans [FOU10].

3.4.3. Résultats et analyses

3.4.3.1. Répartition des fautes

Le Tableau 9 présente la répartition moyenne des bits de configuration fautés sur les campagnes d'injection menées à Bordeaux et les pourcentages correspondants. On a distingué les catégories de bits suivantes, qui correspondent aux différents blocs du FPGA : blocs d'entrée/sortie (IOB), blocs gérant les interconnexions des entrées/sorties (IOI), blocs gérant l'arbre d'horloge (GCLK) du FPGA,

blocs de mémoire RAM et blocs gérant leurs interconnexions (BRAM, BRAMI), blocs CLBs, et enfin les bits gérant les IOB latéraux et contenus dans les colonnes CLB (CLBIO).
On remarque que la plupart des fautes se produisent dans les frames de configuration BRAM ou CLB, qui sont les ressources les plus importantes du FPGA. Il faut toutefois noter que les tirs laser réalisés lors de la campagne étaient essentiellement concentrés sur la partie interne du composant.

Tableau 9: Répartition des fautes parmi les types de bits.

Blocs du FPGA	CLB	CLBI	GCLK	IOB	IOI	BRAM	BRAMI	Total
Nombre de fautes	80.95	0.54	0	0.03	0.03	50.41	5.87	137.85
Pourcentage	58.72	0.39	0	0.02	0.02	36.57	4.26	100

Le Tableau 10 présente la répartition des bits de configuration fautés dans les tuiles CLB. On distingue trois catégories de bits : les bits configurant les interconnexions, les bits configurant les ressources fonctionnelles du CLB (LUTs, portes logiques, éléments mémorisant) et les bits actuellement inconnus (la «22ème trame» des colonnes CLB qui ne peut être manipulée via JBITS). La majorité des erreurs se trouve dans les bits configurant les interconnexions, ce qui est cohérent avec le fait que ce sont les plus nombreux parmi les 1760 bits configurant un CLB.

Tableau 10: Répartition des fautes à l'intérieur des CLBs.

Type de bit	Total	Logique	Interconnexions	Frame 22
Nombre	80.95	34.49	44.15	2.31
Moyenne (%)	58.72	25.02	32.03	1.68

3.4.3.2. Influence de l'état initial du bit

Nous avons effectué les tirs dans une même zone rectangulaire de dimension 0.25*1 mm^2 avec 4 configurations différentes du FPGA. Le premier est complètement vide. Deux applications ont les entrées et les sorties de toutes les bascules configurées par défaut à 0 ou 1 respectivement. Enfin, le dernier circuit est un additionneur 200 bits qui occupe 100 slices dans un coin du FPGA. Le protocole d'injection est le même pour chaque expérience. Le tableau 11 présente le nombre d'erreurs observé.
L'application avec les bascules utilisées forcées à 0 présente la plus grande sensibilité avec un nombre de fautes deux fois plus important que celui de l'additionneur. Ces résultats indiquent une dissymétrie en terme de sensibilité entre les états haut et bas des bits de configuration. Cela implique que la sensibilité d'une application implantée sur le FPGA dépend fortement des détails de l'utilisation des ressources. Ces résultats sont typiques de ce qu'on peut obtenir en utilisant une plateforme de test

laser, et constituent une bonne base pour l'établissement de modèles de fautes permettant d'estimer la vulnérabilité d'une application aux fautes de configuration à partir d'injections de fautes.

Tableau 11: Nombre de fautes en fonction de la configuration initiale.

Configuration	Vide	Bascules à 0	Bascules à 1	Additionneur 200 bits
Nombre d'erreur	78	280	240	135

On note également que les nombres de 0 et de 1 fautés sont du même ordre de grandeur, respectivement 53.09% et 46.91% du nombre total de bits fautés (voir Tableau 13). Ceci doit être mis en relation avec la configuration de la zone scannée par le laser, et le nombre de 0 et de 1 présents à l'intérieur de cette zone. Le Tableau 12 présente le nombre moyen de bits de configuration à 0 et à 1 pour cette zone pour le bitstream golden et les bitstreams fautés. Ces données permettent de comparer les différentes probabilités pour un bit d'être inversé, en fonction de sa valeur initiale : la probabilité d'inverser un 1 est 2,5 fois supérieure à celle d'inverser un 0. Comme 0 est la valeur par défaut pour le Virtex II, on peut en déduire par exemple qu'un tir laser aura de plus grandes chances de supprimer une connexion existante que d'en créer une. Nous reviendrons là-dessus ultérieurement.

Tableau 12: Evaluation de la probabilité d'inversion d'un bit en fonction de son état initial.

Catégorie \ Valeur	Général	Bits à '1'	Bits à '0'
Bitstream golden	1760	212,80	1547,20
Bits faux	9,15	2,37	6,78
Probabilité d'inversion	0,52%	1,11%	0,44%

Le Tableau 13 reprend les informations de répartition présentées au paragraphe précédent, mais en spécifiant la valeur initiale du bit avant faute. Il est divisé en trois parties, les deux premières lignes concernent tous les bits de configuration sans distinction, tandis que les autres ne prennent en compte que les bits initialement à 0 (ligne 3, 4) ou à 1 (ligne 5, 6). Il permet d'aboutir à la même conclusion sur l'influence de l'état initial d'un bit de configuration sur la probabilité de faute.

On remarque que la plupart des fautes se sont produites pour ces campagnes dans les frames de configuration BRAM ou CLB, qui sont les ressources les plus importantes du FPGA. On remarque également que le nombre de « 0 » fautés est du même ordre de grandeur que le nombre de « 1 » fautés (53% contre 47%). Si l'on se réfère à la surface exposée du FPGA et à la densité de « 1 » et de « 0 » dans cette zone, on peut remonter aux probabilités d'inverser un « 1 » ou un « 0 ». Le Tableau 13 donne le nombre moyen de bits par CLB, ce qui donne une bonne idée de ces densités. On remarque alors que la probabilité d'inverser un « 1 » est 2,5 fois plus importante que la probabilité d'inverser un

« 0 ». Comme « 0 » est la valeur par défaut dans le fichier de configuration, les ressources utilisées dans le circuit devraient être plus sensibles aux fautes de configuration que les autres.

Tableau 13: Répartition des fautes détaillée en fonction de l'état initial du bit.

Blocs du FPGA	Total	CLB	CLBIO	GCLK	IOB	IOI	BRAM	BRAMI
Nombre de '0', '1' fautés	137,85	80,95	0,54	0	0,03	0,03	50,41	5,87
Pourcentage	100,00	58,72	0,39	0	0,02	0,02	36,57	4,26
Nombre de '0' fautés	73,18	17,10	0,54	0	0,03	0,03	50,41	5,05
Pourcentage	53,09	12,40	0,39	0	0,02	0,02	36,57	3,66
Nombre de '1' fautés	64,67	63,85	0	0	0	0	0	0,82
Pourcentage	46,91	46,32	0	0	0	0	0	0,59

3.4.3.3. Effet de la taille du spot laser

Nous avons effectué d'autres campagnes d'injection par laser au laboratoire CESTI-LETI de Grenoble, notamment pour analyser l'influence de la taille du spot laser sur la répartition des erreurs dans le FPGA. Ces données ont été analysées avec l'outil SEFEA-ProD.

Nous avons effectué plusieurs campagnes de tirs laser avec des tailles de spot de 40 µm et 8µm. Après chaque tir, la configuration extraite des fichiers de relecture de la configuration a été analysée pour des rapports statistiques sur les caractéristiques des erreurs. Nous détaillons ici les résultats sur la répartition des fautes dans la configuration en fonction de la taille du spot laser. Les modifications architecturales engendrées seront abordées plus loin dans cette partie.

Le Tableau 14 résume la répartition des erreurs pour les tuiles CLB. Le tableau présente le minimum et le maximum de bits fautés par tir laser, le nombre moyen pour la campagne complète, et le pourcentage d'erreurs pour chaque catégorie et pour les deux tailles de faisceau. Quelques bits de configuration ne peuvent pas être analysés par notre outil, ils constituent la catégorie "autres".

Ces résultats confirment la sensibilité des tuiles CLB, un minimum de un bit ayant été modifié à chaque tir. Le nombre moyen de bits modifiés peut être élevé, et dépend clairement de la taille du faisceau laser. Celui de 40 µm modifie en moyenne 9 bits par tir tandis que celui de 8µm n'en modifie que 5. Ces nombres peuvent varier en fonction de la configuration initiale utilisée comme nous l'avons dit dans la partie précédente, puisque la probabilité d'inverser un 1 est plus importante.

Comme nous l'avons vu, une tuile CLB contient une partie de ressources logiques et une partie interconnexion. Pour la partie logique, le nombre moyen de bits modifiés ne varie pas de manière flagrante en fonction de la taille du faisceau laser. Respectivement 2,35 et 1,99 bits sont modifiés par les faisceaux de 40µm et de 8µm. Le pourcentage d'erreur dans cette partie de la configuration est aussi faible.

Tableau 14: Répartition des fautes dans la zone CLB de la mémoire de configuration en fonction du faisceau utilisé.

Faisceau		**40 µm**	**8 µm**
CLB	Min	1	1
	Max	35	17
	Moyen	9,164	5,066
	Pourcentage	99,15%	100%
Logique	Min	0	0
	Max	16	14
	Moyen	2,35	1,99
	Pourcentage	25,61%	39,18%
Interconnexions	Min	0	0
	Max	35	17
	Moyen	6,76	3,03
	Pourcentage	73,82%	59,72%
Autres	Min	0	0
	Max	1	2
	Moyen	0,05	0,06
	Pourcentage	0,57%	1,11%

Ces deux éléments s'expliquent par la faible densité de bits de configuration de ressources logiques par rapport aux bits configurant les interconnexions dans les tuiles CLB. De fait, le nombre d'erreurs moyen pour la partie interconnexion est beaucoup plus élevé, et est fonction de la taille du faisceau laser.

Le Tableau 15 donne plus de détails pour la partie des ressources logiques. Indépendamment de la taille du faisceau laser, les éléments les plus sensibles sont la configuration du contenu des LUTs et les multiplexeurs internes, qui représentent plus de 95% des modifications de configuration.

Tableau 15: Répartition des fautes dans les CLBs en fonction de la taille du faisceau laser.

Taille du faisceau		**40µm**	**8µm**
Contenu LUT	Nombre Moyen	1,74	1,56
	Pourcentage	74,33%	78,46%
Configuration LUT	Nombre Moyen	0,09	0,06
	Pourcentage	3,72%	3,20%
MUX internes	Nombre Moyen	0,51	0,32
	Pourcentage	21,86%	16,28%
Contenu Mémoire	Nombre Moyen	0	0,02
	Pourcentage	0%	0,89%
Configuration Mémoire	Nombre Moyen	0,002	0,01
	Pourcentage	0%	0,64%
Autres ressources	Nombre Moyen	0	0,01
	Pourcentage	0%	0,51%

Le Tableau 16 donne les mêmes informations pour la partie routage de la configuration des tuiles CLB. On voit qu'ici ce sont les bits configurant les entrées de slice et les Hex Lines qui sont les plus sensibles, représentant 70% des erreurs. Cependant, leur sensibilité varie en fonction de la taille du faisceau laser. Avec celui de 40 µm, la probabilité de modifier la configuration d'une entrée de slice est plus grande que celle de modifier la configuration d'une Hex Line. Avec le laser de 8 µm, c'est l'inverse.

Tableau 16: Résultats pour les bits d'interconnexions.

Taille du faisceau		**40µm**	**8µm**
Entrée Slice	Nombre moyen	2,95	0,70
	Pourcentage	43,58%	23,21%
Sortie Slice	Nombre moyen	0,87	0,17
	Pourcentage	12,88%	5,72%
Lignes Doubles	Nombre moyen	0,99	0,34
	Pourcentage	14,75%	11,19%
Lignes Hex	Nombre moyen	1,78	1,54
	Pourcentage	26,28%	50,88%
Lignes longues	Nombre moyen	0,17	0,27
	Pourcentage	2,52%	8,99%

3.4.3.4. Influence de la longueur d'onde et de l'énergie du faisceau laser sur la caractérisation

Le Tableau 17 illustre l'évolution du nombre d'erreurs et leur distribution parmi les différents types de tuiles du FPGA en fonction de l'énergie. On note un net accroissement de la sensibilité à partir de 1,68 nJ. A noter également qu'aucune faute n'a été observée dans les BRAM pour des énergies inférieures à 1,31 nJ. Comme nous l'avons vu, la répartition des erreurs est corrélée avec la zone scannée et sa configuration initiale. Le fait qu'aucune erreur n'ait été produite en dehors des tuiles CLB et BRAM ne signifie pas pour autant que les autres zones ne sont pas sensibles.

Tableau 17: Répartition des fautes en fonction de l'énergie laser.

Type d'élément / Energie (nJ)	CLB	CLBE	GCLK	IOB	IOI	BRAM	BRAMI
0,76	1	0	0	0	0	0	0
0,81	1	0	0	0	0	0	0
1,05	2	0	0	0	0	0	0
1,31	2	0	0	0	0	1	0
1,54	8,4	0	0	0	0	7,6	0
1,68	16,5	0	0	0	0	16,5	0
1,73	18	0	0	0	0	25,3	0

Le Tableau 18 montre, pour la même campagne, le détail de la répartition des erreurs à l'intérieur des tuiles CLB, en prenant en compte la fonctionnalité des bits. On retrouve les catégories "logique", "interconnexions", et "autres" (bits non accédés par nos outils). On note qu'aucune interconnexion n'a été modifiée pour les énergies laser inférieures à 1,73 nJ. La répartition entre la catégorie "logique" et la catégorie "autres" s'explique par la zone scannée lors de cette campagne en particulier. On n'a pas retrouvé ce type de répartition à l'intérieur des tuiles logiques pour les autres expériences (voir tableau résumant l'ensemble de tirs en 3.4.4.2).

Tableau 18: Résultats pour les CLBs en fonction de l'énergie laser.

Type de bits / Energie (nJ)	Total	Logique	Interconnexions	Frame 22
0,76	1	0	0	1
0,81	1	0	0	1
1,05	2	0	0	2
1,31	2	0	0	2
1,54	8,4	0	0	8,4
1,68	16,5	0	0	16,5
1,73	18	0	0,67	17,33

3.4.3.5. Effets sur les connexions

- Interconnexion configurée par un bit

Il y a seulement deux types de motifs différents dans ce cas. Un bitflip peut soit créer une connexion, soit la détruire. Si une perturbation crée ou détruit X connexions, on la comptabilisera comme X motifs d'erreur.

- Interconnexion configurée par plusieurs bits

Pour les connexions configurées par plusieurs bits, nous avons recensés 6 types de motifs d'erreur, en fonction de l'état initial de la connexion et de la modification engendrée par le ou les inversion(s) de bit. Ils sont représentés sur la Figure 58.

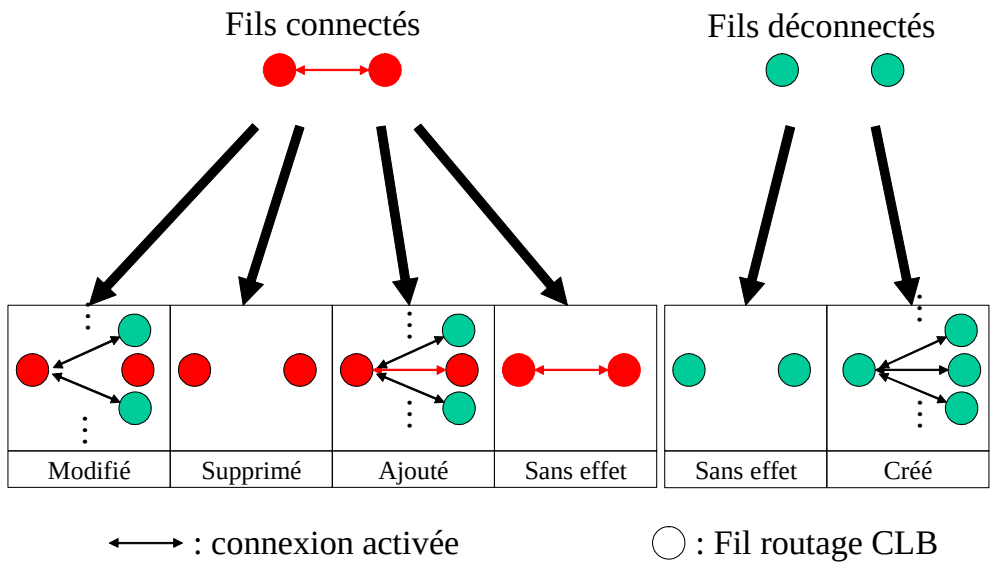

Figure 58: Motifs de modification des connexions

Une inversion de bit affectant une ressource initialement connectée peut mener à 4 situations différentes :

- Situation « modifiée »: le lien initial est supprimé mais d'autre(s) connexion(s) mettant en jeu la ressource initiale sont créée(s).
- Situation « supprimée » : le lien initial est supprimé sans autre modification
- Situation « ajoutée » : le lien initial est maintenu et d'autre(s) connexion(s) mettant en jeu la ressource initiale sont créée(s).
- Situation « sans effet » : le ou les inversions de bits n'engendrent aucun changement.

Deux autres motifs d'erreur apparaissent lorsque la ressource subissant une inversion de bit est initialement non connectée :

- Situation « créée » : une ou plusieurs connexions sont créées sur une ressource initialement non utilisée par le circuit implanté.
- Situation « sans effet » : le ou les inversions de bits n'engendrent aucun changement.

Le Tableau 19 montre la répartition des motifs d'erreur observés à la suite d'une campagne d'injection de fautes. Il montre que, dans 94% des cas, les motifs d'erreurs sont relatifs à des interconnexions configurées par 2 bits. Ceci est cohérent avec le pourcentage qu'occupent ces connexions dans l'architecture du FPGA (90,3%). Malgré la multiplicité d'erreurs importante générée pendant la campagne, nous n'avons pas été capables de produire des inversions de bits mettant en jeu des interconnexions gérées par 3 bits.

La sensibilité des interconnexions aux inversions de bits dépend beaucoup du nombre de bits nécessaires pour les configurer : comme plusieurs connexions configurées par un seul bit peuvent l'être par le même bit, le nombre de motifs d'erreur observés est plus grand que le nombre d'inversions de bits pour de telles connexions. C'est le contraire pour les interconnexions gérées par plusieurs bits, pour lesquelles on observe moins de motifs d'erreur que d'inversions de bits.

Tableau 19: Répartition des fautes en fonction du type de connexion impactée.

Bits de configuration	1 bit	2 bits	3 bits
Nombre d'inversions	18	2290	0
Motifs de modification	91	1407	0

Le Tableau 20 montre la répartition moyenne des motifs d'erreur observés pendant une campagne sur les interconnexions configurées par 2 bits. On remarque tout d'abord que, pour des interconnexions configurées par 2 bits, le lien (ou l'absence de lien) initial est généralement maintenu : lorsque la ressource est initialement inutilisée, dans 82,7 % des cas la situation est inchangée. Si la ressource est

connectée dans la situation initiale, le lien est maintenu dans 50% des cas (situation « ajouté ») et l'on doit alors prendre en compte le routage des ressources voisines pour pouvoir déterminer si le motif d'erreur perturbera l'application.

Si l'on considère le cas d'une ressource initialement inutilisée, créer un lien nécessitera le plus souvent 2 inversions de bits (on considère le cas d'interconnexions configurées par 2 bits). Cependant l'intersection des listes de connexions associées à chaque bit de configuration peut être vide, c'est pourquoi 2 inversions de bits mettant en jeu la même ressource d'interconnexion ne vont pas toujours mener à la création d'un nouveau lien. On observe en effet qu'il faut en moyenne 3 inversions de bits sur la même ressource pour créer un nouveau lien sur une ressource initialement inutilisée (colonne « Créé »). Ceci explique également pourquoi la plupart des ressources initialement inutilisées reste en l'état car, dans 82,7% des cas, seulement 1,4 inversions de bits en moyenne touchent la même ressource.

Tableau 20: Motifs de modification observés.

Etat initial	**Connectée**				**Non connectée**	
Effet sur la connexion	Modifié	Supprimé	Ajouté	Sans effet	Sans effet	Créé
Nombre moyen d'inversion	16.5	30.7	49	0	1613	581
Nombre moyen de modifications observées	7.1	20.4	29	0	1163.1	187.4
Occurrences des modification (%)	0.5%	1.5%	2.1%	0%	82.7%	13.3%
Nombre moyen d'inversion par motif	2.3	1.5	1.7	n/a	1.4	3.1

On a dit qu'il y a en moyenne 9 bits de configuration par ressource d'interconnexion. Ceci permet d'expliquer que la probabilité d'ajouter un nouveau lien à une connexion déjà existante (tirage de 1 parmi 7) est plus grande que la probabilité de supprimer le lien initial (tirage de 1 parmi 2). On remarque cependant qu'en moyenne le nombre d'inversions de bits nécessaire pour créer une situation « ajouté » est plus grand que celui nécessaire pour avoir une situation « supprimé ». Ceci est dû au fait que le motif « ajouté » prend en compte la possibilité d'ajouter plusieurs connexions alors qu'on ne peut en supprimer qu'une.

Pour obtenir un motif « modifié », les inversions de bits doivent détruire le lien initial et en ajouter d'autre(s). Les expériences ont montré qu'il faut au moins deux inversions de bits pour produire ce motif. Cependant, toujours à cause de la structure de configuration (l'intersection des listes de connexions associées à chaque bit de configuration peut être vide), il faut généralement plus de 2

inversions de bits pour observer ce motif (2,3 en moyenne). En tant que combinaison des motifs « supprimé » et « ajouté », cette situation est également plus rare (0,5% des motifs observés).
Le motif « sans effet » n'a pas été observé pour une ressource initialement connectée.

Nous avons finalement étudié le nombre moyen d'inversions de bits par motif en fonction du nombre de connexions créées pour les situations « modifié », « ajouté », et « créé ». Les résultats sont présentés sur la Figure 59.
Si l'on considère le motif « ajouté », il faut en moyenne une inversion de bit pour créer une connexion. Ce n'est pas le cas dans les autres motifs : le cas « modifié » nécessite en moyenne une inversion de bit pour créer une connexion et une autre pour supprimer l'existante, tandis que le cas « créé » nécessite en général 2 bits pour faire une connexion.
En partant de cette situation, chaque nouvelle inversion de bit crée pour tous les motifs une nouvelle connexion. De plus, quand le nombre de connexions créées augmente, le nombre moyen d'inversions de bits nécessaire pour créer une connexion diminue (au début 2 inversions de bits pour une nouvelle connexion, à la fin 1 inversion de bit pour une ou plusieurs nouvelle(s) connexion(s)). Ceci s'explique par le fait que plus le nombre de bits de configuration mis en jeu augmente, et plus le nombre de listes de connexions possibles augmente, et donc le nombre d'intersections non vides.

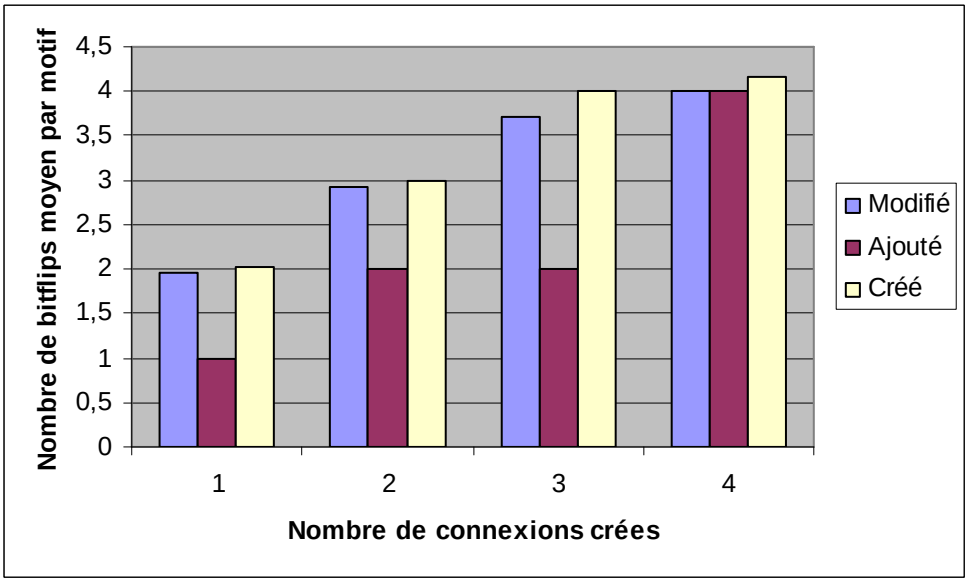

Figure 59: Nombre de connexions modifiées en fonction de la multiplicité des fautes

Afin d'affiner notre compréhension des modifications d'architecture engendrées par les SEUs, nous avons analysé d'autres résultats d'injections laser. Il s'agit d'une part d'avoir un autre ensemble de données pour valider notre première analyse, et d'autre part d'évaluer l'impact de la taille du faisceau laser utilisé dans les modifications d'architecture des interconnexions. Ces données ont été collectées pendant les expériences réalisées au laboratoire CESTI-LETI de Grenoble.

Le Tableau 21 présente la répartition des bits de configuration des connexions modifiées en fonction de l'état initial de la connexion. Cette analyse est également restreinte au type le plus courant d'interconnexion sur le Virtex II, à savoir les connexions configurées par deux bits. Nous retrouvons un pourcentage de connexions modifiées à partir de liens inactifs supérieur à 90%, comme pour les expériences précédentes. Ce résultat n'est pas affecté par la taille du spot laser. Il dépend cependant très clairement de la configuration initiale du FPGA.

Tableau 21: Etat des connexions impactées en fonction du faisceau laser.

Taille du faisceau	**40 µm**	**8 µm**
Initialement connectée	9,75%	5,53%
Initialement déconnectée	90,25%	94,47%

Les Tableau 22 et 23 montrent les effets détaillés d'une inversion de bit en fonction de l'état initial de l'interconnexion affectée. Nous avons vu précédemment qu'un lien initialement connecté a 36% de chance d'être rompu, et 51% de chance de se voir ajouter une autre connexion. Ces résultats sont confirmés par la campagne effectuée avec un faisceau de taille 40 µm, tandis que le faisceau de 8µm a produit des résultats inverses. L'impact de la taille du faisceau laser est ici clair.

Tableau 22: Motifs observés en fonction du laser pour les connexions initialement actives.

Taille du faisceau		**40 µm**	**8 µm**
Modification	Nombre moyen d'inversions	0,01	0,04
	Nombre moyen de modifications	0,03	0,02
	Pourcentage	6,84%	11,11%
Suppression	Nombre moyen d'inversions	0,18	0,06
	Nombre moyen de modifications	0,15	0,06
	Pourcentage	31,97%	44,44%
Ajout	Nombre moyen d'inversions	0,41	0,06
	Nombre moyen de modifications	0,28	0,05
	Pourcentage	58,90%	35,19%
Sans effet	Nombre moyen d'inversions	0,01	0,01
	Nombre moyen de modifications	0,01	0,01
	Pourcentage	2,29%	9,26%

Le nombre moyen de motifs de "suppression" et d'"ajout" (0,21 par tir pour le faisceau de 40 µm) est plus grand que le nombre de motifs de "modification" et "sans effet" (0,02 pour le faisceau de 40 µm).

Pour le motif de "modification", il faut à la fois supprimer une connexion existante, et en ajouter une autre. Il faut un minimum de deux inversions de bits (une pour supprimer et une pour créer, en supposant qu'une des bornes de l'interconnexion soit commune) pour réaliser cela. Une seule inversion suffit pour produire les deux premiers types de motifs d'erreur évoqués. Les résultats observés étaient donc ceux attendus. On observe la même tendance pour les résultats avec le faisceau de 8 µm.

Quand une connexion est initialement déconnectée (voir Tableau 23), nous retrouvons également les mêmes types de résultats que pour les expériences réalisées au laboratoire IMS de Bordeaux. La plupart des tirs sur des ressources initialement déconnectées n'ont produit aucun effet (78,6% des cas pour le faisceau de 40 µm et 81,7% des cas pour le faisceau de 8µm). Il faut en effet un minimum de deux inversions pour créer une connexion du type analysé, ce qui explique ce résultat. Nous avons d'ailleurs un nombre moyen d'inversions de 2,7 pour le motif "création". Ce nombre est quasiment indépendant de la taille du faisceau laser (2,5 pour le faisceau de 8µm).

Tableau 23: Motifs observés pour les connexions inactives.

Taille du faisceau		**40 µm**	**8 µm**
Sans effet	Nombre moyen d'inversions	4,01	2,08
	Nombre moyen de modifications	3,48	1,92
	Pourcentage	78,59%	81,69%
Création	Nombre moyen d'inversions	2,56	1,09
	Nombre moyen de modifications	0,95	0,43
	Pourcentage	21,41%	18,31%

Cette nouvelle campagne a donc validé les premières constatations réalisées à Bordeaux. Il apparaît cependant que des modifications de configuration pour une connexion initialement configurée ne produisent pas systématiquement une modification d'architecture (2,29% de cas "sans effet" avec le faisceau de 40 µm, et 9,26% avec le faisceau de 8 µm).

3.5. Conclusion

Les classe Java JBITS fournies par Xilinx permettent le décodage de la mémoire de configuration du FPGA Virtex II. La structure de configuration du réseau d'interconnexion, ainsi que le rôle de la majorité des bits de configuration des tuiles CLB a ainsi pu être mise à jour. Ces connaissances permettent d'analyser finement le comportement des applications implémentées en fonction des fautes de type SEU dans la mémoire de configuration.

Plusieurs méthodologies d'analyse ont été développées et intégrées dans l'outil SEFEA ProD, qui permet d'obtenir la répartition des fautes dans un bitstream erroné, le rôle des bits de configuration

fautés, ainsi que les modifications architecturales induites. Ceci permet l'élaboration d'un pronostic sur la possibilité de génération de défaillances fonctionnelles en cas de fautes. Nous avons donc défini un classement de criticité des bits de configuration en fonction de ce dernier critère, ce qui permet de mettre en évidence les parties les plus sensibles du bitstream en fonction d'une application donnée. Des résultats expérimentaux ont par ailleurs permis d'identifier des motifs d'erreurs pouvant être obtenus en pratique, et de mettre en évidence sur cette technologie la plus grande sensibilité des bits à 1 dans la configuration.

Le chapitre suivant présente la partie de nos travaux consacrée à la technologie ATMEL AT40K.

4. Etude prédictive de criticité sur la technologie ATMEL

La deuxième technologie ciblée par cette étude est le FPGA AT40K d'ATMEL. Nous avons déjà décrit brièvement l'architecture de ce réseau programmable dans le premier chapitre.
Comme tous les autres composants à configuration SRAM, ce FPGA doit être protégé contre plusieurs types de fautes. Les SEUs sont cependant les plus fréquents en environnement radiatif naturel, comme le confirme la Figure 60, extraite de résultats d'injections de fautes laser réalisées sur un AT40K chez EADS. On rappelle qu'ils peuvent affecter à la fois la mémoire de configuration et les éléments mémoires utilisateurs ; le travail est focalisé sur les erreurs dans la configuration. L'importance de comprendre l'impact réel des SEUs est d'autant plus grande que les MCUs (ou inversions multiples) touchant des bits qui ne sont pas corrélés architecturalement peuvent aussi être analysés comme une accumulation de SEUs.

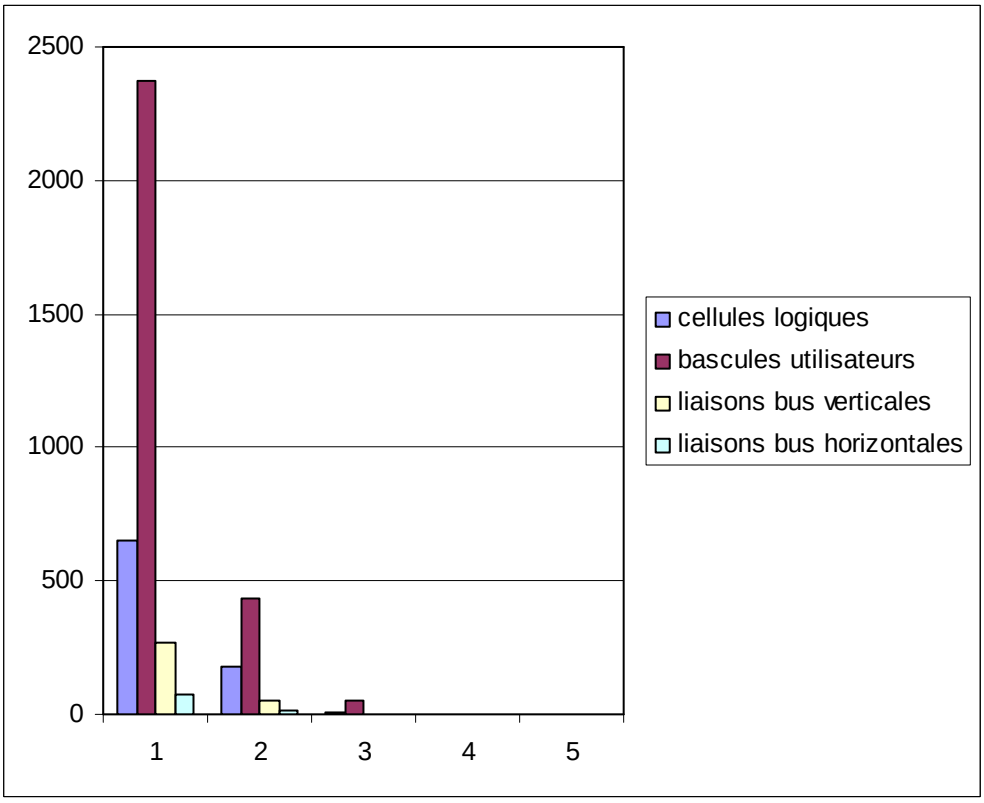

Figure 60: Multiplicité observée pour les fautes laser en fonction du type de bit impacté.

Bien que les perturbations soient transitoires, elles peuvent modifier de manière durable (jusqu'au rafraîchissement de la mémoire de configuration) l'architecture de l'application implantée sur le FPGA. La conséquence est une déviation possible du fonctionnement normal du circuit, c'est-à-dire un SEFI.

Comme nous l'avons déjà vu pour le Virtex II, tous les bits de configuration n'entraînent pas pour autant une erreur de l'application lorsqu'ils sont inversés. Cela dépend de l'architecture du FPGA, de sa structure de configuration, et de l'usage des ressources par l'application. L'évaluation de robustesse pour une application donnée nécessite comme dans le cas des Virtex II des outils spécifiques.

4.1. Définition des objectifs

L'objectif principal de ce chapitre est de proposer comme pour le Virtex II une méthodologie et un outil automatisé pour l'analyse de la criticité aux fautes simples des bits de configuration de l'Atmel AT40K. Notre analyse devra être statique, en opposition aux solutions d'évaluation de robustesse à base d'injections de fautes, et permettra un diagnostic de criticité rapide de la configuration d'une application en cours de développement. Notre méthodologie s'appuie sur un script fourni par Atmel, qui permet la mise en relation des bits de configuration avec leur ressource associée. La définition d'interdépendances entre bits de configuration configurant des ressources corrélées du point de vue architectural permettent de prévoir l'influence d'une inversion de bit de configuration sur l'implantation d'une application. L'objectif est alors de prédire la capacité de fautes simples à générer des SEFIs. La partie résultat de ce chapitre s'attachera à valider notre proposition ; les résultats seront comparés à des injections de fautes par laser et par irradiation de protons.

Le deuxième objectif sera ensuite d'analyser l'influence de l'application implantée sur le diagnostic de criticité, puis de comparer les résultats de criticité avec ceux obtenus sur la technologie Xilinx, permettant ainsi de déterminer l'influence de la plateforme d'implémentation.

4.2. Différents niveaux de criticité

Les classes de criticité sont les mêmes que celles présentées en section 3.3.3., à savoir "Critique", "Transparent" et "Suspect" (système CTS).

Notre analyse porte sur les bits de configuration des cellules logiques, ce qui représente plus de 184000 bits de configuration. La Figure 61 illustre les différentes catégories de bits de configuration traités, associés aux ressources logiques qu'ils configurent. On retrouve les bits configurant les fonctions logiques implantées dans les LUTs (LUT), les bits configurant les multiplexeurs d'accès à la cellule (inMUX), les bits configurant les multiplexeurs de sortie vers les cellules voisines (outMUX), les bits configurant les multiplexeurs internes à la cellule logique (intraMUX), et les bits configurant l'accès aux bus en sortie de cellule (LocalVia).

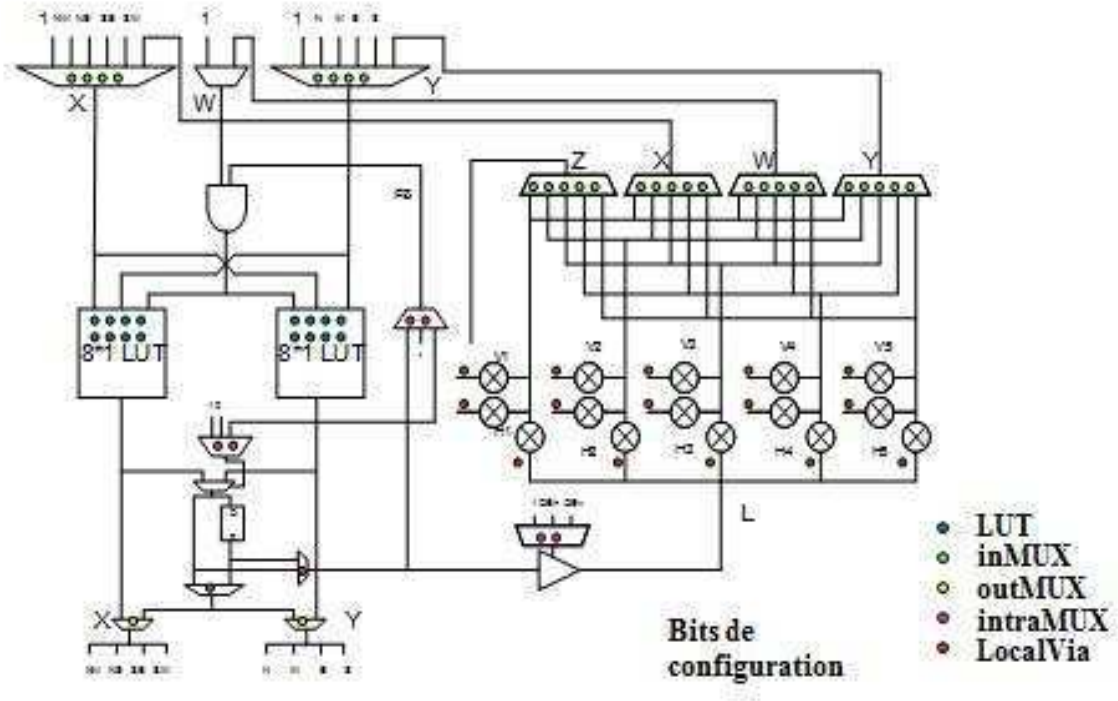

Figure 61: Architecture de la cellule PLB et ses bits de configuration.

La Figure 62 présente les bits LocalVia différemment, aux intersections des bus locaux dont ils configurent l'interconnexion. Elle présente également les bits de connexions des bus express, dont la configuration est aussi gérée dans les bits des cellules logiques.

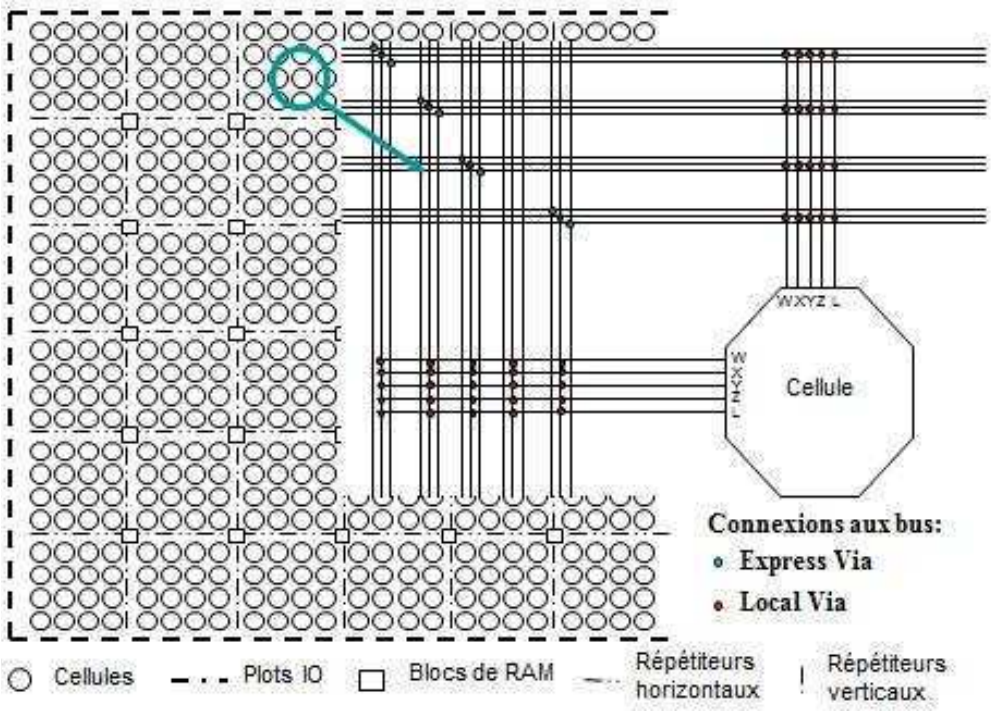

Figure 62: Architecture du réseau et interconnexion des bus.

Notre méthodologie a pour but de déterminer sous quelles conditions chaque bit de configuration peut entraîner un SEFI dans l'hypothèse d'une faute simple. Certains cas sont assez intuitifs, et peuvent s'analyser simplement ; la Figure 63 donne un exemple de cas facile à traiter. On retrouve deux types d'utilisation des multiplexeurs d'entrée de cellule. A gauche, la cellule n'est pas utilisée par l'application, et c'est donc aussi le cas des multiplexeurs d'entrée. Un SEU sur un des bits de configuration du multiplexeur ne peut connecter qu'un fil non alimenté. Un SEU sur un de ces bits de configuration ne peut donc pas avoir d'impact sur le déroulement normal de l'application. Un tel bit de configuration sera classé transparent par notre outil. La partie droite de la figure représente le cas d'un multiplexeur routant un signal vers les générateurs de fonctions de la cellule. Un SEU sur le bit configurant la connexion active, comme illustré sur la cellule, déconnectera un générateur de fonction du reste de l'application. La sortie de la cellule produira une valeur fausse ; ce bit de configuration est donc classé "critique". Cependant, ce niveau d'analyse ne suffit pas pour déterminer le diagnostic de criticité de tous les bits de configuration. Nous détaillons dans la partie suivante les problèmes à résoudre et les solutions qui y ont été apportées.

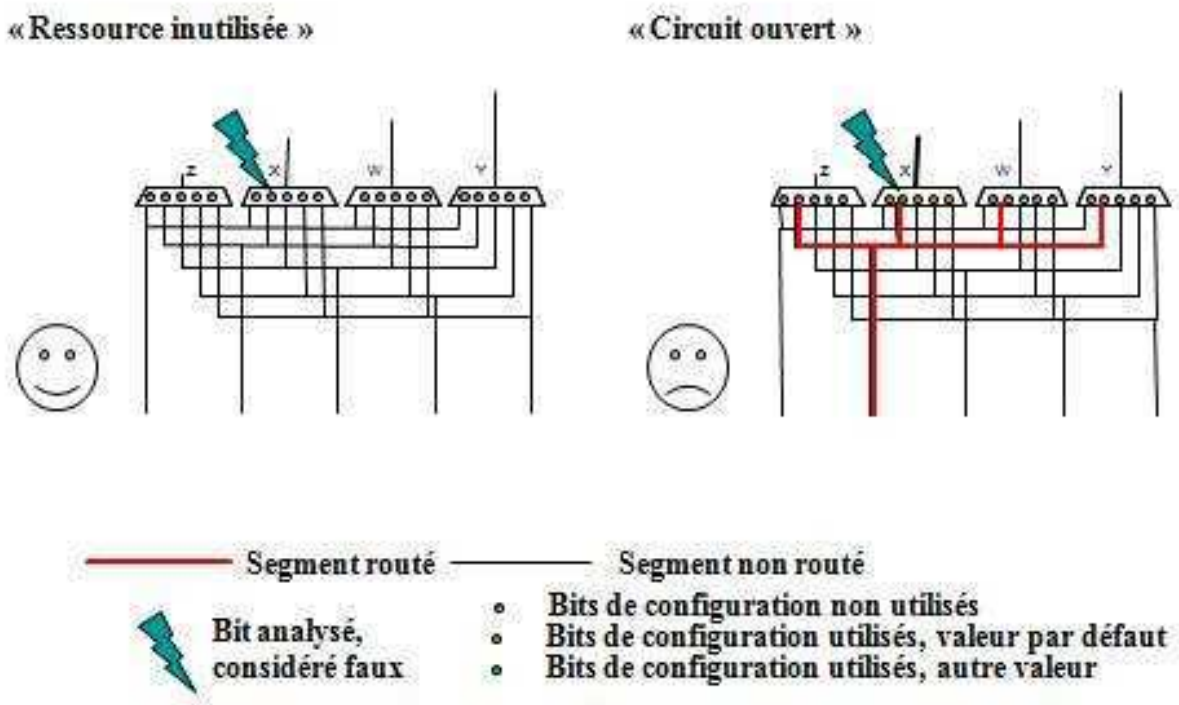

Figure 63: Exemple de bits critiques et transparents dans un PLB.

4.3. Différents niveaux d'analyses

4.3.1. Introduction

On a vu que pour certains cas, la configuration originale du bit analysé et la situation de la cellule à laquelle il appartient suffisent à déterminer la classe de criticité. Ce n'est pas le cas de tous les bits de

configuration. La Figure 64 représente une autre configuration possible d'un multiplexeur d'entrée de cellule logique. On analyse la criticité aux SEUs d'un des bits du multiplexeur dont la valeur est par défaut (un autre bit de cet opérateur est actif). L'inversion d'un tel bit va relier un deuxième bus à la sortie du multiplexeur. Les conséquences peuvent être un cas de conflit si le bus relié est alimenté par un signal actif de l'application. La partie centrale représente les deux cas de figure possibles. Si un des 5 autres bus alimente la cellule logique, un des bits de configuration d'un autre multiplexeur d'entrée de la cellule sera actif. Dans ce cas, un SEU sur le bit analysé amène un conflit sur la sortie du multiplexeur, propageant potentiellement une valeur erronée en entrée des LUTs. Dans ce cas, le bit doit être classé critique. Dans le cas inverse, le bit est transparent. Il apparaît donc que la détermination de la criticité de certains bits de configuration dépend de la configuration de bits de configuration corrélés, qui programment d'autres ressources. L'analyse de criticité doit donc être globale, car la configuration des ressources analysées ne suffit pas à elle seule à déterminer la criticité de ses bits. Nous détaillerons plus loin les différentes corrélations entre ressources.

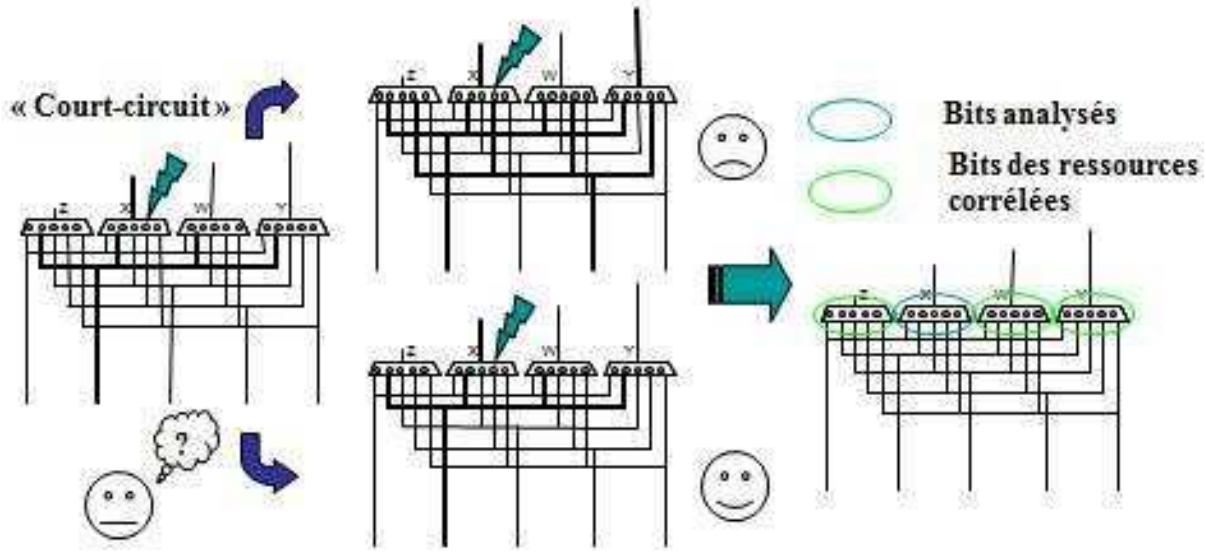

Figure 64: Détermination de criticité et ressources corrélées.

Cependant, la complexité de l'analyse de criticité augmente en fonction du nombre de ressources corrélées à analyser. Elle dépend donc du type de bit considéré, de sa configuration initiale, et de la configuration initiale des ressources qui lui sont jointes. Le temps de calcul associé augmente de manière exponentielle avec le nombre de ressources à analyser. En conséquence, nous avons défini trois algorithmes ayant des caractéristiques différentes en termes de possibilités d'analyse et temps de calcul. L'algorithme "local" présente la complexité la plus faible, et n'utilise que la configuration du bit analysé comme entrée. Ses capacités de classement sont donc moindres, ce qui veut dire que le nombre de bits "Suspects" produits par l'analyse est le plus grand. Deux autres algorithmes ont été développés. L'algorithme "Global" raffine le diagnostic de criticité des bits classés "Suspect" par l'algorithme "local". Il présente une complexité supérieure, puisqu'il utilise la configuration des

ressources corrélées au bit analysé pour établir son classement. La précision de son diagnostic de criticité est supérieure ; le nombre de bits classés "Suspect" par cette analyse est donc moindre. Enfin, un dernier algorithme améliore encore l'analyse sur les bits de configuration des LUTs contenues dans les cellules logiques. Le flot complet d'évaluation est résumé par la Figure 65.

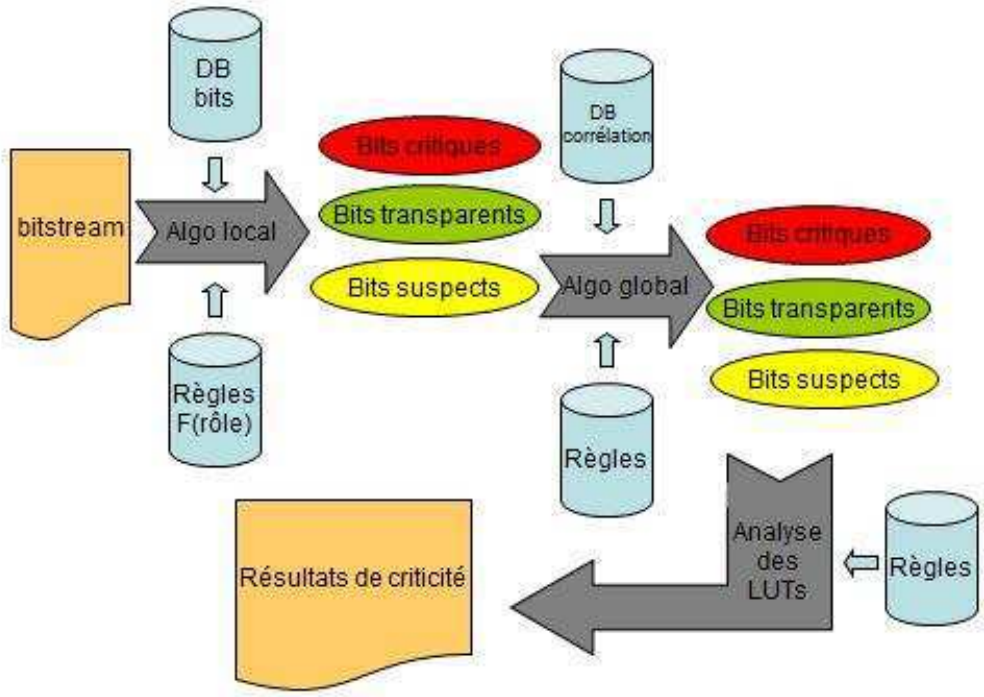

Figure 65: Flot pour le diagnostic statique de criticité.

4.3.2. Algorithme local

L'algorithme Local (voir annexes pour une illustration en "pseudo-code") assigne les classes de criticité "CTS" des bits de configuration en fonction de leur rôle dans la configuration, de leur valeur pour une application donnée, et du statut de leur cellule logique. Le statut représente la distinction entre une cellule vide et une cellule utilisée pour l'implantation d'une fonction logique de l'application. Cet algorithme, qui présente la plus faible complexité, est d'ailleurs générique pour les deux technologies étudiées dans cette thèse (Xilinx Virtex II et Atmel AT40K).

4.3.2.1. Détail de l'analyse des éléments logiques

L'analyse des éléments logiques est assez simpliste pour cet algorithme. Les ressources configurant un opérateur effectivement utilisé par l'application implantée sur le FPGA verront leurs bits de configuration classés "Critique". Celles qui sont dans des cellules implantant au moins une fonction logique pour l'application mais qui sont programmées avec une valeur par défaut, auront leurs bits classés "Suspect". En effet une modification de leur état pourrait perturber l'application en fonction du routage des différents opérateurs d'une cellule logique. Enfin, celles qui appartiennent à des cellules

sans lien architectural avec l'application verront leurs bits de configuration logique classés "Transparent".

4.3.2.2. Détail de l’analyse des éléments de routage

Pour les ressources d'interconnexion, le nombre de bits de configuration programmant la même ressource est pris en compte. Pour les segments configurés par un seul bit de configuration, l'état initial de la ressource et sa relation avec l'application permet de classer la plupart de ces bits comme "Transparents". Si une inversion du bit de configuration signifie la rupture d'une liaison initialement configurée, le bit est au contraire classé "Critique". Dans le cas des connexions configurées par plusieurs bits, une analyse plus complexe doit avoir lieu, prenant en compte la configuration initiale de tous les bits programmant des ressources reliées architecturalement à celle configurée par le bit analysé, comme nous l'avons vu précédemment.

Un grand nombre de bits de configuration sont classés "Suspect" par cet algorithme. En effet, l'impact de la création d'une connexion par exemple, dépendra de l'alimentation par un signal du segment connecté. Ceci constitue la motivation principale pour des algorithmes à complexité accrue, mais présentant de meilleures performances de classification.

4.3.3. Algorithmes Globaux

L'algorithme Global effectue une seconde passe sur les bits classés dans la catégorie "Suspect" par l'algorithme Local. La configuration des ressources corrélées aux éléments configurés par ces bits est analysée. Une description en "pseudo-code" de l'algorithme Global se trouve en Annexe de ce manuscrit.

Nous avons séparé les bits de configuration des cellules logiques en cinq grandes catégories : multiplexeurs d'entrée, multiplexeurs de sortie, multiplexeurs internes, vias locaux, vias express. La catégorie "multiplexeurs d'entrée" regroupe tous les bits configurant les multiplexeurs alimentés par les bus locaux et les connexions directes provenant des 8 cellules voisines. Les bits "multiplexeurs de sortie" configurent les liaisons vers les cellules voisines en sortie de cellule. Les bits de la catégorie "multiplexeurs internes" regroupent les bits configurant l'utilisation des sorties des générateurs de fonctions logiques des cellules, ainsi que les possibilités de rebouclage interne à la cellule. Enfin, les deux autres catégories de bits concernent l'accès en entrée et en sortie de cellule aux bus locaux, ainsi que les interconnexions entre les bus verticaux et les bus horizontaux au coin de chaque cellule. La configuration des LUTs et donc la programmation des fonctions logiques est analysée séparément par l'extension de l'algorithme Global, qui est détaillée dans la partie suivante. Nous allons maintenant présenter les bits de configuration corrélés pour chaque catégorie.

La Figure 66 représente le cas des multiplexeurs d'entrée. Cet exemple a servi de base à l'explication de la nécessité d'étudier les ressources corrélées, donc nous ne répétons pas l'explication. Comme on peut déduire l'ensemble des connexions en entrée de la cellule par l'étude globale de la configuration

des multiplexeurs W, X, Y, et Z, cette analyse est réalisée pour tous les bits suspects de cette catégorie.

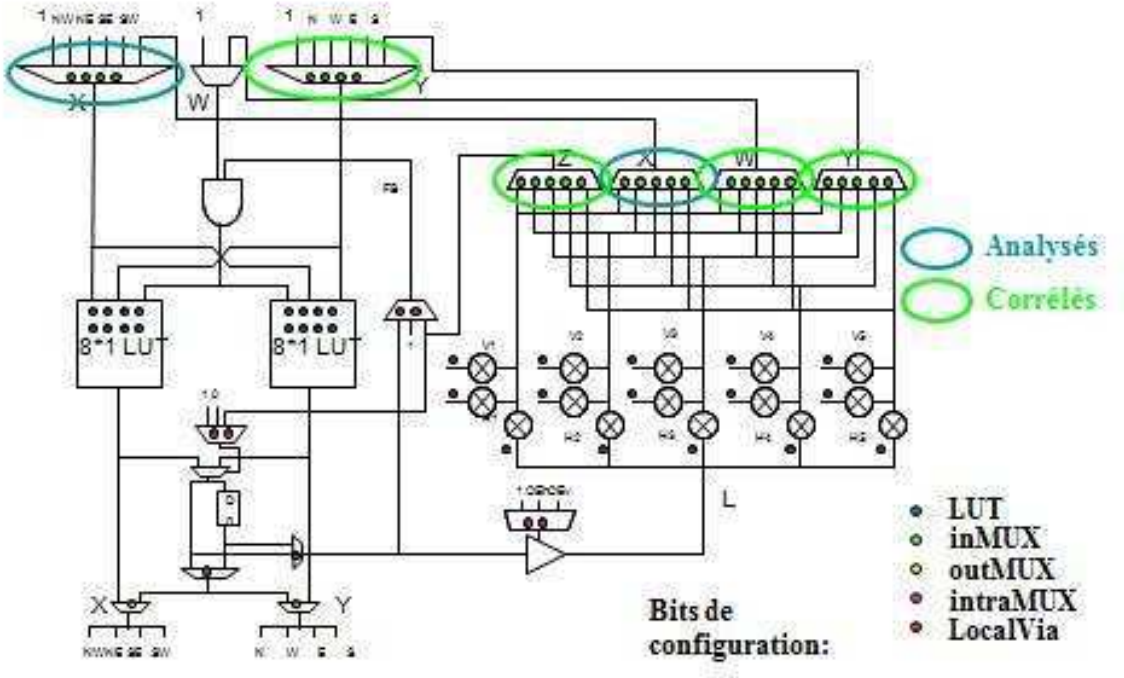

Figure 66: Ressources corrélées pour les multiplexeurs d'entrée.

La Figure 67 présente le cas de la catégorie "multiplexeurs de sortie". Afin de déterminer la classe de criticité de ces bits, il faut connaître les possibilités de conflits engendrées par la création d'une liaison supplémentaire. En effet, s'il s'agit de la rupture d'une liaison initialement configurée, le bit sera classé critique par l'algorithme Local. Il faut déterminer si une ou deux LUTs sont configurés dans la cellule, ainsi que le routage de leur sortie. La première partie est réalisée en étudiant la configuration des multiplexeurs d'entrée de la même cellule, tandis que la seconde consiste à regarder la programmation du bus L qui propage potentiellement les sorties vers les bus locaux. On peut ainsi déterminer si le conflit créé à l'intérieur du multiplexeur de sortie par l'inversion de bit de configuration va correspondre à un vrai conflit électrique. Cette analyse est plus fine que celle qui consiste à étudier la configuration des entrées des cellules voisines (reliées par les sorties en cours d'investigation), et à classer critique tout bit dont l'inversion créerait une liaison vers une cellule utilisée pour l'implantation d'une fonction logique de l'application.

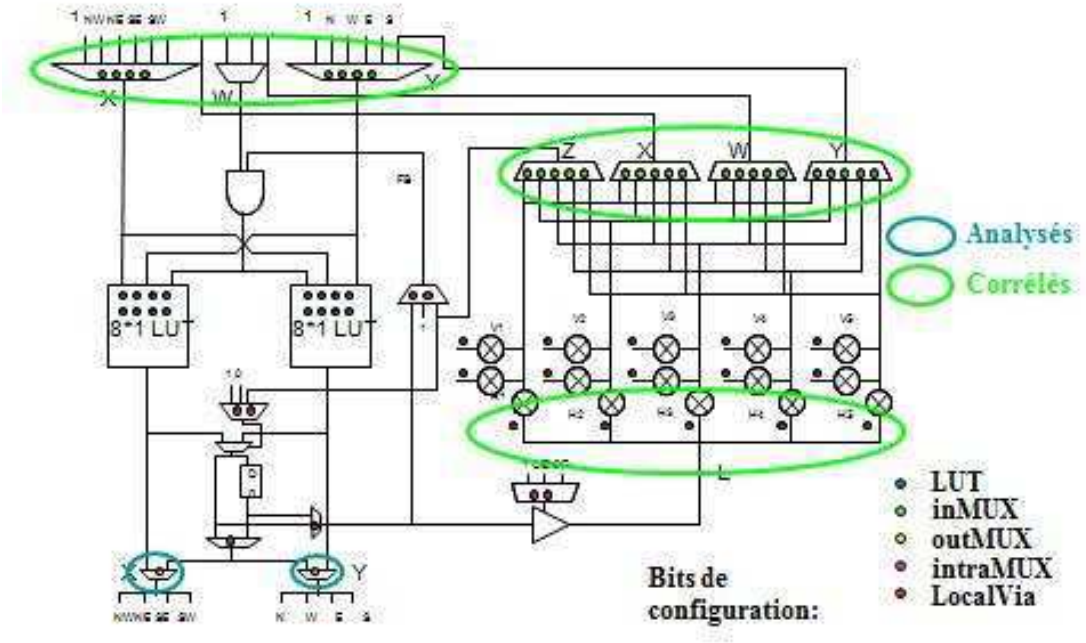

Figure 67: Ressources corrélées pour les multiplexeurs de sorties.

La Figure 68 représente les ressources corrélées pour les multiplexeurs internes. Ici on analyse la configuration des ressources les alimentant, ainsi que la configuration de toutes les sorties potentielles de la cellule afin de discerner les cas de conflits électriques. La configuration des sorties est étudiée pour connaître l'utilisation initiale des générateurs de fonctions logiques. De la même façon que précédemment, on cherche à séparer les conflits électriques réels et factices (ceux entre un fil alimenté et un fil au niveau haute impédance).

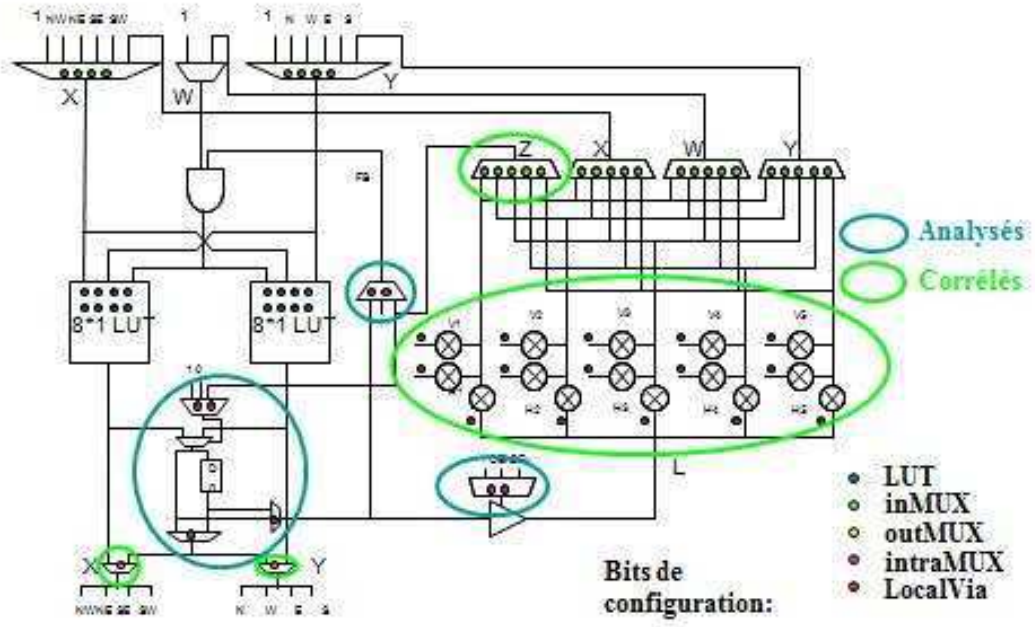

Figure 68: Ressources corrélées pour les multiplexeurs internes.

La Figure 69 représente les ressources corrélées pour les bits de la catégorie "vias locaux" configurant l'accès aux bus en sortie de cellule.

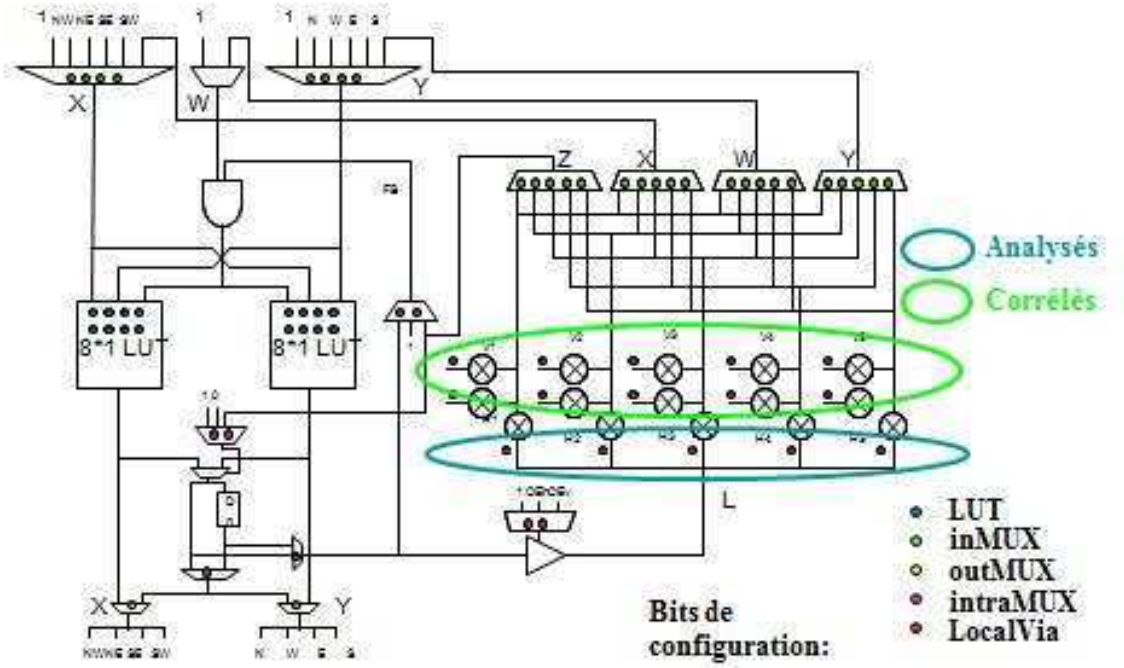

Figure 69: Ressources corrélées pour les bus de sortie.

La création d'une liaison entre un bus utilisé en entrée et la sortie forcerait un conflit électrique sur ce bus. Le bus L est en effet connecté à la bascule pour une configuration par défaut. Il y a donc possibilité de court-circuiter les entrées avec la masse si la bascule n'est pas initialement utilisée, ou avec la sortie logique des LUTs. On doit donc vérifier la configuration des liaisons bus/entrées de cellule afin de classer les bits risquant de provoquer un court-circuit "critique".

Finalement, pour l'analyse du restant des bits de configuration de cette catégorie, ainsi que pour les bits de la catégorie "via express", il est nécessaire d'examiner la configuration des liaisons bus/bus horizontales et verticales (répétiteurs du secteur auquel appartient la cellule logique.

4.3.4. Analyse des LUTs

L'algorithme Global Etendu effectue une dernière passe pour éliminer les bits suspects parmi les bits de configuration des LUTs. Ces bits correspondent aux cellules logiques implantant une fonction combinatoire de l'application. Il est possible, pour les fonctions logiques de moins de quatre entrées (on rappelle que les LUTs sont de taille 16 bits sur l'AT40K), de classer comme "Transparent" des bits de configuration d'une LUT implantant une fonction pour l'application. Dans le cas des fonctions à quatre entrées, tous les bits sont critiques car un SEU modifie la définition de la fonction.

Les cellules logiques contiennent deux LUTs X et Y de 8 bits pouvant s'imbriquer pour former une LUT de 16 bits. Il y a 16 bits XLUTi et YLUTj configurant la définition logique des fonctions implantées par ces LUTs.

Le Tableau 24 montre comment ces bits définissent la fonction lorsque l'entrée Z est à l'état haut. Les bits YLUTj sont utilisés lorsque l'entrée Z est nulle.

Tableau 24: Configuration d'une LUT 8 bits sur AT40K

Signaux\bits	XLUT0	XLUT1	XLUT2	XLUT3	XLUT4	XLUT5	XLUT6	XLUT7
W	0	0	0	0	1	1	1	1
X	0	0	1	1	0	0	1	1
Y	0	1	0	1	0	1	0	1

En fonction du routage des entrées des générateurs de fonctions, certains bits de configuration ne sont jamais accédés, même pour des LUTs implantant des fonctions logiques et non configurées par défaut. La Figure 70 représente quelques uns des différents cas possibles. A gauche toutes les entrées sont routées et l'on doit classer tous les bits critiques. La partie médiane présente le cas de deux fonctions 2 entrées, 4 bits de chaque LUT peuvent être classés transparents. Le ratio est le même pour une fonction 3 entrées implantée sur l'un ou l'autre LUT. Le ratio de la partie droite, une seule fonction de deux entrées, est de 25%. Certaines fonctions enfin ne demandent qu'une entrée (inverseur) et présentent 2 bits critiques pour 14 transparents.

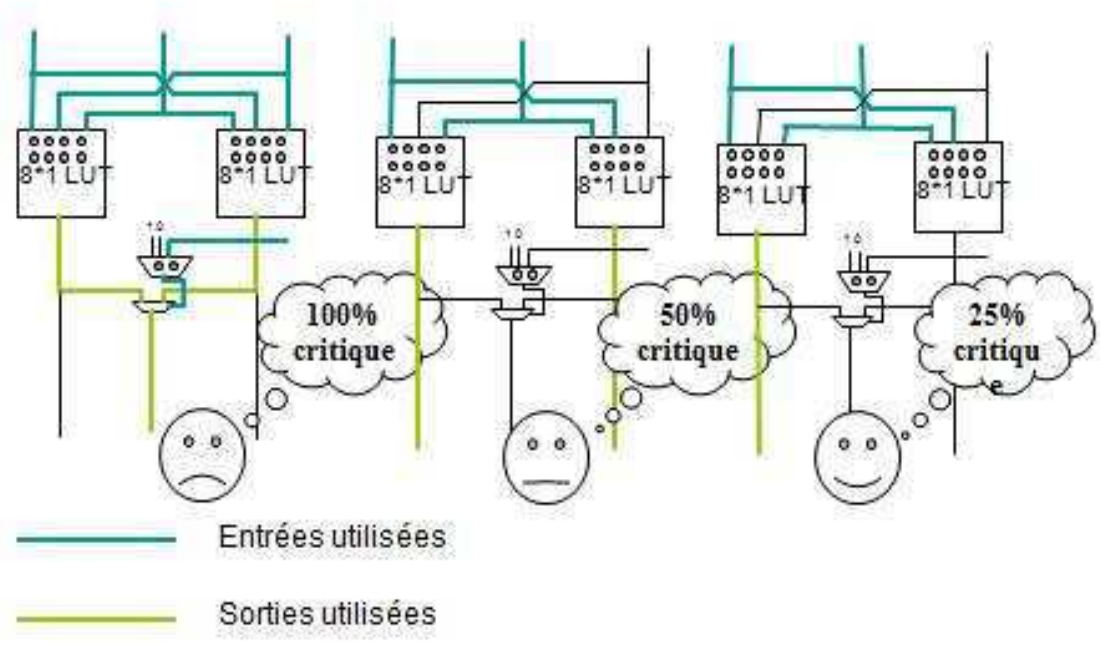

Figure 70: illustration des cas possibles pour les LUTs utilisés.

4.3.5. Récapitulatif

Le Tableau 25 présente le récapitulatif des différentes analyses réalisées par nos outils. On retrouve à chaque fois les ressources ciblées par l'analyse, ainsi que les éléments de configuration considérés par chaque algorithme. Ils sont présentés par ordre de complexité croissante.

Tableau 25: Types d'analyse de criticité et données utilisées.

Algorithmes	Ressources ciblées	Données en entrée
Local	Ressources logiques des PLBs Bits de routage PLB Bits de routage liaison bus/bus	Rôle du bit de configuration Valeur initiale du bit Connectivité de la cellule logique
Global	Bits de configuration suspects	Configuration des ressources corrélées Configuration du chemin de données concerné
Global Etendu	Bits de configuration suspects	Configuration des LUTs

4.4. Présentation de l'outil d'évaluation

Nous n'avons pas pu développer une interface graphique utilisateur comme pour la partie Xilinx. Nos outils d'analyse se présentent donc sous la forme de scripts, lançant diverses applications java. Le fichier de configuration au format md4 est utilisé en entrée. Trois types de rapports sont édités par les trois algorithmes Local, Global, et Global Etendu. Il présente la criticité de tous les bits de configuration des cellules logiques sous la forme d'un tableau indexé par le numéro de cellule et l'adressage du bit de configuration. D'autres rapports sont disponibles, qui rendent compte du taux et du type d'occupation du FPGA par l'application configurée. Nous en reparlerons dans le dernier chapitre de cette étude.

4.5. Etude de cas

Nous avons utilisé une application industrielle afin de confronter nos résultats d'analyse statique de la criticité de la mémoire de configuration avec des résultats d'injections de fautes réalisées par laser d'une part, et par irradiation de protons d'autre part.

4.5.1. L'application : un émetteur-récepteur industriel

Les expériences réalisées ont été basées sur un circuit émetteur-récepteur proche de ceux employés dans le domaine aéronautique. Un envoi de données est conditionné par les contraintes suivantes :

- La trame envoyée (trame « A ») bénéficie d'un contrôle de parité.

- Une seconde trame (trame « B ») est générée de manière à contenir les données inverses (du type Vdd-VtrameA) afin de permettre au composant destinataire d'effectuer une vérification de la symétrie des 2 signaux.
- La fréquence maximale est théoriquement 100Khz.

L'application implantée dans l'AT94K10 d'Atmel produit une trame A de 32 bits de long de durée 16ms par période de 2 secondes.

4.5.2. Protocole de test laser

Des tests laser ont été réalisés chez EADS pour le circuit analysé par nos outils. Dans cette expérience, les trames A et B générées sont toujours les mêmes et servent de références. Lors des essais d'injection de fautes, les trames successives sont enregistrées à chaque cycle de test via l'oscilloscope en sortie du FPGA sur les deux broches dédiées du composant. Elles sont alors comparées aux trames de référence à l'aide d'un calcul mathématique de moyenne sur la valeur de l'ensemble du signal. La finesse de cette comparaison permet de discriminer les trames dont un seul bit apparaît modifié. Le même processus de test est répété pas à pas pour réaliser une cartographie : à chaque pas, après chaque tir laser, d'une part la mémoire de configuration est relue afin de détecter les SEU et d'autre part les trames de l'application sont enregistrées afin de détecter un SEFI.

4.5.3. Protocole de test proton

Une autre expérience a été mise en place par EADS à l'accélérateur de l'Université Catholique de Louvain-la-Neuve (UCL) en Belgique. Le Tableau 26 fournit les données des 3 tests effectués avec des flux de protons de 3 énergies différentes : 22, 50 et 62 MeV.

Tableau 26: Paramètres utilisés pour les irradiations protons [BOC09].

Temps d'irradiation (sec.)	Energie (MeV)	Fluence (p)	Flux moyen
3960	62	5,15E+11	1,30E+8
2636	50	5,34E+11	2,02E+8
2604	22	1,26E+12	4,83E+8

Selon ces paramètres, le composant a été bombardé par des centaines de millions de protons au cours des expériences effectuées pour chacune des énergies. Les protons ont interagi avec le silicium dans les zones actives du FPGA Atmel. Ces interactions ont régulièrement induit des SEU dont certains ont induit des SEFI ayant interrompu l'application testée.

4.5.4. Etude prédictive de criticité de l'application

Le fichier de configuration de l'émetteur-récepteur, utilisé pendant les campagnes d'injections de fautes, a été analysé par notre outil d'analyse statique et nous avons pu établir la classification de criticité CTS pour les différents niveaux d'analyse (algorithmes Local, Global, et Global Etendu). Le Tableau 27 résume les résultats pour les bits configurant les ressources logiques des PLB (plus de 184000 bits de configuration pour le tableau de cellules entier). On voit clairement l'avantage apporté par les algorithmes Globaux par rapport au Local: il y a 51% de bits suspects en moins dans la classification pour l'algorithme Global et 60% de bits suspects en moins pour le Global Etendu. Il reste cependant 17,7% de bits suspects dans la configuration des PLB pour cet exemple.

Tableau 27: Résultats de criticité pour l'émetteur-récepteur industriel.

Algorithmes\ Classes	**Local**	**Global**	**Global Etendu**
Critique	5424 (2,9%)	15013 (8,1%)	20327 (11%)
Transparent	97794 (53,1%)	129449 (70,2%)	131447 (71,3%)
Suspect	81102 (44%)	39858 (21,6%)	32546 (17,7%)

On note que l'analyse locale ne permet pas de classifier la plupart des bits de configuration autrement que "Suspect", et seulement 53,1% de bits sont "Transparents". Avec l'algorithme Global Etendu, 71,3% des bits sont classés "Transparent", nous permettant d'améliorer notre évaluation prédictive du facteur de décote ("derating factor") pour une évaluation pessimiste (au pire des cas) de la robustesse de l'application. Cette augmentation de précision de l'analyse entraîne en revanche une nette augmentation de la complexité de l'algorithme, avec des temps de calcul passant de 163 ms pour l'algorithme local sur un ordinateur classique, à 150s et 160s respectivement pour les algorithmes Global et Global Etendu sur la même plateforme. Ces durées restent néanmoins négligeables d'un point de vue utilisateur.

4.5.5. Résultats des tests laser dynamiques et corrélation laser vs. analyses statiques

Le premier résultat intéressant de la campagne d'injection de fautes par laser est la multiplicité obtenue pour les différentes fautes injectées, afin de la comparer avec l'hypothèse d'inversion unique à partir de laquelle nous avons travaillé pour notre analyse statique de criticité. La Figure 60 a déjà été présentée en début de chapitre.

La Figure 71 présente la répartition des fautes obtenues dans les cellules logiques. Nous avons ici répertorié les SEUs provoqués et les SEFIs engendrés. Les bits modifiés avec la plus grande occurrence sont les bits configurant les vias locaux ainsi que les bits configurant les multiplexeurs

d'entrée, avec respectivement 180 et 173 SEUs. Les LUTs et les vias express ont eux subi respectivement 159 et 79 inversions uniques. Dans tous les cas, les SEUs dans les bits classifiés "Transparent" n'ont pas conduit à une perturbation fonctionnelle. Par ailleurs, peu de SEUs dans les bits "Suspect" ont eu un impact réel, notamment pour les bits de configuration "Vias express".

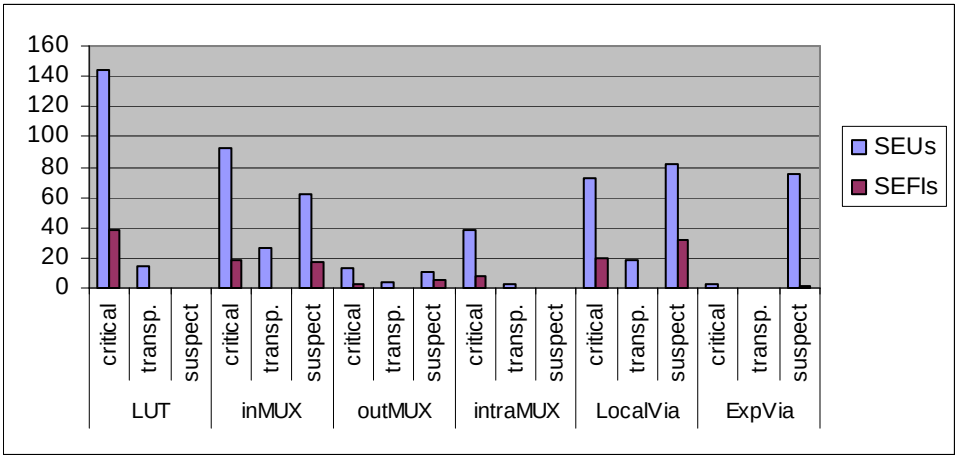

Figure 71: Répartition des fautes lors des injections de fautes laser.

Le Tableau 28 présente de façon synthétique la corrélation des résultats d'injection de fautes avec ceux de notre analyse statique de criticité. Comme nous l'attendions, tous les SEUs touchant des bits critiques n'ont pas engendré de SEFI. En effet, l'impact d'une faute dépend de l'instant d'injection. Cependant notre évaluation de criticité est fiable puisque aucun bit modifié mais analysé comme "Transparent" n'a provoqué de SEFI, et ce quelque soit l'algorithme utilisé.

Tableau 28: Corrélation avec les tests dynamiques laser.

Résultats\ Algorithmes	**Local**	**Global**	**Global Etendu**
SEUs sur des bits critiques	136	268	450
SEUs sur des bits transparents	52	68	68
SEUs sur des bits suspects	620	472	290
SEFIS provoqués par des bits critiques	26	49	87
SEFIS provoqués par des bits transparents	0	0	0
SEFIS provoqués par des bits suspects	122	99	61

Nous observons des SEUs pour toutes les classes. Au moins un SEFI a été créé en impactant des bits "Critiques" pour chaque catégorie, exceptée les bits configurant les via Express. Le Tableau 29 résume le pourcentage de SEUs ayant provoqué des SEFIs pour les bits de configuration classés "Critique". On a observé en moyenne que 19,75% des bits critiques ont induit un SEFI après un SEU. L'exception remarquée des via Express peut s'expliquer par leur emploi relativement faible dans l'application utilisée pour ces expériences.

Il faut également noter que notre classification est probablement trop pessimiste, car un faible nombre de SEFIs provoqués par des bits suspects a été relevé.

Tableau 29: Taux de création de SEFIs par les bits "Critiques" pour les injections de fautes laser.

Type de bit	LUT	InMux	OutMux	IntraMux	LocalVia	ExpVia
SEUs -> SEFIs	26,4%	19,6%	23,1%	22%	27,4%	0%

4.5.6. Résultats des tests dynamiques protons et corrélation protons vs. analyses statiques

La Figure 72 détaille les résultats globaux des expériences d'irradiation aux protons pour les différentes catégories des bits de configuration des PLBs. Les bits les plus fortement modifiés sont les bits de configuration des multiplexeurs d'entrée et les bits configurant les vias Express avec respectivement 23 et 15 SEUs. Les bits Local Via et de configuration des LUTs présentent 12 et 11 inversions respectivement. Du fait du nombre plus faible de fautes créées pendant les expériences d'irradiation proton, il y a un nombre plus faible de SEFIs que pendant les expériences laser, en particulier pour les bits de configuration des multiplexeurs de sortie et les vias Express. Des fautes sont toutefois observées parmi toutes les catégories de bits (45% des inversions sur des bits classés "Suspect", 15% sur des bits "Transparents", et 40% sur des bits "Critiques"). Ces données confirment que notre classification semble trop pessimiste, notamment pour les bits configurant les via Express. Toutefois, elles confirment aussi que les bits classifiés comme "Transparent" n'ont pas induit de SEFI.

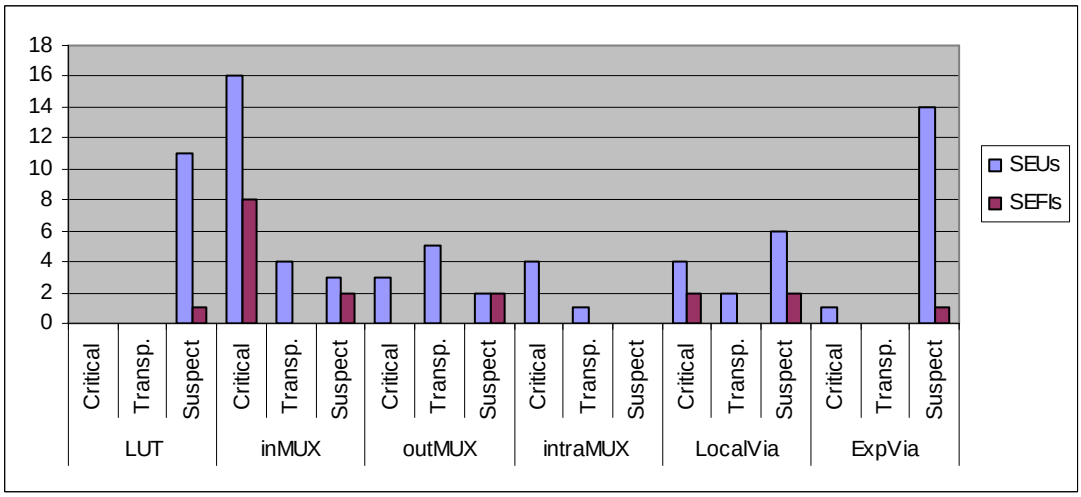

Figure 72: Répartition des fautes lors des irradiations protons.

Le Tableau 30 corrèle l'analyse statique de criticité statique réalisée avec l'algorithme le plus performant avec les résultats de deux des trois campagnes réalisées avec l'accélérateur de particules UCL. La première partie du tableau présente les nombres totaux d'évènements observés sur les

différentes classes de bits de configuration. La deuxième partie présente le nombre de bits concernés par ces évènements (SEUs et SEFIs). On remarque tout d'abord que le nombre d'inversions provoquées et de SEFIs est bien moindre que lors des expériences laser. En moyenne, 5 fois moins de SEUs/SEFIs ont été observés. Toutefois, aucun bit classé "Transparent" n'a engendré de SEFI, notre classification est donc conservative. Les bits ayant provoqués des SEFIs sont en majorité classifiés "Critique" (56%), le reste est "Suspect".

Tableau 30: Corrélation avec les résultats d'injection de fautes par irradiation de protons.

Expériences	Irradiation Protons R1	Irradiation Protons R2	Total	Pourcentage
SEUs sur des bits critiques	11	50	61	41%
SEUs sur des bits transparents	29	8	37	25%
SEUs sur des bits suspects	16	34	50	35%
SEFIS provoqués par des bits critiques	3	7	10	45%
SEFIS provoqués par des bits transparents	0	0	0	0%
SEFIS provoqués par des bits suspects	9	3	12	55%
Bits critiques touchés par au moins un SEU	11	27	38	40%
Bits transparents touchés par au moins un SEU	6	8	14	15%
Bits suspects touchés par au moins un SEU	12	32	44	45%
Bits critiques ayant provoqué au moins un SEFI	3	7	10	56%
Bits transparents ayant provoqué au moins un SEFI	0	0	0	0%
Bits suspects ayant provoqué au moins un SEFI	5	3	8	44%

En ce qui concerne la multiplicité des erreurs de configuration, 75% des fichiers de configuration modifiés par les irradiations protons présentaient des inversions simples, 20% avaient deux bits erronés, 4% avaient 3 bits erronés, et finalement 1% des fichiers présentaient 4 bits erronés. Comme pour les injections de fautes au laser, les SEUs sont donc le cas principal à traiter, même si des erreurs multiples sont présentes. La principale différence est la proportion plus grande d'erreurs dans la configuration du FPGA plutôt que dans les points mémoires utilisateurs.

Le Tableau 31 résume le pourcentage de SEUs produits par des bits classés "Critique" par notre outil d'analyse, qui ont produit des SEFIs pendant les expériences d'irradiation proton. La moyenne est de 20%, un résultat similaire à celui obtenu pendant les expériences d'injection de fautes laser. On remarque la différence du pourcentage pour les LUTs, qui s'explique par le fait qu'aucun des bits considérés "Critique" par nos algorithmes n'a pu être inversé pendant les expériences d'irradiations protons.

Tableau 31: SEFIs produits parmi les bits critiques des PLBs après un SEU.

Type de bit	LUT	inMUX	outMUX	intraMUX	LocalVia	ExpVia
SEUs -> SEFIs	0%	50%	0%	0%	50%	0%

4.5.7. Influence de l'application sur l'évaluation de criticité

Nous avons analysé avec nos outils un ensemble d'exemples différents, provenant d'applications industrielles et de suites de test ("benchmarks").

Tableau 32: Caractéristiques des applications étudiées (pour une implantation sur Atmel AT40K).

Application\ Ressources	Entrées/Sorties	Générateurs de fonctions	Bascules
c3540	72	374	0
c74181	22	23	0
c74283	14	17	0
FIR	41	657	322
HC11	438	2134	184
SBox	19	127	14
Module E/T	96	456	273

Le Tableau 32 résume les principales caractéristiques de ces applications implantées sur AT40K. Les circuits c3540, c74181, et c74283 sont des circuits purement combinatoires et proviennent de l'ensemble d'applications ISCAS. Il s'agit respectivement d'une Unité Arithmétique et Logique sur 8 bits, d'une autre UAL sur 4 bits, et enfin d'un additionneur sur 4 bits également. FIR représente un filtre à réponse impulsionnelle finie, le HC11 est un microcontrôleur 8 bits, et la SBOX est un module d'un cryptoprocesseur AES. Finalement, le module E/T est l'émetteur-récepteur déjà utilisé dans les sections précédentes. On retrouve dans le Tableau 32 le nombre d'entrées/sorties de chaque circuit, et leur complexité estimée par le nombre de générateurs de fonctions qu'ils occupent dans le FPGA. La dernière colonne présente le nombre de bascules des applications.

Tableau 33: Résultats exhaustifs de criticité.

Application et classes de criticité\ Algorithmes	Local	Global	Global Etendu
c3540			
Critiques	5259 (2,9%)	15144 (8,2%)	20068 (10,9%)
Transparents	102546 (55,6%)	133396 (72,4%)	134824 (73,1%)
Suspects	76515 (41,5%)	35780 (19,4%)	29428 (16%)
c74181			
Critiques	342 (0,2%)	658 (0,4%)	860 (0,5%)
Transparents	123012 (66,7%)	160083 (86,8%)	160153 (86,9%)
Suspects	60966 (33,1%)	23579 (12,8%)	23307 (12,6%)
c74283			
Critiques	248 (0,1%)	643 (0,3%)	859 (0,4%)
Transparents	123282 (66,9%)	159821 (86,7%)	159941 (86,8%)
Suspects	60790 (33%)	23856 (12,9%)	23520 (12,8%)
FIR			
Critiques	10246 (5,6%)	28755 (15,6%)	37513 (20,4%)
Transparents	79920 (43,4%)	104014 (56,4%)	108344 (58,8%)
Suspects	94154 (51%)	51551 (28%)	38463 (20,8%)
HC11			
Critiques	26231 (14,2%)	75599 (41%)	96789 (52,5%)
Transparents	3834 (2,1%)	12619 (6,9%)	25861 (14%)
Suspects	154255 (83,7%)	96102 (52,1%)	61670 (33,5%)
SBox			
Critiques	214 (0,1%)	696 (0,4%)	974 (0,5%)
Transparents	123336 (66,9%)	159833 (86,7%)	159875 (86,8%)
Suspects	60770 (33%)	23791 (12,9%)	23471 (12,7%)
R/T module			
Critiques	5424 (2,9%)	15013 (8,1%)	20327 (11%)
Transparents	97794 (53,1%)	129449 (70,2%)	131447 (71,3%)
Suspects	81102 (44%)	39858 (21,6%)	32546 (17,7%)

Le Tableau 33 résume les résultats de criticité pour les bits configurant les ressources logiques des cellules PLB (184320 bits de configuration). On note que l'algorithme Local classe une grande partie

des bits de configuration comme "Suspect". L'amélioration due aux algorithmes globaux est appréciable pour tous les exemples. La différence entre les deux algorithmes globaux dépend en revanche beaucoup de l'exemple considéré. Ceci est dû à la différence de complexité des applications considérées, et plus particulièrement au nombre de générateurs de fonction utilisés par les implantations des fonctions combinatoires.

Dans une approche pour des applications critiques, tous les bits classés "Critique" ou "Suspect" doivent être considérés comme dangereux du point de vue du comportement nominal de l'application. Le Tableau 34 présente donc le nombre de bits de configuration dangereux du point de vue de l'application, classés avec nos trois algorithmes de criticité. Ce chiffre est directement relié aux taux de déviation de fonctionnement normaux (ou au facteur de décote) que les concepteurs doivent prendre en compte dans leur estimation de fiabilité de l'application. Pour tous les exemples, moins d'un tiers des bits de configuration sont dangereux donc l'estimation du taux d'erreur de l'application serait surestimée d'un facteur 3 sans ce type d'analyse. Pour des exemples plus complexes (avec un taux d'occupation des ressources du FPGA plus important), ce ratio décroit, même si la corrélation n'est pas linéaire. Pour les exemples les plus complexes (HC11), plus de 90% des bits de configuration sont potentiellement dangereux. Cependant, le fait que cette analyse soit statique rend un diagnostic pessimiste ; nous avons vu précédemment que le taux d'erreur observé lors des tests dynamiques est toujours inférieur à celui estimé à partir de l'analyse statique de criticité.

Tableau 34: Bits dangereux pour l'ensemble des applications étudiées.

Exemple\ Algorithmes	Local	Global	Global ext.
c3540	44,4%	27,6%	26,9%
c74181	33,3%	13,2%	13,1%
c74283	33,1%	13,2%	13,2%
FIR	56,6%	43,6%	41,2%
HC11	97,9%	93,1%	86%
SBox	33,1%	13,3%	13,2%
R/T module	46,9%	29,7%	28,7%

4.5.8. Comparaison avec l'étude de criticité pour Xilinx

Nous avons souhaité comparer les résultats obtenus au chapitre 3 et ceux obtenus au chapitre 4. En raison des différences d'architecture décrites au chapitre 1, il a été nécessaire de normaliser les résultats afin d'avoir une meilleure comparaison de la criticité sur les deux plateformes. Deux techniques de normalisation ont été utilisées pour cela.

La première méthode de normalisation se base sur la complexité logique globale du dispositif, évaluée comme étant le nombre total de bits de configuration des éléments de calcul logique, c'est-à-dire configurant les LUTs. Le nombre de bits de configuration de chaque catégorie (Critique / Transparent / Suspect) est ensuite divisé par le nombre total de bits configurant les LUTs. Nous n'utilisons pas de pourcentage, puisque les bits de configuration d'interconnexions ne sont pas pris en compte.

La seconde normalisation est basée sur le nombre total de bits effectivement utilisés pour implanter une fonction donnée. Ce nombre est estimé en considérant non seulement les cellules utilisées par le circuit logique mais également les bits des autres cellules (même s'ils sont utilisés uniquement pour router des signaux). Bien sûr, dans ce cas, le nombre de bits critiques / transparents / suspects a été calculé sur le même ensemble.

Le Tableau 35 récapitule les résultats obtenus après la première normalisation pour l'exemple des circuits ISCAS, avec l'algorithme de criticité Local (puisque les autres algorithmes ne sont pas disponibles sur la plateforme Xilinx). Le Tableau 36 présente les mêmes résultats pour la seconde normalisation.

Tableau 35: Comparaison des résultats de criticité Atmel/Xilinx (1ere normalisation, algorithme Local).

Applications \ Classes de criticité	**Critique**	**Transparent**	**Suspect**
Atmel			
c3540	0,1	2,8	2,1
c74181	0,0	3,3	1,7
c74283	0,0	3,3	1,6
Xilinx			
c3540	0,2	13,1	0,5
c74181	0,0	13,4	0,3
c74283	0,0	13,4	0,3

La première normalisation montre que davantage de bits sont identifiés comme "Transparents" dans l'architecture Xilinx que dans celle d'Atmel (proportionnellement aux capacités logiques respectives des deux dispositifs). Le ratio de bits critiques est à peu près équivalent mais le pourcentage de bits suspects est plus élevé dans l'architecture Atmel, ce qui implique un plus grand taux de bits potentiellement dangereux.

Tableau 36: Normalisation en fonction du taux d'occupation logique des implémentations (2eme normalisation, algorithme Local).

Applications \ Classes de criticité	Critique	Transparent	Suspect
Atmel			
c3540	16,2%	0%	83,8%
c74181	16,4%	0%	83,6%
c74283	14,8%	0%	85,2%
Xilinx			
c3540	33,6%	51,5%	14,9%
c74181	35,5%	50,2%	14,3%
c74283	14,4%	74,3%	11,3%

Le Tableau 37 montre que, après l'analyse avec l'algorithme Local, tous les bits de configuration utilisés (c'est à dire. tous les bits des cellules utilisées pour implanter les fonctions) sont considérés comme potentiellement dangereux dans la plateforme Atmel, alors que seuls 25% à 50% des bits sont considérés comme potentiellement dangereux dans l'architecture Xilinx. Cela s'explique, en particulier, par le grand nombre de bits de configuration rattachés à des multiplexeurs inutilisés et à l'encodage redondant des configurations de connexion, puisque seules les inversions de bit unique sont prises en compte.

Le Tableau 37 montre que, après une analyse avec l'algorithme Global Etendu, la plupart des bits de configuration des cellules utilisées de la plateforme Atmel sont toujours considérés comme potentiellement dangereux, mais ce pourcentage dépend des caractéristiques exactes du circuit implanté. Toutefois, précisons que ce résultat n'implique pas que l'architecture Virtex II est plus fiable que l'AT40K. En effet, la configuration du dispositif Virtex utilise davantage de bits, ainsi la probabilité d'avoir une inversion de bit dangereux est supérieure, puisque le nombre de bits dangereux est largement supérieur, comme illustré dans le Tableau 37.

Tableau 37: Pourcentage de bits non sûrs évalué pour les circuits ISCAS sur Xilinx Virtex II et Atmel AT40K, pour le meilleur algorithme disponible sur chaque plateforme.

Applications	Atmel, Global Etendu	Xilinx, Local
c3540	26,9% de 184320 = 49582	5,2% de 2252800 = 117145
c74181	13,1% de 184320 = 24146	2,4% de 2252800 = 54067
c74283	13,2% de 184320 = 24330	2,3% de 2252800 = 51814

Les tableaux précédents montrent qu’un outil d’analyse statique, comme celui présenté dans ce manuscrit, est plus efficace pour les circuits implantés sur la technologie Virtex que pour ceux implantés sur AT40K, puisqu’il permet d’identifier davantage de bits sûrs, et ainsi produire des taux de décote supérieurs. Mais cela ne signifie pas que la technologie Atmel est moins bonne : elle est probablement plus optimisée en nombre de bits de configuration et il est par là même plus difficile de décider de manière statique si un bit est critique ou non.

4.6. Conclusion

Ce chapitre présente une méthode d'analyse statique de la criticité des bits de configuration du FPGA ATMEL AT40K face à la problématique des SEUs. Une classification des bits est établie selon leur potentiel de génération de SEFIs suite à une inversion simple. Nous avons développé plusieurs algorithmes ayant chacun un niveau de complexité et de performance différents. Afin de valider cette approche, l'analyse de plusieurs campagnes d'injections de fautes, à base d'injections laser ou d'irradiations protons, est présentée. La comparaison de la liste de bits de configuration ayant généré des SEFIs et des résultats d'analyse statique de criticité du bitstream permet d'affirmer que cette technique est viable car elle donne très rapidement des résultats qui, bien que pessimistes, sont conservatifs. Une comparaison avec les résultats obtenus sur la plateforme Xilinx a été réalisée et a mis en évidence l'impact de la plateforme d'implantation dans la criticité aux SEUs d'une application.

Le chapitre suivant présente une étude de faisabilité d'une méthodologie de fiabilisation innovante et compétitive par rapport à l'état de l'art, pour les applications implantées sur le FPGA ATMEL AT40K.

5. Méthode de conception durcie, étude de faisabilité

Ce dernier chapitre a pour but l'étude de faisabilité d'une solution de protection des architectures implantées sur FPGA SRAM, qui s'appuie sur nos méthodologies d'évaluation statique de criticité des bits de configuration de ces composants. Nous nous proposons de montrer qu'il est possible de concevoir des systèmes plus robustes contre les SEUs, mais plus compétitifs en termes de surcoût matériel et de dégradation des performances que les solutions présentées dans l'état de l'art.

5.1. Hypothèse de travail

Les expériences réalisées pour l'évaluation de nos outils d'analyse de criticité nous ont permis d'observer de très faibles multiplicités dans des conditions représentant l'effet de radiations naturelles, tant pour les fautes produites par tirs laser que par irradiations protons, confortant l'hypothèse utilisée par nos méthodologies d'analyse. La suite de nos travaux, présentée dans ce chapitre, se base donc sur les mêmes hypothèses, en considérant uniquement le cas des fautes uniques, les SEUs, ou des fautes multiples non corrélées d'un point de vue configuration.

Nous utiliserons un FPGA commercial (le FPGA ATMEL AT40K) comme support pour cette étude de faisabilité. Cela implique donc de travailler avec une architecture prédéfinie et non-modifiable, et exclue également toute possibilité de modification de la structure de configuration afin d'améliorer la robustesse des circuits implantés.

C'est dans ce contexte et avec ces contraintes que nous proposons une approche pour l'amélioration de la fiabilité des applications sur FPGA SRAM face aux problèmes liés aux SEUs.

5.2. Proposition d'une approche de durcissement

Nos travaux se concentrent sur les bits de configuration des LUTs des cellules logiques, bits programmant l'ensemble des fonctions logiques d'une application implémentée sur la carte. La protection de l'application se base sur l'analyse statique de criticité des bits de configuration effectuée par nos algorithmes globaux présentés précédemment. Le principe est ensuite de mettre en œuvre une duplication sélective et locale (propre au secteur considéré) des fonctions logiques configurées par des bits diagnostiqués "critiques". La comparaison des valeurs propagées par la logique originale et sa réplique permet l'émission d'un signal d'erreur propre à chaque secteur logique. Un état actif de ce signal permettra de déclencher une reconfiguration partielle du secteur logique ne produisant pas des valeurs conformes aux spécifications de l'application implantée. Notre proposition utilise les blocs logiques laissés libres par le routeur de l'application sur la plateforme. Ceci permet d'obtenir une amélioration sensible de la robustesse du circuit sans induire de coût matériel réel. Ainsi, cette proposition se démarque nettement des autres architectures pour la robustesse présentées dans l'état de

l'art. Nous verrons dans la suite de ce chapitre l'efficacité que l'on peut espérer d'un tel dispositif en fonction du taux d'occupation initial du FPGA.

5.2.1. Duplication locale et sélective des fonctions logiques

La Figure 73 représente les bits de configuration des cellules logiques de base du FPGA ATMEL AT40K, et en particulier les bits ciblés par notre système de protection, à savoir les 16 bits configurant les fonctions logiques implantées dans les LUTs. Une cellule logique compte 80 bits de configuration, ces bits représentent donc 20% de la configuration des cellules de base.

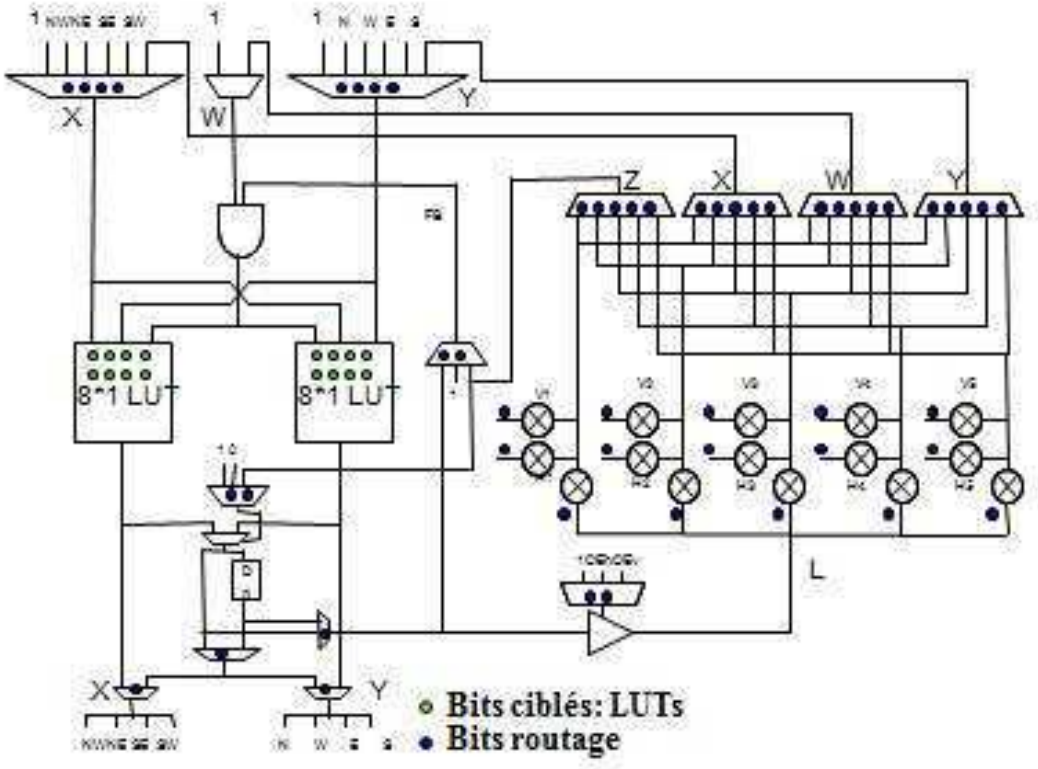

Figure 73: Bits de configuration ciblés dans les cellules de base

Il est d'abord nécessaire de séparer les fonctions logiques à protéger de celles ne pouvant pas produire d'erreurs fonctionnelles. Ceci est réalisé grâce à l'algorithme Global Etendu de l'outil SEFEA-ProD pour FPGA ATMEL. Dans un deuxième temps, on doit séparer les fonctions logiques critiques en plusieurs sous-ensembles :

- Les fonctions logiques de trois variables ou moins, implantées par un LUT 8 bits d'une cellule de base, n'occupant que la moitié de l'espace logique fourni par une cellule de base (sous-ensemble A).
- Les fonctions logiques de quatre variables, implantées par un LUT de 16 bits constitué par la concaténation des deux LUTs 8 bits contenus dans une cellule logique (sous-ensemble B).

- Les fonctions logiques de trois variables ou moins, couplées dans une cellule de base (sous-ensemble C).

Afin d'optimiser le coût matériel de la duplication locale des fonctions logiques critiques, deux schémas sont proposés en fonction des contraintes imposées par l'architecture des cellules de base ATMEL et du nombre de variables d'entrée des fonctions considérées, un pour le sous-ensemble A, et un autre pour les sous-ensembles B et C.

Nous avons implanté des circuits avec des caractéristiques différentes sur l'AT40K (voir section 4.5.8) et étudié l'utilisation faite par les outils de placement/routage des cellules logiques. Il apparait qu'un nombre important (30% en moyenne) de cellules routées n'utilisent qu'un seul de leur LUT 8 bits. La Figure 74 présente le nombre de cellules de ce type (sous-ensemble A) et le nombre total de cellules routées pour chaque application implantée.

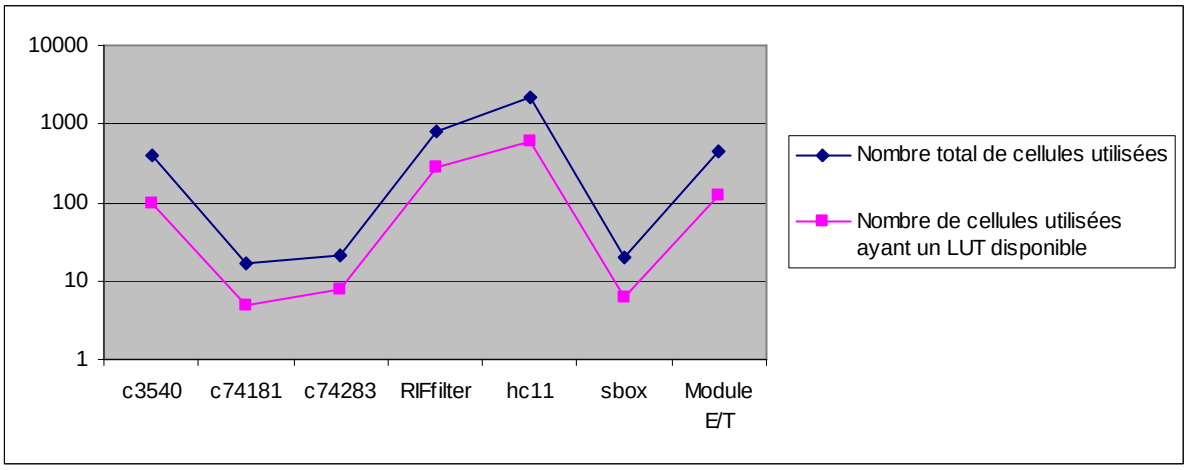

Figure 74: Cellules utilisées et cellules routées ayant un LUT libre sur les circuits étudiés.

L'optimisation du surcoût matériel implique donc deux propositions différentes en fonction du nombre d'entrées des fonctions logiques implantées sur les LUTs dont les bits de configuration sont diagnostiqués "Critiques".

- Schéma pour le sous-ensemble (A): Réplique + original + comparateur sur 2 cellules si fonction 3 entrées.
  - on peut améliorer l'intégration de la protection en fonction de l'occupation du secteur logique de la façon suivante: 2 répliques + 2 originaux + comparateur sur 3 cellules.
- Schéma pour les sous-ensembles (B) et (C): Réplique + original + comparateur sur 3 cellules si fonction 4 entrées.
  - on peut améliorer l'intégration de la protection en fonction de l'occupation du secteur logique de la façon suivante: 2 répliques + 2 originaux + comparateur sur 5 cellules.

Les deux parties suivantes présentent le détail de ces deux schémas.

5.2.2. Implantation pour les fonctions 3 entrées

L'implantation de la duplication logique des fonctions critiques du sous-ensemble A se fait au maximum sur deux cellules de base du même secteur. La réplique de la fonction logique comportant des bits de configuration critiques est implantée dans la même cellule que l'originale. Un autre LUT 8 bits d'une cellule voisine est nécessaire afin de réaliser un XOR permettant d'observer une déviation dans le fonctionnement standard. La Figure 75 représente cette architecture.

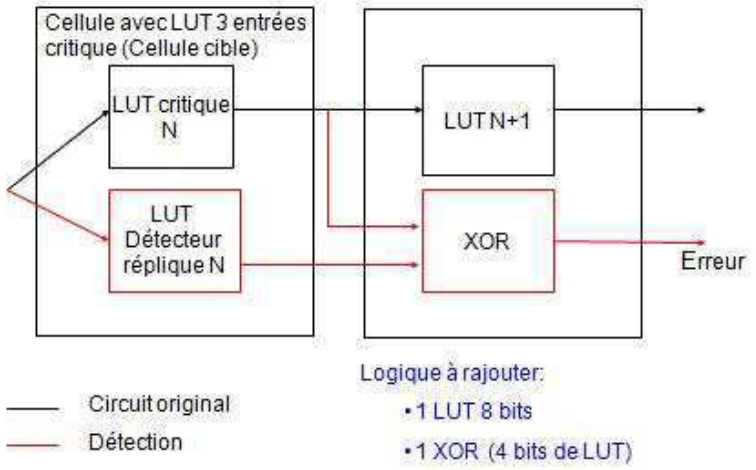

Figure 75: Schéma de principe de duplication locale des fonctions trois entrées.

Le coût matériel de cette duplication s'élève à 12 bits de LUTs. Le routage direct de deux cellules est également modifié. Si le secteur logique considéré présente une cellule de base vide, on peut intégrer ce schéma comme indiqué dans le paragraphe précédent.

5.2.3. Implantation pour les fonctions à quatre entrées

La Figure 76 représente le dispositif de duplication pour les fonctions appartenant aux deux autres sous-ensembles des fonctions critiques à protéger (B et C).

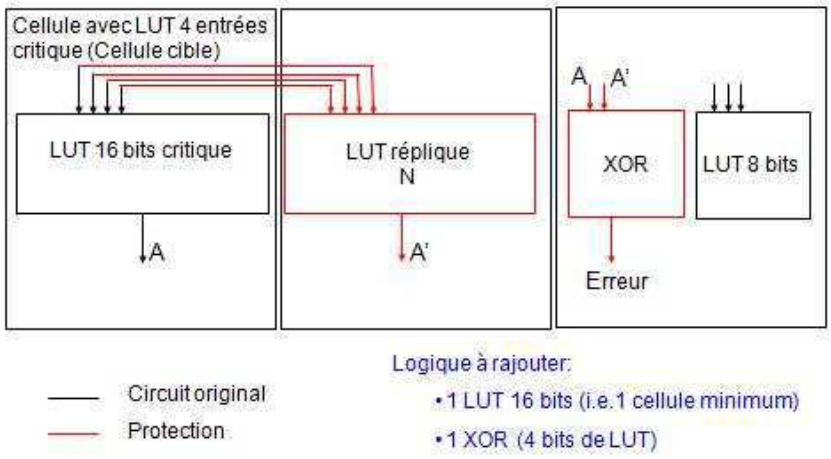

Figure 76:Schéma de principe de duplication locale des fonctions 4 entrées

Il est nécessaire de disposer d'une cellule de base vide dans le même secteur logique afin de pouvoir implanter une réplique de la fonction logique originale. Un autre LUT 8 bits appartenant à une troisième cellule du secteur implante un XOR pour propager un signal d'erreur. Le coût matériel est ici de 20 bits de LUTs sans compter les modifications locales à apporter au routage des signaux. Il est également possible d'intégrer plusieurs XOR dans une cellule de base afin de diminuer le coût de la comparaison.

5.2.4. Routage des signaux d'erreur

Un signal d'erreur est émis par secteur logique (16 cellules de base) du FPGA. Nous l'appelons le signal "d'erreur sortante" du secteur considéré.

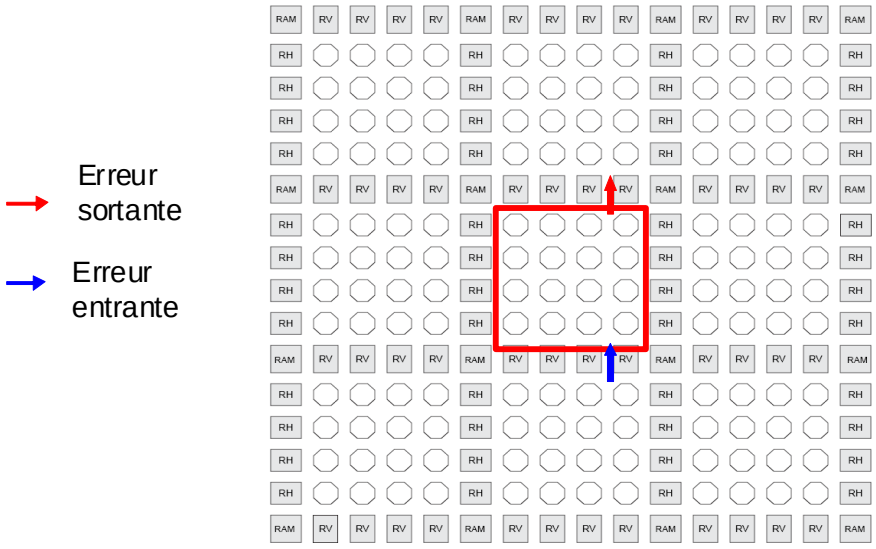

Figure 77: Routage de la combinaison des signaux d'erreur.

La valeur du signal d'erreur sortante est évaluée par un OU logique entre les erreurs produites par les duplications locales propres au secteur, et l'erreur sortante d'un autre secteur logique, propagée sur le réseau de routage du FPGA. Cette dernière peut être considérée comme l'erreur entrante du secteur logique considéré. Le système peut être ainsi représenté par une chaîne logique évaluant l'erreur globale des secteurs contenant des fonctions logiques critiques de l'application. Ceci est représenté Figure 77.

Afin de respecter le chemin critique initial de l'application, il est cependant nécessaire de fractionner le FPGA en plusieurs zones ayant chacune un signal d'erreur associé. Le nombre de zones à créer est propre à chaque application, et dépend à la fois de son chemin critique et du nombre de cellules logiques contenant des fonctions critiques. Il faut noter que le fait d'avoir plusieurs signaux d'erreur n'augmente pas le coût logique de la protection de l'application, mais induit des contraintes supplémentaires sur le placement final de l'application et sur les entrées/sorties.

Une faute dans la configuration du routage des signaux d'erreurs, qui serait synchrone d'une faute dans la logique protégée, est problématique dans le cas où elle entraine un conflit logique. Il faut noter qu'une reconfiguration provoquée par une telle erreur n'a pas d'impact sur la fiabilité de l'application implantée, mais sur ces performances. Cette possibilité n'entre cependant pas dans le cadre de cette étude, consacrée aux fautes simples (type SEU).

5.2.5. Résilience du système dans le cas de fautes doubles

Même si ce système de protection se positionne dans un contexte de fautes uniques, on peut donner quelques éléments quant à sa résilience en cas de fautes doubles. Afin de préserver la qualité du système de détection de faute, il faut conserver le chemin de données partant de l'entrée du bloc logique répliqué jusqu'à la sortie du comparateur. Confrontons ce fait avec nos deux schémas d'implantation introduits précédemment :

- Cas des fonctions du sous-ensemble A :

Les fautes doubles sont indésirables ici car la fonction originale et sa réplique sont implantées dans la même cellule logique. Les fautes contenues à l'intérieur d'un même LUT ne mettront cependant pas en défaut le système. Il en est de même des fautes non corrélées sur le plan architectural.

- Cas des sous-ensembles B et C :

Les fautes doubles peuvent être acceptées dans la réplique de la fonction logique ou dans la cellule originale. Dans les deux cas, la détection pourra être opérationnelle. La probabilité d'une faute double non détectée est non nulle, mais très faible.

5.3. Implantation sur la plateforme ATMEL

Cette partie illustre les possibilités d'implantation de notre proposition sur la plateforme ATMEL AT40K. Nous représentons ici le meilleur cas possible, afin d'illustrer l'intégration maximale d'un tel système. Lorsque le placement original de l'application ne permet pas un tel regroupement des ensembles {fonction logique critique - réplique logique}, un nouveau placement doit être effectué, avec plus ou moins de contraintes en fonction du taux d'occupation original du FPGA de la version non protégée de l'application. Les deux prochaines sous-parties présentent l'implantation des fonctions logiques critiques des sous-ensembles B et C d'une part, et du sous-ensemble A d'autre part.

5.3.1. Schéma 4 entrées, détection maximale

Chaque secteur logique étant constitué de seize cellules de base, il est possible de dupliquer 6 fonctions des sous-ensembles B et C par secteur, tout en incluant la logique nécessaire à la production d'une erreur sortante. Ceci est illustré par la Figure 78.

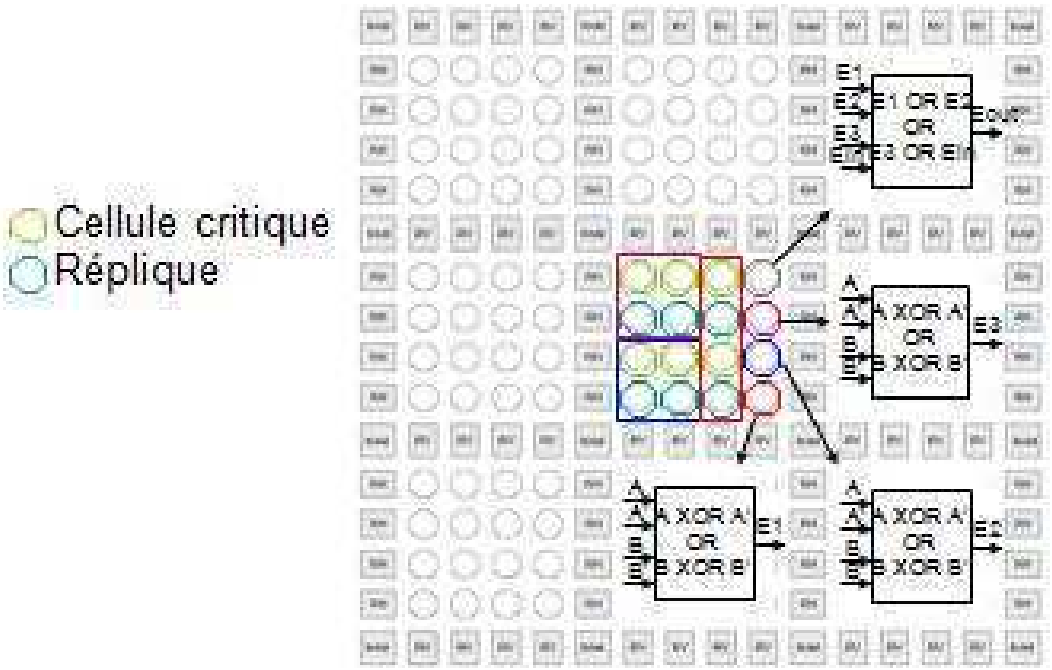

Figure 78: Implantation des fonctions quatre entrées dans les cellules de base de l'AT40K.

Douze cellules permettent d'implanter les fonctions critiques originales et leurs répliques (partie gauche du secteur). Trois autres (dans la partie droite du secteur) produisent des erreurs partielles (E1, E2, E3), chacune correspondant à deux paires « fonction critique/réplique » (sorties A, A' et B, B'). Une dernière cellule permet d'implanter l'erreur sortante (Eout) du secteur à partir des trois erreurs partielles et de l'erreur entrante (Ein). Les fonctions XOR et OR impliquées sont indiquées sur la figure.

Cet agencement représente la meilleure intégration possible de notre système. On peut ainsi considérer qu'il sera possible de protéger jusqu'à 6 fonctions critiques des sous-ensembles B et C par secteur de 16 cellules logiques à coût nul, c'est à dire en utilisant les blocs logiques inutilisés par le placement original de l'application. Un taux d'occupation plus important des ressources du FPGA implique une diminution du nombre de fonctions protégeables sans coût réel.

5.3.2. Schéma 3 entrées, détection maximale

De la même façon, il est possible de dupliquer 9 fonctions du sous-ensemble A par secteur, tout en incluant la logique nécessaire à la production d'une erreur sortante. Ceci est illustré par la Figure 79.

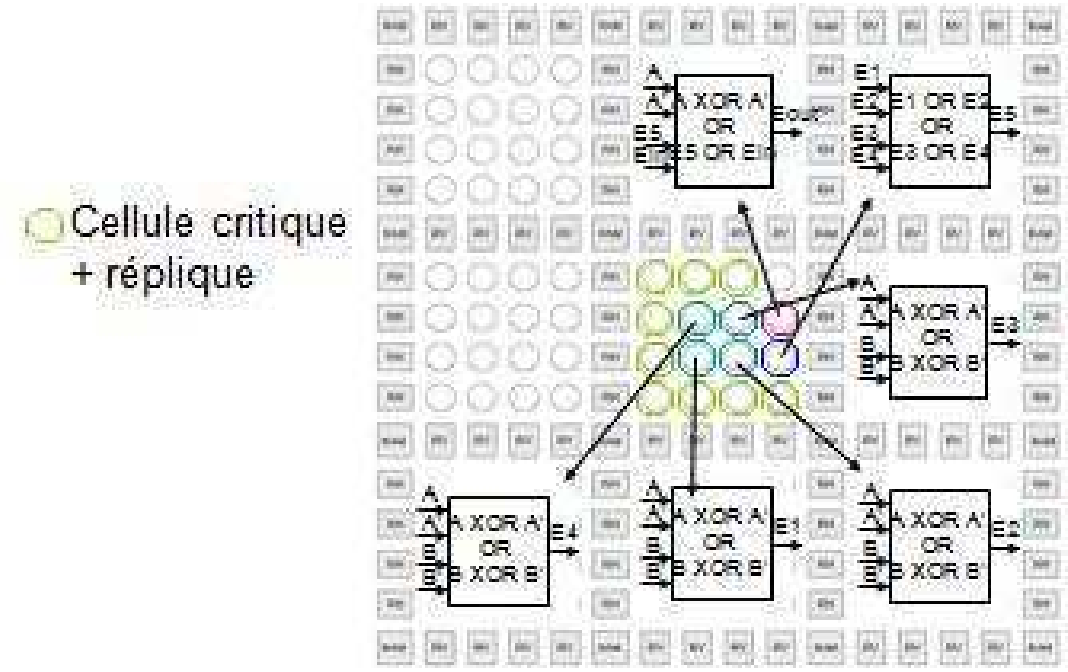

Figure 79: Implantation des fonctions trois entrées dans les cellules de base de l'AT40K.

Chaque cellule représentée en vert sur la figure ci-dessus contient une fonction critique et sa réplique. Il faut 6 LUT 16 bits (et donc 6 cellules logiques) pour produire le signal d'erreur sortant d'un tel secteur. Quatre cellules produisent les signaux d'erreurs partielles E1, E2, E3, et E4 correspondant à 8 paires "fonction critique/réplique". Deux autres comparent les sorties de la dernière paire et produisent le signal d'erreur du secteur (Eout) en faisant un "OU" logique de toutes les erreurs partielles. On peut considérer qu'il sera possible de protéger jusqu'à 9 fonctions critiques du sous-ensemble A par secteur de 16 cellules logiques à coût nul, c'est à dire en utilisant les blocs logiques inutilisés par le placement original de l'application.

5.3.3. Problèmes liés à l'encombrement des secteurs logiques

Dans le cas où il n'y a pas assez de cellules vides dans les secteurs logiques contenant des fonctions critiques, il est nécessaire de modifier le placement original afin d'implanter l'un des schémas présentés. S'il n'y a pas de secteur logique libre, il n'est alors pas possible d'obtenir une protection complète sans surcoût matériel réel. Dans ce cas, on peut quand même protéger une partie des fonctions critiques, la proportion protégeable pouvant-être exprimée mathématiquement, en fonction du taux d'occupation du FPGA.

5.4. Estimation mathématique du coût matériel réel

Cette partie vise à exprimer mathématiquement les conditions nécessaires pour protéger les fonctions critiques d'une application implantée sur AT40K avec notre méthodologie. On trouvera également l'expression de l'ensemble protégeable dans le cas où une protection complète n'est pas possible.

5.4.1. Cas optimal, conditions nécessaires pour la détection à coût matériel nul

On considère un FGPA AT40K ATMEL de M cellules. L'application implantée sur cette plateforme comporte N cellules se répartissant ainsi :

- X cellules comportant des fonctions critiques des ensembles B et C détaillés précédemment (cellules type BC).
- N-X cellules avec des fonctions critiques de l'ensemble A (cellules type A).

On peut estimer le surcoût matériel engendré par la protection de chaque type de fonction à partir des schémas d'implémentation sur cette plateforme :

- Protection de Y cellules de type A : coût = 2/3*Y

Six cellules sont nécessaires pour la protection de 9 cellules contenant des fonctions critiques. Le coût matériel pour protéger Y cellules sera donc 2/3*Y cellules.

- Protection Z cellules de type BC : coût = 5/3*Z

Dix cellules sont nécessaires pour la protection de 6 cellules contenant des fonctions critiques. Le coût matériel pour protéger Z cellules seront donc 5/3*Z cellules.

On peut déduire de ces formules la condition de protection à coût matériel nul pour une application implantée sur AT40K, c'est à dire les conditions nécessaires pour pouvoir inclure la protection dans les espaces laissés vacant par le placement original de l'application (sous réserve d'un routage ne conduisant pas à des congestions incontournables) :

- M-N >= 5/3*X + 2/3*(N-X).
- M>= X +5/3* N.

5.4.2. Détection partielle

Dans le cas où cette condition n'est pas remplie, on peut néanmoins exprimer le pourcentage de cellules protégeables pour chaque type.

Expression mathématique d'une détection partielle :

- $M >= X + 5/3 * N + \Delta$ où Δ est le nombre de cellules manquantes pour une détection complète

Le pourcentage des cellules de type A protégeables est déterminé comme suit (surcoût matériel divisé par l'espace laissé libre dans le FPGA par le placement original de l'application) :

- $Pa = 3*(M-N)/2*(N-X)$, $0<Pa<1$ avec $\Delta > 0$

Celui des cellules de type BC protégeables :

- $Pbc = 3*(M-N)/(5*X)$, $0<Pbc<1$ avec $\Delta > 0$

5.4.3. Illustration de faisabilité

Pour donner un aperçu des possibilités de cette technique, nous avons évalué le coût matériel de la protection du filtre RIF utilisé pour les expériences d'injections de fautes décrites aux chapitres 3 et 4.

5.4.3.1. Le filtre RIF

Le FPGA contient 2304 cellules réparties dans un tableau de 48 colonnes et 48 lignes. Le filtre, placé grâce au routeur IDS (voir la Figure 80), occupe 814 cellules du FPGA, comportant toutes au moins une fonction critique :

- 284 cellules de type A
- 530 cellules de type BC

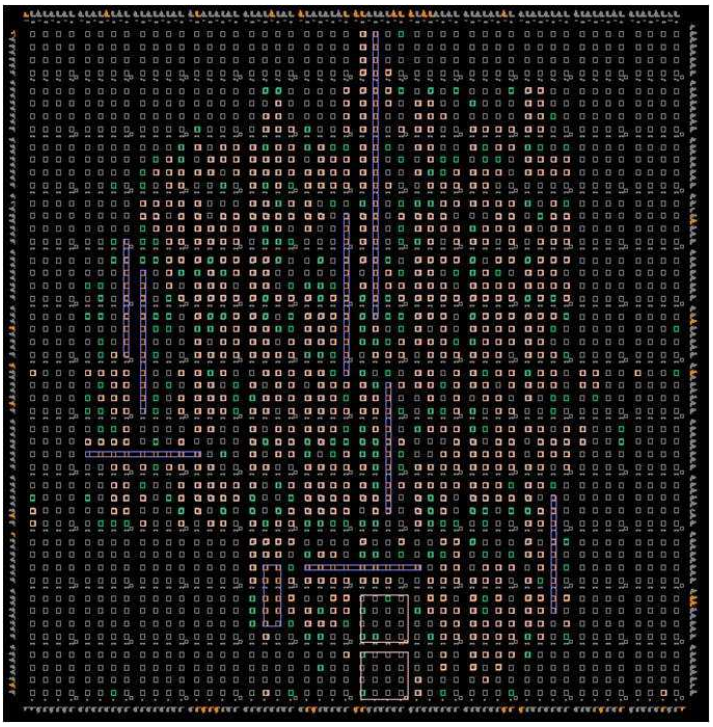

Figure 80: Implantation du FIR sur l'AT40K

5.4.3.2. Perspectives d'implémentation sur AT40K

Le coût matériel nécessaire à la protection des 284 cellules de type A est calculé grâce aux formules précédentes, et est estimé à 190 cellules. Le coût pour les cellules de types BC s'élève lui à 884 cellules de base. Le surcoût total est donc de 1074 cellules, ce qui représente 132% de la surface initialement occupée. On peut donc router complètement sur le composant une version du filtre protégée contre les SEUs avec un taux d'occupation final de 82%.

5.5. Conclusion et perspectives

Il est possible d'envisager de nouvelles approches pour le durcissement de circuits implantés sur des FPGA à mémoire SRAM commerciaux, sans modification de leur structure, mais en exploitant au mieux les ressources initialement inutilisées par l'application. En utilisant de la redondance matérielle locale, déterminée en fonction des niveaux de criticité fournis par les outils présentés dans les chapitres précédents, il est possible d'augmenter sensiblement la robustesse des applications pour un coût matériel réel nul (c'est à dire. en ne rajoutant pas de contraintes sur le taux d'occupation du FPGA par l'application originale) ou négligeable par rapport aux approches classiques (si le taux d'occupation est déjà élevé). Il s'agit toutefois d'un compromis puisque l'objectif n'est pas d'atteindre le même niveau de robustesse qu'une implantation TMR, mais plutôt d'améliorer la robustesse à moindre coût et en implantant des fonctions plus complexes sur un FPGA donné.

Conclusion de l’étude et perspectives

Les FPGAs SRAM connaissent un engouement certain auprès des concepteurs, notamment grâce à leurs possibilités de reconfiguration en ligne et/ou partielle et leur faible coût par rapport aux autres technologies configurables ou par rapport aux ASICs. Leur utilisation dans des domaines demandant un haut niveau de fiabilité, par exemple pour des applications destinées à l'aérospatial ou au secteur de la défense, implique cependant de nouvelles contraintes pour ce type de circuits. Ils doivent être capables de fonctionner normalement dans des environnements hostiles, comme par exemple dans des radiations ioniques dans l'espace ou en subissant un bombardement de particules du type protons/neutrons dans l'atmosphère.

Les effets singuliers induits par les interactions de ces particules avec les circuits constituent la principale menace. Le plus fréquent est l'effet singulier d'inversion (Single Event Upset en anglais, SEU) qui consiste en l'inversion d'un point mémoire embarqué. Les SEUs peuvent affecter à la fois les éléments mémoires utilisateurs embarqués (blocs de mémoire RAM embarqués et bascules utilisateurs) comme la mémoire de configuration SRAM du FPGA. Les fautes dans la mémoire de configuration peuvent endommager de manière définitive la définition de l'application implantée par le circuit, induisant une forte probabilité de disfonctionnement jusqu'au prochain rafraîchissement de la mémoire de configuration.

Afin de pouvoir protéger efficacement un circuit implanté sur FPGA SRAM contre ces perturbations, il est fondamental de bien comprendre le type de fautes pouvant survenir et leurs conséquences au niveau architectural et donc applicatif. Sur la base de la modélisation de fautes obtenue, il est ensuite possible de proposer des techniques de protection adaptées, cherchant à atteindre de bons compromis entre surcoûts d'implantation et couverture de fautes (donc moins coûteuses que l'approche TMR classiquement proposée).

A cet effet, il était important de rappeler les caractéristiques générales des réseaux programmables d'une part, et de ceux configurés par mémoire SRAM d'autre part; avant de s'intéresser aux plateformes cibles de l'étude, le Virtex II de chez Xilinx, et l'AT40K de chez Atmel. Dans les deux cas, nous avons rappelé les détails utiles à la compréhension de nos travaux, que ce soit au niveau architectural, ou du point de vue de leur mémoire de configuration.

Afin de situer l'étude d'un point de vue technique, nous avons ensuite exposé l'état actuel des recherches concernant l'analyse de fautes sur FPGA SRAM et leurs protections contre le problème des SEUs. Nous avons abordé différentes techniques d'analyse, des tests réels aux expériences d'injections de fautes laser, ainsi que différentes techniques de protection utilisant un ou plusieurs types de redondance (matérielle, d'information, temporelle). On aura remarqué que la majeure partie des

techniques de protection se base encore sur l'architecture TMR et ses déclinaisons, entrainant toujours un surcoût matériel important.

Enfin, nous avons décrit les trois principales contributions à l'état de l'art apportées par cette thèse. Tout d'abord nous avons procédé à l'analyse complète de la structure de configuration du FPGA Virtex II ainsi que la mise en place de différentes méthodologies d'analyse de l'effet des fautes de configuration pour ce circuit. L'étude des architectures et des structures de configuration respectives des FPGAs Xilinx et Atmel a permis de déterminer les conditions architecturales qui provoquent la génération de SEFIs à partir de SEUs sur ces technologies. Ainsi, nous avons pu implémenter des algorithmes de criticité aux fautes des bits de configuration en fonction de la configuration originale pour les deux plateformes utilisées. Les résultats ont été présentés et comparés, et leur validation a été effectué en comparant la classification obtenue avec des résultats de tests dynamiques réalisés par injection de fautes laser ou par irradiation proton. Enfin, une technique innovante de conception durcie des applications a été proposée pour le FPGA Atmel AT40K, basée sur un durcissement sélectif des zones mémoires les plus critiques.

Ces travaux démontrent le potentiel des méthodologies d'analyse statique de la configuration pour l'augmentation de robustesse des applications implantées sur des FPGAs à mémoire SRAM, à la fois par les possibilités de diagnostic offertes au concepteur, et par les possibilités d'incorporer cette technique dans un flot de durcissement sélectif. Ils mettent également en lumière l'importance de la structure de la mémoire de configuration dans le processus de génération de SEFIs à partir de SEUs. En effet, l'étude des modifications architecturales engendrées sur les interconnexions du FPGA Xilinx d'une part, et la comparaison des résultats de criticité normalisés sur les plateformes AT40K et Virtex II d'autre part, tend à montrer que l'augmentation de complexité de la structure de configuration permet de limiter les problèmes liés aux SEUs. Comme représenté chapitre 3, les points d'interconnexion du Virtex II ont une probabilité non négligeable de masquer les SEUs parce qu'ils utilisent pour la plupart d'entre eux une combinaison de bits pour déterminer leur état. De la même manière, la comparaison des résultats de criticité réalisée sur la série ISCAS indique que le nombre de bits de configuration dangereux est proportionnellement plus grand sur la technologie ATMEL, alors que ce FPGA présente un système de configuration plus simple que le Virtex II. Il parait dès lors utile de s'intéresser à l'avenir à cette problématique, par exemple en étudiant les effets de l'insertion de redondance d'information dans la matrice de configuration, de manière systématique ou alors partiellement sur des ressources identifiées comme structurellement sensibles, afin de limiter la génération de SEFIs sans nécessairement augmenter le coût matériel des applications. Il conviendra pour se faire d'obtenir un compromis entre la capacité totale de la mémoire de configuration (définissant le taux d'inversion de bits), le taux de couverture de fautes obtenu, et la taille globale de la puce.

Bibliographie

[ALA92] Sami A. Al-Arian, Mehmet B. Gumusel, "Hardware partition in time redundancy technique for fault tolerance", Proceedings of IEEE Southeastcon'92.

[ALD05] M. Alderighi, A. Candelori, F.Casini, S. D'Angelo, M.Mancini, A. Paccagnella, S. Pastore, and G. R. Sechi, "SEU Sensitivity of Virtex Configuration Logic", IEEE Trans. Nucl. Sci., vol. 52, no. 6, pp. 2462-7, 2005.

[ANT01] L. Antoni, R. Leveugle, B. Fehér, "Using Run-Time Reconfiguration for Fault Injection Applications", IEEE Instrumentation and Measurement Technology Conference (IMTC'01), vol 3, pp. 1773-1777, Mai 2001.

[ANT03] L. Antoni, R. Leveugle, B. Fehér, "Using Run-Time Reconfiguration for Fault Injection Applications", IEEE Trans. on Instrumentation and Measurement, vol 52, issue 5, pp. 1468-1473, Octobre 2003.

[ATM03] "Military Reprogrammable FPGAs with FreeRAM, AT40KAL Preliminary", ATMEL, 2003.

[BAR97] J. Barth, "Radiation Environments", IEEE NSREC Short Course, Session I, July 21, 1997.

[BAT09] N. Battezzati, F. Decuzzi, M. Violante, "Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs", 15th IEEE International On-Line Testing Symposium, IOLTS 2009.

[BER04] P. Bernardi, M. Sonza Reorda, L. Sterpone, M. Violante, "On the evaluation of SEU sensitiveness in SRAM-based FPGAs", Proceedings of the 10th IEEE international On-Line testing Symposium (IOLTS'04).

[BOC07] A. Bocquillon, G. Foucard, F. Miller, N. Buard, R. Leveugle, C. Daniel, S. Rakers, T. Carriere, V. Pouget, R. Velazco, "Highlights of laser testing capabilities regarding the understanding of SEE in SRAM based FPGAs", European Radiation Effects on Components and Systems Conference Proceedings, 2007.

[BOC09] Alexandre Bocquillon, « Evaluation de la sensibilité des FPGAs SRAM-based face aux erreurs induites par les radiations naturelles », thèse effectuée au laboratoire TIMA, 2009.

[BOL07] Cristiana Bolchini, Antonio Miele, Marco D. Santambrogio, "TMR and Partial Reconfiguration to mitigate SEU faults in FPGAs", 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'07).

[BON07] Sandro Bonacini, "Développement de circuits logiques programmables résistants aux aléas logiques en technologie CMOS submicormétrique", thèse effectuée au sein du groupe de microélectronique du Laboratoire Européen pour la Recherche Nnucléaire (CERN), 2007.

[CAL96] T. Calin, M. Nicolaidis, R. Velazco, "Upset Hardened Memory Design for SubMicron CMOS Technology", IEEE Transactions on Nuclear Science, vol. 43, n°6, 1996.

[CAN09] Gaëtan Canivet, "Analyse des effets d'attaques par fautes et conception sécurisée sur plateforme re-configurable", thèse effectuée au laboratoire TIMA, 2009.

[CAS06] Maico Cassel, Fernanda Lima Kastensmidt, “Evaluating One Hot Encoding Finite State Machines for SEU Reliability in SRAM-based FPGAs”, Proceedings of the 12th IEEE International On-Line testing Symposium (IOLTS’06).

[CHE00] A. Chee, L. Braby, and T. Conroy, “Potential doses to passengers and crew of supersonic transports”, Health Phys., Vol. 79, pp. 547, 2000.

[CSA03] David R. Csajkowski, Manish P. Pagey, Murat Goksel, David J. Bozek, “TTMR and H-Core applied to Reconfigurable FPGAs to improve radiation performances”, Xilinx Workshop SpaceMicro v5, 2003.

[DUZ02] S. Duzellier, S. Bourdarie, R. Velazco, B. Nicolescu, R. Ecoffet, “SEE in-flight data for two static 32KB memories on high earth orbit”, Radiation Effects Data Workshop, pp. 1-6, 2002.

[DUZ97] S. Duzellier, D. Falgubre, R. Ecoffet, I. Tsourilo, “EXEQ II and III: On-board experiments for the study of single events”, 4th Radiation and Its Effects on Components and Systems (RADECS’97), Cannes, France, pp. 504-509, 1997.

[FAL94] D. Falguere, S. Duzellier, R. Ecoffet, "SEE In-Flight Measurement on the MIR Orbital Station", IEEE Transaction on Nuclear Science, Vol. 41, 1994.

[FIL05] Caio Kinzel Filho, Fernanda Lima Kastensmidt, Luigi Carro, “Improving Reliability of SRAM-based FPGAs by Inserting Redundant Routing”, 8th European conference on Radiation and its effects on Components and Systems (RADECS’05), 2005.

[FOU10] Gilles Foucard, ‘Taux d’erreurs dues aux radiations pour des applications implémentées dans des FPGAs à base de mémoire SRAM : prédictions versus mesures » ; thèse effectuée au laboratoire TIMA, 2010.

[GER07] Manuel G. Gericota, Luis F. Lecos, Gustavo R; Alves, José M. Ferreira, “On-Line Self-Healing of Circuits Implemented on Reconfigurable FPGAs”, 13th IEEE International On-Line Testing Symposium (IOLTS’07).

[GOL00] P. Goldhagen, “Overview of aircraft radiation exposure and recent ER-2 measurements”, Health Phys., vol. 79, p. 526, 2000.

[GUI08] Y.Guillemenet, I. Hassoune, L. Torres, G. Sassatelli, "FPGA non-volatiles à base de cellules mémoires magnétiques à écriture assistée thermiquement", Journée Nationale du Réseau Doctoral en Microélectronique (JNRDM), May 2008

[KAF06] Leoš Kafka, Ondřej Novák, “FPGA-based Fault Simulator”, 9th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS’06), Prague, République Tchèque, pp. 218-219, Avril 2006.

[LES05] A. Lesea, S. Drimer, J.J. Fabula, C. Carmichael, P. Alfke, “The rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs”, IEEE Trans. Device Mater. Rel., 5(3), Sept. 2005.

[LIM03] Fernanda Lima, Luigi Carro, Ricardo Reis, “Designing Fault Tolerant Systems on SRAM-based FPGAs”, Proceedings of Design Automation Conference, 2003.

[LIM05] F. Lima Kastensmidt, L. Sterpone, L. Carro, M. Sonza Reorda, "On the optimal design of Triple Modular redundancy for SRAM-based FPGAs", Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'05).

[MAI09] Vincent Maingot, "Conception sécurisée contre les attaques par fautes et par canaux caches", thèse effectuée au laboratoire TIMA, 2009.

[MOR05] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs", IEEE Trans. Nucl. Sci., vol52, no. 6, pp. 2438-45, 2005.

[NOR01] E. Normand, "Correlation of inflight neutron dosimeter and SEU measurements with atmospheric neutron model", IEEE Transaction on Nuclear Science, Vol. 48, pp. 1996-2003, 2001.

[POS08] J. Postel-Pellerin, "Fiabilité des Mémoires Non-Volatiles de type Flash en architectures NOR et NAND", Thèse de doctorat, Décembre 2008

[POU01] V.Pouget, H. Lapuyade, P. Fouillat, D. Lewis, S. Buchner, "Theorical Investigation o an Equivalent Laser LET", Microelectronics Reliability, Vol. 41, pp. 1513-1518, 2001.

[SHI08] Yuriy Shiyanovskii, Francis Wolff, Chris Papachristou, "SRAM cell design protected from SEU upsets", 14th IEEE International On-Line Testing Symposium (IOLTS'08), 2008.

[SOH00] J. H. Sohn, "The effects of single event upsets in avionics systems", M.S. thesis, Iowa State Univ., May 2000.

[STE06] Luca Sterpone, Massimo Violante, "A New reliability-Oriented Place and Route Algorithm for SRAM-Based FPGAs". IEEE Transactions on computers, vol. 55, #6, p732 – 745, 2006.

[TOM04] J.N. Tombs, F. Muňoz, V. Baena, A. Torralba, L.G. Franquelo, A. Fernández-León, F. Tortosa-López, D. Gonzáles-Gutiérrez, "A Hardware Approach for SEU Immunity using Xilinx FPGA's", XIX Conf. on Design of Circuits and Integrated System (DCIS'04), Bordeaux, France, Novembre 2004.

[XIL00] "Correcting Single-Event Upsets Through Virtex partial Configuration", Xilinx Application notes XAPP216, 2000.

[XIL05a] "Virtex II Platforms FPGAs: Complete Data Sheet", Xilinx, 2005.

[XIL05b] ""¨Virtex–II Platform FPGA User Guide", Xilinx, 2005.

[ZHA07] W. Zhao, E. Belhaire, C. Chappert, "Circuit intégré non-volatile et reconfigurable basé sur la mémoire magnétique", Colloque du GDR SOC-SIP (System-On-Chip & System-In-Package), Juin 2007

Publications de l'auteur

Revues internationales :

- V. Maingot, J. B. Ferron, R. Leveugle, V. Pouget, A. Douin, "Configuration errors analysis in SRAM-based FPGAs: software tool and practical results", Microelectronics Reliability, Elsevier, vol. 47, no. 9-11, September-November 2007, pp. 1836-1840

Conférences internationales avec actes :

- V. Pouget, A. Douin, D. Lewis, P. Fouillat, G. Foucard, P. Peronnard, V. Maingot, J. B. Ferron, L. Anghel, R. Leveugle, R. Velazco "Tools and methodology development for pulsed laser fault injection in SRAM-based FPGAs" 8th Latin-American Test Workshop (LATW), March 12-14, 2007, pp. 167-172, IMS, Bordeaux, France
- G. Canivet, J.Clédière, J.B. Ferron, F. Valette, M. Renaudin, R. Leveugle, "Detailed analyses of single laser shot effects in the configuration of a Virtex-II FPGA", 14th IEEE International On-Line Testing Symposium (IOLTS), Rhodes, Greece, July 7–9, 2008.
- G. Foucard, P. Peronnard, R. Velazco, J.B. Ferron, A. Doin, V. Pouget, A. Bocquillon, F. Miller ,G. Berger , F. Charlier , F. Boldrin, "Methodologies and Tools for the Evaluation of the Sensitivity to Radiation of SRAM-based FPGAs", 23th Conference on Design of Circuits and Integrated Systems (DCIS), november 2008
- JB Ferron, L Anghel, R Leveugle, F Miller, A Bocquillon, G Mantelet, "A Methodology and Tool for Predictive Analysis of Configuration Bit Criticality in SRAM-based FPGAs: Experimental Results", 3rd International Conference on Signals, Circuits, and Systems (SCS'09), Djerba, Tunisia, November 6-8, 2009.
- J.B. Ferron, L. Anghel, R. Leveugle, "Analysis of configuration bit criticality in designs implemented with SRAM-based FPGAs", IEEE Symposium on Industrial Electronics & Applications (ISIEA), Langkawi, Malaysia, September 25-28, 2011, pp. 83-88.

Conférences internationales sans actes :

- JB Ferron, R Leveugle, "Predictive Analysis of Configuration Bit Criticality in SRAM-based FPGAs", Methodology, results, and tools", présentation à RADFAC2009.
- JB Ferron, L. Anghel, R. Leveugle, "Criticality of configuration Bits in SRAM_based FPGAs: Predictive analysis and Experimental Results", présentation à "3rd IEEE Workshop on Design for Reliability and Variability", DRVW2011.

Manifestations nationales avec actes :

- JB Ferron, L Anghel, R Leveugle, "Predictive Analysis of Configuration Bit Criticality in SRAM-based FPGAs", Colloque National du GDR SoC-SiP, 2009
- JB Ferron, "Proposition d'un flot pour la fiabilisation des FPGAs SRAM contre les fautes dans la mémoire de configuration", JNRDM 2010.

Manifestations sans actes :

- V. Maingot, J.B. Ferron, G. Canivet, R. Leveugle, "Fault Attacks on SRAM-based FPGAs: Analysis of Laser-induced Faults in a Virtex II", USE-IT Workshop, 2007

Glossaire

Activation: Une faute est “active” quand elle affecte l’état du composant

Arbre d’horloge: Chemin de distribution de l’horloge dans le FPGA

ASIC: Application Specific Integrated Circuit

Bascule D : Flip flop en anglais. Elément mémoire sensible à un front (généralement montant) de l’horloge. L’entrée est recopiée sur la sortie au moment du front de l’horloge.

Bitstream: Fichier de configuration d'un FPGA.

BRAM : Bloc de RAM dans le FPGA

BRAMI : Bloc d’interconnexions des BRAMs

Buffer à 3 états: Eléments fonctionnels permettant d’implémenter une architecture TMR

CLB: Configurable Logic Bloc, tuile contenant les éléments logiques et séquentiels élémentaires dans le FPGA Virtex II.

CLBIO: Ensemble des bits de configuration des IOBs latéraux

CUT: circuit sous test

DCM: Digital Clock Manager, tuile du FPGA Virtex II gérant la transmission du signal d’horloge aux autres tuiles fonctionnelles.

Dependability: anglais pour surêté

Device : Composant

DUT : Device under test

Erreur: Une erreur est une faute qui a été activée et a donc modifiée l’état du composant.

Faute : Une faute est une valeur erronée dans un composant qui pourrer causer un erreur.

Fiabilité : Un système est fiable quand il peut assurer une continuité de fonctionnement.

FPGA: Field Programmable Gate Array, circuits logiques programmables

Frame: Colonne d’un bit de large dans la matrice de configuration

GCLK: Global Clock.

Golden : Read back après un « run » d’une application sans injection de fautes dans la mémoire de configuration

GRM: General Routing Matrix

Latch: Bascule sensible aux niveaux plutôt qu’aux fronts

LUT: Look Up Table

Matrice de configuration: Mémoire SRAM dont le rôle est de stocker la configuration du FPGA

Matrice d’interconnexion (switch matrix en anglais): Matrice qui s’occupe de la communication du CLB avec l’extérieur

MBU (Multiple Bit Upsets): Un effet singulier qui cause la transition de plusieurs points mémoires au même instant.

Motif d’erreur (error pattern):

Multiplicité spatiale: Nombre maximal de bits commutés pendant un instant d'injection

Multiplicité temporelle : Nombre maximal d'instants d'injection pour une faute et donc pour une expérience

IOB: Input Output Blocs ce sont eux qui permettent de faire entrer ou sortir des données de la puce

IOI : Bloc d'interconnexions des IOB

MUX: Multiplexeur

Paquet: Section d'un bitstream formatée pour être chargée sur le FPGA.

PIP: Programmable Interconnect Points

RAM: Random Access Memory : une mémoire volatile qui peut être accedée en lecture/écriture à n'importe quelle adresse.

Reconfiguration dynamique: Le fait de pouvoir reconfigurer le FPGA alors qu'une application est en cours

Reliability: anglais pour fiabilité.

Scrubbing: rafraichissement périodique d'une mémoire

SEE: Single Event Effect, événement singulier, typiquement causé par un impact de particule

SET: Single Event Transient, un SEE qui cause l'inversion d'un bit dans un calcul combinatoire

SEU: Single Event Upset, un SEE qui cause l'inversion d'un point mémoire.

SEFI: Single Event Functional Interrupt

SelectRAM: Mémoire RAM utilisée par Xilinx dans ces devices Virtex II. Ecriture synchrone et lecture asynchrone.

Slice: Module fonctionnel d'un CLB contenant les éléments séquentiels et logiques.

SoC: System on Chip, Système sur puce

Soft Error: Erreurs modifiant les données contenues dans le FPGA, par opposition aux « hard error » qui modifient la structure du circuit

SRAM: Static Random Access Memory

TMR: Triple Modular Redundancy, méthode de protection matérielle des circuits par dupplication des éléments logiques et votes sur les sorties.

Tuile: Motif élémentaire du FPGA

Wafer: Plaque de silicium sur laquelle sont gravées plusieurs puces

Table des acronymes

BRAM: Block RAM

BT: Bit Line

CAN: Convertisseur Analogique Numérique

CED: Concurrent Error Detection

CLB: Conbinantial Logic Block

CNA: Convertisseur Numérique Analogique

CPLD: Complex Programmable Logic Devices

DCM: Digital Clock Manager

DSP: Digital Signal Processing

DUT: Device Under Test

DWC: Duplication With Comparison

EEPROM: Electrical Erasable ¨Programmable Read Only Memory

EPROM: Erasable Programmable Read Only Memory

FAR: Frame Address Register

FDRI: Frame Data Input Register

FDRO: Frame Data Output Register

FF: Flip Flop

FPGA: Field Programmable Gate Arrays

FSM: Finite State Machine

GAL: Generic Array Logic

IOB: Input Output block

LET: Linear Energy Transfer

LUT: Look Up Table

MBU: Multiple Bit Upset

MCU: Multiple Cell Upset

PAL: Programmable Array Logic

PC ROM: Phase Change ROM

PLA: Programmable logic Array

PROM: Programmable Read Only Memory

PIP: Programmable Interconnect Point

PLB: programmable Logic Block

RAM: Random Access Memory

SEE: Single Event Effect

SEFI: Single Event Functional Interrupts

SEU: Single Event Upset

SRAM: Static Random Access Memory

RAM: Random Access Memory

RESO: Re- computing with Shift Operands

REWSO: Re-computing With Swapped Operands

TMR: Triple Modular Redundancy

TSTMR: Time Shared Triple Modular Redundancy

WL: Word Line

XTMR: Xilinx Triple Modular Redundancy

Annexes

Cette partie est consacrée à des documents techniques pouvant éclairer la lecture du manuscrit. Nous n'avons pas jugé utile de les inclure dans le texte principal cependant.

A1. Détails sur l'architecture du FPGA Virtex II et la configuration des tuiles CLB

Nous allons maintenant aborder la configuration des éléments fonctionnels logiques et séquentiels contenus dans les 4 slices de chaque CLB.

❖ Architecture et configuration d'un slice

Chaque slice contient les ressources suivantes :

- 2 générateurs de fonctions à 4 entrées, F et G
- une chaîne de retenue
- des portes logiques pour le calcul arithmétique
- 2 éléments mémoires
- des multiplexeurs

➢ Configuration des générateurs de fonctions

Les générateurs de fonctions peuvent implémenter un LUT à 4 entrées, une RAM de 16 bits, ou un registre à décalage de 16 bits. Quelque soit la configuration adoptée, les 16 bits qui contiennent la programmation des générateurs de fonctions se trouvent dans la matrice de configuration.

➢ Configuration des éléments mémoires

Ils peuvent implémenter au choix des bascules D sensibles aux fronts montants de l'horloge ou des latchs sensibles aux niveaux de l'horloge.

➢ Ressources de routage globales

Les « long lines » : Ce sont des bus qui parcourent le FPGA de haut en bas (long lines verticales) et de gauche à droite (« long lines » horizontales). Il passe un bus de ce type entre chaque tuile. Ces liaisons sont bidirectionnelles.

➢ Ressources de routage locales

Trois types de liaisons locales sont disponibles, ce sont par ordre décroissant de longueur les « hex line », les « double line » et les connections directes.

Les « double line » mettent en relation un CLB avec son CLB mitoyen et le suivant. Chaque double a 3 ports, « begin », « middle » et « end ». Le double peut être alimenté par son port « begin » et peut alimenter d'autres ressources par ses ports « middle » et « end ». Les « Doubles » sont séparés en 4 groupes de 10 fils selon leur orientation (nord, sud, est ou ouest).

Les « hex line » permettent de communiquer avec un CLB éloigné de 3 ou 6 blocs horizontalement ou verticalement, et fonctionnent de la même façon que les « doubles » (Ils possèdent eux aussi 3 ports « begin », « middle » et « end »).

Les connexions directes permettent de communiquer avec tous les voisins proches, y compris les voisins diagonaux. Ce sont les sorties du multiplexeur de sortie qui sont connectées aux matrices d'interconnexions des CLB voisins.

En plus de ces ressources, des liaisons rapides existent à l'intérieur même d'un CLB afin de pouvoir reboucler certains signaux: elles permettent principalement de relier les sorties du LUT aux entrées du slice.

A2. Logique de configuration du FPGA Virtex II

La Figure 81 représente la logique de configuration nécessaire pour le chargement ou la relecture du bitstream sur le FPGA Virtex II. Voir partie 1.3.1.5 pour de plus amples détails.

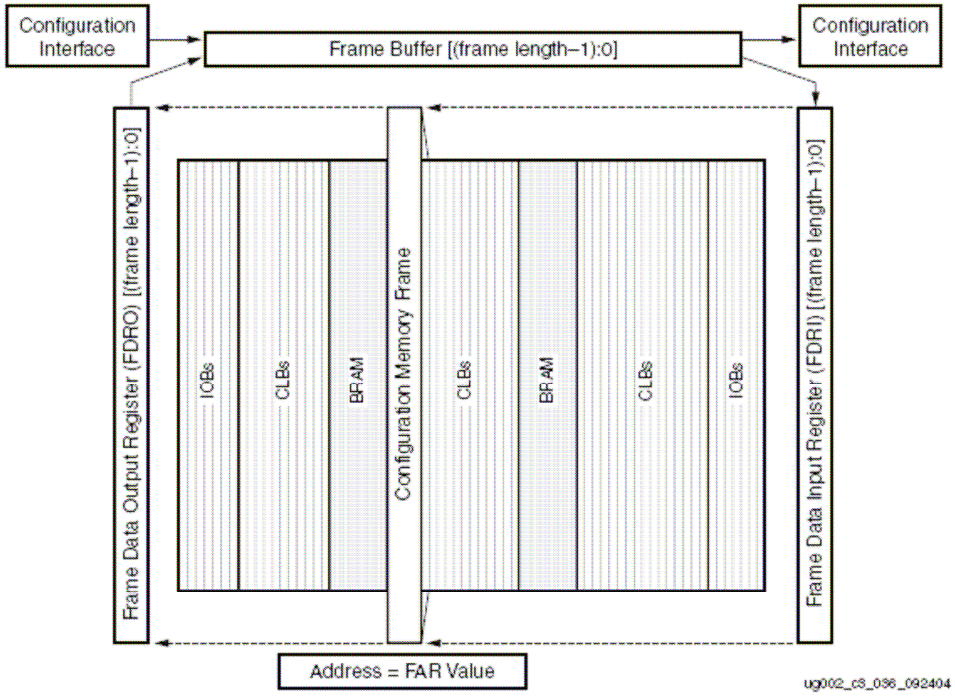

Figure 81: Logique de configuration sur le Virtex II [XIL05b].

A3. Détails sur l'architecture du circuit Atmel AT40K

Ci-dessus (Figure 82 et Figure 83) sont présentés quelques modes d'utilisation courants des cellules de base du FPGA ATMEL AT40K:

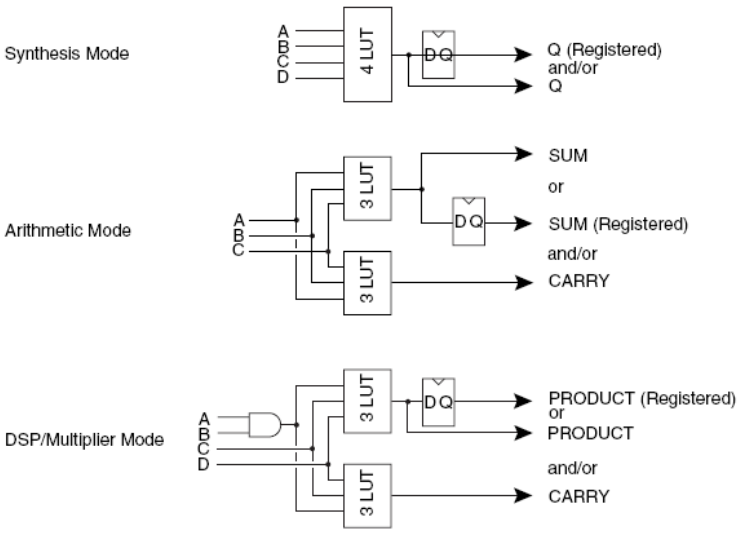

Figure 82: Modes d'utilisation courants (a) [ATM03].

- Mode synthèse: Le mode générique permettant d'implémenter n'importe quelle fonction logique de base, particulièrement appréciées des outils de synthèse logique actuelle. La sortie peut alimenter une bascule, un registre trois états, ou être rebouclée vers l'entrée de la cellule.
- Mode arithmétique: L'architecture de la cellule de l'AT40K est adaptée pour implémenter un additionneur 1 bit. On utilise trois entrées (une pour la retenue) une LUT pour l'addition, et une LUT pour la retenue.
- Mode multiplieur: En cascadant une porte ET en entrée de la version additionneur, on peut implémenter des multiplieurs, facilement cascadés grâce aux ressources de connexion directes de l'AT40K (voir paragraphe 1.2.2.3).
- Mode compteur: Le multiplexeur « feedback » alimentés par les sorties des LUT permettent d'implémenter des compteurs facilement.
- Mode multiplexeur/registre trois états: Un mode bien adapté aux applications orientée télécommunication où les données doivent être distribuées vers de nombreux chemins.

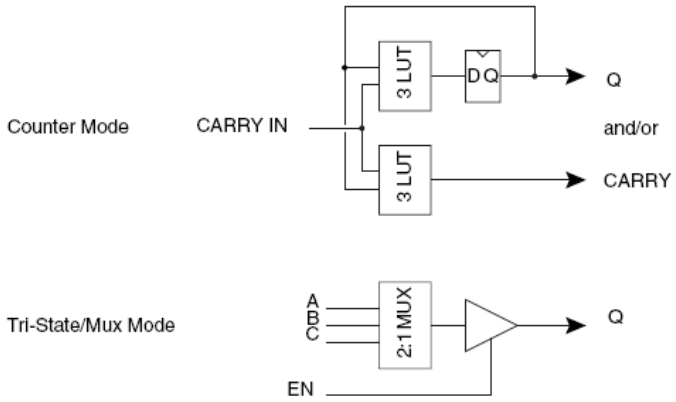

Figure 83: Modes d'utilisation courants (b) [ATM03].

A4. Implémentation de code correcteur d'erreur pour la détection de fautes par relecture de configuration

La Figure 84 représente la logique nécessaire à la mise en place d'un code correcteur d'erreur pour la détection de faute lors de relecture de configuration. Cette technique permet de protéger de manière simple un FPGA SRAM en reconfigurant les frames comportant des erreurs.

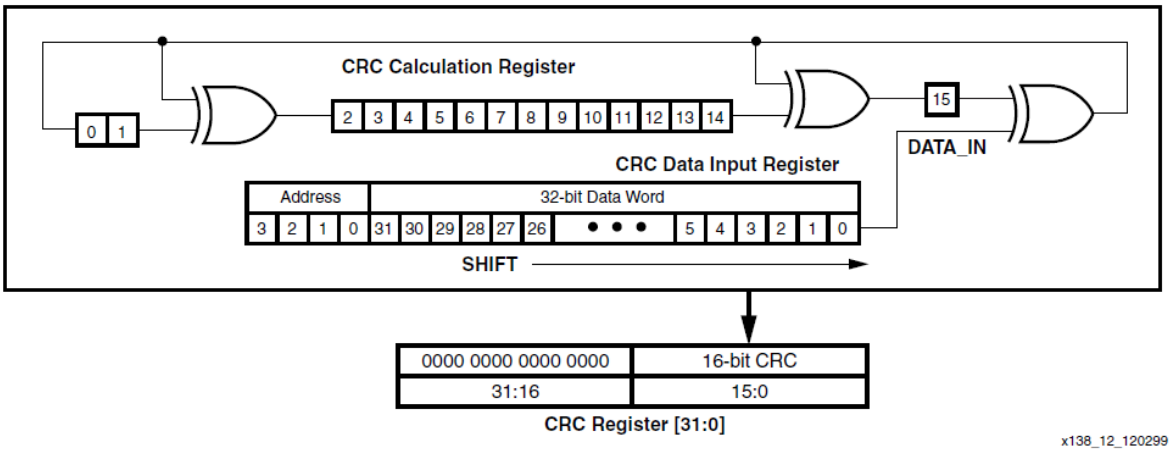

Figure 84: Implémentation série du code correcteur 16 bit XILINX [XIL00]

A5. Algorithmes de criticité ATMEL

A5.1. Algorithmes local

- if bit_type == logic then
  - if cell connected to network then
    - if configuration == default then
      - firstCriticality = suspect
    - else
      - firstCriticality = critical
  - else
    - firstCriticality = transparent
- else // bit_type = wiring
  - if initial_bit_status == used then
    - firstCriticality = critical
  - else
    - if resource configurating bits number > 1 then
      - if other_bits_status == used then
        - firstCriticality is suspect
      - else
        - firstCriticality = transparent
    - else
      - if network topology is modified then
        - firstCriticality = suspect
      - else
        - firstCriticality = transparent

A5.2. Algorithme global

- if bit firstCriticality == suspect then
  - switch (bit_type)
    - case input muxes
      - Neighboring cells outputs check
      - Buses connections check (repeaters)
    - case output muxes
      - Neighboring cells input muxes check
    - case intra muxes (FF output management)

- Type of the output cell connection used check
- case Local Via
  - Cell shortcut check
  - Local buses/cell connections check
- case Express Via
  - Repeaters check

A5.3. Algorithme global Etendu

int number_inputmux_used =0;

String MUXused_name = null;

- if bit_critichierarchic == suspect && bit_role == LUT_content then
  - if ZBUS == used then -- 4 input function
    - bit_critichierarchicext == critical
  - else
    - if XLUT_output == used && YLUT_output == used then -- 3 entrées alimentent 2 fonctions
      - number_inputmux_used = getMUXnumber();
      - MUXused_name = getMUXused();
      - switch (number_inputmux_used)
        - case 1:
          - switch (MUXused_name)
            - case W:
              - if (bit_role == LUT_content corresponding at X=0 or Y=0) then -- comme le LUT maintient alors X et Y au niveau haut, on peut dire:
                - bit_critichierarchicext == transparent; -- correspond à 75% d'un LUT
              - else
                - bit_critichierarchicext == critical;
            - case X:

- if (bit_role == LUT_content corresponding at W=0 or Y=0) then
  - bit_critichierarchicext == transparent; -- correspond à 75% d'un LUT
- else
  - bit_critichierarchicext == critical;

- case Y:
  - if (bit_role == LUT_content corresponding at W=0 or X=0) then
    - bit_critichierarchicext == transparent; -- correspond à 75% d'un LUT
  - else
    - bit_critichierarchicext == critical;

- case 2:
  - switch (MUXused_name)
    - case WX:
      - if (bit_role == LUT_content corresponding at Y=0) then
        - bit_critichierarchicext == transparent; -- correspond à 50% d'un LUT
      - else
        - bit_critichierarchicext == critical;
    - case WY:
      - if (bit_role == LUT_content corresponding at X=0) then
        - bit_critichierarchicext == transparent; -- correspond à 50% d'un LUT
      - else
        - bit_critichierarchicext == critical;

- case XY:
  - if (bit_role == LUT_content corresponding at W=0) then
    - bit_critichierarchicext == transparent; -- correspond à 50% d'un LUT
  - else
    - bit_critichierarchicext == critical;
- case 3:
  - bit_critichierarchicext == critical;

- else -- un seul LUT utilisé, fonction 3 entrées
  - if (XLUT == used) then
    - if (bit_index == 7) then -- bit configurant content de XLUT!
      - bit_critichierarchicext == critical;
    - else
      - bit_critichierarchicext == transparent;
  - else
    - if (bit_index == 6) then -- bit configurant content de YLUT!
      - bit_critichierarchicext == critical;
    - else
      - bit_critichierarchicext == transparent;

TITRE

Analyse statique de l'effet des erreurs de configuration dans des FPGA configurés par SRAM et amélioration de robustesse

RESUME

Cette thèse s'intéresse en premier lieu à l'analyse des effets fonctionnels des erreurs dans la configuration de FPGAs à base de SRAM. Ces erreurs peuvent provenir de perturbations naturelles (rayonnements, particules) ou d'attaques volontaires, par exemple avec un laser. La famille Virtex II de Xilinx est utilisée comme premier cas pratique d'expérimentation, puis une comparaison est réalisée avec la famille AT40K de chez ATMEL. Ceci a permis de mieux comprendre l'impact réel de différentes sources de perturbations, et les motifs d'erreur devant réellement être pris en compte pour améliorer la robustesse d'un circuit implanté sur ce type de technologie. Cette étude a nécessité le développement d'outils de conception spécifiques, permettant d'automatiser les analyses. Une méthodologie innovante est proposée pour l'évaluation de la sensibilité de la mémoire de configuration aux SEUs : une classification des bits de configuration est établie en fonction des effets produits par leur inversion sur le fonctionnement normal de l'application. Ceci permet de déterminer les zones les plus critiques, autorisant le développement de stratégies de protection sélectives et à faible coût.

MOTS-CLES

FPGAs SRAM, mémoire de configuration, analyse statique de criticité, robustesse.

TITLE

Static analysis of configuration error effects in SRAM-based FPGAs and robustness improvement

ABSTRACT

This thesis deals primarily with the analysis of the functional effects of errors in the configuration of SRAM-based FPGAs. These errors can be due either to natural perturbations (radiations, particles) or to malicious attacks, for example with a laser. The Xilinx Virtex II family is used as first case study, then a comparison is made with the ATMEL AT40K family. This work allowed us a better understanding of the real impact of perturbations, and of the error patterns that need to be taken into account when improving the robustness of a circuit implemented on this type of technology. This study required the development of specific design tools to automate the analyses. An innovative methodology is proposed for the evaluation of the configuration memory sensitivity to SEUs: a classification of configuration bits is made with respect to the effects produced on the application by a single bit-flip. This enables us to identify the most critical areas, and to propose selective hardening solutions, improving the global reliability of the application at low cost.

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

ISBN 978-2-84813-186-3