UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Nanoélectronique et Nanotechnologie**

Arrêté ministériel : 7 août 2006

Présentée par

Mohamed BEN JRAD

Thèse dirigée par **Régis LEVEUGLE**

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

Robustesse par conception de circuits implantés sur FPGA SRAM et validation par injection de fautes

Thèse soutenue publiquement le **1er juillet 2013**,
devant le jury composé de :

M. Frédéric PETROT
Professeur, Université de Grenoble, Grenoble-INP, Président
M. Abbas DANDACHE
Professeur, Université de Lorraine, Rapporteur
M. Bruno ROUZEYRE
Professeur, Université de Montpellier II, Rapporteur
M. Régis LEVEUGLE
Professeur, Université de Grenoble, Grenoble-INP, Directeur de thèse
M. Antonin BOUGEROL
Responsable lot de travaux, Thales Optronique S.A. Elancourt, Examinateur

A mes parents Anouar & Saloua, ma sœur Azza et ma femme Salma
A la mémoire de mes grands parents Mohamed & Souad, Hédi & Chema

Remerciements

Ces travaux de thèse ont été réalisés au sein du laboratoire TIMA de Grenoble, dont je tiens à remercier la directrice, Dominique Borrione, pour m'y avoir accueilli, ainsi que l'ensemble du personnel administratif : Anne-Laure, Laurence, Sophie, Younes, Claire et Marie-Christine, ainsi que notre service informatique Nicolas, Ahmed et Fréd.

Je remercie également les membres de mon jury, en commençant par Frédéric Pétrot, Professeur à l'INP Grenoble et chef du groupe SLS, pour m'avoir fait l'honneur d'être le président de mon jury. J'adresse mes remerciements à Abbas Dandache, Directeur du laboratoire LICM et Professeur à l'université de Metz, et Bruno Rouzeyre, Professeur à l'Université de Montpellier 2, qui ont accepté de participer à ce jury en tant que rapporteurs. Je remercie Antonin Bougerol, responsable lot de travaux à Thalès, d'avoir apporté son expérience en tant qu'examinateur.

Enfin je tiens tout particulièrement à remercier « Le Grand Patron » Régis Leveugle, Professeur à Grenoble INP, pour m'avoir proposé ce sujet de thèse, pour son encadrement, sa patience et ses conseils tout au long de ces années et pour la confiance qu'il m'a apportée.

Je souhaite également remercier Alexandre Chagoya et Robin Rolland pour leur disponibilité.

Cette thèse n'aurait pu être possible sans le la bonne humeur de plusieurs personnes qui j'ai eu la chance de rencontrer et de côtoyer. Je remercie tout particulièrement, J.B. et Marion, Paolo et Anna, Diego et Gabriella, Pierre et Solène, Maryam, Giota, Fabien, Adrien, Vladimir, Thierry, Wassim, les deux Gilles ainsi que les frères Fall, qui ont partagé avec moi, ces quatre années au laboratoire.

Je veux également dire merci aux grands chercheurs de ce laboratoire Mihail Nicolaïdis, Raul Velazco, Nacer Zergainoh et Skandar Basrour pour tous ces débats enrichissants au cours des pauses café.

Je remercie également les autres amis d'ici et d'ailleurs avec qui j'ai passé d'agréables moments de détente et d'évasion, je pense à Souchou et Choupette, Guillaume et Hela, Amine et Dorra, Tom et Xav, Amine et Sarrour, Issam et Henda, Nizou.

Je finis par remercier les personnes qui me sont les plus chères. Un énorme merci à ma mère, mon père et ma sœur (dbouba en gros sans faire de jeu de mots), tata Fatma et tonton Fethi, Med Amine et Meriem et bien évidemment ma femme Salma, pour leur soutien, leurs encouragements et tout le reste… durant toutes ces années sans qui je n'y serais pas arrivé.

Table des matières

Liste des figures

Chapitre I : Contexte et Objectifs

Chapitre II : Présentation des FPGA utilisé

Chapitre III : Evaluation des effets des perturbations dans les mémoires de configuration des FPGA SRAM

Chapitre IV : Approche de protection pragmatique de systèmes implantés sur FPGA SRAM

Annexe A : Détails sur l'architecture Virtex

Annexe B : Détails sur le composant ICAP

Annexe C : Mise en place de la méthode de durcissement du circuit en mode ECO

Liste des tableaux

Chapitre I : Contexte et Objectifs

Chapitre II : Présentation des FPGA utilisé

Chapitre III : Evaluation des effets des perturbations dans les mémoires de configuration des FPGA SRAM

Chapitre IV: Approche de protection pragmatique de systèmes implantés sur FPGA SRAM

Annexe A : Détails sur l'architecture Virtex

Annexe B : Détails sur le composant ICAP

Annexe C : Mise en place de la méthode de durcissement du circuit en mode ECO

Introduction générale

L'évolution des technologies microélectroniques augmente la sensibilité des circuits intégrés face à leur environnement. En effet, leurs dimensions de plus en plus petites et leurs tensions d'alimentation de plus en plus basses, contribuent à rendre les nouvelles technologies microélectroniques moins résistantes face à différents parasites (rayonnements, particules ou autres perturbations), même au niveau de la mer. Ces parasites peuvent engendrer des fautes transitoires et modifier le comportement des circuits et les données qu'ils traitent. Ces fautes peuvent induire des comportements inacceptables non seulement dans les applications critiques mais aussi dans les applications grand public.

Dans le cas d'un circuit implanté dans un réseau programmable (FPGA) configuré par une mémoire SRAM, de telles modifications peuvent survenir soit dans les données manipulées par l'application, soit dans la mémoire de configuration. Dans les deux cas, les données erronées peuvent perturber l'application exécutée. Dans le second cas, l'application peut aussi être modifiée (changement de fonction). Or la confiance en de tels systèmes dépend de leur capacité à réagir de façon sûre lorsque des fautes surviennent pendant l'exécution.

C'est la raison pour laquelle la conception de systèmes destinés à être utilisés dans des conditions environnementales difficiles exige une maîtrise et une prédiction précise des effets réels que peuvent avoir les fautes. Si une analyse de robustesse est réalisée tôt dans le flot de conception, avant que le circuit ne soit fabriqué, l'amélioration des caractéristiques du circuit pourra être faite à moindre coût soit en modifiant la description originale (ajout de mécanismes architecturaux de protection par exemple) soit en définissant des contraintes à plus bas niveau (choix d'une technologie spécifique ou de cellules de bibliothèque plus robustes).

Cette analyse de sûreté peut être menée grâce à des techniques d'injection de fautes. Le principe est de comparer le comportement nominal du circuit (sans injection de fautes) avec son comportement en présence de fautes, survenant lors de l'exécution d'une application. Des campagnes d'injection de fautes peuvent être réalisées suivant plusieurs approches, en particulier la simulation ou l'émulation pour des approches de haut niveau.

Le chapitre 1 de ce manuscrit sera consacré à une présentation des notions générales de la sûreté de fonctionnement. Dans un premier temps, nous verrons la terminologie nécessaire à

la compréhension du sujet. L'étude des phénomènes mis en jeux dans les circuits intégrés permet ensuite d'établir une liste des modèles de fautes qui peuvent être utilisés. Nous nous focaliserons sur les phénomènes liés aux applications terrestres, notamment parce que leur importance est grandissante dans le domaine de l'analyse de sûreté. Il fera aussi l'objet de l'état de l'art des techniques d'injection de fautes. Celui-ci nous permettra de faire le point sur les avantages et les inconvénients des techniques existantes et d'affiner nos choix en vue du développement d'un environnement répondant à nos objectifs. Suite à cela une présentation des techniques de détection et de protection sera développée.

Le chapitre 2 résume les principales caractéristiques des FPGA qui seront utilisés dans le cadre de notre travail. L'objectif de ce chapitre est de préciser la terminologie définie par les différents fabricants et de souligner les différences architecturales entre différentes familles de composants. Les mécanismes de protection prévus par les constructeurs seront présentés également dans ce chapitre.

Le chapitre 3 a pour premier but de présenter les différents environnements d'analyse que nous avons développés. Nous présentons comment tirer profit d'un processeur dans un environnement FPGA afin d'obtenir un environnement flexible et performant, puis ce chapitre présentera notamment une méthode originale permettant l'exploitation par abstraction de motifs d'erreurs réalistes dans la configuration de FPGAs SRAM. Cette méthode a été mise en œuvre pour la famille Virtex II, dans notre environnement d'injection de fautes conçu pour ces FPGAs. Les performances de plusieurs environnements et plateformes sont détaillées et comparées du point de vue de l'injection de fautes. Enfin, nous résumons quelques résultats obtenus sur des études de cas complexes, basées sur les processeurs Leon2 et Leon3. Outre l'illustration des capacités de nos plateformes d'injection, ces études de cas mettent en lumière que des approches fonctionnelles de protection peuvent être efficaces pour une implantation ASIC, mais le sont beaucoup moins pour une implantation sur FPGA SRAM dans le cas d'erreurs multiples dans la configuration.

Le chapitre 4 porte enfin sur la proposition et l'implantation d'une approche de protection pragmatique sur une sélection d'architectures de type FPGA à mémoire SRAM. Une évaluation du surcoût en fonction du circuit et de la cible sera présentée dans ce chapitre, suivie d'une validation du taux de couverture par injection de fautes.

Chapitre I

Contexte et Objectifs

L'objectif de ce chapitre est de présenter les notions générales de la sûreté de fonctionnement. Dans un premier temps nous verrons les sources de défaillances des circuits électroniques, leurs modèles et leur impact sur les FPGA. Ensuite suivra une présentation de l'état de l'art des méthodes d'injection de fautes. Nous allons clore ce chapitre par la présentation des différentes méthodes de protection existantes.

I.1. Sources de défaillance des circuits électroniques

La sûreté de fonctionnement (*dependability*) d'un système est définie comme la « propriété qui permet de placer une confiance justifiée dans le service qu'il délivre » **[Lapr.-04]**. En d'autres termes c'est la propriété, lorsque le système est mis en œuvre, de respecter sa fonctionnalité et de rendre un service conformément à une référence prédéterminée malgré la possibilité d'occurrence de certaines perturbations. Par opposition, une défaillance du système est une déviation de la spécification du service rendu par le système **[Pies.-06]**. La propriété de sûreté de fonctionnement est associée à plusieurs attributs, tels que la fiabilité (*reliability*), la disponibilité (*availability*), l'innocuité (*safety*) ou encore la confidentialité (*security*).

Même si le terme « fonctionnement » est parfois omis, on parle alors simplement de « sûreté», il faut le garder en mémoire car il a une importance capitale. En effet le niveau de sûreté d'un circuit dépend de l'environnement dans lequel il se trouve (exposition aux perturbations) mais également de la tâche qu'il doit effectuer et de sa charge de travail. Suivant sa charge de travail un circuit peut se comporter différemment en présence d'une ou plusieurs fautes. Le niveau de sûreté d'un circuit est évalué par rapport au service rendu. La notion de service rendu est donc déterminante. Si le service est tel que prédéfini en terme de fonctionnalité et de performance on dit qu'il est correct.

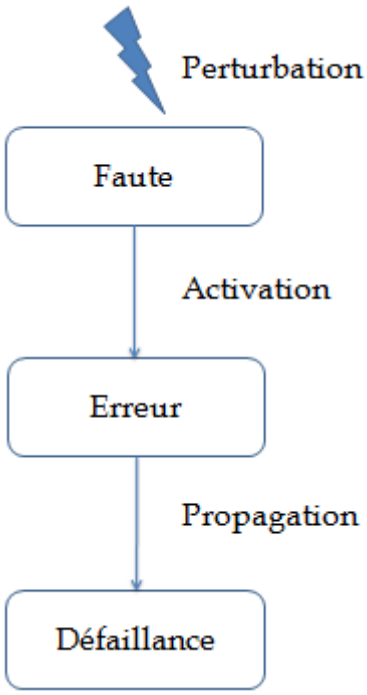


Figure 1-1 : Schéma Faute-Erreur-Défaillance

Une défaillance (*failure*) survient lorsque « le service délivré dévie du service correct, soit parce qu'il n'est plus conforme à la spécification, soit parce que la spécification ne décrit pas de manière adéquate la fonction du système » **[Lapr.-04]**. Une erreur, état anormal du système, peut entraîner une défaillance si elle se propage et devient observable de l'extérieur. L'origine d'une erreur est une faute dans le système, telle une valeur logique incorrecte. Cet enchaînement est très souvent illustré par un schéma comme celui de la figure 1-1. Si une faute survient dans une partie utilisée du circuit elle sera activée et engendrera, sous certaines conditions, une erreur de fonctionnement. Dans le cas contraire, si la faute se trouve dans une partie non utilisée du circuit, elle est dite latente.

On distingue deux grandes catégories d'erreurs : les erreurs permanentes et transitoires.

- Une erreur permanente (*hard-error*) correspond à la destruction partielle du circuit.
- Une erreur transitoire (*soft-error*) cause un état erroné temporaire et n'engendre pas la destruction du circuit. En effet, ce dernier reprend son fonctionnement normal après la disparition de cette erreur. L'erreur transitoire est donc un état temporaire du système se caractérisant par une modification non définitive de données. Par exemple, une erreur transitoire dans un registre disparaît après réécriture de ce registre. Cependant, avant son masquage, l'erreur peut s'être propagée en créant d'autres erreurs dans le système. Notre étude s'est portée uniquement sur ce type d'erreurs pour plusieurs raisons, dont notamment la probabilité d'occurrence plus grande pour une erreur transitoire qu'une erreur permanente.

Cependant, il arrive qu'une erreur soit liée à un fonctionnement dans des conditions particulières. Dans ce cas, elle se manifeste d'une manière répétée mais non systématique. Cette catégorie d'erreurs est dite intermittente et elle sera assimilée à une erreur transitoire dans la suite du document.

Les fautes sont provoquées par des phénomènes pouvant être d'origine naturelle ou intentionnelle. Ces phénomènes peuvent être singuliers et extérieurs au circuit (impacts de particules) mais aussi internes, de type absence d'intégrité du signal (bruit, crosstalks …).

Parmi les fautes d'origines naturelles, nous pouvons citer les rayonnements cosmiques. Ces derniers sont une menace touchant particulièrement les applications spatiales et aéronautiques, qui vu la nature des environnements dans lesquels elles évoluent, sont très sensibles aux rayonnements cosmiques. Cependant, à cause des évolutions technologiques, impliquant une croissance de la vitesse et du nombre de transistors ainsi qu'une décroissance de la tension d'alimentation, même les applications utilisées au niveau de la mer ne sont plus épargnées par ce genre de menace.

Outre les rayonnements cosmiques, il existe dans l'environnement d'un circuit d'autres sources de radiations qui peuvent produire des fautes, à savoir les interactions Neutron-Boron10 (^{10}B)**[Baum.-01],** ou encore les impuretés radioactives provenant essentiellement des matériaux qui composent le boîtier du circuit, dont les particules alpha (α).

Les phénomènes internes de type perte d'intégrité du signal augmentent avec la réduction des dimensions. La perte de l'intégrité du signal se manifeste par la détérioration de la qualité des signaux engendrés dans un circuit et leur capacité à bien représenter les valeurs logiques souhaitées. L'altération du signal peut être engendrée par des interférences dues aux autres éléments du circuit ou du système, mais aussi provenant de rayonnements électromagnétiques. Parmi les causes de perte de l'intégrité du signal il y a notamment la diaphonie capacitive et inductive **[Angh.-00]**, la pollution des alimentations et les variations de l'horloge.

Alors que la sécurité est devenue une propriété de plus en plus recherchée pour de nombreuses applications, même grand public, utilisant des données sensées rester secrètes comme une clé de chiffrement **[BarE.-06],** il ne faut plus négliger les fautes d'origine intentionnelle. Il devient indispensable de considérer les attaques utilisées pour modifier le comportement d'un système et ainsi obtenir des droits supplémentaires ou la clé de chiffrement. Ces fautes intentionnelles peuvent être réalisées par des injections optiques (flash, laser) ou encore par la modification de l'environnement du circuit ; pour ce faire plusieurs techniques existent dont notamment la variation de la température ou de la tension d'alimentation, la variation de l'horloge ou du reset…

I.2. Les modèles de fautes ou d'erreurs

Un modèle de fautes (ou plutôt d'erreurs) est une abstraction de l'état incorrect d'un circuit. Etant donné que nous souhaitons faire de l'injection de fautes pour évaluer les conséquences de

ces fautes, il est nécessaire d'avoir un ou plusieurs modèles de fautes qui représentent le plus fidèlement possible les phénomènes physiques présentés dans le paragraphe I.1. De plus les modèles de fautes doivent être applicables dans le cadre de l'approche choisie pour l'injection. Les effets singuliers des fautes sont appelés SEEs (*Single Event Effects*), et peuvent être divisés en deux classes d'effets:

- Les dégradations permanentes et irréversibles ou erreurs matérielles (*Single Event Latch-up* (SEL), *Single Event Gate Rupture* (SEGR), *Single Event Snapback* (SES)). Ce type ne fera pas l'objet de notre étude.
- Les effets singuliers réversibles dont les transitoires dans la logique (SET) ou les erreurs dans les éléments de mémorisation (SEU, MBU, MCU) :
    - Le SEU, *Single Event Upset*, correspond au changement d'état logique d'un point mémoire unique suite au passage d'une particule unique. Ce changement accidentel de niveau logique dans une mémoire est réversible (le point mémoire pourra être corrigé par le processus normal d'écriture) et ne conduit pas à la destruction du composant.
    - Le SET, *Single Event Transient*, correspond à l'apparition d'un courant parasite suite au passage d'une particule unique dans le composant. La propagation d'un SET à travers une chaîne logique jusqu'à un élément de mémorisation peut transformer sous certaines conditions un SET en SEU et donc induire une erreur logique. La propagation peut aussi induire plusieurs bits erronés.
    - Le MBU, *Multiple Bit Upset*, correspond au basculement de plusieurs points mémoires du même mot logique, provoqué par une particule unique. Les charges produites par la particule se diffusent sur plusieurs jonctions voisines. En effet, avec l'intégration, les jonctions des transistors voisins deviennent très proches, favorisant les évènements de type MBU.
    - Le MCU, *Multiple Cells Upset*, correspond à plusieurs SEUs touchant des bits n'appartenant pas au même mot mémoire **[JEDE.-07]**. C'est ce type d'erreur qui nous intéresse le plus pour le chapitre 3. En effet, nous avons recueilli des MCUs provenant de résultats réels d'injection de fautes à partir de perturbations physiques (Laser). Ces derniers seront reproduits dans différentes zones du FPGA par translation et seront appelés dans la suite du rapport des « motifs d'erreurs ».

D'autres modèles de fautes peuvent être considérés dans certaines situations de perturbations, comme le modèle des collages à 0 ou à 1 (transitoires ou permanents). Ce type de modèle ne

sera pas explicitement pris en compte dans la suite du document, mais les approches qui vont être présentées pourraient facilement être adaptées pour les prendre en compte.

I.3. Impacts des fautes sur les FPGA de type SRAM

Les circuits programmables (Programmable Logic Devices - PLD) ont été introduits dans les années 1970. Dans un premier temps,uniquement disponibles avec des tailles réduites (quelques centaines de portes), ils sont devenus plus importants avec les CPLD (Complex PLD). En 1985, Xilinx fabrique les premiers LCAs (Logic Cell Arrays) pour combler l'espace entre les ASICs et les CPLDs. L'utilisation du terme FPGA apparait avec les FPGAs à antifusibles d'Actel puis elle a été généralisée. Un FPGA est un circuit intégré constitué d'éléments logiques programmables, d'éléments mémoires et d'interconnexions programmables.

Les principaux avantages des FPGAs actuels sont un prototypage rapide, un coût de production limité pour des petites séries, la possibilité de reconfiguration dynamique ou encore l'accélération apportée pour certains calculs (par rapport a une approche logicielle). L'utilisation de FPGA est facilitée par la commercialisation de plateformes complètes de prototypage. Celles-ci intègrent non seulement un FPGA mais également un certain nombre de périphériques (mémoire, afficheurs…) notamment pour la communication avec l'ordinateur hôte (connecteurs série, ethernet…). L'ordinateur hôte, sur lequel sont exécutés un ou plusieurs modules logiciels, permet notamment à l'utilisateur de commander le FPGA, d'y télécharger ou d'en récupérer des données. Certaines plateformes de prototypage se connectent directement à la carte mère de l'ordinateur hôte (slot PCI).

Il existe plusieurs types de FPGA dont notamment les FPGA basés sur une SRAM, les FPGA à mémoire flash ou encore à EPROM ou à antifusibles. Les FPGAs à base de SRAM sont largement les plus utilisés sur le marché et ils sont moins coûteux que leurs concurrents utilisant des fusibles, une EPROM ou une mémoire Flash. Ils sont fabriqués avec les technologies CMOS les plus avancées et sont reconfigurables à volonté. Les deux principaux inconvénients des FPGAs SRAM sont qu'ils doivent être reconfigurés après chaque coupure de l'alimentation (mémoire SRAM volatile) et la sensibilité aux fautes transitoires.

Alors que ce deuxième inconvénient tend à augmenter avec les avancées technologiques, on assiste, selon les prévisions de **[ITRS-02]**, pour toutes les technologies, à une décroissance quadratique de la charge de commutation, une croissance linéaire de la vitesse, une décroissance linéaire de la tension d'alimentation mais également une croissance quadratique du nombre de transistors (illustrée dans le Tableau I-I).

Tableau I-I Evolution des paramètres technologiques [ITRS-02]

	2004	**2005**	**2007**	**2010**	**2016**
Densité de transistors - SRAM (10^9.cm^{-2})	393	504	827	1718	7208

L'augmentation de la fréquence d'horloge entraîne des variations de courant plus rapides (di/dt), ce qui augmente les bruits de commutation et altère l'intégrité du signal. D'un autre côté, pour les technologies avancées, les effets des impuretés radioactives, et donc des particules α, augmentent exponentiellement. Il faut également noter que pour les éléments à forte charge critique (logique combinatoire) le taux d'erreur est plus influencé par les neutrons que par les particules α. Alors que pour les éléments à faible charge critique (petites cellules mémoires, amplificateurs) les particules α sont prédominantes dans le calcul du taux d'erreurs transitoires.

En ce qui concerne les interactions neutrons/^{10}B, il est possible de les éviter en remplaçant le ^{10}B par un autre matériau comme par exemple le ^{11}B, dans le dopage de type P et dans la formation de couches diélectriques. Le **Erreur ! Source du renvoi introuvable.**II, tiré de **[Gasi.-01]**, montre les effets des particules alphas, des neutrons et du ^{10}B, dans le taux des erreurs transitoires pour deux technologies SRAM différentes (avec et sans ^{10}B).

Tableau I-II Importance des sources sur le taux d'erreurs transitoires en un temps donné

	SRAM 0,25µm (avec ^{10}B)	**SRAM 0,18µm (sans ^{10}B)**
Particules α	4%	18%
Neutrons à haute énergie	15%	82%
Fission du ^{10}B	81%	0%

L'utilisation des FPGA SRAM est donc délicate si des contraintes de sûreté importantes sont à satisfaire, ce qui est le cas par exemple en avionique ou pour certains équipements automobiles. La modification d'un bit de la configuration stockée en SRAM peut avoir un impact variable sur l'application, en fonction de l'implémentation du circuit sur le FPGA et en fonction du rôle du bit modifié. Pour les applications critiques, il est donc nécessaire de pouvoir identifier précisément les bits de la configuration pouvant avoir un impact fonctionnel important. Par ailleurs, toute commutation de bit dans un FPGA SRAM peut entraîner une modification non maîtrisée du circuit implanté et des conséquences désastreuses pour l'application en cours. La modification est maintenue tant que la configuration initiale n'est pas restaurée, voire dans certains cas jusqu'à un cycle complet de ré-initialisation (on parle alors de faute rémanente). Ces

commutations peuvent intervenir également dans les parties mémoires utilisées par le circuit, mais leur effet est alors temporaire car il n'altère pas durablement l'architecture du circuit.

Dans le contexte d'injection de fautes, l'utilisation de prototypes a été proposée pour améliorer et accélérer la réalisation des campagnes d'injection. Pour implémenter de tels prototypes, les FPGA SRAM semblent être un excellent choix. Mais il est nécessaire de bien distinguer ce qui peut survenir dans la cible finale d'implantation et ce qui correspond juste à un prototypage permettant d'accélérer l'évaluation de certains effets. Ainsi, l'injection de fautes rémanentes n'a de sens que pour l'évaluation de circuits devant effectivement être implantés sur FPGA dans le produit final. Si le produit final est un circuit à fabrication spécifique (ASIC), les injections dans la configuration ne sont utilisées que sur une base transitoire, pour représenter la perturbation temporaire d'un signal ou d'un point mémoire.
Outre les fautes survenant dans la mémoire de configuration, les perturbations survenant dans un FPGA peuvent aussi conduire à des interruptions fonctionnelles (SEFI – Single Event Functional Interrupt), correspondant le plus souvent à l'apparition d'un état aberrant dans un bloc support du FPGA (logique de configuration, générateur d'horloge, etc.). Dans de nombreux cas, de telles interruptions nécessitent un redémarrage du système. Ce type d'effet ne sera pas considéré dans la suite du document.

I.4. Techniques d'injection de fautes

L'injection de fautes s'est avérée être une technique efficace pour l'analyse de sûreté, depuis la fin des années 80. Les approches existantes peuvent être classées en deux catégories : les approches d'injection qui s'appliquent lorsque le circuit à analyser est déjà fabriqué et les techniques qui permettent une analyse plus tôt dans le flot de conception, typiquement au niveau transferts de registres ou au niveau portes. Une liste non exhaustive des approches d'injection de fautes est présentée dans la figure 1-2. Les méthodes qui vont être présentées ont été choisies pour leurs caractéristiques spécifiques donnant une idée concrète sur l'ensemble de l'état de l'art. Nous nous limiterons par ailleurs aux méthodes permettant d'évaluer les effets de fautes de type SEU, en nous concentrant sur les méthodes généralistes, applicables à tout circuit numérique.

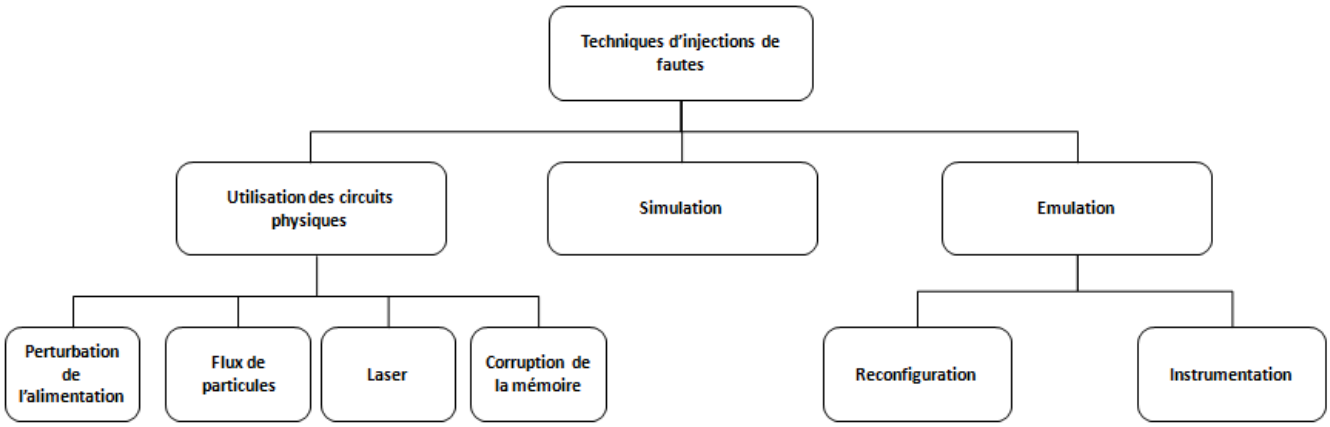


Figure 1-2. Exemples de méthodes d'injection de fautes

I.4.1 Approches utilisant des circuits physiques

L'approche la plus utilisée pour caractériser un circuit existant est de le placer dans un faisceau de particules. Cette approche, outre un coût élevé, ne permet souvent d'observer qu'un petit nombre d'évènements. Par ailleurs, aucun contrôle spatial ou temporel n'est possible sur les fautes injectées. Une autre approche a donc été significativement développée ces dernières années : l'utilisation de bancs laser.

Le laser permet d'injecter des fautes représentatives des erreurs transitoires de façon non intrusive et non destructive dans des circuits fonctionnant à fréquence nominale au moment de l'injection. Le laser offre une très bonne contrôlabilité dans le temps et dans l'espace. En effet, les techniques d'excitation par laser permettent d'irradier avec précision des zones de systèmes complexes (Processeur : cache, registres, pipeline d'un processeur, logique de prédiction de branchement, etc... DRAM : Logique de contrôle, zone mémoire, etc..). Afin de reproduire la nature impulsionnelle et singulière d'une particule, les lasers sont pulsés avec des durées d'impulsion ultra courtes allant du pico à la centaine de femto secondes. Cependant, avec les avancées technologiques actuelles qui tendent à diminuer la taille des circuits, la taille des faisceaux lumineux risque de devenir une limitation.

Les possibilités de simuler expérimentalement l'effet d'un SEU à l'aide d'un laser pulsé font l'objet d'un intérêt particulier depuis les années 60 avec les travaux de H. Habing **[Habi.-65].** Plus récemment, **[Jaul.-09]** a poursuivi le travail de son équipe **[Poug.-99]** pour arriver à une formalisation de l'interaction laser-semiconducteur. Pendant les deux dernières décennies, des effets de rayonnement sur des cellules de SRAM ont été largement étudiés **[Zout.-87] [More.-95]**

[Dodd-96]. D'ailleurs, cela a permis aussi de comprendre les mécanismes de base responsables des SEUs dans les mémoires statiques.

La méthode proposée dans **[Samp.-97]** offre une fréquence de fonctionnement de 50kHz et une précision spatiale de 0,1µm. Le laser et l'environnement mis en œuvre au laboratoire IMS ont des caractéristiques similaires : précision spatiale de 0,1µm, taille du spot de 1µm mais une fréquence de fonctionnement beaucoup plus élevée (80MHz). L'automatisation du banc permet de réaliser des cartographies basées sur l'évolution de nombreux paramètres de l'onde électrique (temps de montée, valeur maximale, valeur crête à crête, etc…) avec une résolution spatiale inférieure au micron. La précision temporelle de l'impulsion laser permet de mettre en œuvre des études dynamiques. Deux limitations rapportées dans **[Duze.-00]** sont les pulsations de fuites qui peuvent entrainer des bit-flips « parasites » et la taille du spot laser (1µm) qui est proche de la taille de la cellule étudiée (4,8µm2).

Le laser peut également être utilisé pour caractériser la sensibilité d'un circuit aux SEU **[Lewi.-05]**. La méthode se montre très efficace mais doit être approfondie pour des technologies actuelles.

Le travail effectué dans **[Cani.-08]** analyse les effets d'un seul tir laser pulsé sur la mémoire de configuration d'un FPGA. Cette étude tient compte de plusieurs diamètres de spots, ciblés sur plusieurs types de blocs logiques et compare leurs effets. Nous utiliserons les résultats obtenus dans cette étude dans le chapitre III.

D'autres approches peuvent être utilisées pour analyser la sensibilité d'un circuit aux SEUs, lorsque le circuit physique est disponible. En particulier, des approches ont été proposées dans le cas des microprocesseurs, telle la méthode CEU **[Vela.-00]**. Nous ne détaillerons pas ces approches, qui sont spécifiques à un type de circuits.

I.4.2. Approches à base de simulation

Les outils de simulation ont été développés pour valider le fonctionnement d'un circuit. Ils peuvent également être utilisés pour l'analyse de sûreté d'un système à différentes étapes du flot de conception. Le principe est de comparer par simulation le fonctionnement du système sans et avec injection de fautes, l'injection de fautes étant effectuée via le simulateur.

La simulation de descriptions de haut niveau (High-level Description Language - HDL) permet de vérifier un circuit de façon fonctionnelle avant synthèse. La simulation niveau portes permet

de vérifier également le timing du circuit. La contrepartie est que les simulations niveau portes sont plus lentes de plusieurs ordres de grandeur que les simulations RTL.

Les simulateurs analogiques ou les simulateurs mixtes permettent d'utiliser des modèles de fautes encore plus précis. Mais les temps de simulation deviennent alors prohibitifs.

Dans cette partie nous allons surtout nous focaliser sur les approches d'injection de fautes par simulation au niveau RTL :

- Exploitation des commandes du simulateur

Il est possible d'utiliser les routines d'un langage spécifique (TCL par exemple) ainsi que les commandes du simulateur, lors de la simulation RTL de la description du circuit à analyser, afin d'injecter une ou plusieurs fautes en forçant des signaux internes.

MEFISTO (Multi-level Error/Fault Injection Simulation Tool) **[Jenn-94]** est un outil qui utilise les commandes du simulateur, la manipulation de signaux et la manipulation de variables pour l'injection. La manipulation de signaux cible l'injection de fautes de collage permanentes et transitoires dans une description structurelle alors que la manipulation de variables vise l'injection de bit-flips au niveau comportemental.

Un autre outil de simulation RTL a été développé au sein de l'université de Turin. Il exploite les mécanismes de debug du simulateur commercial Modelsim ainsi que l'interface proposée par le langage TCL (Tool Command Language) **[Corn.-00]**. L'analyse de la description comportementale (VHDL RTL) permet de générer une liste de fautes en fonction du modèle de fautes choisi. Le « simulateur de fautes » se compose d'un ensemble de routines qui interagissent avec le simulateur.

- Simulation avec description instrumentée

Parallèlement à l'approche utilisant les commandes utilisateurs comme cité précédemment, le laboratoire LAAS a développé une approche à base d'instrumentation, MEFISTO-L **[Boue-98]**. Cette instrumentation est effectuée par l'addition de saboteurs et de sondes. Les saboteurs sont utilisés pour modifier la valeur ou le délai de certains signaux alors que les sondes permettent d'observer la valeur des signaux. La description originale est analysée pour générer les modèles de fautes, les cibles potentielles et les signaux à observer. L'approche se base également sur un simulateur commercial pour la compilation et la simulation de la description.

L'injection de fautes avec instrumentation a été également utilisée dans **[Hadj.-05]**. Dans ce travail, une nouvelle approche pour la génération de mutants a été présentée, permettant l'instrumentation d'un circuit pour des modèles de fautes hétérogènes. La modification est alors faite en modifiant finement la description RTL des blocs du circuit à analyser, contrairement à l'approche à base de saboteurs qui rajoute des blocs entre les blocs existant. L'approche permet d'injecter des fautes de type SEU, MBU ou MCU dans n'importe quel bloc, ce qui n'est pas possible avec des saboteurs, mais la modification de la description du circuit est beaucoup plus difficile à automatiser.

I.4.3. Approches à base d'émulation

La simulation permet de réaliser des analyses avec beaucoup de flexibilité, mais nécessite des temps expérimentaux très élevés (ou des moyens de calcul très puissants), surtout dans le cas de circuits complexes, exécutant des applications nécessitant un grand nombre de cycles. L'émulation représente un gain de temps important par rapport à la simulation. Autre avantage, la reconfiguration dynamique de certains FPGA est aussi une caractéristique qui peut être exploitée pour l'injection de fautes. L'utilisation du prototypage s'est largement répandue avec les améliorations des circuits programmables récents (vitesse, taille de la logique programmable…).

Parmi les premières techniques développées sur la base de l'émulation, nous pouvons citer celle à base de reconfiguration présentée dans **[Burg.-96]**, celles à base d'instrumentation présentées dans **[Hwan.-98]** et **[Seda.-98]** et enfin celle combinant reconfiguration et instrumentation proposée dans **[Chen.-99]**. Ces premières techniques ne visaient pas l'analyse de sûreté, mais plutôt l'accélération de la simulation de fautes pour le test de fin de fabrication.

I.4.3.1. Injection de fautes avec instrumentation

Comme indiqué pour la simulation, le principe de l'instrumentation correspond à la modification de la description initiale afin de permettre l'injection de fautes. L'instrumentation se fait soit en insérant un ou plusieurs blocs entre les blocs du circuit original, soit en modifiant un ou plusieurs blocs de ce circuit. Un circuit instrumenté doit se comporter de façon nominale lorsque l'injection de fautes n'est pas activée. L'instrumentation permet de modifier la valeur d'éléments internes : par exemple les signaux pour une instrumentation HDL ou les nœuds pour une instrumentation niveau portes. Le circuit instrumenté est ensuite synthétisé puis émulé. La modification peut aussi être faite après synthèse, sur la description niveau portes.

La pertinence du prototypage pour l'analyse de sûreté a été mise en avant au TIMA en 1999 dans **[Leve.-99]**. L'instrumentation et la génération de modèles comportementaux pour représenter les modes de propagation d'erreur sont également présentées comme des techniques performantes pour l'analyse de sûreté.

La thèse **[Vanh.-08]** se base sur cette technique pour développer une méthodologie et un environnement améliorant l'étude de la robustesse de circuits intégrés numériques. L'approche proposée met en œuvre un prototype matériel d'une version instrumentée du circuit à analyser. L'environnement comprend trois niveaux d'exécution dont un niveau logiciel embarqué qui permet d'accélérer les expériences en conservant une grande flexibilité : l'utilisateur peut obtenir le meilleur compromis entre complexité de l'analyse et durée des expériences. **[Vanh.-08]** propose également de nouvelles techniques d'instrumentation et de contrôle des injections afin d'améliorer les performances de l'environnement. Une évaluation prédictive de ces performances renseigne l'utilisateur sur les paramètres les plus influents et sur la durée de l'analyse pour un circuit et une implantation de l'environnement donnés.

Enfin la méthodologie est appliquée pour l'analyse de deux systèmes significatifs dont un système matériel/logiciel construit autour d'un microprocesseur SparcV8.

De nombreux autres travaux ont été réalisés à travers le monde sur des principes similaires ; il serait trop long de tous les détailler, d'autant plus que l'approche que nous proposerons dans cette thèse n'utile pas l'instrumentation.

I.4.3.2. Injection de fautes par reconfiguration

Pour les FPGAs basés sur de la SRAM, le fichier de configuration contient les données ainsi que les commandes de configuration. Les cellules SRAM de configuration peuvent être vues, en simplifiant, comme un registre à décalage. Les données de configuration sont téléchargées sur le FPGA de façon sérielle depuis l'ordinateur hôte. Pour certains composants, ces données peuvent ensuite être relues à partir du FPGA ainsi que le contenu de toutes les cellules mémoires ; ces deux opérations sont appelées respectivement readback et capture.

Une fois qu'un FPGA a été configuré, il existe deux manières de le reconfigurer : de façon statique (Compile-Time Reconfiguration - CTR) ou de façon dynamique (Run-Time Reconfiguration - RTR).

- Dans le cas d'une reconfiguration statique (CTR), pour implémenter une quelconque modification il est nécessaire de recompiler et re-synthétiser les blocs modifiés, de régénérer le fichier de configuration pour le système complet et de télécharger cette nouvelle configuration sur le FPGA.

- La reconfiguration dynamique (RTR) offre la possibilité de reconfigurer certains FPGA durant le déroulement d'une application. Il est possible de reconfigurer tout le circuit (Global RTR) ou parfois uniquement une partie spécifique de celui-ci, on parle alors de reconfiguration partielle (Local RTR). Lors d'une reconfiguration partielle, le sous-ensemble non reconfiguré peut, avec certains FPGA, continuer à fonctionner normalement.

L'injection de fautes par reconfiguration se fait par modification directe des données de configuration, sans nécessiter de modifications de la description du circuit, donc sans instrumentation. L'un des avantages est d'éviter le risque d'une erreur lors de l'instrumentation, conduisant à des erreurs lors de l'évaluation de sûreté. Le deuxième avantage est le gain de ressources, permettant d'évaluer des circuits plus complexes sur un FPGA donné.

Le principal objectif de la reconfiguration dynamique est d'économiser le temps induit par les différentes étapes (compilation, synthèse, génération du fichier de configuration complet) de la reconfiguration statique. Les données de configuration peuvent être conservées lors de la première configuration ou bien relues sur le FPGA (readback). Cette technique permet d'injecter des collages permanents et transitoires dans les blocs combinatoires, ceci en modifiant par exemple les tables de configuration (Look-Up Table - LUT). L'injection de SEUs dans les éléments mémoires requiert la lecture du contenu des éléments mémoire (capture).

Les expériences présentées dans **[Anto.-01]** ont été effectuées avec deux FPGA Xilinx distincts, l'un d'entre eux disposant de mécanismes de reconfiguration partielle. Les bitstreams (fichiers de configuration Xilinx) y sont modifiés avec JBits, un outil Java qui permet une lecture de la configuration du FPGA et une reconfiguration simple et rapide. Les résultats obtenus démontrent la faisabilité de l'approche et montrent que celle-ci permet des économies de temps importantes dans certaines conditions. Il semble cependant que de nombreuses contraintes ont été rencontrées. Une description plus complète de l'approche ainsi que des résultats obtenus sont proposés dans **[Anto.-03-1]** et **[Anto.-03-2]**.

L'endo-reconfiguration (utilisée dans **[Ster.-07]**) étend la notion de reconfiguration dynamique. Elle suppose que des circuits sur le même FPGA sont utilisés pour contrôler la reconfiguration des autres parties du FPGA. Cette approche intéresse particulièrement les concepteurs à cause de l'autonomie qu'elle apporte. Nous reviendrons plus précisément sur cette approche dans le cadre de nos propositions.

I.4.4. Bilan de l'état de l'art sur les techniques d'injection de fautes

La méthode d'injection de fautes par reconfiguration présente plusieurs avantages par rapport à ses concurrentes, voir Tableau I-III. En effet, par rapport à l'injection par moyens physiques (qui seraient possibles aussi à partir d'un prototype), l'injection de fautes par reconfiguration présente des coûts inférieurs ainsi que des temps expérimentaux plus faibles et elle ne présente pas de contraintes sur la disponibilité des sources de perturbations physiques. Par ailleurs, les techniques d'injection par reconfiguration permettent de réduire les durées de campagnes par rapport aux techniques basées sur la simulation (une campagne d'injection par reconfiguration est 25 fois plus rapide qu'une campagne d'injection par simulation fonctionnelle). Enfin, les techniques d'injection de fautes par instrumentation nécessitent d'augmenter sensiblement la complexité du circuit à implanter et induisent notamment l'ajout d'un certain nombre d'E/S au circuit à analyser pour pouvoir commander les injections.

Tableau I-III Bilan de l'état de l'art des méthodes d'injection de fautes

		Coûts	Temps expérimentaux	Disponibilité des sources	Modification du circuit initial
Injection de fautes par moyens physiques		✗	✗	✗	✓
Injection de fautes par simulation		✓	✗	✓	✓
Injection de fautes par Emulation	Instrumentation	✓	✓	✓	✗
	Reconfiguration	✓	✓	✓	✓

Après l'étude de l'état de l'art, nous avons donc opté pour la technique d'injection de fautes par reconfiguration partielle du FPGA. Nous allons nous baser sur le concept d'endo-configuration utilisé dans **[Ster.07]** pour développer notre environnement d'injection de fautes. Cette approche nous permettra à la fois d'injecter des erreurs dans la configuration du FPGA et dans

les mémoires utilisateur, alors que les approches basées sur l'instrumentation ne permettent d'injecter que dans les mémoires utilisateur.

I.5. Méthodes de protection

Une analyse de sûreté par injection de fautes conduit le plus souvent à identifier des zones critiques dans un circuit, qui doivent être protégées pour répondre aux besoins de l'application. Le but des techniques de tolérance aux fautes est de limiter les effets d'une faute, ce qui signifie augmenter la probabilité qu'une erreur soit gérée ou tolérée par le système. Une caractéristique commune de toutes les techniques de tolérance aux fautes est l'utilisation de la redondance. Nous nous proposons ici de faire une brève comparaison des techniques de base de tolérance aux fautes existantes en termes de capacité de détection (paragraphe I.5.1) et de correction d'erreur (paragraphe I.5.2). Cette étude ne cherche pas à être exhaustive, mais à montrer les principales directions pouvant être suivies pour protéger un circuit.

I.5.1 Détection d'erreurs

La détection d'erreur génère un signal d'erreur ou un message dans le système. Elle peut être basée sur la détection préventive ou sur une vérification concurrente :

- La détection préventive permet surtout de contrôler le système afin de détecter les erreurs latentes et les fautes dormantes. En effet, elle agit essentiellement hors-ligne, quand la prestation de service normale est suspendue. Notre étude ne va pas porter sur ce type de détection, qui ne permet pas une réactivité suffisante face à des fautes transitoires comme les SEU.
- La vérification concurrente est une technique en ligne qui a lieu pendant la prestation de service normale. **[Bick.-10]** définit la détection d'erreurs concurrente (Concurrent Error Detection, CED) comme un processus de détection et de rapport d'erreurs tout en réalisant, en même temps, les opérations normales du système.

I.5.1.1 Redondance matérielle

La redondance matérielle est une approche couramment utilisée. Elle fait référence à l'ajout de ressources matérielles supplémentaires, tel que le doublement du système, en utilisant un comparateur en sortie pour détecter les erreurs. Il est tenu compte ici de la structure du circuit et non pas de la fonctionnalité. La duplication matérielle est tout aussi efficace pour les fautes transitoires que pour les fautes intermittentes ou permanentes. Cependant, les exigences en

surface et en puissance sont très élevées. Elle peut être classée en deux sous-types : (i) duplication avec comparaison et (ii) duplication avec redondance complémentaire.

i. La duplication avec comparaison (Duplication With Comparison, DWC) ou double redondance modulaire (Dual Modular Redundancy, DMR), illustrée dans la figure 1-3, est une technique de détection d'erreur simple et facile à mettre en œuvre. Elle possède une bonne capacité de détection d'erreur, sauf pour les défauts de conception, les erreurs dans le comparateur, ou les combinaisons d'erreurs simultanées dans les deux modules. Le coût matériel du circuit augmente de plus de 100 %.

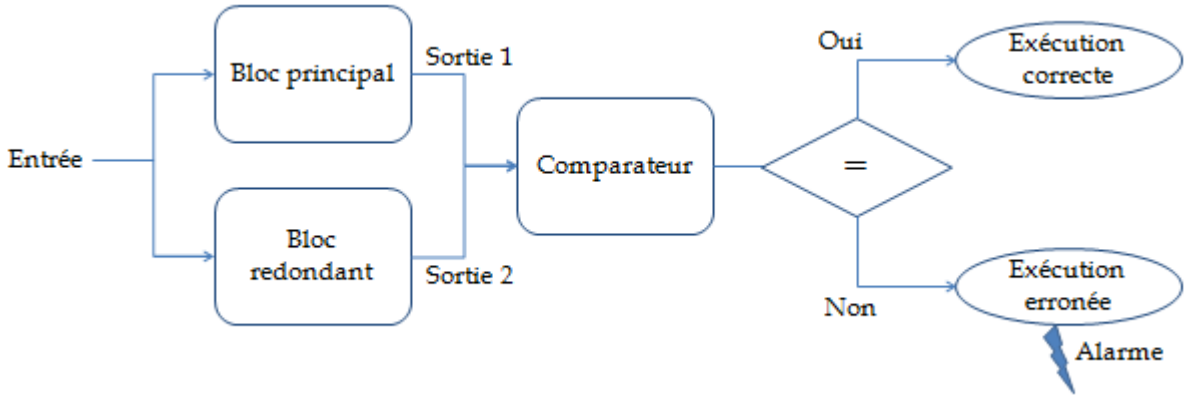


Figure 1-3. Double Redondance modulaire

ii. La technique nommée « duplication avec redondance complémentaire » (Duplication with Complementary Redundancy, DWCR) est similaire à la DWC à la différence près que les signaux d'entrée, les signaux de contrôle en sortie, ainsi que les signaux de données internes de chacun des deux modules sont de polarité opposée pour limiter le risque d'erreurs similaires simultanées dans les deux modules afin d'éviter la défaillance du système. Ici aussi, l'augmentation de surface et de puissance consommée est supérieure à 100 % et cette méthode augmente la complexité de la conception par rapport à une simple duplication. D'autres types de diversité entre modules peuvent être envisagés, avec les mêmes inconvénients.

I.5.1.2. Redondance temporelle

La redondance temporelle désigne une technique qui nécessite une seule unité effectuant une même opération deux fois de suite. Si une différence est constatée entre les deux calculs successifs, cela signifie qu'une faute est apparue lors d'un des calculs. Il s'agit d'une technique de réplication temporelle, sans considération de la fonctionnalité du circuit.

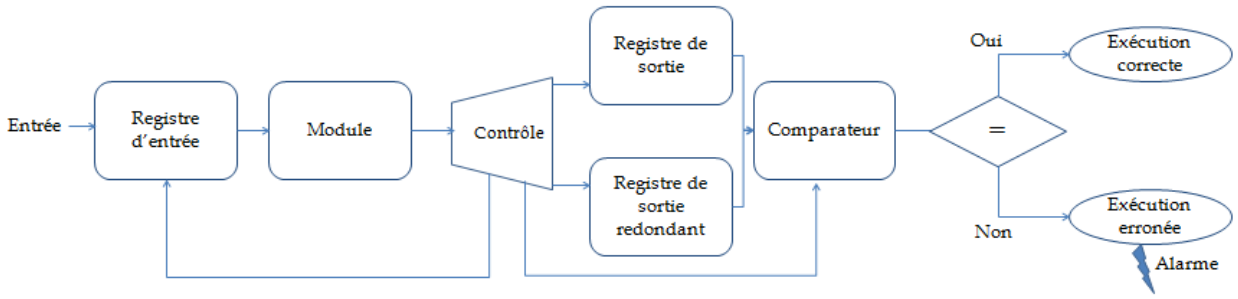


Figure 1-4. Redondance temporelle

Dans cette technique, certaines fautes intermittentes et les fautes transitoires sont détectées mais les fautes permanentes ne le sont pas, sauf dans le cas d'un double calcul avec diversité. La redondance temporelle affecte directement les performances du système, même si le coût matériel est généralement inférieur à celui de la redondance matérielle. Ainsi les systèmes basés sur la redondance temporelle sont comparativement plus lents. Afin de surmonter cet inconvénient, de nombreux systèmes utilisent le pipelining pour masquer le problème de latence. Les conséquences énergétiques de la redondance temporelle sont souvent semblables à celles d'une redondance matérielle.

I.5.1.3. Redondance d'information

La redondance d'information a été présente dès le début de l'informatique, sous la forme des codes détecteurs. Déjà dans les années '40 Shannon et Hamming ont posé les bases théoriques, pratiques et statistiques de cette discipline, qui n'a pas cessé d'évoluer au cours des années grâce aussi au développement des télécommunications, qui l'utilisent massivement.
Tous les codes sont basés sur le même principe : à partir des bits d'information à protéger (que l'on appellera « mot ») on va obtenir des bits de code à travers un calcul mathématique souvent très complexe, mais correspondant toujours à des opérations entre bits.

Au moment du contrôle, il faut vérifier que les bits de code sont toujours compatibles avec les bits du mot : en cas contraire une erreur est détectée et, si le code le permet, corrigée. Une fois encore on introduit des composants critiques (codeurs et décodeurs), qui sont donc souvent implantés de façon auto-contrôlable (en Dual Rail par exemple).

Le code le plus connu est sans doute le code de parité : un bit de code est ajouté au mot de telle façon que le nombre de bit à 1 dans le mot soit toujours pair ou impair. Cet artifice permet de détecter toutes les transitions erronées d'un nombre impair de bits, mais pas les autres erreurs.

Le grand avantage de ce code est sa simplicité d'implantation (un arbre de XOR ou XNOR suffit) et son surcoût est modéré (un seul bit est ajouté par mot, avec le codeur et le décodeur).

Une autre famille de codes très répandue est celle des codes de Hamming. Ici les mots de code sont choisis de façon à avoir une certaine distance minimale « d » entre eux. Cette distance est calculée par une fonction P (poids de Hamming), qui correspond au nombre de bits différents. Presque tous les codes développés plus récemment suivent le même principe que les codes de Hamming : l'évolution consiste principalement à trouver des règles mathématiques qui permettent de définir des codes avec de bonnes distances minimales qui demandent peu de bits de contrôle ou qui soient faciles à calculer/vérifier pour augmenter les performances. On cite par exemple les Codes à Redondance Cyclique ou CRC, les codes de Berger et les codes à résidu.

La somme de contrôle (Checksum)

Une autre forme de redondance d'information, très répandue dans le domaine des réseaux, consiste à stocker une donnée supplémentaire appelée somme de contrôle ou empreinte dans le bloc de données à protéger. Avant d'utiliser ces données protégées, le mécanisme de détection recalcule son empreinte et compare la valeur obtenue à celle stockée.

I.5.1.4. Détection avec des assertions

Une assertion est une expression logique permettant d'effectuer en ligne un contrôle de vraisemblance sur les objets d'un programme ou le comportement d'un circuit. La valeur de l'expression logique est à VRAI si l'état est correct et à FAUX dans le cas contraire ; dans ce dernier cas, une exception est levée. Trois types de contrôles peuvent être mis en œuvre par des assertions, à savoir les contrôles temporels et d'exécution, les contrôles de vraisemblance et les contrôles de données structurées. Il s'agit d'une technique de réplication basée sur la fonctionnalité du circuit. D'autres méthodes sont basées sur la surveillance de la fonctionnalité, comme les méthodes de vérification de flot de contrôle ; nous en verrons un exemple au chapitre III.

I.5.2. Correction d'erreurs

I.5. 2. 1. Redondance matérielle

Pour faire de la correction d'erreurs avec de la redondance matérielle, à partir d'une architecture DMR (présentée dans le paragraphe 1.5.1.1), il suffit d'ajouter un troisième bloc et de remplacer le comparateur par un voteur. Cette nouvelle architecture, appelée architecture

TMR, est illustrée dans la figure 1-5. Dans cette technique, les effets des fautes sont masqués. Tous les composants fonctionnent simultanément et leurs sorties sont envoyées vers un voteur. La sortie du voteur sera correcte si au moins deux des composants sont non défectueux. Les techniques de redondance statique sont caractérisées pour être simples, mais elles ont un fort surcoût en surface et en consommation, variant entre 300 et 350% par rapport à une conception non-redondante **[John.-08].**

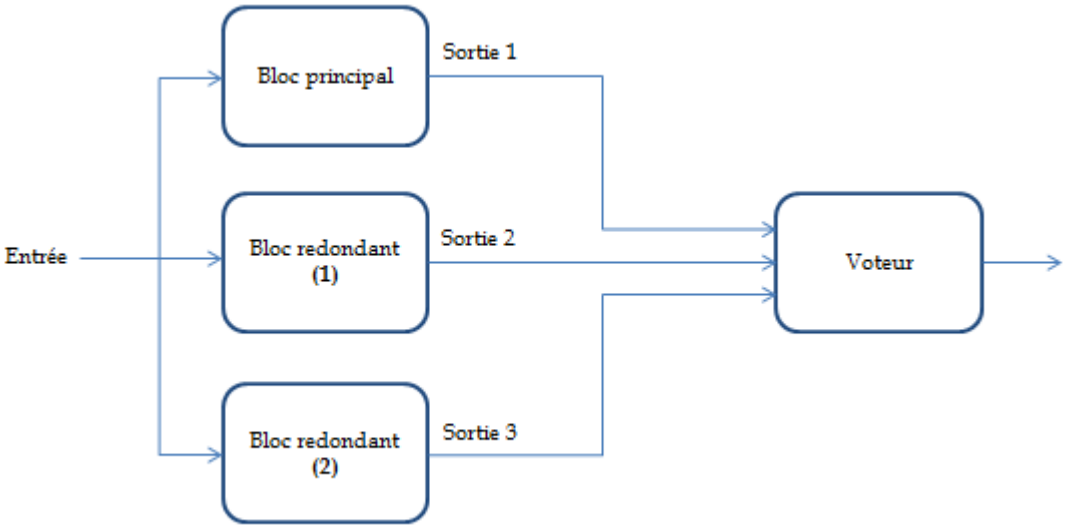


Figure 1-5. Protection par triplication

La TMR peut être employée pour traiter les erreurs de données à bit unique (SET, persistantes ou non) survenant dans une cellule **[Carm.-01].**

Le point faible de cette architecture réside dans son voteur. En effet, si une faute se produit dans ce dernier tout le système échoue. Toutefois, le voteur est généralement petit, et donc souvent considéré comme fiable. La redondance matérielle peut aller aussi jusqu'à la quadruplication des modules à protéger, comme cela a été fait dans **[Nikn.-11].** Une approche plus générale de la redondance N-modulaire est discutée dans **[Kore.-07].** Comme pour la duplication, de nombreuses variantes existent incluant notamment la diversification des blocs.

I.5. 2. 2. Redondance temporelle

Pour corriger les erreurs avec une architecture de redondance temporelle, il faut répéter le calcul à trois intervalles de temps différents puis voter sur les trois résultats obtenus. Cette méthode implique donc trois fois plus de cycles d'horloge pour exécuter la même tâche, ce qui réduit l'utilisation de cette méthode aux systèmes avec peu ou pas de contraintes temporelles. Elle offre des surcoûts en surface plus faibles par rapport à la TMR décrite dans le paragraphe

1.5.2.1. Néanmoins la redondance temporelle ne peut corriger que les erreurs dues aux fautes transitoires à condition que la durée de la faute soit inférieure au temps de calcul.

I.5. 2. 3. Redondance mixte matérielle / temporelle

Pour implanter des systèmes tolérants aux fautes, il y a toujours des compromis à faire à cause de la redondance essentiellement. En effet, une approche basée totalement sur la redondance matérielle peut causer un important surcoût en surface, alors que la redondance temporelle engendrer des pénalités élevées sur la performance. Une approche mixte peut donc être employée, comme illustré dans la figure 1-6 **[Lima-03]**.

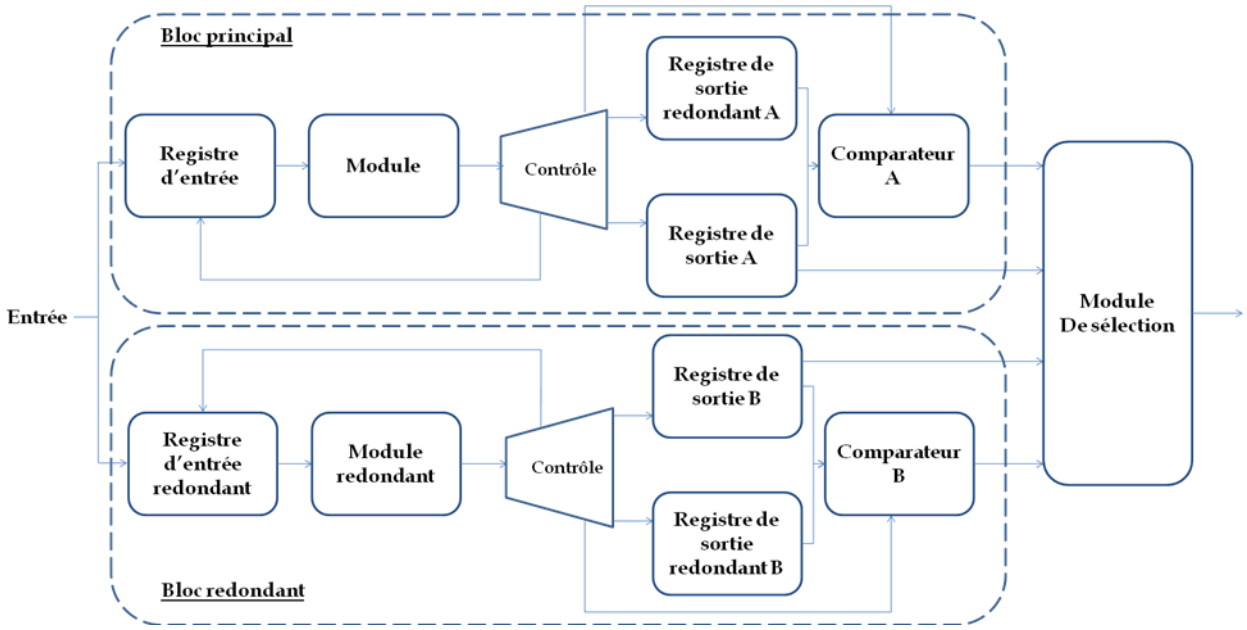


Figure 1-6. Redondance mixte

I.5. 2. 4. Redondance d'information

Comme pour la détection, la redondance d'information pour la correction des erreurs nécessite le calcul d'un code à partir des bits d'information à protéger à travers un calcul mathématique complexe, mais correspondant toujours à des opérations entre bits.

Les codes correcteurs d'erreurs (*Error Correcting Codes, ECC)* peuvent représenter une solution moins coûteuse que les autres techniques de redondance. Ils sont couramment utilisés pour protéger les mémoires. Un ECC est caractérisé par son nombre de bits supplémentaires, le matériel supplémentaire pour le codage et le décodage et enfin son temps de latence. Par ailleurs, il possède deux paramètres clés à savoir le nombre de bits erronés qui peuvent être détectés et le nombre de bits erronés qui peuvent être corrigés.

Les ECC les plus couramment utilisés dans les circuits numériques sont les codes de Hamming, Hsiao et Reed-Solomon :

- Les codes de Hamming sont les codes les plus simples. Ils sont capables de corriger une seule erreur, et d'en détecter deux (SEC-DED), mais pas les deux simultanément. Ce sont les premiers codes ECC linéaires. Ils sont très utiles dans les cas où la probabilité d'une seule erreur est significative.
- Les codes Hsiao (aussi appelés codes de Hamming évolués) sont d'autres codes communément utilisés pour la protection/correction des erreurs dans les mémoires. Ils permettent un encodage et une détection d'erreur plus rapide que les codes Hamming.
- Les codes Reed-Solomon figurent parmi les codes les plus puissants pouvant être construits en utilisant des polynômes générateurs appropriés et sont particulièrement bien adaptés aux applications où les erreurs se produisent en rafales. Ces codes cycliques nécessitent toutefois un circuit de codage et de décodage complexe.

I.5.3. Méthodes de durcissement spécifiques aux FPGAs SRAM

Différentes approches ont été proposées pour augmenter la robustesse de circuits implantés sur FPGA configuré par SRAM. Ces approches se classent essentiellement en trois catégories principales : utiliser une technologie plus robuste, modifier la structure du FPGA en remplaçant les points mémoires par des points mémoire durcis ou en ajoutant des dispositifs de détection ou correction d'erreur, ou encore modifier le circuit à implanter pour lui permettre de mieux détecter ou tolérer les perturbations.

Dans le cadre de cette thèse, nous cherchons à améliorer l'utilisation de dispositifs commerciaux ; nous ne détaillerons donc pas ce qui nécessiterait la conception d'un nouveau FPGA. Parmi les approches basées sur l'amélioration du circuit à implanter, de nombreuses propositions sont basées sur l'implantation d'une architecture TMR, éventuellement un peu simplifiée ou complétée par des améliorations du routage, ou encore par de la reconfiguration dynamique partielle. Nous ne détaillerons dans ce qui suit que les techniques ne faisant pas l'hypothèse d'une utilisation de redondance massive.

Un cas particulier de redondance matérielle spécifique aux FPGAs tire profit de la propriété du FPGA à stocker son comportement et sa configuration dans des mémoires. La technique la plus directe consiste à comparer périodiquement, par un système tiers, la configuration actuelle et une copie de référence, le duplica **[Carm.-09]**. Des techniques plus évoluées permettent de faire

réaliser ces tests par une partie du circuit, alors que le reste du circuit continue à exécuter la fonction utilisateur **[Shni.-98] [Emme.-07].** La latence de détection est toutefois assez grande.

[Naza.-12] présente une nouvelle variante de redondance matérielle. Elle consiste à faire de la DMR sur chaque LUT utilisée, au lieu de le faire sur des modules de composants. Cette technique est dite DMR à "grain fin" par opposition à la technique classique DMR à gros grain (coarse grained DMR). La DMR à "grain fin" permet d'avoir des temps de latence inférieurs à ceux d'une DMR traditionnelle, mais présentent l'inconvénient du surcoût mémoire à cause du besoin supplémentaire en comparateurs.

Pour réduire le surcoût mémoire, **[Naza.-12]** propose d'utiliser les ressources intrinsèques du FPGA, dont notamment la chaine de propagation de la retenue qui existe dans les FPGAs Virtex V et est souvent inexploitée par les outils de synthèse. Pour arriver à des surcoûts optimaux atteignant 3,2% pour un temps de latence 7,68 fois inférieur aux temps de latence moyen des techniques traditionnelles, cette technique doit en plus être couplée avec une stratégie de routage optimale pour permettre l'utilisation au mieux des ressources du FPGA.

Dans **[Ferr.-12-1]**, une nouvelle technique de protection a été proposée pour les FPGAs AT40k. Celle-ci a pour but de protéger les bits les plus critiques tout en exploitant les composants inutilisés dans le FPGA. Le principe a été validé sur une étude de cas, mais sans automatisation de l'approche.

[JuYu.-10-1] utilise également les ressources intrinsèques des FPGAs Virtex et Altera pour masquer les erreurs de bit dans les LUTs. Pour ce faire, ils dupliquent la LUT à protéger et connectent sa sortie ainsi que la sortie de son duplica par une porte AND ou OR, selon le type du bit-flip à considérer (si le bit est à 1 le masquage est avec une porte AND, s'il est à 0 il faut utiliser une porte OR). L'avantage de cette méthode est qu'elle utilise les ressources non exploitées dans le FPGA et permet donc de faire le masquage à moindre coût. L'inconvénient est que cette technique ne permet de masquer que 50% des bit-flips (si le masquage utilise des portes AND les bitflip 1->0 ne seront pas masqués et pour le OR ce sont les bitflip 0->1 qui seront ignorés) sans oublier qu'une erreur dans la logique dupliquée amènerait à un résultat erroné. Cette technique a été améliorée dans **[JuYu.-10-2]** en faisant de la décomposition pour les fonctions trop grandes pour être protégées par duplication et le but de cette technique est d'augmenter le temps moyen de bon fonctionnement, MTTF (Mean Time To Failure). L'approche présentée reste essentiellement théorique.

I.6. Conclusion

Nous avons vu dans ce chapitre différents problèmes liés à la sûreté de fonctionnement. Ces problèmes sont multiples et croissants essentiellement avec la réduction des dimensions des procédés microélectroniques. Les fautes transitoires, quelque soient leurs origines naturelles ou malveillantes et quelques soient leurs multiplicités, doivent désormais être prises en compte même pour les applications au niveau de la mer. Les circuits programmables FPGA configurés par SRAM sont tout particulièrement concernés par ces phénomènes.

L'analyse de l'état de l'art des méthodes d'injection de fautes a permis de souligner les avantages et les inconvénients des approches utilisées pour évaluer, tôt dans le flot de conception, la robustesse d'un circuit. Parmi ces méthodes, nous avons choisi la technique utilisant une endo-reconfiguration partielle, permettant d'injecter aussi bien dans la mémoire de configuration que dans les éléments de mémorisation utilisateur. Cette approche sera à la base des environnements proposés dans le chapitre III.

Par la suite, nous avons survolé les méthodes de protection des circuits afin de les comparer selon plusieurs critères tels que le taux de couverture ou encore les différents surcoûts. Les solutions traditionnelles impliquant une redondance excessive sont pénalisées en surface, en puissance et/ou en performance. L'utilisation de codes est essentiellement limitée par leur capacité à gérer des erreurs multiples sans induire des coûts équivalents.
Les techniques de durcissement spécifiques aux FPGAs SRAM, et applicables pour des FPGA commerciaux, souffrent soit d'un taux de couverture assez faible, soit d'une latence de détection élevée, soit d'un surcoût élevé, soit d'une absence d'automatisation. Nous chercherons à proposer dans le chapitre IV une approche automatisée et généralisable, qui s'inscrit dans la catégorie des techniques de durcissement spécifiques aux FPGAs SRAM, permettant d'augmenter la robustesse avec des pénalités et une latence de détection plus faibles et en tenant compte de la possibilité d'erreurs multiples.

Afin de compléter les bases nécessaires à la compréhension de notre démarche d'injection de fautes et de protection, nous allons présenter dans le chapitre II les différentes familles de réseaux programmables de type SRAM que nous avons sélectionnées pour notre étude.

Chapitre II

Présentation des FPGAs étudiés

Dans cette section, nous allons présenter les architectures sur lesquelles nous allons travailler ainsi que les différentes possibilités qu'elles offrent, que ce soit pour l'injection de fautes ou pour la protection intégrée.

Au cours de nos travaux nous nous sommes intéressés à 3 constructeurs différents : ALTERA, ATMEL et XILINX. Les FPGAs étudiés sont respectivement le Stratix IV pour ALTERA, l'AT40K pour ATMEL et le Virtex II Pro puis le Virtex V pour XILINX.

II.1. Virtex

II.1.1. Présentation générale des FPGAs Virtex

Dès l'origine, les FPGA, tels que Xilinx les a inventés, avaient la réputation de mettre à la disposition de l'utilisateur une conception rapide, fiable et simple. Les progrès technologiques ont permis d'accéder à des matrices logiques programmables de plusieurs millions de portes. Cette complexité actuelle reste toutefois gérable et permet la réalisation d'applications très performantes moyennant une bonne connaissance des ressources offertes et le respect d'une méthodologie de conception.

La Figure 2-1 montre l'architecture simplifiée d'un FPGA Xilinx (allant de la famille Virtex II Pro jusqu'à Virtex V). Les principales caractéristiques de ces familles sont :

- Complexités allant de 1500 à plus de 8 millions de portes
- Faible consommation
- Grande souplesse d'utilisation des entrées/sorties avec adaptation d'impédance et configuration en mode différentiel
- Fonctions mémoire (distribuée et blocs de Ram)
- Dispositifs de gestion des horloges (DCM)
- Multiplieurs câblés et microprocesseurs intégrés dans certains composants.

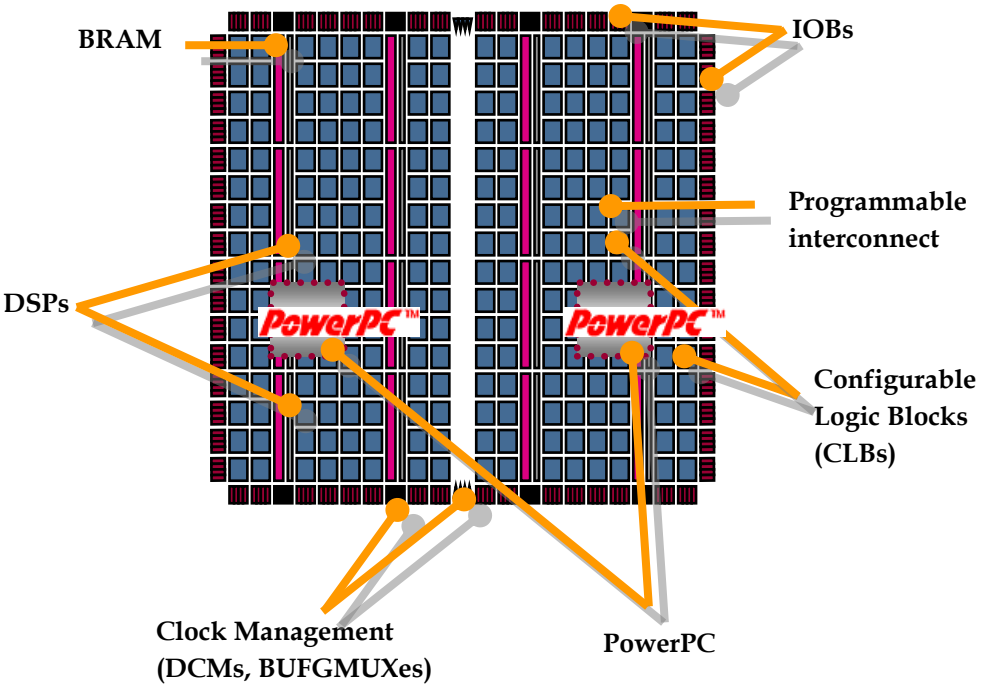


Figure 2-1. Architecture d'un FPGA Virtex [Xili.-08]

Un des modules de base du FPGA, à savoir le bloc logique configurable (ou CLB), est lui-même constitué de slices. La Figure 2-2 illustre l'architecture simplifiée d'une slice.

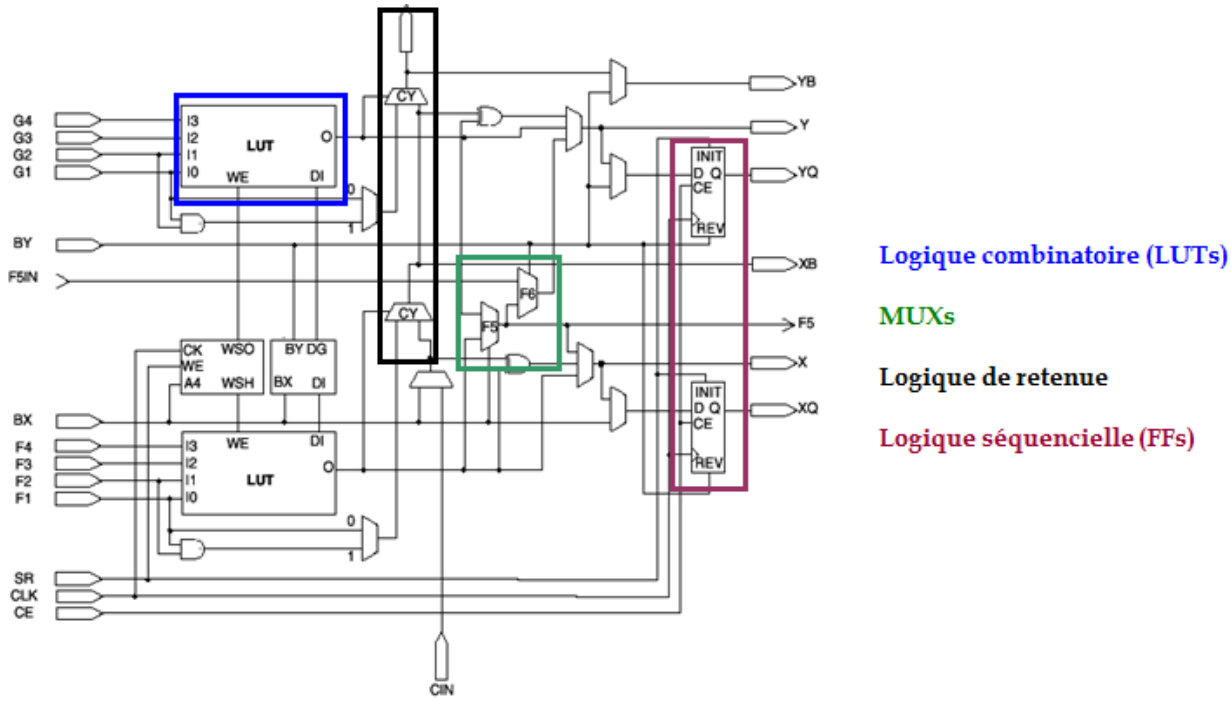


Figure 2-2. Architecture simplifiée d'une slice [Xili.-08]

La logique combinatoire est implantée grâce aux LUT (Look-Up Table) contenues dans chaque slice. Ces LUTs peuvent également être configurées comme des éléments de mémoire synchrone, simple ou double-port (de 16 à 64 bits en fonction du composant), ou encore comme registres à décalage. Il existe donc trois modes de configuration de ces LUT. Le fonctionnement en mode combinatoire est obtenu en lisant le contenu pointé par les signaux d'entrée, comme illustré dans la figure 2.3.

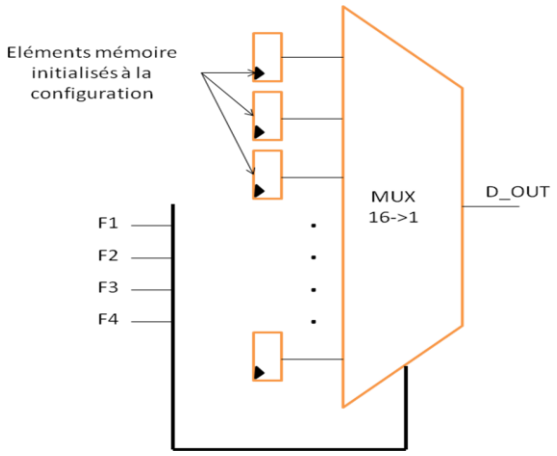


Figure 2-3. Configuration en mode combinatoire

Autrement dit, les LUTs sont des mémoires dont le contenu est initialisé lors de la configuration du FPGA. De ce fait, elles permettent à l'utilisateur d'en disposer en mode «élément mémoire» dans chacun des slices si nécessaire (Figure 2.4).

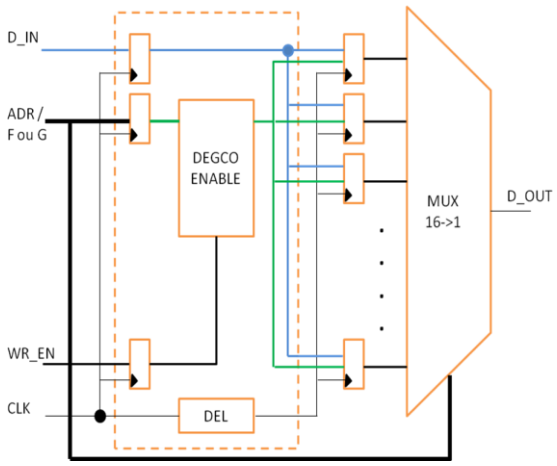


Figure 2-4. Configuration en mode "élément mémoire"

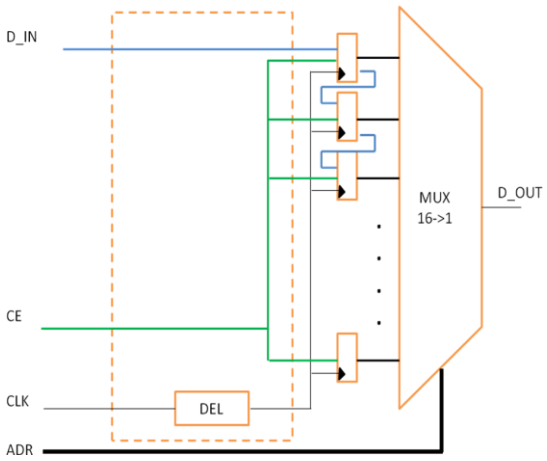


Figure 2-5. Configuration en mode "registre à décalage"

La Figure 2.5 décrit le mode de configuration particulier en registre à décalage de longueur programmable jusqu'à 16 bits. Par ailleurs, une logique supplémentaire utile pour la réalisation

des fonctions arithmétiques est disponible dans chaque slice. Grâce à ces éléments, et au style d'écriture adapté, des modules de type accumulateur chargeable en addition/soustraction pourront être implantés à raison de 2 bits par slice.

Les bascules dans chaque slice ont aussi des caractéristiques importantes pour le concepteur. En particulier, elles sont initialisées systématiquement à la mise sous tension (par défaut à la valeur '0'), et peuvent être utilisables indépendamment de la logique combinatoire disponible dans le même slice. En outre, chaque bascule bénéficie de broches de contrôle. Une entrée dédiée de validation de l'horloge (Clock Enable) permet d'activer ou de suspendre le fonctionnement de chacune des bascules individuellement, et ceci sans avoir à insérer de la logique combinatoire sur le chemin de l'horloge. La polarité des signaux d'horloge, de «Clock Enable », de set et reset est programmable pour chacune des bascules. Autrement dit, ces signaux peuvent être individuellement actifs au niveau haut ou au niveau bas. Et la compréhension de ces signaux-là est très importante pour la reconfiguration des bascules.

II.1.1.1. FPGA Virtex-II Pro

La plateforme Virtex utilisée pour une bonne partie de nos études est basée sur un dispositif XC2VP30 Virtex II Pro (V2P), qui intègre un processeur PowerPC 405 dont la fréquence est 100 MHz. Le lien de communication avec le PC hôte est un lien série dont la transmission de données peut être configurée entre 2400 et 115200 bits/s. Dans l'architecture V2P, le bitstream stocké dans la mémoire de configuration est divisé en unités élémentaires appelées frame. La frame est un ensemble de bits de configuration similaires sur toute une colonne du dispositif. La taille de la colonne dépend de la taille globale du FPGA.

Les CLBs de la famille Virtex II Pro sont composés de 4 slices qui contiennent la logique séquentielle et combinatoire, ces slices communiquant à travers la switch Matrix ou les fast Connects (voir Figure 2-6).

Dans les CLBs il y a deux sortes de Slices :

- Left-hand SLICEM (Memory) : dans ces slices les LUTs peuvent être configurés comme étant des éléments de mémoire synchrone, ou encore comme des registres à décalage de 16 bits.
- Right-hand SLICEL (Logic) : dans ces slices les LUTs sont seulement utilisées comme logique combinatoire.

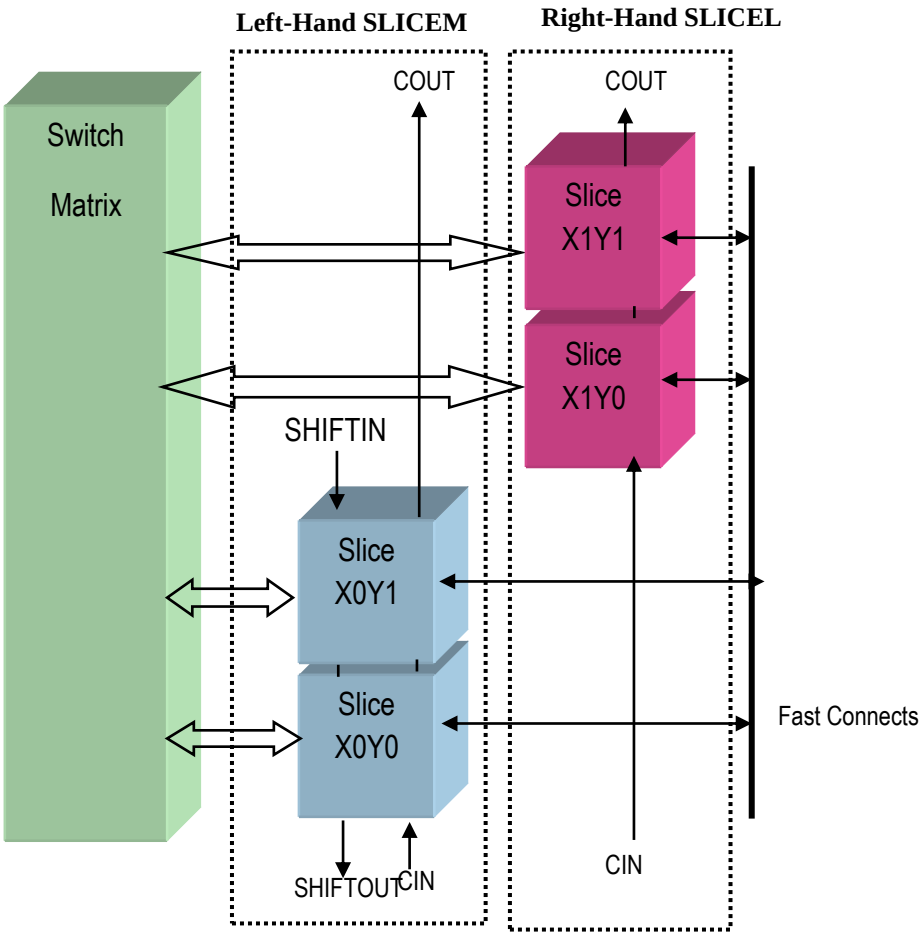


Figure 2-6. Architecture d'un CLB du FPGA Virtex II Pro [Xili.-08]

Xilinx fournit des IPs soft écrites en langage VHDL qui permettent d'exploiter le PowerPC405 et de lui fournir tous les périphériques nécessaires à son fonctionnement. Les premières IPs à implémenter autour du processeur sont le PLB « Processor Local Bus » et le «On Chip Memory Controller ». Le PLB est un bus de communication sur lequel viennent se greffer d'autres IPs, quant au «On Chip Memory Controller » il permet de connecter le PowerPC405 avec les Bram's du FPGA ou bien avec des RAMs externes où sera logé le code à exécuter. L'interface OPB HWICAP tourne à une fréquence de 100 MHz et permet de gérer les endo-reconfiguration dynamiques.

II.1.1.2. FPGA Virtex-V

La plateforme Virtex V utilisée est basée sur un dispositif Xilinx Virtex XC5VLX110T V (V5). Ce composant bénéficie d'une technologie de fabrication CMOS plus agressive par rapport au précédent (65nm au lieu de 0.13um/90nm). Toutefois, le microprocesseur n'est pas disponible en dur dans le dispositif, et le seul processeur embarqué est un cœur synthétisable, à savoir le MicroBlaze, qui doit être mis en œuvre dans la logique configurable. Dans cette architecture,

l'unité de reconfiguration élémentaire reste la frame, comme dans l'architecture V2P. Un comparatif détaillé entre les deux architectures est présenté dans l'annexe A.

Le processeur MicroBlaze fonctionne à 125 MHz, une fréquence légèrement plus élevée que le PowerPC dans la plate-forme V2P et le lien de communication avec le PC hôte est une liaison série similaire à celle du dispositif V2P. La configuration utilise un câble de programmation USB-JTAG. L'interface HWICAP XPS peut désormais fonctionner à 125 MHz.

En ce qui concerne la logique combinatoire, les LUTs sont passées de fonctions logiques à 4 entrées à des fonctions logiques à 6 entrées. Par ailleurs une CLB est composée désormais de 2 slices qui contiennent chacune 4 LUTs et 4 bascules comme nous pouvons le voir dans la figure 2-7.

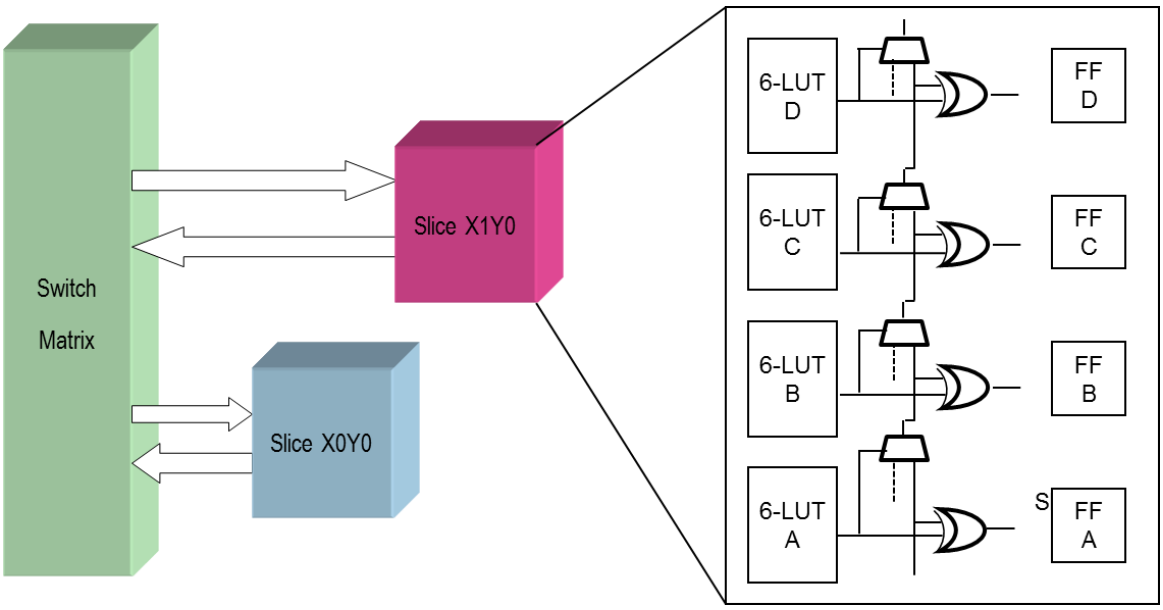


Figure 2-7. CLB d'un FPGA Virtex V

II.1.2. Les mécanismes de reconfiguration partielle

La reconfiguration d'un FPGA peut nécessiter un temps assez long, ce qui peut constituer pour notre utilisation une limitation des techniques basées sur le prototypage, surtout si de nombreuses reconfigurations sont nécessaires pour réaliser l'injection des fautes. Cependant, les FPGA Virtex permettent de faire des reconfigurations partielles, ce qui signifie qu'un bitstream réduit reconfigure une petite partie des composants internes. Cette reconfiguration est dite dynamique (DPR- (Dynamic Partial Reconfiguration) si elle est effectuée pendant le fonctionnement du FPGA. En utilisant cette possibilité, seule une partie du réseau doit être reconfigurée lorsque des modifications sont requises, ce qui conduit à des gains de temps

notables lorsque seules quelques différences existent entre deux configurations successives, ce qui est le cas pour des injections de fautes. Pour les FPGA Xilinx, il existe deux méthodes documentées pour la réalisation du DPR : DPR modulaire ou par la différence **[Upeg.-06][Xili.-04-3]** :

- La DPR modulaire (Module Based Flow) permet au concepteur de diviser le système en modules. Pour chaque module, le concepteur doit générer un bitstream de configuration à partir de la description HDL en utilisant la synthèse, le mapping et les procédures de routage, et ce indépendamment des autres modules.

  Un bitstream initial complet doit par ailleurs être généré, et à partir de là, les bitstreams partiels pour chaque partie reconfigurable sont générés.

- La DPR par la différence **[Xili.-07-1]** oblige le concepteur à éditer les changements bas-niveau. En utilisant FPGA Editor qui est un outil d'édition bas niveau, le concepteur peut changer la configuration de plusieurs types de composants, tels que les LUT, les RAMs, les Flip Flops, les standards d'entrée/sortie… Après l'édition de ces changements les bitstreams partiels sont générés contenant seulement la différence entre les deux versions du circuit.

Dans la plupart des cas où les tâches de reconfiguration sont réparties entre un PC et le FPGA, la bande passante entre le PC et la carte devient le principal facteur limitant. Il serait donc plus intéressant que le pilotage des campagnes d'injections de fautes soit fait par le processeur embarqué dans FPGA (endo-configuration), en utilisant les capacités internes de configuration avec le port ICAP (Internal Configuration Access Port), afin d'optimiser le temps de traitement et de minimiser la communication entre le PC hôte et le FPGA. Les composants à utiliser sont alors :

- HWICAP : **[Xili.-04-1]** est le module qui permet au PowerPC de lire et de modifier la configuration mémoire interne du FPGA par le biais de l'ICAP au moment de l'exécution, ce qui permet à l'utilisateur d'écrire un programme pour le processeur embarqué qui modifie la structure et la fonctionnalité du circuit. La version initiale du HWICAP inclut le support de la lecture des ressources et la modification des composants des CLBs. En outre, il permet de télécharger un bitstream partiel généré par ISE 8.2 ou PlanAhead 8.2, le bitstream partiel est chargé de la mémoire locale à l'ICAP.

- ICAP : **[Xili.-04-2]** est le module fondamental utilisé pour effectuer la reconfiguration interne des FPGA Virtex à partir de la version V2P. L'ICAP est située dans le coin inférieur droit du FPGA. Il est utilisé pour accéder à la configuration du FPGA ainsi que'aux registres de transfert des données de configuration en utilisant le protocole SelectMAP. Une description plus détaillée est présentée dans l'annexe B.

II.1.3 Protection des Virtex

Dans cette section nous allons présenter les protections disponibles dans les FPGAs Virtex, en mettant en relief ce que les récentes versions ont apporté de plus en matière de sécurisation.

II.1.3.1. Protection du Virtex II Pro

La logique programmable permet de mettre à jour à distance le FPGA d'une manière sûre et fiable, par exemple pour corriger le circuit ou effectuer des opérations de mise à jour. Les bitstreams des Virtex II Pro peuvent être sécurisés par la technologie Xilinx SecureChip, implémentée avec le standard triple DES (TDES). Le bitstream encrypté peut être téléchargé et ensuite décrypté pendant son chargement dans le FPGA à travers une clé obtenue pendant la fabrication du système. Bien que cette protection soit utile pour lutter contre des modifications non autorisées ou réduire les risques de clonage, elle n'apporte aucune robustesse face aux perturbations.

II.1.3.2. Protection du Virtex-V

Dans les versions récentes des FPGA Virtex, Xilinx a non seulement changé l'algorithme de chiffrement du bitstream lors de la configuration du dispositif mais a aussi rajouté des mécanismes de protection sur le bitstream chargé.

- **Chiffrement du bitstream**

La protection utilisée pour le Virtex-V est semblable à celle utilisée dans le Virtex-II Pro, mais le TDES a été remplacé par l'algorithme AES. Le système AES des FPGAs Virtex-V consiste en un logiciel de chiffrement du bitsream et un bitstream de décryptage sur puce avec une mémoire dédiée pour mémoriser la clé de chiffrement. L'utilisation du logiciel Xilinx ISE 8.2, permet à l'utilisateur de générer la clé de chiffrement et le bitstream chiffré.

Pour faire des injections de fautes, nous devons connaître avec précision certaines informations, telles que la localisation des bits de contenu et de configuration des bascules et des LUTs dans le bitstream. Il faudra donc désactiver le chiffrement.

❖ **Correction de SEU intégrée**

Les FPGAs Virtex V fournissent aussi un composant spécialisé, appelé Frame ECC (code correcteur d'erreurs), pour la détection et l'identification des erreurs simples et doubles dans les frames de données. Pour chaque frame lue dans la mémoire de configuration, le module Frame ECC calcule le code de Hamming ainsi que la parité globale pour les données de la frame, et compare les résultats obtenus avec ceux stockés dans la mémoire de configuration. En se basant sur cette comparaison, le module Frame ECC peut produire 4 types d'indications suivant le syndrome qu'il génère :

- Aucune erreur: syndrome [11] =0, syndrome [10:0] =0
- Un bit d'erreur: syndrome [11] =1, syndrome [10:0]≠0
- deux bits d'erreur: syndrome [11] =0, syndrome [10:0] ≠ 0
- Erreur dans le bit de parité : syndrome [11] =1, syndrome [10:0] =0

Le module ECC ne gère pas plus de deux bits erronés par frame. Les FPGAs Virtex V mettent en place un nouveau dispositif qui effectue en continu et en tâche de fond des readbacks des données de configuration du circuit. Cette fonctionnalité peut être utilisée en complément à la fonction de Frame ECC pour les opérations avancées telles que la correction des SEUs. Une fois activée, la logique de configuration dédiée fait des readbacks en permanence pour vérifier le CRC du contenu de la mémoire de configuration. Si le CRC indique une erreur, alors la Frame ECC est utilisée pour déterminer s'il s'agit d'un seul bit fauté (erreur corrigible) ou plusieurs bits (erreur incorrigible). Pendant la première configuration, la valeur CRC est sauvegardée en tant que valeur de référence pour les comparaisons ultérieures.

Quand une incohérence du CRC est détecté, une erreur est signalée et l'indicateur d'erreur reste levé jusqu'à la comparaison suivante si l'erreur n'a pas été corrigée entre temps. Si aucune nouvelle valeur CRC de référence n'est calculée, l'indicateur d'erreur reste également levé.

Lorsque l'utilisateur accède à la logique de configuration par le biais d'une commande ICAP, JTAG, ou un SelectMap, le readback s'arrête automatiquement sans affecter l'accès à la configuration, et l'erreur est effacée. Lorsque l'utilisateur termine cette opération, les readbacks reprennent automatiquement.

Une erreur de logique causée par un SEU pourrait également entraîner une corruption prolongée des données. Cela pourrait être tolérable si la réparation est effectuée rapidement et n'a pas d'effets durables. Cependant, une erreur de logique qui affecte les chemins de contrôle ou les états de fonctionnement du circuit peut avoir des effets durables. Une solution consiste à

effectuer une réinitialisation de tous les circuits critiques en cas de détection d'une erreur, et revenir au fonctionnement normal une fois que l'erreur a été corrigée.

- **Stratégies pour le traitement des SEUs de configuration**

Il existe 3 stratégies différentes pour la correction des SEUs sur les FPGAs Virtex V. Ces stratégies peuvent être utilisées séparément ou combinées.

1. **La maintenance programmée:** La stratégie de maintenance programmée (ou préventive) exploite toutes les opportunités disponibles pour reconfigurer le dispositif pendant son fonctionnement normal et en particulier toutes les parties non utilisées activement dans le système. Dans cette stratégie, il ne s'agit pas de déterminer si un SEU s'est produit ou pas ; la reconfiguration permet de réparer les corruptions qui ont eu lieu. Idéalement la reconfiguration est faite tous les MTBF du circuit. Cette stratégie peut ne pas être adéquate pour les applications avioniques, mais elle peut être couplée à une autre méthodologie de détection et de correction des SEUs pour améliorer la fiabilité du système.
2. **La maintenance d'urgence :** La stratégie de maintenance d'urgence consiste à forcer une reconfiguration en cas de détection d'un SEU, plutôt que de tenter d'identifier et de corriger l'erreur. Bien que cela puisse être considéré comme une approche sûre, elle nécessite une planification supplémentaire pour assurer un redémarrage normal. La clé de cette stratégie réside dans la détection d'un SEU dans les cellules de configuration. Pour que cela soit possible un bloc intégré scanne continuellement les cellules de configuration du composant et calcule la valeur du CRC sur 32 bits. Si la valeur du CRC calculée par le scan d'urgence diffère de la valeur de référence calculée immédiatement après la configuration du FPGA, un changement de la configuration s'est produit et la perturbation est signalée en entraînant la broche INIT_B à l'état Bas. La stratégie de maintenance d'urgence est bénéfique lorsque la conception peut tolérer un défaut de configuration pendant la durée de relecture du CRC, plus le temps de réaction nécessaire pour initialiser la reconfiguration.
3. **Stratégie de correction :** Comme pour la maintenance d'urgence, le dispositif de CRC du Virtex V est mis à profit pour faciliter la détection des changements dans les cellules de configuration. Une fois l'erreur détectée elle doit être par la suite localisée, l'emplacement exact est déterminé par le code correcteur ECC, présenté un peu plus haut dans ce chapitre, via le SYNDROME. En principe, la correction est simple. La frame de 1.312 bits corrompue est lue dans une mémoire tampon, le bit altéré est isolé en

interprétant le syndrome 12-bit, et il est alors corrigé. La correction inverse simplement le bit inversé par le SEU.

Enfin, la trame corrigée est réécrite dans les cellules de configuration. Une fois qu'une erreur a été corrigée, la relecture du circuit CRC réinitialise les signaux d'erreur. Le scan suivant du dispositif permet de confirmer que la configuration du FPGA est rétablie. Bien que les bits ECC permettent de corriger les erreurs d'un seul bit, il n'est pas possible de corriger des erreurs de bits multiples dans la même frame (1312 bits). Les erreurs multiples peuvent être réparées, à condition que les erreurs soient dans des frames différentes ce qui correspond à un MCU.

II.2. AT40K

II.2.1. Présentation générale du FPGA AT40K

La troisième plateforme utilisée pour notre étude est basée sur un composant Atmel AT40KEL40, fabriqué en technologie CMOS 0.35µm, donc nettement moins fine.

Le cœur de l'architecture AT40K d'Atmel réside dans un tableau symétrique de cellules identiques, comme le montre la figure 2-8. Ce tableau est continu sur le long du FPGA, excepté sur les bus répétiteurs (verticaux et horizontaux) qui sont espacés de quatre cellules logiques (voir figure 2-9). Ces répétiteurs assurent la continuité électrique des signaux.

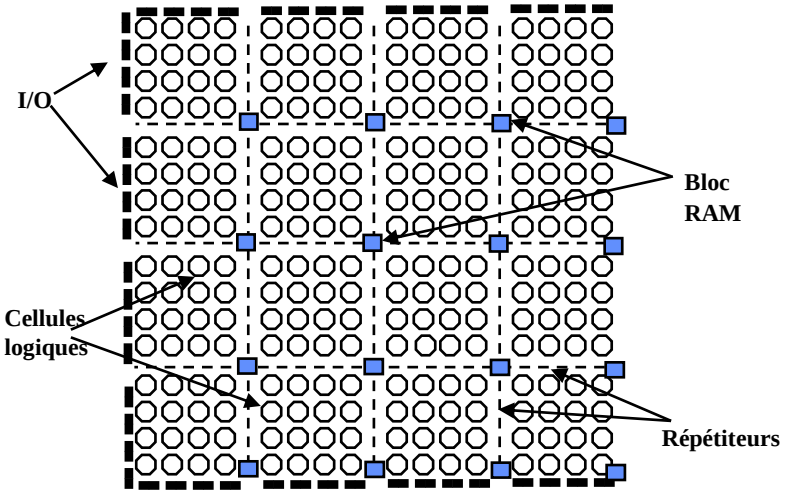


Figure 2-8. Vue d'ensemble sur l'architecture de l'AT40K [Atme.-01-1]

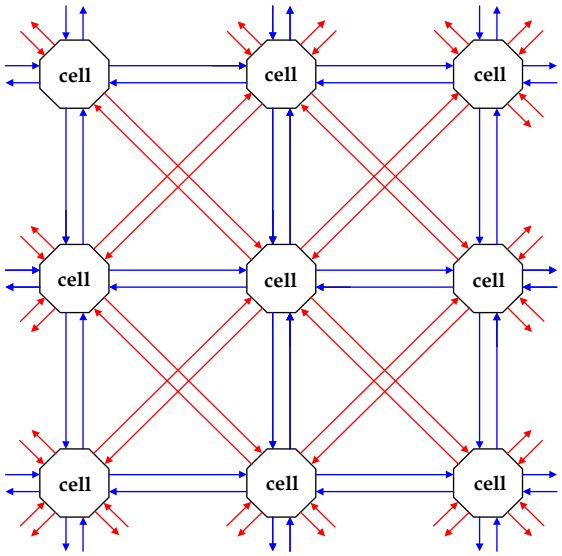


Figure 2-9. Connexions entre les cellules logiques [Atme.-01-1]

Ce FPGA comporte aussi des blocs RAM de 32 mots, aux intersections des colonnes et des lignes des répétiteurs.

La cartographie mémoire de l'AT40K peut être représentée par 4 dimensions comme le présente la figure 2-10 :

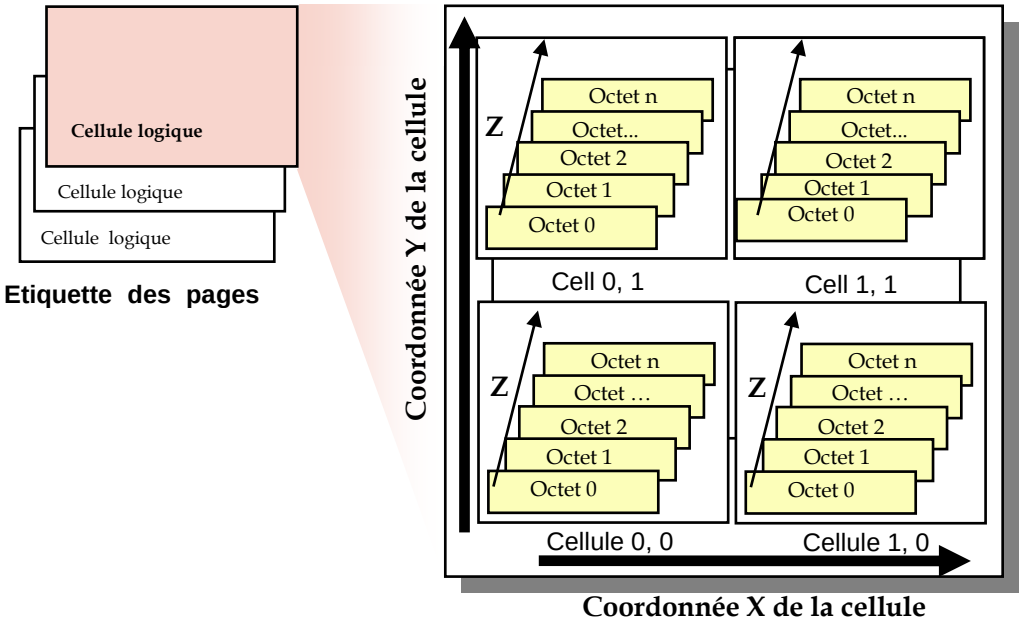


Figure 2-10. Cartographie de la mémoire logique de l'AT40K [Atme.-01-2]

- L'étiquette définit la page qui va être accédée ; une page représente un regroupement de cellules logiques.
- X et Y définissent la localisation d'une cellule logique dans une page donnée du FPGA.
- Z représente l'adresse d'un octet donné dans une cellule logique.

II.2.2. Configuration de l'AT40K

Il existe trois modes de configuration principaux pour l'AT40K :

- Mode Série maître : Le FPGA se configure tout seul dans ce mode. Après un reset de la mise sous tension, il charge les données de configuration contenues 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.

Tableau II-I Modes de configuration d'un FPGA AT40K

Mode	Description	M2	M1	M0	Horloge	Type de donnée	Remarques
0	Série Maitre	0	0	0	Output	Serial	AutoConfiguration, EEPROM série
1	Série Esclave	0	0	1	Input	Serial	Microprocessor, EEPROM série
7		1	1				
2	Paralléle Esclave	0	1	0	Input	8 ou 16 bits	Microprocessor, Parallèle EEPROM
4	RAM Synchronous	1	0	0	Input	8 ou 16 bits	Port parallèle microprocesseur, 24 bit d'adressage en entrée
6	Parallel UP Esclave	1	1	0	Input	8 ou 16 bits	Parallèle EPROM, 20 bits d'adressage en sortie

❖ **Reconfiguration par adressage direct:**

La famille AT40K supporte l'écriture et la lecture des données à partir de la configuration SRAM du FPGA à travers le mode configuration synchrone RAM qui est le mode 4.

Cette interface ne requiert pas de machine à état durant le processus de téléchargement du bitstream.

Le processus de chargement commence quand CON (broche de l'enclenchement de la configuration) et CS0 (chip select de la configuration du FPGA) sont à l'état bas. L'interface de configuration est demandée, et elle consiste en une horloge, un signal de sélection Ecriture/Lecture, un flag d'erreur, ainsi qu'un bitstream « partiel », composé de 24 bits d'adresse et 8 bits de données (voir figure 2-11).

La logique de configuration peut réclamer 8 autres bits de données, via un bit de contrôle, dans le cas où les données sélectionnées sont trop larges.

L'ensemble de la mémoire de configuration de la SRAM est accessible en écriture et en lecture par l'utilisateur à travers ce mode.

Que ce soit pour un cycle d'écriture ou de lecture, les données, l'adresse et le signal Ecriture/Lecture sont fournis simultanément au dispositif, sauf que dans le premier cas l'écriture dans la configuration de la SRAM se produit sur le front descendant de l'horloge juste après le premier front montant de cette dernière. Alors que pour un cycle de lecture les données sont disponibles au deuxième front montant de l'horloge.

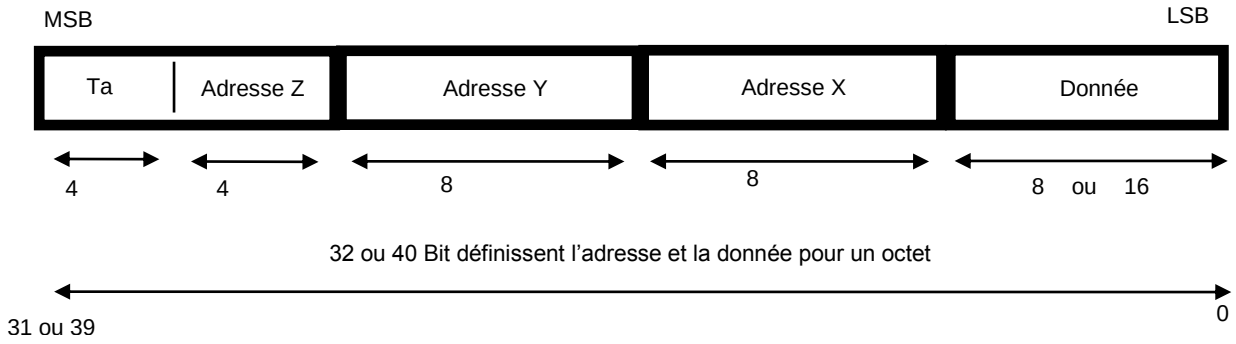


Figure 2-11. Composition d'un bitstream partiel [Atme.-01-2]

La reconfiguration partielle peut être réalisée en modifiant une ou quelques adresses seulement dans la mémoire de configuration. L'écriture de données dans une structure n'a aucun impact sur toutes les autres structures, ce qui signifie que moins de bits peuvent être rechargés par rapport aux approches de reconfiguration basées sur les frames.

Les principales limitations de ce composant sont l'absence de processeur embarqué et la petite capacité en logique configurable qui doit être totalement réservée pour notre application au

circuit sous évaluation. Par conséquent, pour cette plateforme, la configuration devra être totalement pilotée par un PC hôte. La taille réduite de la bande passante devrait pouvoir être compensée par le nombre limité de bits à envoyer au dispositif pendant la reconfiguration. Nous ne pouvons pas donner plus de détails sur ce mode, compte tenu de la clause de non divulgation que nous avons signé avec ATMEL afin de protéger les utilisateurs des FPGA AT40K.

II.2.3. Protection de l'AT40K

La famille des FPGAs AT40K intègre une fonction de contrôle Checksum. Au cours d'un chargement de configuration, une somme de contrôle est calculée après chaque mot (de 8 ou 16 bits) du bitstream chargé dans le FPGA. Pendant le chargement, l'utilisateur peut écrire dans une série de registres dans une fenêtre spéciale connue sous le nom Checksum Page.

Après power-on-reset, le contrôle et les graines sont remis à zéro. Juste avant le début d'un chargement de configuration, la valeur de départ est chargée dans le checksum. Chaque fois que le Evaluate Checksum register(s) est accédé en écriture, une nouvelle somme de contrôle est calculée. Si la nouvelle valeur n'est pas égale à "FF" (ou "FFFF" pour 16 bits), une erreur est signalée (la broche INIT est remise à 0).

Le chargement du bitstream ne peut pas se terminer avec une erreur de la fonction de somme de contrôle.

II.3. Stratix IV

II.3.1. Présentation générale du FPGA Stratix IV

A la demande d'un partenaire industriel, nous nous sommes aussi intéressés à la famille Stratix IV d'Altera. Ces FPGAs en technologie 40 nm offrent une grande densité et une haute performance avec une faible consommation d'énergie.

Les blocs logiques sont nommés « LABs » (Logic Array Blocs). Ils sont composés par des modules logiques adaptatifs (ALMs - Adaptative Logic Modules) qui sont configurables pour implémenter des fonctions logiques, arithmétiques et des registres.

Chaque LAB est constitué de 10 ALMs, diverses chaines de retenue, des chaines arithmétiques partagées, des signaux de contrôle de LAB, des interconnexions locales et des lignes de connexion des chaines des registres. La figure 2-12 et la figure 2-13 montrent les interconnexions entre ces éléments.

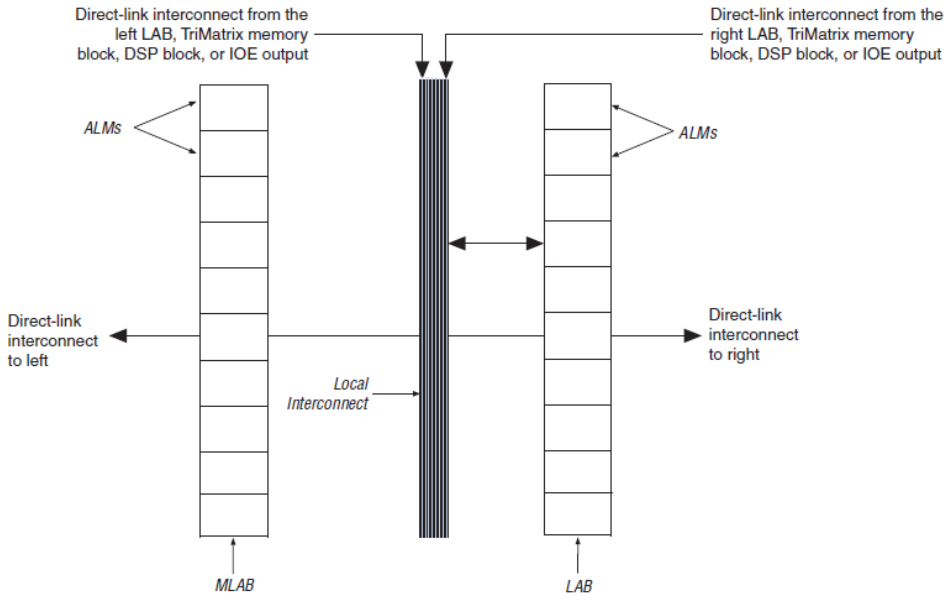


Figure 2-12. Eléments de base du Stratix IV [Alte.-11]

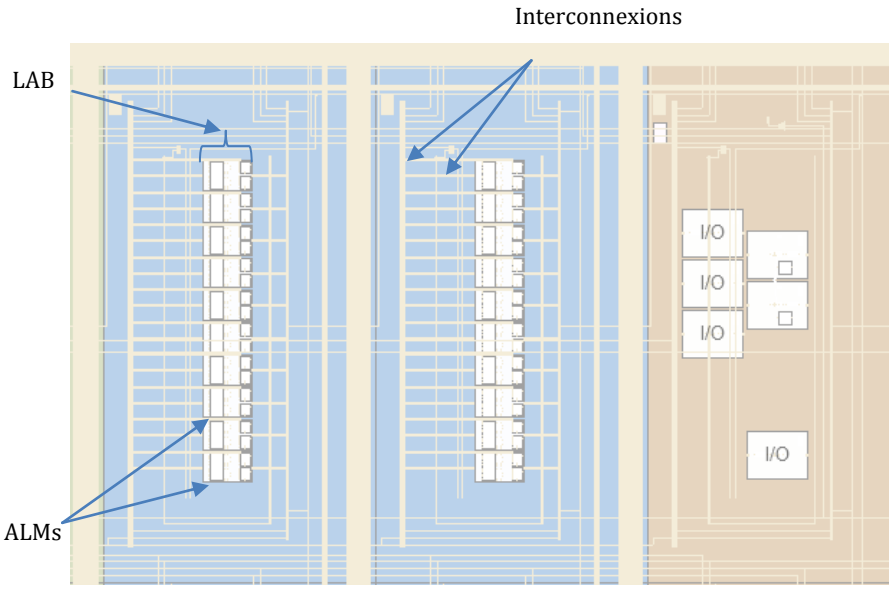


Figure 2-13. Architecture du Stratix IV

Chaque LAB de Stratix IV possède un bloc mémoire dérivé nommé MLAB qui rajoute des capacités mémoires aux LUTs. MLAB et LAB coexistent toujours en paire dans toutes les familles des FPGAs Stratix IV.

Chaque ALM contient une variété de ressources à base de LUTs et peut être divisé entre 2 LUTs adaptatives combinatoires (ALUTs) et 2 registres.

Outre les ressources adaptatives à base de LUT, chaque ALM contient 2 registres programmables, 2 additionneurs complets dédiés, une chaine de retenue, une chaine arithmétique partagée et une chaine de registre.

Pour les fonctions combinatoires, le registre est contourné et la sortie de la LUT est reliée directement à la sortie de l'ALM.

Une présentation du bloc ALM est donnée par les figures 2-14 et 2-15.

En permettant aux ALUTs adjacentes de partager la logique et les entrées, l'ALM réduit les ressources logiques nécessaires à une fonction donnée, ainsi que le nombre de niveaux logiques nécessaires dans un chemin critique donné. En outre, deux fonctions indépendantes peuvent être insérées dans une seule ALM, ce qui réduit les besoins en ressources logiques.

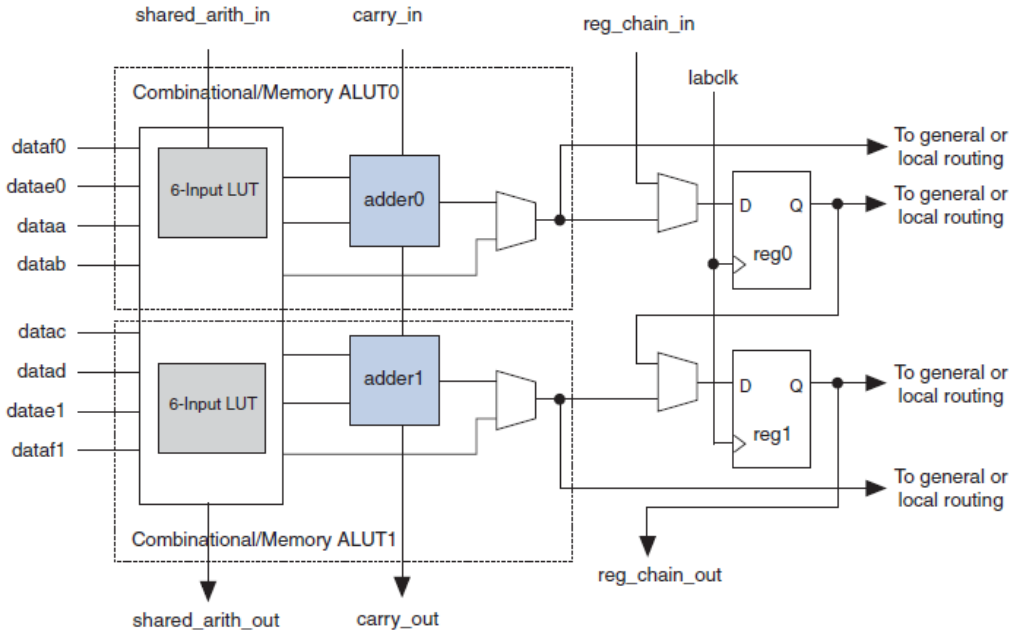


Figure 2-14. Composition d'un ALM [Alte.-11]

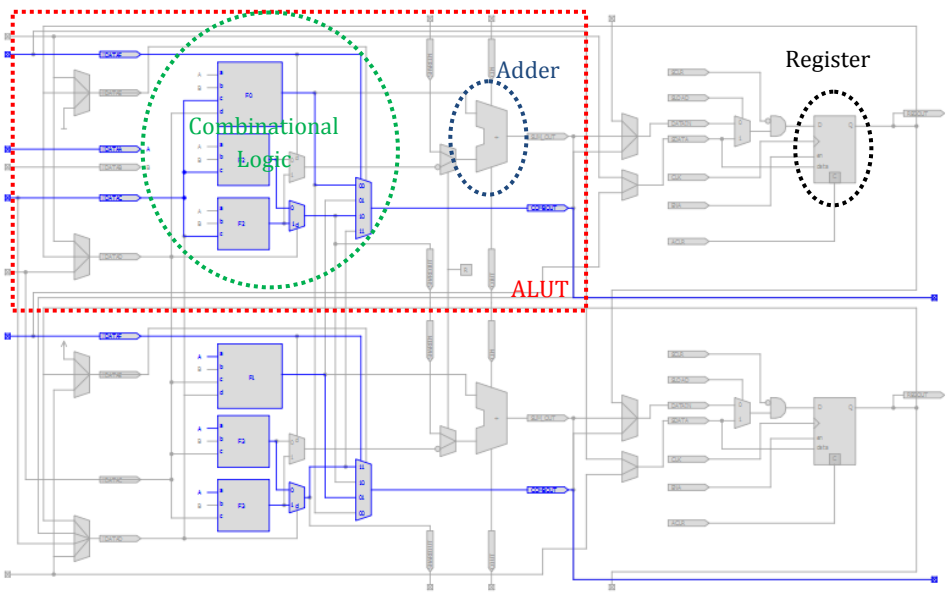


Figure 2-15. Architecture détaillée d'un ALM

❖ Modes d'opération des ALMs

Chaque ALM du Stratix IV opère dans un des modes suivants: Normal, Extended LUT, Arithmetic, Shared Arithmetic et LUT-register. Chaque mode utilise les ressources de l'ALM différemment.

L'outil logiciel Quartus II choisit automatiquement le mode approprié pour les fonctions communes à savoir les compteurs, les additionneurs, les soustracteurs et les autres fonctions arithmétiques. Les 2 modes les plus utilisés sont le mode normal et le mode étendu.

❖ *Mode normal*

Ce mode est approprié pour les fonctions combinatoires. Dans ce mode, jusqu'à 8 entrées de données de l'interconnexion locale du LAB sont des entrées pour la logique combinatoire.

Ce mode permet d'insérer 2 fonctions dans une seule ALM ou bien une seule fonction avec 6 entrées au maximum. L'ALM supporte certaines combinaisons de fonctions complètement indépendantes et des combinaisons variées de fonctions ayant des entrées communes.

La figure 2-16 montre les combinaisons des LUTs supportées dans le mode normal.

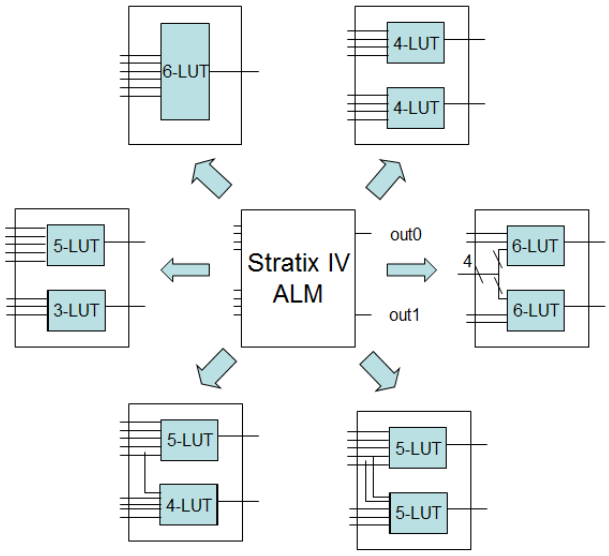


Figure 2-16. Les différentes combinaisons d'implémentation d'un ALM

❖ *Mode étendu*

Ce mode est utilisé pour implanter un ensemble spécifique de fonctions à 7 entrées.

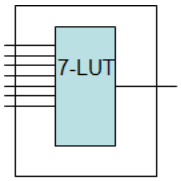


Figure 2-17. ALM en Extended mode

II.3.2. Reconfiguration d'un FPGA Stratix

La reconfiguration partielle utilisée pour les autres plateformes étudiées n'est pas encore supportée par les outils logiciels d'Altera. Il est possible de reconfigurer partiellement le Stratix V pendant que le reste du dispositif continue de s'exécuter, cependant cela n'est pas possible avec le dispositif de notre étude qui est le Stratix-IV.

Les Stratix IV GX et GT permettent la reconfiguration dynamique totale, mais ces dispositifs ne permettent pas de faire des readback des fichiers de reconfiguration, donc bien évidemment les

informations sur le circuit ne peuvent pas être copiées. Par conséquent, il n'est pas possible non plus pour ces composants de vérifier les injections de bit-flips pour réaliser des campagnes.
Une technique d'injection de fautes est possible, qui consiste en l'utilisation de l'instruction JTAG EDERROR_INJECT qui permet de contrôler le contenu du registre JTAG d'injection de fautes pour que la valeur des bits du readback soit modifiée à un endroit particulier de la première frame de données. Un fichier .jam peut être utilisé pour automatiser le processus. Toutefois, l'impossibilité de vérifier le résultat rend cette plateforme peu adaptée à ce genre d'utilisation, malgré sa configuration par SRAM.

II.3.3. Protection d'un FPGA Stratix

En ce qui concerne la protection, tous les FPGAs Stratix sont dotés d'un système de contrôle de redondance cyclique (*Cyclic Redundancy Check* - CRC) qui permet de détecter les bit-flips dans n'importe quel bit de la mémoire de configuration (*Configuration Random Access Memory* - CRAM) et d'indiquer la présence d'une erreur via la broche CRC_ERROR.
Pendant la reconfiguration du FPGA, le mécanisme de détection des erreurs CRC permet de détecter les corruptions au niveau du bitstream de configuration pendant que ce bitstream est transféré d'un dispositif externe vers le FPGA. Avec le mode utilisateur, le dispositif de détection des erreurs CRC détecte les SEUs et détermine leur type et leur localisation. La vérification CRC est contrôlée à travers l'outil de conception Quartus II.
Outre la capacité de détection des erreurs, les Stratix V (et versions ultérieures) permettent également le support du scrubbing interne qui permet de corriger les soft erreurs sans l'outil de reconfiguration.
Cependant, le support du scrubbing interne n'est pas disponible sur la plateforme Stratix IV choisie pour notre étude de cas. La vérification CRC reste toutefois possible pour détecter les SEUs et déclencher la reconfiguration du dispositif en cas d'erreur. Dans un contexte ou les soft erreurs ne sont pas très fréquents et de multiplicité réduite, le mécanisme CRC peut être suffisant pour atteindre le niveau attendu de protection pour l'application.

II.4. Conclusion

Ce deuxième chapitre a permis de rappeler les caractéristiques générales des réseaux programmables, et de préciser celles de trois familles configurées par mémoire SRAM et qui seront les cibles de nos études : les FPGAs Virtex II Pro et Virtex V de chez Xilinx, l'AT40K de chez Atmel et le Stratix IV de chez Altera. Pour toutes ces plateformes, nous avons résumé la terminologie et les informations importantes pour la compréhension de nos travaux :

- Pour les FPGAs Virtex, les mécanismes de l'ICAP seront adoptés et exploités pour la reconfiguration. La mise en œuvre de l'environnement d'injection de fautes par reconfiguration sur les FPGAs Virtex sera détaillée dans le chapitre III.

  Par ailleurs, l'étude des protections intrinsèques du Virtex V a montré que seulement deux SEUs, dans les éléments mémoires de configuration, peuvent être détectés et qu'un seul SEU peut être corrigé. Dans le chapitre IV, nous proposerons une méthode de protection capable de détecter des erreurs multiples à faible coût, sur ce type de plateforme.
- Les FPGAs AT40K offrent un mode de reconfiguration par adressage direct. Ce mode peut offrir un temps de reconfiguration réduit par rapport aux autres plateformes, il sera donc testé dans le chapitre III.
- Pour les FPGAs Stratix, seule la protection CRC est disponible et permet aussi la détection seulement des SEU ; une méthode similaire à celle proposée sur Virtex sera présentée et validée dans le chapitre IV. Cette plateforme ne sera pas considérée comme plateforme d'injection de fautes, compte tenu des limitations identifiées pour ce type d'application.

Chapitre III - Evaluation des effets des perturbations dans les mémoires de configuration des FPGA SRAM

Dans la première partie de ce chapitre nous allons résumer les objectifs de nos environnements d'injection de fautes. Nous présenterons ensuite une méthode originale d'abstraction de motifs d'erreurs permettant la création d'une base de données issue d'attaques physiques réelles afin de réaliser des campagnes d'injection de fautes par émulation les plus réalistes possibles. Ces principes seront appliqués tout d'abord dans le cadre d'un environnement d'injection de fautes par reconfiguration partielle sur un type de FPGAs Virtex. D'autres plateformes sont ensuite comparées, notamment une plateforme basée sur le FPGA AT40K, en utilisant le mode d'adressage direct. Plusieurs exemples d'études de cas seront enfin résumés, afin d'illustrer les possibilités des plateformes implantées.

III. 1. Objectif

Notre étude s'intéresse à l'évaluation des effets des perturbations dans les mémoires de configuration des FPGA SRAM (ce qui inclut les perturbations dans les bascules utilisateur ou les blocs mémoires intégrés). Nous allons prendre en compte les modèles d'erreur présentés au chapitre I, essentiellement les erreurs transitoires et rémanentes. Les erreurs transitoires résident dans les éléments mémoire utilisateur (bascules ou blocs mémoire), tandis que les erreurs rémanentes surviennent dans les bits de configuration du FPGA qui définissent les fonctions et les interconnexions. Nous allons nous intéresser à ces types d'erreurs sous leurs différentes formes : bit-flips unique (SEU) ou multiple (MBF), cette dernière catégorie se divisant en deux familles, les MBUs et les MCUs.

Pour ce faire nous avons développé un premier environnement d'injection de fautes sur une carte avec un composant V2P. Cet environnement permet d'évaluer la robustesse du circuit à tester (DUT) selon le modèle d'erreurs défini pour une campagne. Cet environnement permet de modifier aussi bien les bits de configuration que les éléments de mémoire utilisateur. Il sera détaillé dans la partie III.3.

Au-delà de ces modèles génériques, nous avons cherché à évaluer l'effet de motifs d'erreurs réalistes, sans pour autant avoir accès systématiquement à des sources physiques de perturbation. Une méthodologie basée sur l'abstraction de motifs d'erreurs pré-caractérisés a donc été proposée. Cette méthodologie est présentée dans la section suivante.

III. 2. Méthode d'abstraction de motifs d'erreur physiquement réalistes

III. 2.1. Motifs d'erreurs

Lorsque des évaluations de robustesse d'une application donnée sont réalisées en utilisant des perturbations physiques (faisceau de particules, laser …), les durées d'expérimentation sont limitées à cause de la disponibilité et du coût des équipements. En conséquence, il n'est pas possible le plus souvent d'obtenir une évaluation complète de la robustesse car seules certaines parties du composant sont exposées aux erreurs.

Les campagnes d'injection de fautes en phase de conception permettent au contraire, en général, d'injecter beaucoup plus d'erreurs et de mieux répartir les positions et les instants où les erreurs sont injectées. Toutefois, elles sont habituellement effectuées sur la base d'un modèle d'erreur générique, le plus souvent le modèle SEU impliquant une seule inversion de bits à la fois dans un circuit. Dans certains cas, les erreurs sur plusieurs bits sont prises en considération, en supposant une valeur de multiplicité donnée et une distribution aléatoire. Ceci n'est pas forcément le plus proche de la réalité d'une perturbation (surtout en cas d'impact de particule ou d'attaque volontaire localisée).

Notre but est donc de mettre en place une méthodologie d'injection permettant de "rejouer" des scénarii réalistes de perturbation ou d'attaque. Afin d'illustrer nos motivations, nous pouvons utiliser des configurations d'erreurs provenant de données d'attaques physiques par laser sur un composant Xilinx V2, effectuées avec les bancs laser du CESTI-LETI du CEA Grenoble [Cani.-08]. Ces bancs laser 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 premier banc laser est un banc laser fibré et permet d'obtenir une taille de spot de l'ordre du diamètre extérieur de la fibre c'est-à-dire de l'ordre de 100 µm de diamètre. Le second banc laser est un banc microscope qui permet d'obtenir plusieurs tailles de spot puisque le banc 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.

Ces deux bancs laser sont constitués des mêmes équipements tels que des tables XYZ motorisées afin d'automatiser les déplacements lors d'une étude d'une zone du composant ou du composant entier. Ces tables XYZ peuvent être pilotées, soit par liaison série RS232, soit par un joystick et la précision de ces dernières est nettement inférieure au micromètre.

Les bancs laser possèdent des amplificateurs de puissance pour les diodes laser qui permettent de fournir la puissance nécessaire au déclenchement du faisceau laser. Cette puissance peut influer sur les effets des tirs lasers tout comme la durée de l'éclairement. Par ailleurs, selon l'effet désiré dans les zones actives du composant, il est possible de modifier plusieurs paramètres des lasers tels que la taille du spot (grossissement, fibre, etc…), la longueur d'onde de la diode laser, la focalisation du faisceau dont le réglage n'est pas quelque chose de simple ou la valeur de l'énergie émise. Tous les changements pouvant être faits sur le banc conduisent à de nouvelles conditions expérimentales.
Plusieurs campagnes de tirs laser ont été effectuées 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.

Tableau III-I Répartition des fautes dans les CLBs en fonction de la taille du faisceau laser et de l'incrément de déplacement entre deux tirs

	Spot	**40µm**		**8µm**	
	Incrément spatial	**20µm**	**10µm**	**20µm**	**10µm**
CLB	**Moyenne**	9,373	9,164	4,571	5,066
	Pourcentage	99,10%	99,15%	100%	100%
Logique	**Moyenne**	2,339	2,347	1,705	1,985
	Pourcentage	24,96%	25,61%	37,29%	39,18%
Interconnexion	**Moyenne**	6,528	6,764	2,829	3,025
	Pourcentage	74,32%	73,82%	61,88%	59,72%
Inconnues	**Moyenne**	0,068	0,052	0,038	0,056
	Pourcentage	0,72%	0,57%	0,83%	1,11%

Pour analyser les effets des injections de fautes dans la configuration de certains FPGA, un outil a été développé au laboratoire TIMA, l'outil SEFEA-ProD **[Ferr.-12-2]**. Cet outil utilise notamment des classes Java fournies par Xilinx, nommées JBits 3.0 qui permettent d'accéder à la configuration des composants de la famille Virtex II. Ces classes Java ne sont malheureusement plus actualisées pour les composants des familles plus récentes que la famille Virtex II. SEFEA-ProD ne modifie pas les fichiers de configuration mais compare un fichier de configuration, que nous appellerons dans la suite du manuscrit le **golden readback**, avec un fichier dans lequel des modifications ont pu être apportées par des perturbations. En comparant les différents fichiers

bits à bits, une analyse des modifications est faite en fonction des différents éléments des tuiles CLB et de la valeur initiale du bit. Une synthèse, en fonction du spot et du pas de déplacement, sur la moyenne et le pourcentage des éléments touchés par les bit-flips est présentée dans le tableau III-I. Comme décrit dans ce tableau, les injections de fautes par laser sur les CLBs induisent l'inversion de 4,8 bits à 9 bits en moyenne selon la taille du spot (8µm et 40µm respectivement). A chaque tir laser sur les CLBs, au moins un bit est forcé quelque soit la taille du spot utilisé, ce qui démontre la vulnérabilité de ce type d'éléments (ceci dépend bien sûr aussi de l'énergie utilisée et des autres paramètres expérimentaux). En ce qui concerne le nombre de bits touchés au maximum, nous notons une différence entre les deux tailles de spot. En effet, avec un spot de 8µm nous avons touché un maximum de 16 bits alors qu'avec le spot de 40µm nous avons modifié au maximum 34 bits par tir.

Une analyse plus fine de la répartition des fautes à l'intérieur des CLBs permet de déterminer à quelle catégorie appartiennent les bits modifiés : à un élément de logique, d'interconnexion ou à un élément "encore inconnu" c'est-à-dire sur lequel nous n'avons pas d'information précise. Cette analyse montre que les bit-flips sont répartis entre les interconnexions et la logique avec respectivement une moyenne de 66% et 34%. Même si les interconnexions paraissent à première vue plus sensibles, deux facteurs nous poussent à relativiser cette observation : (1) le nombre de bits d'interconnexions dans un CLB est 5 fois supérieur au nombre de bits de la logique, (2) un bit-flip dans l'interconnexion peut déboucher sur plusieurs scénarii, parmi lesquels nous trouvons des cas de non altération du comportement du circuit. Ces scénarii sont illustrés dans le tableau III-II.

Tableau III-II Résultats d'injection laser sur les bits d'interconnexion

		Spot	40µm		8µm	
		Incrément spatial	**20µm**	**10µm**	**20µm**	**10µm**
Connectée	**Pourcentage total de modifications**		9,228%	9,750%	4,363%	5,527%
	Modification	**Nb moyen de Bit-Flips**	0,076	0,013	0,038	0,038
		Nb moyen de modifications	0,025	0,033	0,019	0,015
		Pourcentage de modifications	5,451%	6,841%	18,176%	11,109%
	Suppression	**Nb moyen de Bit-Flips**	0,161	0,181	0,038	0,064
		Nb moyen de modifications	0,152	0,153	0,038	0,061
		Pourcentage de modifications	32,726%	31,969%	36,374%	44,436%
	Ajout	**Nb moyen de Bit-Flips**	0,169	0,406	0,029	0,064
		Nb moyen de modifications	0,449	0,282	0,029	0,048
		Pourcentage de modifications	60,002%	58,903%	27,275%	35,191%
	Aucun effet	**Nb moyen de Bit-Flips**	0,008	0,011	0,019	0,013
		Nb moyen de modifications	0,008	0,011	0,019	0,013
		Pourcentage de modifications	1,821%	2,287%	18,176%	9,264%
Non Connectée	**Pourcentage total de modifications**		90,772%	90,250%	95,635%	94,473%
	Aucun effet	**Nb moyen de Bit-Flips**	4,085	4,001	2,143	2,079
		Nb moyen de modifications	3,542	3,478	1,952	1,919
		Pourcentage de modifications	77,264%	78,588%	85,062%	81,690%
	Création	**Nb moyen de Bit-Flips**	2,813	2,559	0,838	1,087
		Nb moyen de modifications	1,042	0,947	0,343	0,430
		Pourcentage de modifications	22,736%	21,412%	14,938%	18,310%

Dans le cas où une connexion était initialement présente, plusieurs cas peuvent se produire :

- ✓ Modification : suppression de la connexion initialement présente et création d'une ou plusieurs nouvelles connexions,
- ✓ Suppression : suppression de la connexion initialement présente,
- ✓ Ajout: ajout d'une ou plusieurs connexions à celle initialement présente,
- ✓ Aucun effet: conservation de la connexion initialement présente et aucune nouvelle connexion créée.

En moyenne, dans 92,78% des cas les connexions concernées par les bit-flips ne sont pas initialement présentes, ce qui signifie que pour tous ces cas, il existe une faible probabilité d'altération du comportement initial du circuit. En effet, la création d'un nouveau segment d'interconnexion n'implique pas la création d'une interconnexion complète jusqu'à un nœud pouvant par exemple être mis en court-circuit.

Dans les 7,22% des cas restants, les bit-flips concernent les interconnexions existantes. Parmi les effets enregistrés de ces cas, les suppressions et les modifications représentent à elles seules 40%

des bit-flips et affectent d'une manière significative le comportement du circuit. Ceci représente en moyenne 3,19% des tirs.

En d'autres termes, 3,19% seulement des 66% des bit-flips touchant les interconnexions sont considérés comme vraiment critiques pour le circuit. C'est pour cette raison que dans la suite du document (et plus particulièrement le chapitre IV), nous allons nous concentrer essentiellement sur les bit-flips touchant la logique. Toutefois, pour analyser l'effet d'une perturbation sur un circuit implanté dans le FPGA, il est nécessaire de prendre en compte le motif d'erreurs global. Ce motif peut en effet inclure à la fois des bits de configuration des interconnexions et des bits de configuration de la logique.

III. 2.2. Base de données de motifs d'erreurs

Pour injecter des erreurs plus réalistes, il est donc utile de se baser sur des pré-caractérisations de la cible technologique. Cette pré-caractérisation peut être faite une seule fois pour un dispositif donné et une source de perturbation donnée (par exemple, certains flux de particules, des champs électromagnétiques ou un certain type de laser). La pré-caractérisation peut être faite de manière statique, en utilisant éventuellement une configuration du dispositif qui couvre la plupart des modes de configuration possibles pour les CLB et les blocs de mémoire embarquée.

Comme nous l'avons présenté dans le chapitre II, les FPGAs sont essentiellement composés d'un grand nombre de blocs logiques configurables identiques, répétés dans le composant selon une matrice à deux dimensions. Pour les FPGAs modernes, plusieurs types de blocs co-existent (logique configurable, mémoire embarquée, multiplieurs, etc.) mais le principe d'une répétition régulière, dans la structure globale, reste valable. Une perturbation observée dans une zone précise peut donc être reproduite à l'identique dans une autre zone similaire du FPGA (nous faisons ici abstraction d'éventuelles variations technologiques).

L'idée proposée consiste donc à enregistrer sur une petite zone du FPGA les évènements obtenus et les motifs d'erreur observés, que ce soit dans la configuration ou dans les éléments de mémorisation utilisés par l'application exécutée par le circuit, puis à en déduire des motifs d'erreur généralisés (ou relatifs), pouvant être re-localisés à différents autres endroits dans le FPGA. Ceci permet de réaliser une évaluation de robustesse en deux temps. Tout d'abord, l'exposition par exemple sous faisceau laser, décrite dans le paragraphe III.1.1, permet d'enregistrer des motifs d'erreur réalistes pour le type de composant visé. Ensuite, ces motifs

sont transformés en motifs généralisés re-localisables, qui peuvent être injectés à de nombreux endroits dans le FPGA, en phase de conception,, pendant l'exécution d'une l'application. Le principe de l'abstraction est illustré par la figure 3-1.

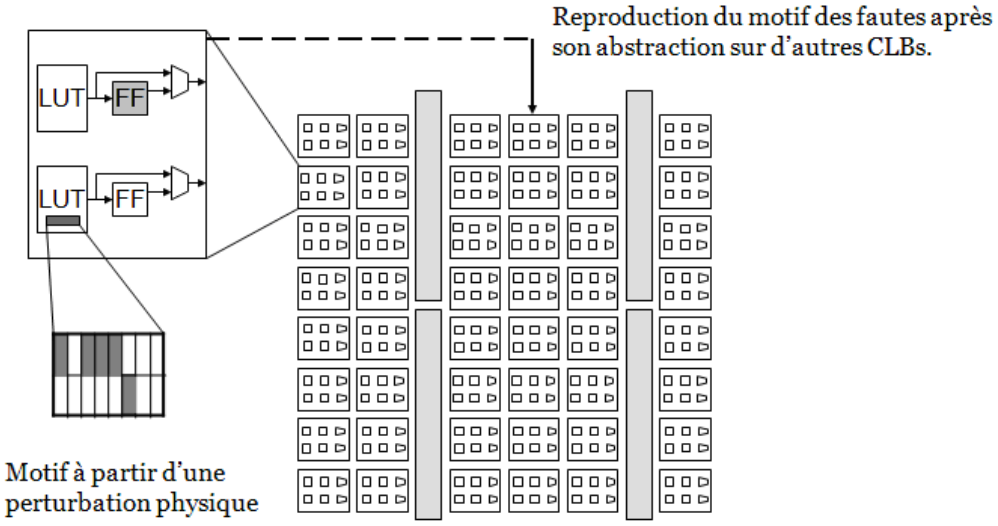


Figure 3-1. Abstraction des motifs d'erreurs

Le coût d'un tel système est faible ; il est donc possible de réaliser des analyses de robustesse très complètes, éventuellement en réalisant les expériences d'injection sur plusieurs systèmes en parallèle. Cette approche permet de tirer profit des techniques d'injection de fautes par émulation, plus exhaustives et moins coûteuses que les expérimentations sous faisceau de particules ou sous laser, tout en ayant la certitude d'injecter des motifs d'erreur parfaitement réalistes. Par ailleurs, les motifs obtenus une fois pour un composant et une source de perturbation donnés sont utilisables pour étudier la robustesse d'autant de circuits ou d'applications que souhaité.

Cette approche est complètement générale et peut être utilisée quelque soit la source de perturbations envisagée (y compris à partir de l'enregistrement de perturbations naturelles dans un environnement difficile comme par exemple l'espace).

❖ Constitution d'une base de données de motifs d'erreurs :

Après le recueil des rapports d'erreur avec l'outil SEFEA-ProD, comme expliqué dans le paragraphe III.1.1, nous créons pour chaque rapport un enregistrement « motif d'erreur » dans une base de données. A ces motifs d'erreurs nous associons la liste des bits. Un enregistrement « erreur » aura comme principales caractéristiques :

- La tuile le contenant dans le FPGA,
- La position relative dans la tuile (numéro de frame et numéro de bit),
- Le sous composant auquel il appartient (exemple : LutG2, interco3),
- La position dans le sous composant,
- Un indice de décalage X, Y s'il y a des bits fautés dans plusieurs CLB.

Suite au recueil des motifs initiaux, la base de données de motifs relatifs est créée, nous permettant de reproduire un même motif d'erreur dans des zones similaires à la zone initiale qui a subi la perturbation. Cette procédure d'abstraction est illustrée dans la figure 3-2.

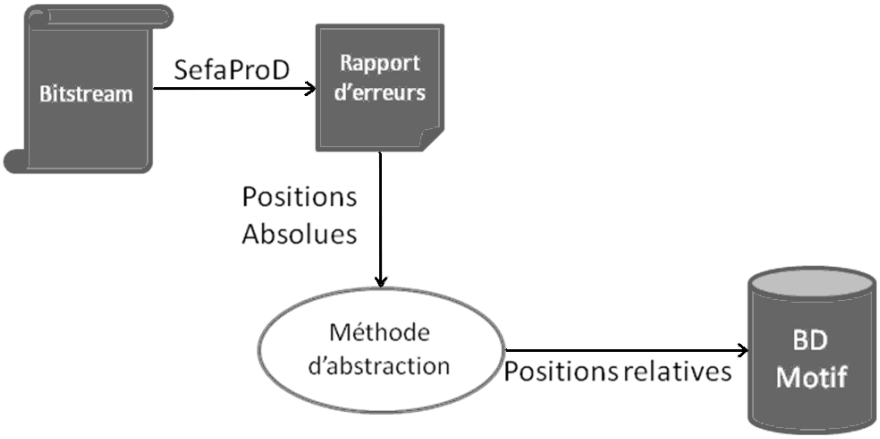


Figure 3-2. Procédure d'abstraction

La base de données que nous avons constituée à partir d'un ensemble d'expériences laser sur un composant Xilinx V2. Cette base de données est composée de 5435 motifs d'erreurs correspondant à des attaques laser, en privilégiant les motifs ayant au moins un bit-flip dans la logique et essentiellement dans la logique combinatoire. Ces motifs comportent entre 1 et 41 bits fautés dans la configuration, avec une moyenne de 11,7 bits erronés dans les bits du contenu des LUT. Le motif peut toucher entre 1 et 6 LUTs (de une à 4 slices) avec une moyenne de 1,7 LUT altérée par expérimentation.

Nous avons par ailleurs classifié, dans le tableau III-III, les parties touchées en fonction de leurs emplacements dans le CLB : G est la partie haute du CLB et F est la partie inférieure. La figure 3-3 est une synthèse de la répartition des nombres de bit-flips des motifs d'erreur de notre base de données.

Tableau III-III Répartition des bitflips par slice

N° Slice	**Slice 0**		**Slice 1**		**Slice 2**		**Slice 3**		Total
Partie du CLB	**F**	**G**	**F**	**G**	**F**	**G**	**F**	**G**	
# de bitflips	2090	20546	2201	11174	1264	8121	1962	11062	58420
% de bitflips	3,58%	35,17%	3,77%	19,13%	2,16%	13,90%	3,36%	18,94%	

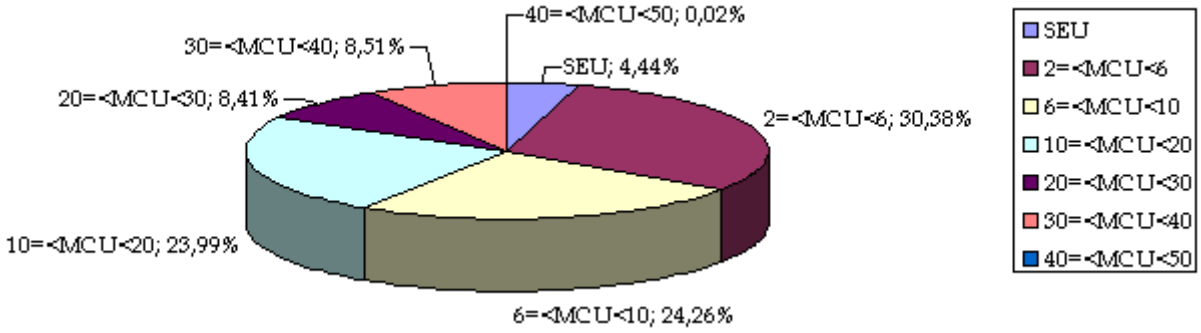


Figure 3-3. Répartition des motifs par nombre de bits fautés

Nous pouvons constater que seulement 4,44% des expérimentations ont abouti à un SEU et que le reste des expérimentations ont engendré plusieurs bits fautés à la fois. Les motifs d'erreurs que nous avons obtenus sont des MCU de multiplicités diverses, parfois élevées.

III.3. Environnement d'Injection de Fautes

Comme nous l'avons vu dans le paragraphe I.4, il existe différentes manières d'injecter des fautes dans la mémoire de configuration d'un FPGA. L'utilisation de l'endo-reconfiguration permet de réduire l'échange de données avec un PC hôte, conduisant à de meilleures performances lors de la campagne d'injection de fautes. De plus cette technique n'est pas très intrusive par rapport à un circuit d'origine et permet l'évaluation de circuits plus complexes sur un FPGA donné. C'est pour toutes ces raisons que nous allons construire notre environnement d'injection de fautes autour de cette technique.

Notre outil d'injection de fautes a été mis en œuvre initialement pour un FPGA Virtex II Pro, et utilise, pour effectuer et gérer les campagnes d'injection de fautes, l'IP ICAP et un des processeurs embarqués, à savoir le PowerPC. Cet environnement est portable, avec quelques modifications, sur d'autres FPGA de la famille Virtex, du moment qu'ils sont plus récents que le Virtex II. Il permet l'injection d'une erreur de bit unique ou multiple dans la configuration et les bascules utilisateur qui se trouvent dans les blocs logiques de configuration (CLB). L'injection de fautes peut être déclenchée à tout moment pendant l'exécution d'une application sur le FPGA.

La figure 3-4 illustre tout le flot d'injection des motifs réalistes, depuis les tirs laser jusqu'à arriver aux indicateurs de robustesse. Bien évidemment pour injecter des motifs d'erreurs sur une autre plateforme, il faut faire une nouvelle pré-caractérisation en injection via un moyen physique déterminé sur la plateforme d'évaluation ciblée.

Bien que le principe soit illustré sur les campagnes laser disponibles, il est généralisable à d'autres plateformes et à d'autres sources de perturbations.

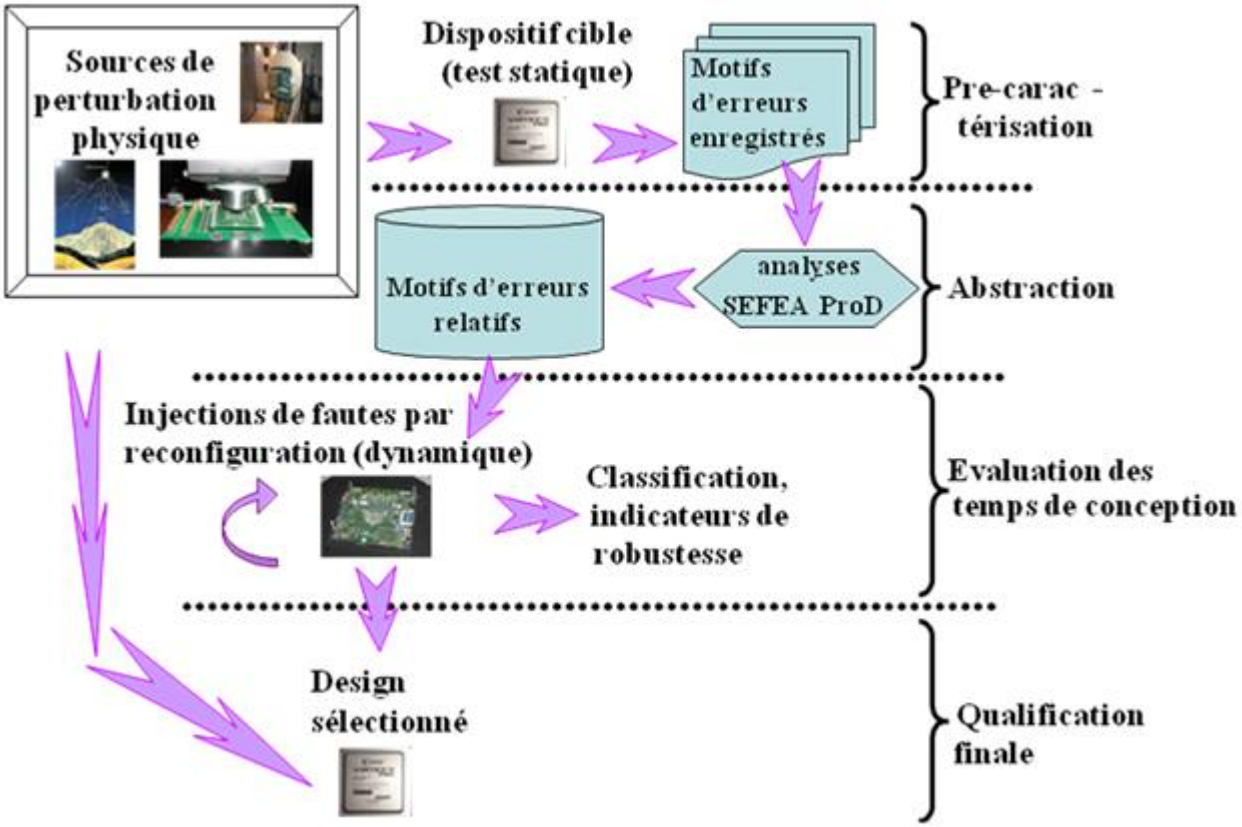


Figure 3-4. Flot d'injection des motifs d'erreurs

III.3.1 Utilisation du PowerPC

Un système d'endo-reconfiguration sur les FPGA Virtex de Xilinx est possible par l'utilisation du bloc IP (HwICAP) permettant de mettre en œuvre l'interface ICAP (présentée dans le chapitre II et détaillée en Annexe B). Un bitstream partiel est alors écrit sur l'ICAP, qui reconfigure par la suite les portions voulues du FPGA. La communication avec l'ICAP peut être assurée à travers un processeur PowerPC embarqué via un programme C.

Les mémoires de stockage des données et instructions du PowerPC ainsi que des bitstreams partiels sont présentées dans l'Annexe B.

III.3.2. Création de l'environnement d'injection de fautes

Le flot d'injection de fautes est représenté dans la figure 3-5 et se décompose comme suit :

- Synthèse du circuit à analyser avec le reste des composants du FPGA (microprocesseur, bus de communication, mécanisme d'injection HWICAP…) ;
- Placement routage de la netlist totale du circuit ;
- Lecture du bitstream avec FPGAEditor ;
- Sélection et extraction des listes de modules dans lesquels nous allons faire l'injection de fautes avec notre outil d'extraction ;
- Exécution de la campagne d'injection de fautes en fonction des éléments visés et des caractéristiques définies par l'utilisateur (détaillées dans le paragraphe III.3) ;
- Recueil et classification des résultats d'injection.

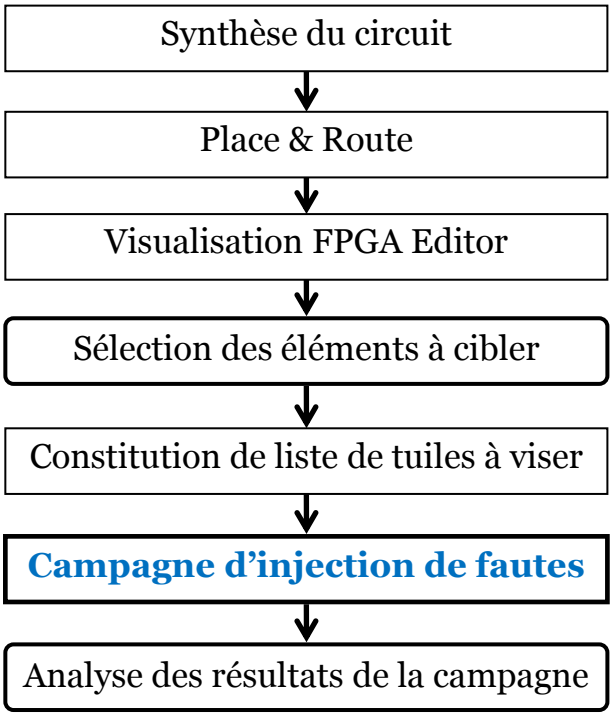


Figure 3-5. Flot d'injection de fautes

III.3.3. Description du mécanisme d'injection selon la cible

Dans cette section nous allons présenter les différents flots d'injection de fautes en fonction de la cible (éléments mémoire ou bits de configuration). Pour bien introduire ces différents flots nous devons d'abord présenter la composition en bits de configuration d'une slice. La moitié d'une slice est illustré d'une manière simplifiée dans la figure 3-6.

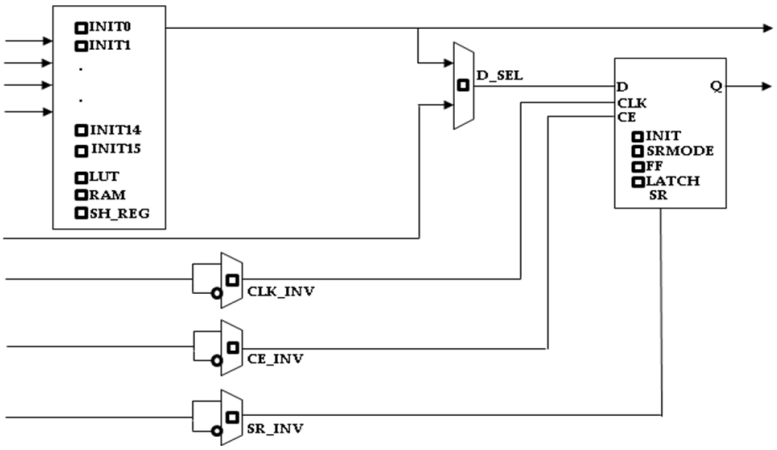


Figure 3-6. Schéma simplifié de la moitié d'une slice

Comme nous l'avons présenté dans le chapitre II, une slice est composée essentiellement de bascules, de LUTs et de multiplexeurs, les interconnexions mises à part. Nous allons présenter les différents bits de configuration de ces 3 éléments :

❖ LUT :

- Les bits INIT de 0 à 15 (pour la V2P, et de 0 à 63 pour la V5) déterminent le résultat de la table de vérité quand la LUT est configurée en logique combinatoire et deviennent des éléments mémoire quand elle est configurée en RAM ou en registre à décalage.
- Les bits de configuration de la LUT déterminent en quoi cette dernière est configurée (LUT, RAM ou registre à décalage).

❖ Bascule : Les bascules d'une même slice partagent les mêmes signaux exceptés les entrées D et les sorties Q

- Le bit de configuration SRMODE détermine la valeur de la bascule lors d'un reset.
- Le bit INIT stocke la valeur du contenu de la bascule au chargement du bitstream et donne la valeur d'une bascule à un instant T en appliquant la commande *capture*
- Les bits de configuration de la bascule déterminent en quoi cette dernière est configurée (verrou ou bascule)

❖ MUX :

- Le multiplexeur SR_INV contrôle le signal de reset des bascules d'une même slice
- Le multiplexeur CLK_INV contrôle le signal d'horloge des bascules d'une même slice
- Le multiplexeur D_SEL contrôle le signal d'entrée que doit recevoir la bascule
- Le multiplexeur CE_INV contrôle le signal de validation de l'horloge des bascules d'une même slice

Nous pouvons à présent présenter le flot général détaillé (figure 3-7) qui sera suivi des cas particuliers en fonction de la cible de l'injection.

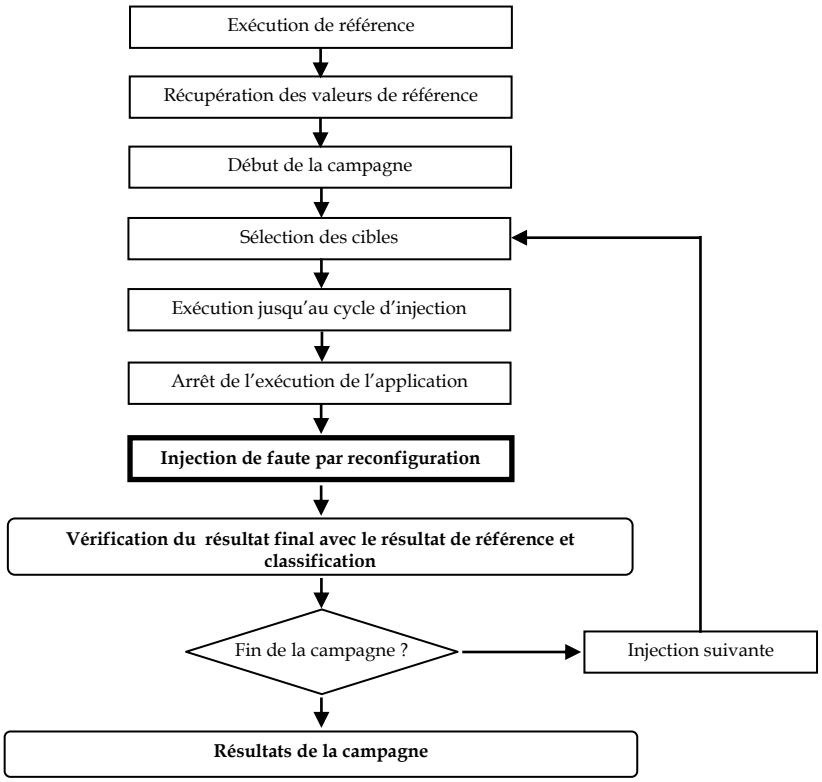


Figure 3-7. Flot général détaillé d'injection de fautes

Tout d'abord une exécution de référence de l'application servira à la récupération des valeurs de bonne fin de programme qui vont nous permettre de classifier les résultats des injections.

Par la suite la campagne d'injection peut commencer, avec la sélection aléatoire du cycle et des cibles. L'application est exécutée jusqu'au cycle d'injection. Une fois le cycle d'injection atteint, le circuit testé est mis en pause et l'injection de fautes est effectuée puis le circuit reprend son fonctionnement jusqu'à la fin de l'application. Ces étapes sont réitérées jusqu'à la fin de la campagne d'injection de fautes.

L'étape « injection de faute par reconfiguration » peut différer selon la cible (LUT ou bascule). Les différents procédés utilisés sont illustrés dans les figures 3-8, 3-9, 3-10 et 3-11.

❖ Injection dans les LUTs

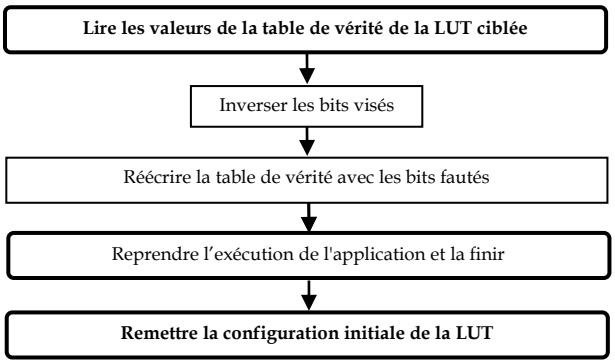


Figure 3-8. Procédé 1 : Flot d'injection dans une LUT

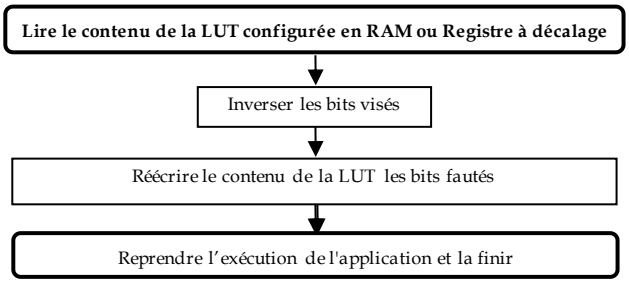


Figure 3-9. Procédé 2 : Flot d'injection dans une LUT configurée en RAM

La figure 3-8 permet d'illustrer le cas d'injection de fautes dans les bits de configuration par l'inversion des bits de contenu d'une LUT. La figure 3-9 décrit le cas d'injection de fautes dans les LUTs configurées en RAM. Les procédés 1 et 2 se ressemblent, la seule différence consiste dans le fait qu'en reconfigurant les bits de configuration, nous devons remettre la configuration initiale après l'injection. Sans cela, la faute injectée serait maintenue lors de l'expérience d'injection suivante. Alors que si la LUT est configurée en RAM ou en registre à décalage ce ne sont plus des bits de configuration mais des éléments mémoires qui changent avec l'exécution de l'application et donc la remise de la valeur initiale à la fin de chaque expérimentation est inutile.

❖ Injection dans les bascules

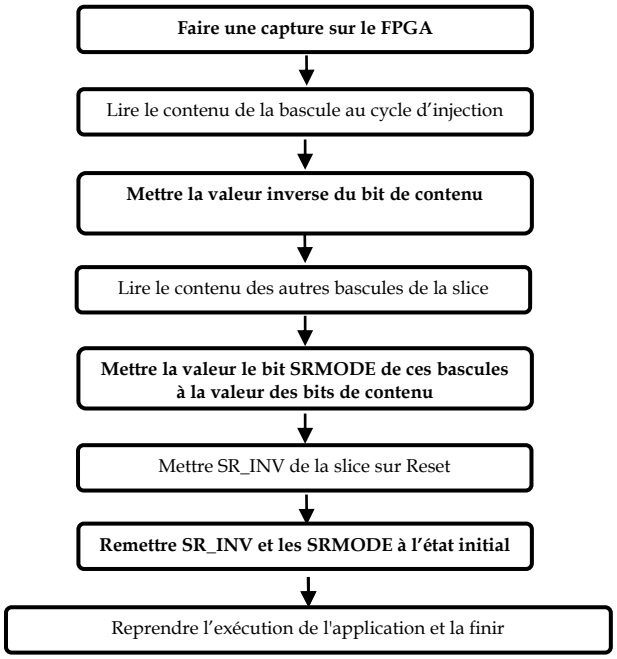


Figure 3-10. Procédé 3 : Flot d'injection dans une bascule en utilisant les bits d'initialisation

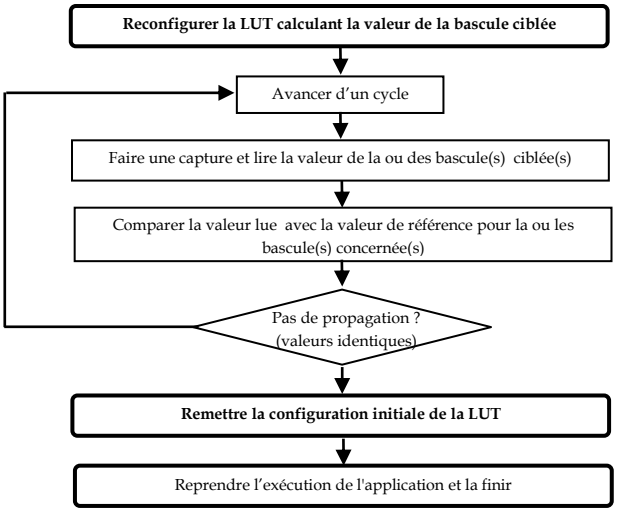


Figure 3-11. Procédé 4 : Flot d'injection dans une bascule via la LUT

La figure 3-10 illustre l'utilisation des bits SR des bascules pour réaliser de l'injection de fautes dans les bascules: Tout d'abord, une capture est appliquée, ce qui permet de charger dans les INIT de toutes les bascules la valeur de celles-ci au cycle d'injection. Après la récupération de la valeur de la bascule, celle-ci est inversée et chargée dans le bit du SRMODE. Par la suite le bit SR_INV est inversé et pour que seule la valeur de la bascule ciblée soit modifiée les SRMODE des autres bascules de la slice doivent être chargés avec la valeur des INIT de leur bascule respective. Pour finir il faut remettre les bits SRMODE des bascules et SR_INV de la slice à leur état d'origine pour que l'erreur ne devienne pas une erreur rémanente.

Nous avons, par ailleurs, expérimenté une autre technique d'altération de la bascule, en exploitant la relation LUT-bascule pour émuler le comportement d'un SEU dans une bascule. L'idée est la suivante : nous inversons tous les bits de configuration de la LUT de façon à ce que la fonction soit la fonction inverse de celle originellement implémentée et que le résultat de sortie, qui est l'entrée de la bascule, soit un résultat erroné. Nous gardons la configuration inverse de la fonction jusqu'à ce que le résultat de calcul se propage dans la bascule.

Pour ce procédé, l'exécution de référence de l'application sert à la récupération des valeurs de bonne fin de programme mais aussi à la génération d'un **golden readback** des contenus des bascules ciblées cycle par cycle. Ce **golden readback** est l'élément de référence pour savoir quand le contenu de la bascule a été altéré par la reconfiguration de la LUT en charge du calcul, et à ce moment, nous la configuration originale de la LUT est restaurée. Ce procédé que nous nommerons pour le reste du document le procédé 4 est détaillé dans la figure 3-11.

III.4. Définition des campagnes d'injection de fautes

Pour définir une campagne d'injections, nous devons spécifier les quatre entrées suivantes:

- **La description du circuit à analyser :** Pour nos campagnes nous utiliserons les descriptions RTL synthétisables de nos circuits cibles. Différents circuits ont été utilisés dans le cadre de nos études. Pour le premier cas d'étude qui sera présenté dans ce document (paragraphe III.6), nous utiliserons deux versions du processeur Leon2, une version distribuée sur le site de Gaisler Research avec une licence GNU GPL, et une version durcie tolérante aux fautes, développée au laboratoire TIMA **[Port.-06]**. Pour la deuxième étude de cas (paragraphe III.7), nous ciblerons le processeur Leon3 surveillé par un Watchdog **[Berg.-09]**.

- **L'application à exécuter :** Le comportement du circuit ne peut pas être étudié sans application s'exécutant sur ce dernier. Nous avons choisi pour nos cas d'étude des applications couvrant des domaines divers : FIR un filtre à réponse impulsionnelle, Mtmx une application de multiplication de matrices, Sieve une application de calcul des nombres premiers utilisant le crible d'Ératosthène et enfin l'algorithme de chiffrement/déchiffrement AES.
- **Les données relatives aux fautes à injecter (modèles, cibles) :** L'un des modèles précédemment décrit doit être sélectionné Pour notre étude de cas Leon2-Leon2FT, le modèle MCU sera par exemple choisi et la sélection des multiplicités spatiales sera faite aléatoirement à partir de la base de données des motifs d'erreur présentée dans le paragraphe III.2.
- **Le type de campagne :** Il existe trois types de campagne possibles : déterministe, aléatoire (ou pseudo-aléatoire) et exhaustive. Le type de campagne détermine le choix de la liste de fautes. Pour une campagne déterministe, l'utilisateur définit, pour chaque faute, la cible et le moment de l'injection. Lors d'une campagne exhaustive, des fautes sont injectées dans chaque cible et à tous les cycles d'horloge de l'application.

  La liste de fautes peut également être générée de façon aléatoire, c'est la meilleure solution quand le nombre d'erreurs possibles qu'un circuit peut subir est importante et qu'il est impossible en termes de ressources et de temps de faire toutes les injections. Toutefois, le principal problème d'une telle sélection est la détermination de la confiance dans les résultats obtenus. Des études, telle celle présentée dans **[Leve.-09]**, permettent de résoudre ce problème. Une formule qui permet de calculer le nombre de fautes à injecter est la suivante :

$$n = \frac{N}{1 + e^2 * \frac{N-1}{t^2 * p * (1-p)}}$$

Où : n : Taille des échantillons.
N : Taille de la population mère.
p : Proportion d'erreur supposée
e : Marge d'erreur
t : Taux de confiance

Cette formule permet de donner le nombre n d'injections à réaliser, en fonction des autres paramètres cités dans l'équation. Le paramètre p correspond au pourcentage d'erreurs conduisant au cas analysé (par exemple, une défaillance de l'application). Pour la population mère, son calcul est fait de manière à prendre en compte toutes les cibles concernées éventuellement par l'opération d'injections de fautes (banc de registres,

mémoires…), avec la multiplicité d'erreurs choisie, et pour l'ensemble des cycles d'horloge exigé par l'application.

Pour toutes les campagnes présentées dans ce chapitre, nous avons choisi une taille d'échantillon pour une marge d'erreur de 5% et un taux de confiance de 95% (t=1,96).

III.5. Etude comparative de plateformes AT40k et Virtex

Avant de présenter quelques résultats de campagnes pour montrer l'utilisation de notre plateforme d'injections, nous allons analyser sur trois exemples l'impact du type de composant choisi sur l'efficacité de l'injection par reconfiguration dynamique. Ce paragraphe fait l'objet d'une comparaison des temps de reconfiguration pour trois plateformes différentes.

III.5.1. Présentation des plateformes comparées

La première plateforme est celle mentionnée précédemment, basée sur un composant Xilinx XC2VP30 Virtex II Pro (V2P). La seconde plateforme est basée sur un Xilinx XC5VLX110T Virtex V (V5) et la troisième est basée sur un Atmel AT40KEL40 AT40K. Ces trois types de composants ont été présentés respectivement dans les paragraphes II.1.1.1, II.1.1.2 et II.2. Les deux premières plateformes sont assez similaires, vu que toutes les deux se basent sur l'endo-reconfiguration via l'ICAP. Toutefois, le composant V5 ne disposant pas de cœur de processeur directement intégré, le processeur PowerPC du V2P devra être remplacé par un processeur synthétisable (MicroBlaze). La plateforme est donc plus efficace en terme de technologie, mais elle a aussi un handicap potentiel. Pour la plateforme Atmel, c'est l'utilisation de l'adressage direct lors de la reconfiguration qui la rend attractive, alors que sa technologie est nettement plus ancienne.

III.5.2. Focalisation sur les durées de reconfiguration

Effectuer des injections de fautes nécessite bien évidemment l'écriture de nouvelles valeurs dans la configuration, mais aussi la lecture des valeurs existantes, soit pour déclencher un bit-flip, soit pour vérifier que l'injection a été efficace. Nous rapportons donc ici les temps de reconfiguration élémentaires (en lecture et en écriture). Chaque action élémentaire peut être répétée un nombre variable de fois pour plusieurs campagnes d'injection de fautes, selon les spécifications fournies par le concepteur. L'incidence globale sur la durée de la campagne est fonction du nombre de cycles exécutés pour chaque expérience (c'est à dire, chaque exécution de l'application) et du modèle de fautes (en particulier, la multiplicité spatiale et/ou temporelle des erreurs). Le temps global nécessité par les injections lors d'une campagne donnée n'est donc

significatif que pour cette campagne particulière, alors que la durée d'une action élémentaire permet de prédire les pénalités induites par le processus d'injection.

III.5.3. Résultats comparatifs sur les durées de reconfiguration

Les premières expériences ont été effectuées sur la plateforme V2P. Les délais d'injection de fautes avec la reconfiguration partielle ont été mesurés avec un Timer en fonction des tuiles visées (LUT, bascule ou BRAM) et du nombre de bits reconfigurés. La figure 3-12 illustre le temps nécessaire pour l'injection d'un motif d'erreurs donné (ciblant des LUT) en fonction du nombre de bits à modifier. Le tableau III-IV résume les durées que nous avons obtenues pour les différentes opérations exécutées par l'ICAP.

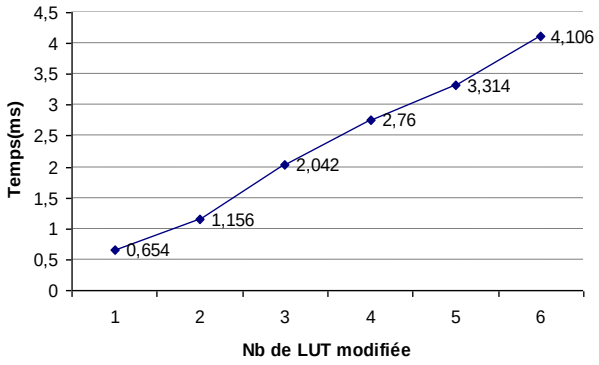


Figure 3-12. Délais de reconfiguration en fonction du nombre de LUTs modifiées

Tableau III-IV. Durée des readbacks et des reconfigurations

Opération	Temps (ms)
Lecture de 100 LUT	41,19821
Reconfiguration de 100 LUT	47,66516
Lecture de 100 FF	17,1135
Reconfiguration de 100 FF	17,86155
Lecture de 100 Frame	40,27013

Concernant l'AT40K nous avons exploité le mode de configuration RAM synchrone communément appelé le mode 4 (présenté dans le chapitre II). Ce dernier permet de configurer le FPGA avec des bitstream partiels avec une trame de 32 ou 40 bits : 24 bits pour l'adresse et 8 ou 16 bits pour les données selon le besoin. Dans ce mode, l'adresse, la donnée et l'autorisation d'écriture sont appliqués sur le front montant de l'horloge et le FPGA est vu comme un simple espace d'adressage de mémoire mappée. Ceci nous permet un accès en lecture et en écriture sur toute la configuration du FPGA.

L'AT40K n'est pas muni d'un microprocesseur embarqué AVR comme les FPGAs plus récents de cette famille, nous n'avons donc pas pu utiliser une gestion intégrée de la reconfiguration comme pour la plateforme V2P. La reconfiguration a été commandée à partir du PC hôte et nous nous sommes appuyés sur les Hard Macro, qui peuvent être créées par l'outil Figaro puis chargées dans le FPGA comme étant des bitstreams partiels, pour évaluer les durées de reconfiguration.

Le tableau III-V résume le temps nécessaire pour lire et écrire un élément de base sur les trois plateformes employées. Le temps correspondant à une frame est bien sûr seulement indiqué pour les architectures Xilinx. Les durées présentées sont des moyennes obtenues sur un ensemble de mesures. Il est à noter que dans le cas de la plateforme V5 les mémoires caches du MicroBlaze ont été activées. En effet, quand elles sont désactivées, les temps d'exécution sont presque multipliés par dix. Bien entendu, le temps passé dans le programme de commande d'injection afin de déterminer l'emplacement de l'injection n'est pas pris en compte, nous nous intéressons seulement aux temps d'écriture et de lecture.

L'utilisation d'une plateforme plus récente (V5 au lieu de V2P) peut être bénéfique lorsque les injections doivent être faites dans les bascules utilisateurs. Toutefois, le gain n'est que d'environ 50% lors de la lecture et moins de 10% lors de l'écriture. Pour le même type d'opération, la plateforme AT40K présente de 50% jusqu'à 70% d'amélioration par rapport à la V5, alors qu'elle est fabriquée dans une technologie beaucoup plus ancienne, et en dépit du contrôle externe de la reconfiguration. Dans le contexte de l'injection de fautes, la technologie du composant n'est donc manifestement pas le paramètre le plus important.

Tableau III-V Temps d'accès pour les trois plateformes V2P, V5 et AT40

	Durée de lecture (µs)			Durée d'écriture (µs)		
	V2P	V5	AT40	V2P	V5	AT40
LUT	4,41	27	0,27	4,48	31,5	0,31
FF	1,17	0,59	0,28	1,18	1,1	0,3
Frame	0,57	0,48	NA	0,63	0,51	NA

Lorsque les injections nécessitent le changement du contenu d'une LUT, la plateforme AT40K présente un net avantage à la fois en lecture et en écriture. Dans ce cas, la plateforme V5 est environ 6 fois plus lente que la plateforme V2P, alors que le nombre de bits d'une frame est plus petit et que les mesures des temps de reconfiguration pour une seule frame ont montré que la V5 nécessite moins de temps que la V2P. L'utilisation d'un processeur synthétisé sur la logique programmable, le MicroBlaze à la place du PowerPC, peut expliquer ces résultats. Cela peut difficilement être quantifié, mais nous n'avons pas remarqué une réelle amélioration dans

le temps de reconfiguration sur la plateforme V5, en dépit de l'amélioration de la technologie et de l'accès direct au processeur à partir du HwICAP en utilisant le bus PLB (au lieu de passer par un bus OPB, puis un pont OPB-PLB pour atteindre le processeur dans la plate-forme V2P - les figures 3-13 et 3-14 illustrent la différence entre les deux architectures).

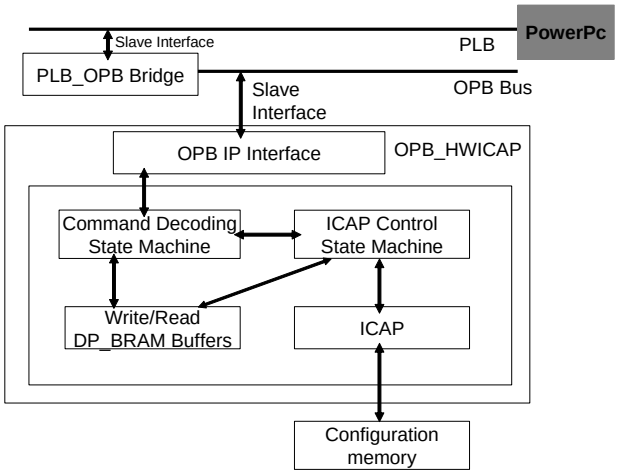


Figure 3-13. Vue d'ensemble du contrôle de reconfiguration dans une plateforme V2P

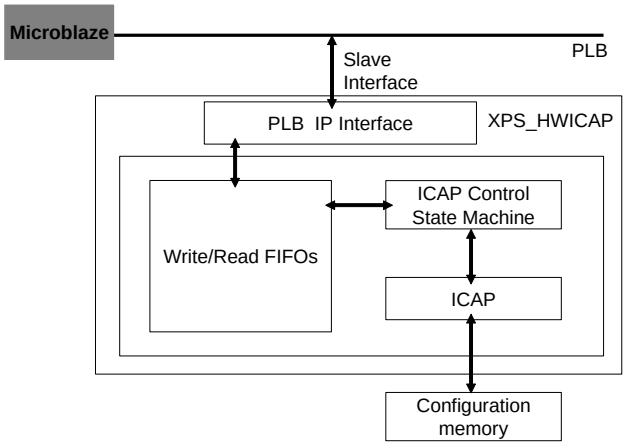


Figure 3-14. Vue d'ensemble du contrôle de reconfiguration dans une plateforme V5

Enfin, la raison principale de la perte de performance par rapport à V2P est probablement l'évolution de l'architecture du composant, nécessitant plus de chargements de configuration pour une injection dans les LUT. Les LUTs de ATK, V2P et V5 ont respectivement 3, 4 et 6 entrées par LUT. Par conséquent, elles contiennent 8, 16 et 64 bits de données de configuration. Dans l'architecture V5, le plus grand nombre de bits dans les LUTs peut impliquer un plus grand nombre de frames pour reconfigurer l'ensemble d'une LUT. Quoi qu'il en soit, le temps pour les injections dans les LUTs de la V5 peut être 100 fois plus grand que l'équivalent dans une LUT Atmel. Grâce à l'adressage direct, ce résultat n'est pas directement lié à la capacité du composant donc cet avantage devrait être maintenu pour des composants plus grands que celui employé, mais utilisant un mode de configuration identique.

En conclusion sur cette partie, plusieurs caractéristiques semblent donc avoir un impact notable sur l'efficacité d'une injection de fautes par reconfiguration partielle, et donc le choix d'une plateforme pour ce type d'application. Le choix d'un composant récent, dans la technologie la plus agressive, peut permettre d'augmenter la fréquence d'horloge du prototype et et d'évaluer la robustesse de circuits plus complexes ; toutefois, le temps perdu par les injections peut augmenter au lieu d'être réduit. A l'inverse, un composant permettant une reconfiguration très locale peut permettre un gain de temps, malgré une technologie plus ancienne.

III.6. Etude de cas Leon2

L'environnement d'abstraction et d'injection de fautes a été utilisé sur un FPGA Virtex II Pro, avec le processeur Leon2 exécutant les applications FIR, Mtmx, Sieve et AES. Deux versions du Leon2 ont été utilisées. La première version est la version distribuée sur le site de Gaisler Research avec une licence GNU GPL. Cette version ne comprend aucune technique d'amélioration de robustesse. La deuxième version a été développée au sein du laboratoire TIMA il y a quelques années. Elle représente une variante durcie et flexible du processeur Leon2 **[Port.-06],** avec plusieurs versions de protections possibles.

Dans la version utilisée, la logique combinatoire du pipeline est protégée grâce à un mécanisme dynamique de vérification de la parité. Connaissant la fonction du bloc combinatoire (Bloc L dans la figure 3-15), un composant spécifique (Bloc L' dans la même figure) prédit, à partir des valeurs d'entrée, la valeur des bits de code de sortie pour pouvoir faire une comparaison avec les bits calculés à partir de la vraie sortie. Etant donné que la principale caractéristique du pipeline est d'avoir la logique combinatoire partitionnée sur plusieurs cycles grâce à l'introduction de registres internes qui permettent de « couper » les opérations, la figure 3-15 illustre une étape du pipeline du Leon 2 durci.

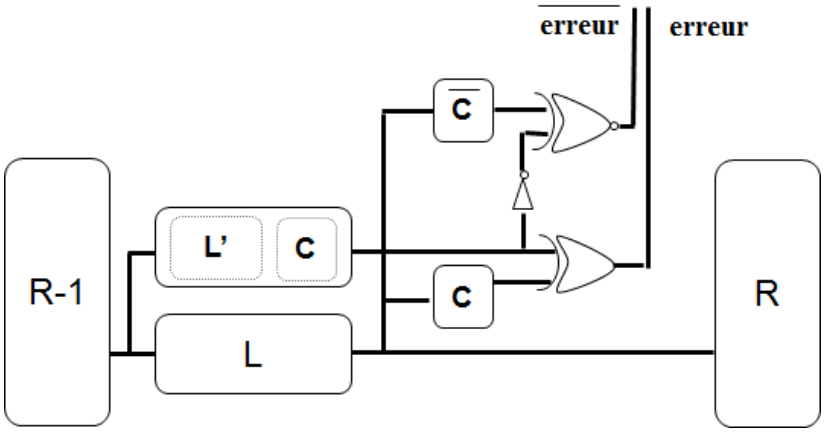


Figure 3-15. Protection par prédiction de code d'un étage du pipeline ajoutée au Leon2 [Port.-06]

L'objectif principal de cette étude de cas est d'évaluer l'efficacité d'une telle approche de protection fonctionnelle, face à des erreurs survenant dans la configuration du FPGA. La protection évaluée vise à protéger une implantation sous forme d'un ASIC pré-caractérisé ; nous cherchons donc à savoir quelle efficacité est obtenue pour des motifs d'erreurs réalistes dans le cas d'une implantation sur FPGA SRAM.

III.6.1. Spécifications de la campagne d'injection

Le modèle d'erreur traditionnel SEU a été sélectionné pour une première partie de la campagne, en ciblant soit les LUT soit les bascules utilisateur. La base de données de motifs d'erreurs présentée précédemment a aussi été utilisée lors de ces expériences.Les motifs injectés ont alors été sélectionnés pour être concentrés sur une LUT donnée et impliquaient 2 à 15 bits fautés. Ils ont été injectés pour chaque programme et pour les deux versions du Leon2 dans les CLB qui implémentent les 5 étages du pipeline de l'unité entière. Pour avoir une marge d'erreur de la classification égale à 5% avec un taux de confiance de 95%, le nombre d'injections réalisées a varié de 8000 jusqu'à 160000 en fonction de l'étage de pipeline. Chaque motif d'erreurs injecté a été classé selon le comportement du système après l'injection : "Silencieuse" s'il n'y a aucun effet; "Erreur" si le résultat retourné est faux mais que l'application reste active et peut réaliser d'autres calculs ; "Défaillance" si le programme se termine d'une manière inopinée et que le système a un comportement indéterminé; et "Crash" si l'erreur est irrécupérable jusqu'à la réinitialisation du système. Dans le cas de la version protégée du Leon, chaque classe outre la "Silencieuse" est divisée en deux sous-classes "détectée" et "non détectée".

III.6.2. Résultats de l'étude de cas et discussion

Nous avons noté dans le paragraphe III.2.2 que notre base de données contient 10 fois plus de bits fautés dans la partie G des CLB que dans la partie F. Nous avons donc classé, dans le tableau III-VI, les LUTs implantant le Leon2 en fonction de leurs emplacements dans le CLB pour voir si le taux de répartition est proportionnel au taux des bits fautés. La répartition étant quasi égale nous avons favorisé la sélection des motifs d'erreurs touchant aussi bien les LUT G que les LUT F, afin de ne pas altérer l'analyse avec des motifs visant essentiellement la partie haute du CLB.

Tableau III-VI. Répartition des LUTs du processeur Leon2 selon leur emplacement dans le CLB

	Fetch	Decode	Execute	Memory	Write	Total
Logique implantée dans une LUT F	100	223	615	267	37	1242
Logique implantée dans une LUT G	100	230	651	272	37	1290

Les résultats d'injection de fautes sont résumés dans le tableau III-VII, qui montre que les résultats d'injection de SEU dans les LUTs et les bascules sont plus optimistes que les injections de motifs d'erreurs réalistes dans les LUTs, et ce pour toutes les applications testées.

En effet, dans plus de 90% des cas (91% en moyenne pour les 4 applications), les injections de motifs engendrent un comportement incontrôlé du système donc soit une défaillance soit un crash. Ce taux descend à 80% pour l'application de chiffrement AES mais reste quand même

très élevé. Les injections de motifs d'erreurs sont suivies par les injections de SEUs dans les LUTs qui affichent des taux de comportements très perturbés entre 65 et 89%. Enfin, les injections de SEU dans les bascules utilisateur sont les moins perturbatrices, avec un taux de crash ou de défaillance ne dépassant pas 40%.

Tableau III-VII Résultats d'injection de fautes dans le processeur Leon2

		Silencieuse	Erreur	Défaillance	Crash
FIR	SEU FF	54,61%	11,63%	10,88%	22,89%
	SEU LUT	13,14%	6,12%	50,39%	30,35%
	Motifs d'erreur	3,40%	0,98%	60,54%	35,08%
Mtmx	SEU FF	52,59%	12,03%	14,55%	20,83%
	SEU LUT	18,68%	5,52%	28,44%	47,36%
	Motifs d'erreur	7,61%	0%	31,67%	60,72%
Sieve	SEU FF	47,77%	12,11%	23,46%	16,66%
	SEU LUT	10,17%	3,53%	52,63%	33,67%
	Motifs d'erreur	4,90%	0%	40,08%	55,02%
AES	SEU FF	57,96%	8,52%	24,67%	8,85%
	SEU LUT	34,35%	1,78%	25,49%	38,37%
	Motifs d'erreur	18,41%	0,52%	44,21%	36,85%

Les résultats montrent également qu'un très faible pourcentage de motifs conduit à des sorties erronées sans défaillance ou crash (moins de 1% pour toutes les applications), alors que les injections dans les bascules conduisent en moyenne à 11% de cas de sorties erronées contre une moyenne de 4% pour les SEUs dans les LUTs.

Une erreur dans la logique combinatoire étant une faute rémanente, elle a des effets répétitifs sur le comportement du circuit : à chaque fois que cette fonction de calcul entre en jeu, elle peut donner un résultat erroné qui va se propager à d'autres éléments du FPGA, et aboutira dans la plupart des cas à des fautes avec multiplicité temporelle. Comme nous l'avons présenté dans le paragraphe III.1.2, les motifs d'erreurs que nous injectons peuvent contenir plusieurs bit-flips dans la logique combinatoire, ce qui rajoute à la multiplicité temporelle une multiplicité spatiale qui augmente encore la probabilité d'altération du programme.

Les fautes silencieuses sont plus fréquentes avec les injections de SEU dans les bascules (53% en moyenne), suivies des injections de SEU dans les LUT (19% en moyenne) et en dernière position les injections de motifs d'erreurs (8% en moyenne).

Quand les mécanismes de durcissement sont ajoutés au processeur, les résultats d'injections restent sensiblement les mêmes. Les résultats des campagnes sur la version du Leon2 protégé

sont présentés dans le tableau III-VIII. La figure 3-16 illustre la classification des fautes injectées pour chaque application étudiée.

Tableau III-VIII Taux de détection des motifs d'erreur dans le Leon protégé

Application	Motifs non silencieux	Motifs détectés	% Detection
Fir	294934	61666	21.06%
Mtmx	248126	54492	21.96%
Sieve	251745	58596	23.28%
AES	139408	35635	27.13%

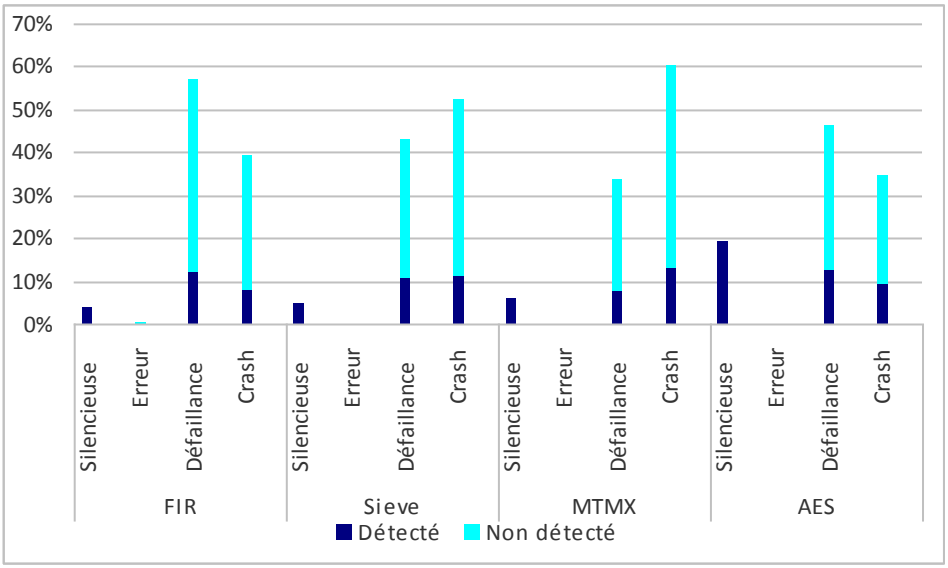


Figure 3-16. Classification des fautes injectées pour chaque application étudiée

Seules 8% des injections en moyenne sont restées silencieuses. Cependant, ce taux est très variable d'une application à une autre : moins de 4% pour FIR mais 19% pour AES. Aucune « erreur » n'est enregistrée sauf avec FIR pour moins de 1% des injections. Les classes « défaillance » et « crash » sont par contre très majoritaires avec une moyenne de 91% des injections sur les 4 applications comme dans le cas précédent (malgré les modifications du pipeline). Parmi ces injections, de 21% à 27% ont été détectées par le mécanisme de parité lié à l'application. L'application AES se caractérise par le plus haut taux d'injections silencieuses et le meilleur taux de détection.

Les résultats obtenus sur la version protégée du processeur Leon2 montrent que les motifs d'erreur réalistes dans la mémoire de configuration peuvent causer dans la plupart des cas un comportement incontrôlable par le système en dépit de la protection implantée, avec un taux de

détection limité aux alentours de 20-25%. Les protections n'étant pas destinées à couvrir les erreurs de configuration, ce taux de protection n'est pas négligeable. Cependant, il est obtenu avec un coût élevé, et n'est donc pas satisfaisant. Des techniques plus ciblées sur les erreurs de configuration sont donc nécessaires lorsque les applications critiques sont mises en œuvre sur de tels dispositifs, ou lorsque une protection efficace doit être obtenue avec un coût réduit.

III.7. Etude de cas Leon3

Une seconde étude de cas va être présentée, afin d'illustrer l'utilisation d'une autre plateforme et l'efficacité d'une approche différente de protection fonctionnelle. Pour cette étude le processeur Leon3 a été implanté sur un FPGA Xilinx de la famille Virtex V avec un coprocesseur de vérification de flot de contrôle et exécute une application de chiffrement AES.

Cette fois, il ne sera donc pas possible d'utiliser la base de données de motifs d'erreurs, obtenue pour la technologie V2P. Afin de tester la robustesse de la protection, nous avons injecté des fautes en particulier sur des registres très critiques du pipeline : Fetch-PC (adresse de recherche d'instruction), Decode-Inst (instruction à exécuter) et Decode-CWP (identification de la fenêtre de registres en cours d'utilisation dans l'architecture Sparc). Des campagnes supplémentaires ont été également effectuées sur le banc de registres implanté dans des LUTs configurées en RAM.

Nous avons conservé la classification utilisée dans l'étude de cas précédente, en rajoutant les erreur de retard, limitées à +/- 1500 cycle car quasiment toutes les erreurs (ou résultats erronés) que nous avons observé sur ce prototype sont accompagnées d'un délai dans l'exécution.

III.7.1. Description des techniques d'injection de fautes utilisées

III.7.1.1. Technique d'injection de fautes dans les registres PC, INST et CWP

La procédure 3, présentée dans le paragraphe III.3.3, nécessite la connaissance des fonctions des bits dans une frame. Or pour la plateforme V5, l'outil SEFEA-ProD n'est pas disponible. Par contre nous sommes parvenus, par « rétro-conception », à identifier les bits de contenu des bascules et des LUTs. Ceci nous a permis de mettre en pratique le procédé 4 présenté en III.3.3, procédé qui exploite le fait que la bascule visée est chargée à partir de la LUT comme illustré dans la figure 3-17.

L'inconvénient de ce procédé c'est que nous sommes limités à injecter dans les bascules qui obtiennent leur nouvelle valeur à partir des LUTs, et que nous devons réitérer l'injection dans la LUT jusqu'à ce qu'un résultat erroné soit chargé dans la bascule. A titre d'exemple, nous avons

observé que pour le registre Fetch_PC la propagation du bit fauté de la LUT à la bascule se produit dans 41% des cas au cycle suivant la reconfiguration de la LUT, pour l'application utilisée.

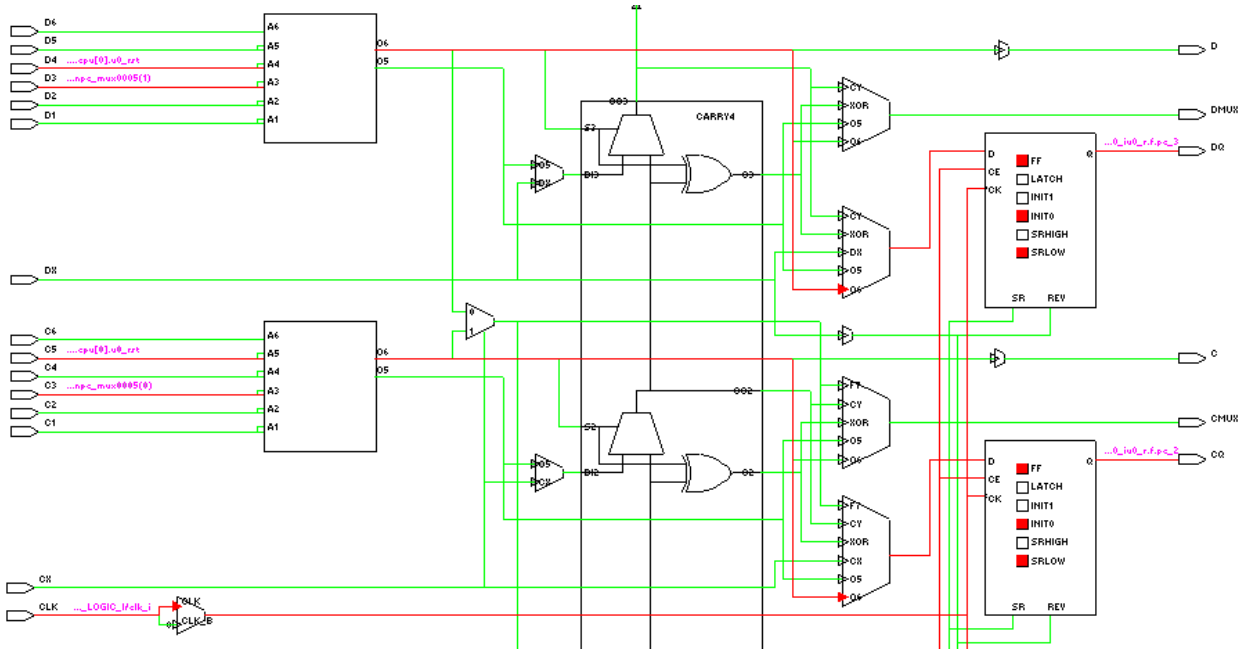


Figure 3-17. Relation bascule/LUT

Les motifs d'erreurs ne pouvant être exploités, nous avons injecté avec cette technique des erreurs de multiplicité allant de 1 à 5. Les erreurs de multiplicité supérieure à 1 seront considérées comme des MCUs même si elles peuvent dans certains cas être des MBU. Elles sont toutefois contrôlées au niveau spatial, pour correspondre à des bits physiquement proches dans la mémoire de configuration.

III.7.1.2. Technique d'injection de fautes dans le banc de registres

Pour les injections de fautes dans le banc de registres nous avons utilisé le procédé 2, introduit dans le paragraphe III.3.3. Etant donné que chaque LUT est constituée de 64 bits de contenu, celle-ci engloberait donc deux registres. Pour injecter une faute, nous choisissons aléatoirement 4 entrées :

- le cycle ;
- Les coordonnées d'une LUT ;
- Partie basse ou haute dans la LUT (1 : (0-> 31) ou 2: (32-> 64)) ;
- L'index du bit dans la partie sélectionnée de la LUT.

} Choix de la cible

III.7.2. Présentation des résultats obtenus sur le seul processeur Leon3

Dans ce qui suit, nous allons considérer toute "non bonne terminaison du programme au bon cycle", comme un cas d'altération du programme. Par conséquent, nous regrouperons dans cette catégorie les retards, les défaillances et les crashs.

III.7.2.1. Résultats d'injection dans Fetch-PC

Comme le montre le tableau III-IX, les injections dans le registre Fetch-PC perturbent énormément (et sans surprise) le déroulement du programme. Toutefois, en analysant les résultats obtenus nous avons remarqué que le Leon3 est insensible à la modification de certains bits de son Fetch-PC, à savoir les bits 26 à 29. Ce résultat montre l'intérêt des campagnes effectuées, non seulement pour l'analyse de robustesse, mais aussi pour la validation fonctionnelle d'un circuit devant être inclus dans une application critique.

Tableau III-IX Résultats d'injections dans Fetch-PC

	# fautes silencieuses	# d'injections	% altération du programme
SEU	51	573	91,10%
MCU2	40	573	93,02%
MCU3	16	573	97,21%
MCU4	8	573	98,60%
MCU5	14	573	97,56%

Tableau III-X Détails des résultats d'altération du programme suite à l'injection dans Fetch-PC

	Erreurs	Retards	Crashs	Défaillances
SEU	3,26%	9,77%	83,91%	3,07%
MCU2	2,57%	1,80%	94,34%	1,29%
MCU3	0,47%	0,47%	93,14%	5,91%
MCU4	0,68%	0,23%	91,57%	7,52%
MCU5	0,53%	1,59%	89,22%	8,48%

Si nous analysons de plus près les cas d'altération du programme, détaillés dans le tableau III-X, nous remarquons qu'en moyenne, pour toutes les multiplicités confondues, plus de 9 injections sur 10 produisent un crash ou une défaillance. Par ailleurs, les taux d'erreurs et de retard atteignent respectivement 3% et 9% des injections SEUs puis diminuent avec l'augmentation de la multiplicité.

III.7.2.2. Résultats d'injection dans Decode-Inst

Dans les résultats du tableau III-XI, nous remarquons que l'altération du programme est beaucoup moins élevée pour le registre Decode-Inst que pour le registre Fetch-PC, malgré l'utilisation du même procédé d'injection qui favorise la propagation des erreurs. Les fautes silencieuses sont dues à la non utilisation de certains bits du registre dans un certain nombre d'instructions. Cependant, le taux d'altération augmente avec la multiplicité des fautes.

Tableau III-XI Résultats d'injections dans Decode-Inst

	# fautes silencieuses	# d'injections	% altération du programme
SEU	278	506	45,06%
MCU2	227	307	26,06%
MCU3	168	496	66,13%
MCU4	104	498	79,12%
MCU5	139	847	83,59%

Dans le tableau III-XII nous pouvons remarquer que le taux de Crash et de Défaillance tend à augmenter avec la multiplicité.

Tableau III-XII Détails des résultats d'altération du programme suite à l'injection dans Decode-INST

	Erreurs	Retards	Crashs	Défaillances
SEU	13,16%	21,93%	40,79%	24,12%
MCU2	13,75%	11,25%	61,25%	13,75%
MCU3	5,49%	17,99%	45,73%	30,79%
MCU4	5,08%	15,74%	52,03%	26,90%
MCU5	7,06%	14,12%	60,59%	18,08%

III.7.2.3. Résultats d'injection dans Decode-CWP

Dans l'architecture Sparc le nombre de fenêtres de registres varie entre 2 et 32 selon l'implémentation. Dans notre version du Leon3, il existe seulement 8 fenêtres. Le registre CWP est donc composé de 3 bits seulement, c'est la raison pour laquelle nous allons y injecter des SEUs et des MBU2 seulement.

Le taux d'altération est aussi élevé que pour le registre PC de l'étage Fetch (Voir tableau III-XIII). Ceci s'explique par le fait que la modification de ce registre aboutit à la modification de tous les registres du banc de registres régis par le système de fenêtrage : %l, %i et %o (les registres locaux, les registres input et les registres output).

Tableau III-XIII Résultats d'injections dans Decode-CWP

	# d'injection	# fautes silencieuses	% altération du programme
SEU	935	36	96,15%
MCU2	500	18	96,40%

Les fautes silencieuses surviennent exclusivement pendant le boot matériel, pour la simple raison que le fenêtrage n'est pas encore utilisé. C'est pour cette raison qu'il n'y a pas de grande différence au niveau du taux d'altération entre MCU2 et SEU, les 4% de fautes silencieuses correspondant à l'injection dans la phase du boot matériel.

III.7.2.4. Résultats d'injection dans le banc de registres

Le banc de registres étant configuré dans plus de 400 LUTs il nous a été difficile de repérer précisément les registres utilisés dans le banc de registres. La localisation est d'autant plus compliquée par le système de fenêtrage qui fait que mis à part les registre globaux %g les autres registres ne sont pas fixes. Pour ces raisons nous avons décidé d'injecter des fautes de type MCU5 dans les LUTs configurant le banc de registres, mais sans cibler des cycles précis pour chaque registre. Les résultats de cette campagne sont présentés dans le tableau III-XIV.

Tableau III-XIV Résultats d'injections dans le banc de registres

	# d'injection	# fautes silencieuses	% altération du programme
MCU5	3929	3310	15,75%

Pour comprendre ces résultats, 3 considérations doivent être prises en compte : (1) seulement le quart du banc de registres est utilisé par le Leon3 dans le prototype implanté, (2) l'application AES n'a recourt qu'à la moitié du banc de registres utilisé, (3) l'injection n'a d'effet sur l'application que lorsque l'élément mémoire fauté appartient à un registre de la fenêtre en cours d'utilisation. Ceci nous amène à relativiser ce taux et nous permet de considérer que le taux d'altération suite aux injections dans le banc de registres est considérable.

III.7.3. Leon3 protégé avec la méthode IDSM

Le processeur Leon3 a été protégé dans un second temps avec la méthode de vérification de flot de contrôle IDSM **[Berg.-09]**, conçue et développée au sein du laboratoire TIMA Pour cela, nous avons greffé au processeur Leon3 un coprocesseur de surveillance (ou « watchdog ») pour vérifier qu'il effectue les bonnes instructions dans le bon ordre.

Le watchdog surveille le flux circulant entre l'unité d'exécution du processeur et sa mémoire cache, à chaque cycle de l'exécution de l'application à vérifier (dans notre cas d'étude, cette application est toujours l'algorithme de chiffrement AES). Un schéma simplifié de l'architecture cible est fourni dans la figure 3-18.

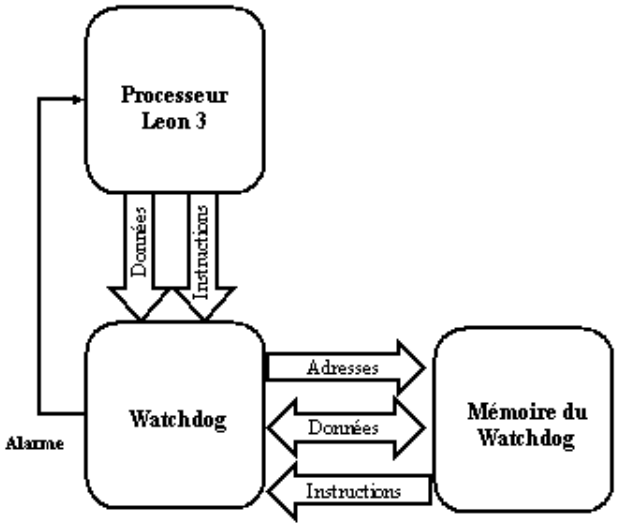


Figure 3-18. Architecture d'un système protégé avec IDSM

Une fois le watchdog ajouté au LEON3, ce système (LEON+watchdog) a été implanté sur un FPGA Xilinx de la famille Virtex V. Les résultats détaillés des taux d'occupation du FPGA Virtex V, par le Leon3 et le watchdog, en nombre de bascules, de LUTs et de blocs Ram utilisés sont présentés dans le tableau III-XV.

Tableau III-XV-Occupation du FPGA Virtex 5 par les Leon3 et le Watchdog

		Leon3mp		Watchdog	
		#	% Virtex5	#	% Virtex5
Bascules		3503	5%	862	1,25%
LUTs	Total	6704	9%	2016	2,92%
	Logique	6294	9%	1976	2,86%
	Mémoire	410	2%	40	0,06%
Bloc RAM		6	4%	0	0,00%

Dans cette partie nous allons reprendre les registres visés dans la première partie de l'étude du processeur Leon3 en ignorant les fautes silencieuses que nous avons vu précédemment. Les catégories présentées dans cette partie seront :

- Exception : cette catégorie correspond à la « Défaillance» dans l'ancienne classification, les exceptions sont un comportement imprédictible du Leon et ce sont des erreurs que la méthode de vérification de flot de contrôle ne peut pas gérer. En effet, il arrive qu'après injection de fautes, le Leon3 saute à une mauvaise adresse, puis à une adresse dans la "trap table" et pour finir stoppe avant que le watchdog n'ait pu s'en apercevoir. Ces cas, bien qu'indétectables par la méthode, seront répertoriés et inclus dans le calcul du taux de détection car (1) ils correspondent à un mauvais comportement du processeur, que nous avons identifié grâce à nos expériences et qui devrait être corrigé et (2) cela pourrait être détecté par un Timer.
- Erreur : cette catégorie regroupe toutes les injections ayant conduit à une altération du déroulement de l'application mis à part la catégorie « Défaillance», à savoir erreur de calcul, crash ou retard

La capacité de détection du watchdog se limite à la surveillance comportementale. En effet, ce dernier, dans sa conception actuelle, ne peut pas détecter les fautes dont les conséquences sont temporelles et se traduisent par des délais d'exécution (délais positifs ou négatifs).

Le tableau III-XVI résume les résultats de détection des fautes dans le registre Fetch-PC.

Tableau III-XVI Résultats de détection de fautes dans FETCH-PC avec la méthode IDSM

		Détecté	Non détecté	Exception	% de détection	% d'exception
MBU5	Boot matériel	5	34	1	**12,5%**	**2,5%**
	Boot Logiciel	72	14	17	**69,9%**	**16,5%**
	Pgm Principal	342	35	39	**82,21%**	**9,38%**
	Total	419	83	57	**74,96%**	**10,2%**
MBU4	Boot matériel	5	22	0	**18,52%**	**0%**
	Boot Logiciel	69	15	15	**69,7%**	**15,15%**
	Pgm Principal	375	31	33	**85,42%**	**7,52%**
	Total	449	68	48	**79,47%**	**8,5%**
MBU3	Boot matériel	1	31	0	**3,13%**	**0%**
	Boot Logiciel	73	13	16	**71,57%**	**15,69%**
	Pgm Principal	374	24	25	**88,42%**	**5,91%**
	Total	448	68	41	**80,43%**	**7,36%**
MBU2	Boot matériel	2	34	0	**5,56%**	**0%**
	Boot Logiciel	83	15	10	**76,85%**	**9,26%**
	Pgm Principal	358	26	5	**92,03%**	**1,29%**
	Total	443	75	15	**83,11%**	**2,81%**
SEU	Boot matériel	0	25	1	**0%**	**3,85%**
	Boot Logiciel	86	12	4	**84,31%**	**3,92%**
	Pgm Principal	362	21	11	**91,88%**	**2,79%**
	Total	448	58	16	**85,82%**	**3,07%**

Le meilleur taux de détection, pour toutes les multiplicités, est obtenu pendant le programme principal, jusqu'à 92,03% pour des fautes MBU2. Nous remarquons que plus la multiplicité augmente plus le taux de détection global diminue, entre 74,96% pour les MBU5 et 85,82% pour les SEU. Toutefois, le taux de détection reste élevé même pour de grandes multiplicités alors que la plupart des méthodes ne peuvent pas gérer des erreurs multiples. Quelques fautes injectées pendant le boot matériel ont été détectées, bien que le watchdog ne soit pas opérationnel pendant cette phase. Ceci vient essentiellement du fait que les effets de ces fautes se sont prolongés au delà de cette phase, atteignant la phase de boot logiciel. Nous enregistrons pour la multiplicité 5 le meilleur taux de détection dans le boot matériel à savoir 12,5%. Pour les multiplicités 3 et 4, les exceptions dépassent 50% des injections non détectées pendant le boot logiciel et l'exécution du programme principal.

Les résultats présentés dans le tableau III-VII montrent que le taux de détection dans CWP, avoisine les 80% et 90% pour les MBU2 et SEU respectivement. Le taux de détection des

injections de fautes de multiplicité 2 pendant le boot logiciel dépasse celui relatif au programme principal.

Tableau II-XVII Résultats de détection de fautes dans le Decode-CWP avec la méthode IDSM

		Détecté	Non détecté	% de détection
MBU2	Boot matériel	0	0	-
	Boot Logiciel	88	23	**79,28%**
	Pgm Principal	288	83	**77,63%**
	Total	376	106	**78,01%**
SEU	Boot matériel	0	0	-
	Boot Logiciel	177	29	**85,92%**
	Pgm Principal	606	87	**87,45%**
	Total	783	116	**87,1%**

Pour ce registre nous n'avons pas enregistré d'exception. Mais le taux de détection est affaibli par le fait que le changement de la valeur du CWP entraine l'altération des registres locaux, d'input et d'output du banc de registres. Plusieurs registres parmi ceux-ci n'auront pas sur le flot de contrôle.

Dans le registre D-INST, la détection augmente selon la phase de l'exécution du Leon3, comme illustré dans le tableau III-XVIII.

Tableau III-XVIII- Résultats de détection de fautes dans le Decode_INST avec la méthode IDSM

		Détecté	Non détecté	Exception	% de détection	% d'exception
MBU5	Boot matériel	1	22	0	**4,35%**	**0,00%**
	Boot Logiciel	54	39	42	**40,00%**	**31,11%**
	Pgm Principal	450	13	86	**81,97%**	**15,66%**
	Total	505	74	128	**71,43%**	**18,10%**
MBU4	Boot matériel	1	9	1	**9,09%**	**9,09%**
	Boot Logiciel	20	19	39	**25,64%**	**50,00%**
	Pgm Principal	231	7	66	**75,99%**	**21,71%**
	Total	252	35	106	**64,12%**	**26,97%**
MBU3	Boot matériel	0	11	1	**0,00%**	**8,33%**
	Boot Logiciel	15	12	20	**31,91%**	**42,55%**
	Pgm Principal	181	8	80	**67,29%**	**29,74%**
	Total	196	31	101	**59,76%**	**30,79%**
SEU	Boot matériel	0	6	0	**0,00%**	**0,00%**
	Boot Logiciel	20	18	10	**41,67%**	**20,83%**
	Pgm Principal	114	15	45	**65,52%**	**25,86%**
	Total	134	39	55	**58,77%**	**24,12%**

Pendant le boot matériel nous notons les pires taux de détection allant de 0% à 9% en fonction de la multiplicité de l'erreur. Pendant la phase d'initialisation, la méthode IDSM permet de détecter entre 30% et 40% des erreurs, cependant, durant cette phase, le taux d'exceptions est très élevé, ce qui a causé une détection moins performante qu'espéré. Et pour finir, la phase de l'exécution du programme principal (main), semble être la phase dans laquelle la détection est la plus performante sans oublier que ce taux est faussé par le taux assez élevé des exceptions (entre 18 et 29%).

Les résultats du tableau III-XIX montrent les résultats des injections de SEU dans les LUTs configurant le banc de registres du Leon3.

Tableau III-XIX Résultats de détection de fautes dans le banc de registres avec la méthode IDSM

		Détecté	Non détecté	Exception	% de détection	% d'exception
SEU	Boot matériel	0	4	1	0,00%	25,00%
	Boot Logiciel	40	9	24	54,79%	48,98%
	Pgm Principal	110	824	187	9,81%	20,02%
	Total	119	868	203	10,00%	20,57%

Les résultats du tableau III-XIX montrent que le banc de registres souffre d'un taux de détection bas, car il comprend plusieurs registres qui n'entrent essentiellement que dans le flot de données. Le taux de détection peut donc être considéré comme respectable surtout qu'une grande proportion des erreurs non détectées est due à des exceptions.

Nous allons finir notre étude par le tableau III-XX qui résume le résultat d'injections dans les bits de configuration, pour deux campagnes de ce type: la première avec des injections de SEU et la deuxième avec des MBU5.

Tableau III-XX Résultats de détection de fautes dans la configuration avec la méthode IDSM

		Détecté	Non détecté	Exception	% de détection	% d'exception
MBU3	Boot matériel	14	50	5	20,29%	7,81%
	Boot Logiciel	67	166	28	25,67%	12,02%
	Pgm Principal	213	596	102	23,38%	12,61%
	Total	294	812	135	23,69%	12,21%
SEU	Boot matériel	2	56	1	3,39%	1,72%
	Boot Logiciel	52	105	19	29,55%	12,10%
	Pgm Principal	84	383	52	16,18%	11,13%
	Total	138	544	72	18,30%	10,56%

Contrairement aux injections dans les registres, le meilleur taux de détection est obtenu pendant la phase du boot logiciel pour les SEU. Ceci peut être expliqué par le fait que l'injection a été faite, pendant cette phase là, mais qu'elle n'a eu d'impact réel que dans la phase suivante. Il ne faut pas oublier que l'injection dans les bits de configuration ont des répercussions récurrentes sur l'application. Si nous ne tenons par compte des erreurs d'exception, le taux de détection reste assez convenable par rapport à une méthode de flot de contrôle, car à la base elle n'a pas été conçue pour détecter des erreurs dans les bits de configuration. Toutefois, les résultats montrent qu'une telle méthode ne peut pas être une bonne approche pour se protéger efficacement d'erreurs dans la configuration du FPGA.

Une méthode de durcissement à proposer doit donc, contrairement à la méthode IDSM, prendre en compte les défaillances et les délais, ou bien garantir une détection sans latence. Même si la méthode IDSM est la seule méthode de ce type qui à notre connaissance ne néglige pas le boot logiciel, la méthode de protection à concevoir pour une protection sur FPGA SRAM doit garantir un meilleur taux de détection, indépendamment de l'instant d'injection (pendant le boot matériel, le boot logiciel ou le programme principal).

III.8. Conclusion

Dans ce chapitre, nous avons présenté une nouvelle méthodologie d'injection de fautes, basée sur l'abstraction de motifs d'erreurs, obtenus à partir d'expériences de pré-caractérisation physiques. Cette méthode optimise les temps d'injections, car elle réduit l'infinité de combinaisons de bits dans lesquelles nous pouvons injecter les fautes multiples, en une liste de motifs d'erreurs simultanées réalistes (une base de données de plus de 5000 motifs pour notre exemple), pouvant être très complexes en termes de multiplicité d'erreurs. Elle tire ensuite profit de la régularité structurelle et géométrique des FPGAs pour rendre génériques les motifs d'erreurs observés sur une zone réduite du FPGA. Notre méthodologie d'abstraction a été ensuite couplée à un environnement d'injection de fautes dans un FPGA via l'ICAP. Cet environnement réduit les communications entre le FPGA et le PC hôte, grâce à la technique d'endo-reconfiguration, ce qui nous fait gagner en performances.

Une étude comparative entre les mécanismes de reconfiguration de 3 plateformes a été également réalisée. Elle a montré que l'utilisation de dispositifs récents et/ou de technologies de fabrication plus avancées n'apporte pas nécessairement des avantages significatifs de performance pour les injections de fautes à base d'émulation. Cela peut être vrai dans le contexte d'une émulation autonome qui peut bénéficier d'une augmentation de la fréquence

d'horloge, mais qui peut également souffrir de limitations telles que le manque de flexibilité et la complexité de l'évolution. Lorsque l'injection est commandée par un processeur intégré, l'amélioration de la technologie n'est pas nécessairement bénéfique puisque les avantages peuvent être supprimés par d'autres changements (par exemple, augmentation de la taille des LUTs dans les dispositifs de la nouvelle génération). Dans le contexte de l'injection de fautes, les LUTs de petite taille sont plus efficaces lorsque la reconfiguration est basée sur les frames. En outre, les processeurs synthétisables semblent moins intéressants pour les performances de l'endo-reconfiguration.

Nous avons exploré pour la première fois l'utilisation d'un dispositif à adressage direct et nous avons montré les gains importants en temps de reconfiguration, produits par ce type de dispositifs. Bien que le FPGA AT40K, utilisé dans ces expériences, soit nettement plus petit que les autres, il peut être supposé que les performances en temps de reconfiguration d'un tel dispositif ne vont pas être détériorées lorsque l'on augmente la taille du composant (l'adressage direct ne sera pas aussi sensible que la configuration en série lors de la mise à l'échelle). Nous pensons donc que l'adressage direct est une caractéristique très attrayante pour les plateformes utilisées dans le cadre des injections de fautes à base d'émulation, bien que les plateformes Xilinx soient aujourd'hui mieux adaptées pour l'évaluation de la robustesse des grands circuits.

Deux études de cas ont aussi été présentées. La première, basée sur le processeur Leon2, a permis de montrer sur une plateforme Virtex II Pro que les motifs d'erreurs réalistes conduisent à des évaluations de robustesse moins optimistes que les modèles d'erreurs classiques. Elle a aussi montré qu'une méthode de protection adaptée à un ASIC ne permet pas de satisfaire convenablement les contraintes de robustesse lors d'une implantation sur FPGA SRAM et que les erreurs de configuration doivent donc être gérées selon d'autres principes. La deuxième étude de cas a ciblé un processeur Leon3 surveillé par un Watchdog sur une plateforme Virtex V. Durant cette étude de cas, nous avons expérimenté une nouvelle méthode d'injection de fautes indirecte dans les bascules. Cette étude a par ailleurs démontré certains dysfonctionnements dans le Leon3 et, outre l'évaluation d'efficacité de la méthode de vérification de flot de contrôle, a permis de constater à nouveau qu'une approche fonctionnelle ne permet pas d'obtenir un taux de couverture élevé face à des erreurs de configuration.

Un autre type de méthode de protection va donc être présenté dans le chapitre IV. Cette méthode vise à augmenter la robustesse face aux erreurs dans la configuration, mais avec un coût très réduit par rapport aux méthodes traditionnelles. Notre méthodologie d'injection de fautes nous permettra de valider le principe proposé.

Chapitre IV:

Approche de protection pragmatique de systèmes implantés sur FPGA SRAM

Ce dernier chapitre a pour but l'implantation et l'évaluation d'une solution de protection des circuits implantés sur deux FPGA SRAM sélectionnés. Cette solution doit mettre à profit les caractéristiques des architectures cibles pour diminuer les coûts de protection.

Une présentation générale de l'approche fera l'objet du paragraphe IV.1. Elle sera suivie de deux études pour l'adaptation de l'approche sur les architectures Xilinx Virtex et Altera Stratix, respectivement dans le paragraphe IV.2 et IV.3. Avant de clore le chapitre, nous discuterons des pistes explorées pour l'amélioration de l'approche.

IV.1. Présentation générale de l'approche

Les campagnes d'injection de fautes réalisées nous ont démontré que l'injection de fautes dans les bits de configuration (et plus particulièrement dans la logique combinatoire), en SEU et surtout en MBF, est plus invasive et plus perturbante pour le bon déroulement de l'application que les injections dans les bascules utilisateur. Le but de notre approche est donc d'améliorer la robustesse d'une application face aux fautes dans la configuration, sans toutefois utiliser les approches classiques présentées au chapitre I, qui induisent des coûts très élevés en ressources et en consommation. Notre objectif n'est pas de concurrencer une protection par triplication, mais au contraire d'améliorer la résistance aux fautes à faible coût, voire à coût nul. La cible concerne donc des systèmes embarqués devant avoir la meilleure qualité possible et donc la meilleure fiabilité possible sans pour autant concerner des systèmes dits "critiques".

Dans un premier temps, notre approche tire profit des caractéristiques des architectures cibles, afin de répliquer les fonctions qui peuvent être placées dans une seule cellule logique. Cette duplication partielle est conditionnée par l'inutilisation, dans l'application, des ressources qui implémentent le duplica de la fonction à protéger donc n'induit pas de coût réel d'implantation au niveau des ressources (même si la consommation du circuit peut en être augmentée).

Dans un deuxième temps, la logique de détection (comparaisons puis combinaison des signaux d'alarme) est implantée dans des LUTs supplémentaires nous permettant d'obtenir un signal

final de sortie pour la détection des erreurs. Ceci se traduit par l'implantation de la logique de détection comme suit (pour des LUT à 6 entrées) :

- Des comparateurs à 6 entrées qui comparent chacun trois paires de fonctions/réplicas
- Des détecteurs d'erreur pour combiner la sortie de 6 comparateurs ou autres détecteurs.

Il faut noter que les LUT supplémentaires peuvent dans de nombreux cas être présents mais inutilisés dans le FPGA. En effet, une famille de FPGAs se compose d'une série de composants de tailles discrètes ; il est relativement rare que la capacité du FPGA (ou du moins la capacité réellement utilisable lors du placement routage) corresponde exactement aux ressources nécessaires pour l'application. Les LUT inutilisées, mais accessibles pour le routage, peuvent donc permettre dans certains cas une implantation complète de la protection à coût nul, pour tout le circuit implanté sur le FPGA. Dans d'autres cas, les LUT supplémentaires peuvent induire un surcoût, mais ce surcoût n'est pas forcément visible au niveau de l'application. En effet, l'approche peut être limitée à un ou plusieurs blocs fonctionnels plus critiques que les autres. Dans ce cas, un surcoût au niveau du bloc protégé peut être absorbé au niveau de l'application par une répartition différente des budgets de ressources entre les différents blocs du circuit. Bien sûr, cela implique en général d'avoir un surcoût relativement faible au niveau du bloc protégé.

Le flot de principe pour le durcissement est illustré par la figure 4-1.

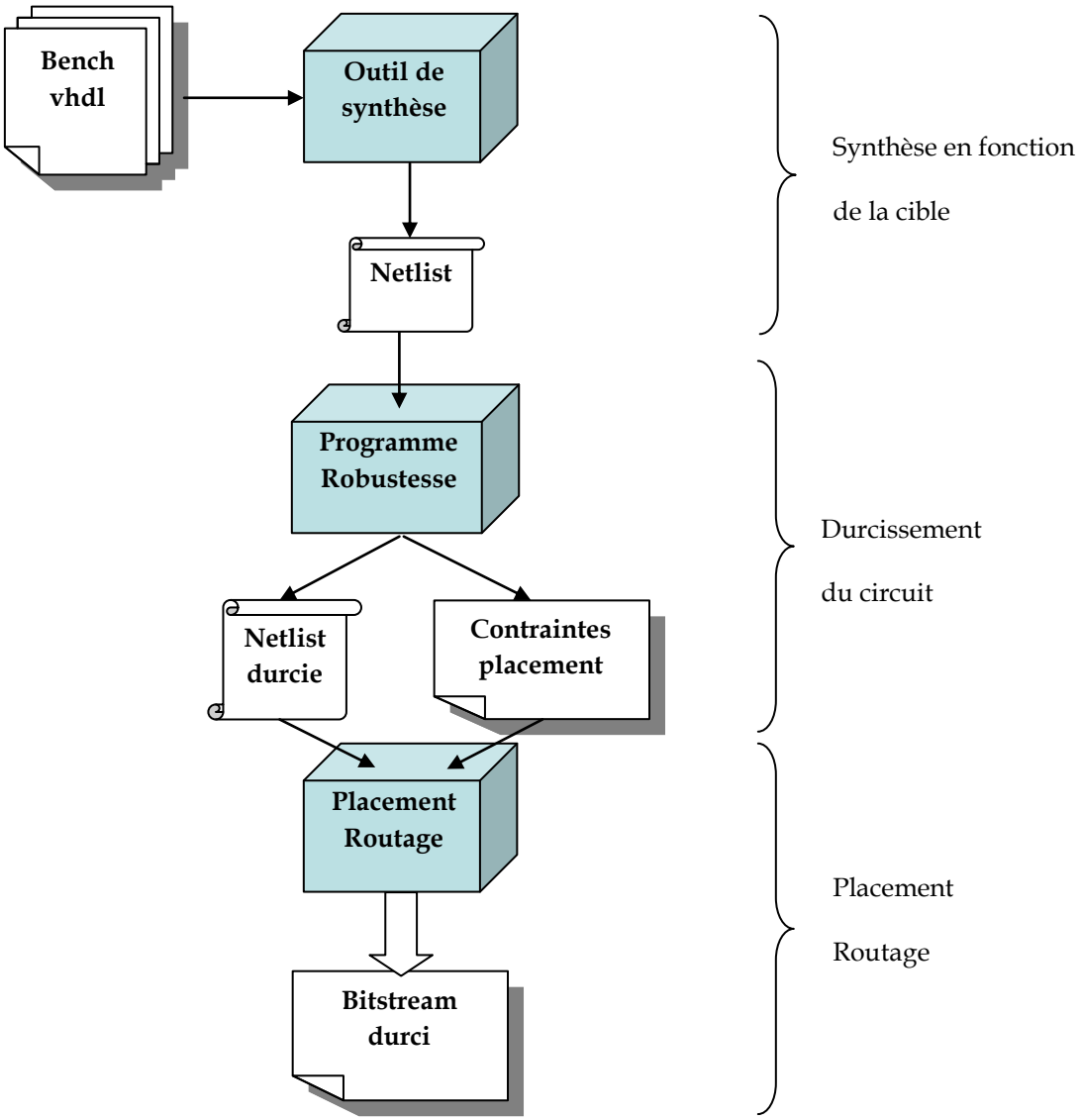


Figure 4-1. Flot de principe pour le durcissement

Nous allons nous intéresser dans la suite aux architectures des FPGAs Virtex V et Stratix IV et à la démarche à suivre pour exploiter au mieux les avantages que peuvent offrir chacune de ces deux plateformes.

Notre approche prendra en compte les faiblesses des protections sur les bitstreams fournis par les constructeurs qui sont généralement des codes correcteurs. En effet ces codes correcteurs sont essentiellement limités par leur capacité à gérer des erreurs multiples et impliquent des délais de détection importants dus aux calculs périodiques. Notre approche permettra de détecter immédiatement des erreurs multiples.

IV.2. Adaptation de la méthode sur une architecture Virtex

IV.2.1. Présentation des caractéristiques Virtex favorables à l'approche

La logique combinatoire des Virtex V est composée en apparence de LUTs à 6 entrées mais en réalité chaque LUT à 6 entrées est formée par deux LUTs à 5 entrées communes et deux sorties distinctes, comme illustré dans la figure 4-2.

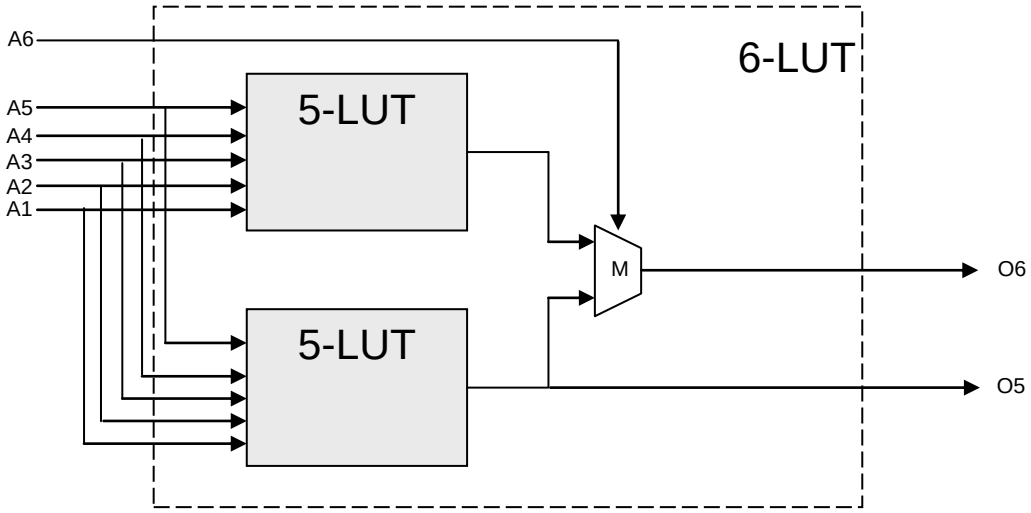


Figure 4-2. LUT à 6-entrées Virtex V

Lors d'une étude de faisabilité, nous avons remarqué que dans 90% des cas seule la première sortie est utilisée ce qui signifierait que la deuxième LUT n'est pas mise en œuvre dans l'implantation du circuit. Cette deuxième LUT pourrait donc servir comme réplica "gratuit" de la fonction originale (voir figure 4-3).

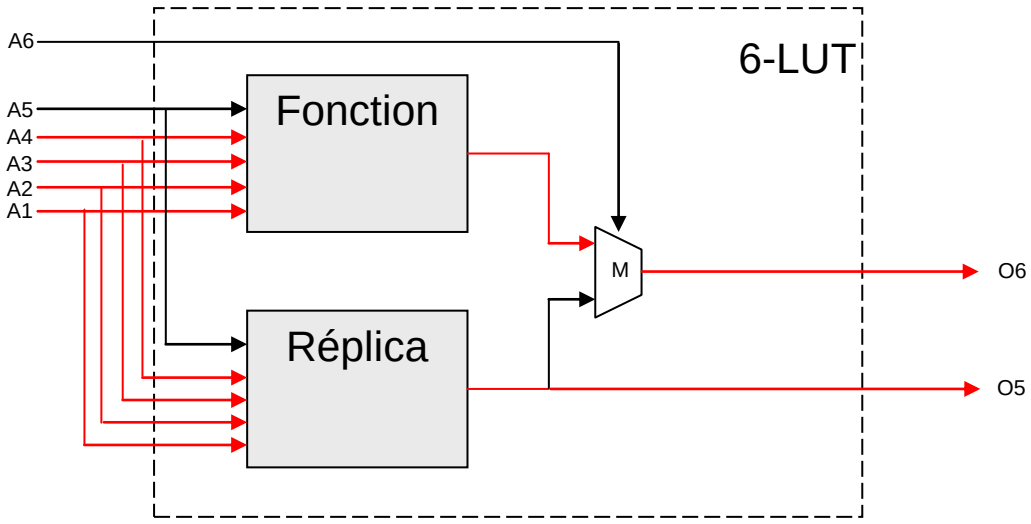


Figure 4-3. Approche de protection sous Virtex V

La condition majeure pour qu'une fonction puisse être protégée dans la même LUT à 6 entrées que celle qui l'implémente est que cette fonction ait moins de 6 entrées. De cette manière la première partie de la LUT implémentera la fonction originale et la deuxième partie fera office de réplica, ces deux LUTs se partageront les mêmes entrées.

Le tableau IV-II donne une évaluation des taux de protection que nous pouvons espérer avec ce principe. Nous avons pour cela utilisé des circuits précédemment présentés tels que les processeurs Leon2 et Leon3 ainsi que les benchmarks ITC'99 présentés dans le tableau IV-I.

Tableau IV-I. Description des benchmarks ITC

Nom	Fonction
b01	Une FSM qui compare des flux en série
b02	Une FSM qui reconnaît les numéros décimaux codés binaire (BCD)
b03	Arbitrage de ressources
b04	Calcul de min et de max
b05	Élaboration le contenu d'une mémoire
b06	Interruption de gestionnaire
b07	Compteur de points sur une ligne droite
b08	Repérage d'inclusions dans des séquences de nombres
b09	Convertisseur série-série
b10	Système de vote
b11	Brouilleur de chaîne de caractères avec chiffrement variable
b12	Jeu 1 joueur (prédiction d'une séquence)
b13	Interface pour capteurs météos
b14	Sous ensemble du processeur Viper
b15	Sous ensemble du processeur 80386

Au vu de cette évaluation, nous pouvons conclure que la majeure partie des applications étudiées est implantée sur des ressources logiques partiellement exploitées (jusqu'à 100% pour b02 et b06). L'application b12 constitue le pire cas, avec l'utilisation de 36,86% seulement de LUTs pouvant être dupliquées sans surcoût.

Tableau IV-II Evaluation initiale de l'approche sur V5

	Application	nb LUTs/Application	LUTs dupliquées sans coût	Couverture
ITC	**b01**	10	9	90%
	b02	4	4	100%
	b03	38	22	57,89%
	b04	114	83	72,81%
	b05	184	88	47,83%
	b06	8	8	100%
	b07	99	60	60,61%
	b08	23	11	47,83%
	b09	49	41	83,67%
	b10	38	19	50%
	b11	100	55	55%
	b12	255	94	36,86%
	b13	49	35	71,43%
	b14	1161	602	51,85%
	b15	1712	708	41,35%
Divers	**Multiplieur**	93	46	49,46%
	Filtre fir	238	171	71,85%
	Leon3	6857	2960	43,17%
	Leon2	4063	1838	45,24

IV.2.2. Résultats de l'application de la méthode sur une architecture Xilinx

A partir des fichiers VHDL, nous générons pour une cible V5 une première netlist avec l'outil ISE 12.3. Cette netlist est par la suite convertie en une netlist en format texte, modifiée avec notre programme de durcissement de telle sorte que pour les LUTs à 6 entrées dont la deuxième sortie n'est pas utilisée, un réplica de la fonction originale est implanté, comme décrit dans le paragraphe IV.2.1.

Par ailleurs, notre programme de durcissement génère un deuxième fichier, à savoir le fichier de contraintes de placement dont l'extension est UCF.

Enfin, nous intégrons la netlist durcie dans le circuit complet (ou notre plateforme d'injections de fautes pour nos expériences) avec l'outil EDK 12.3, en plaçant le circuit durci selon les contraintes de placement routage générées. Toute la procédure est résumée par la figure 4-4.

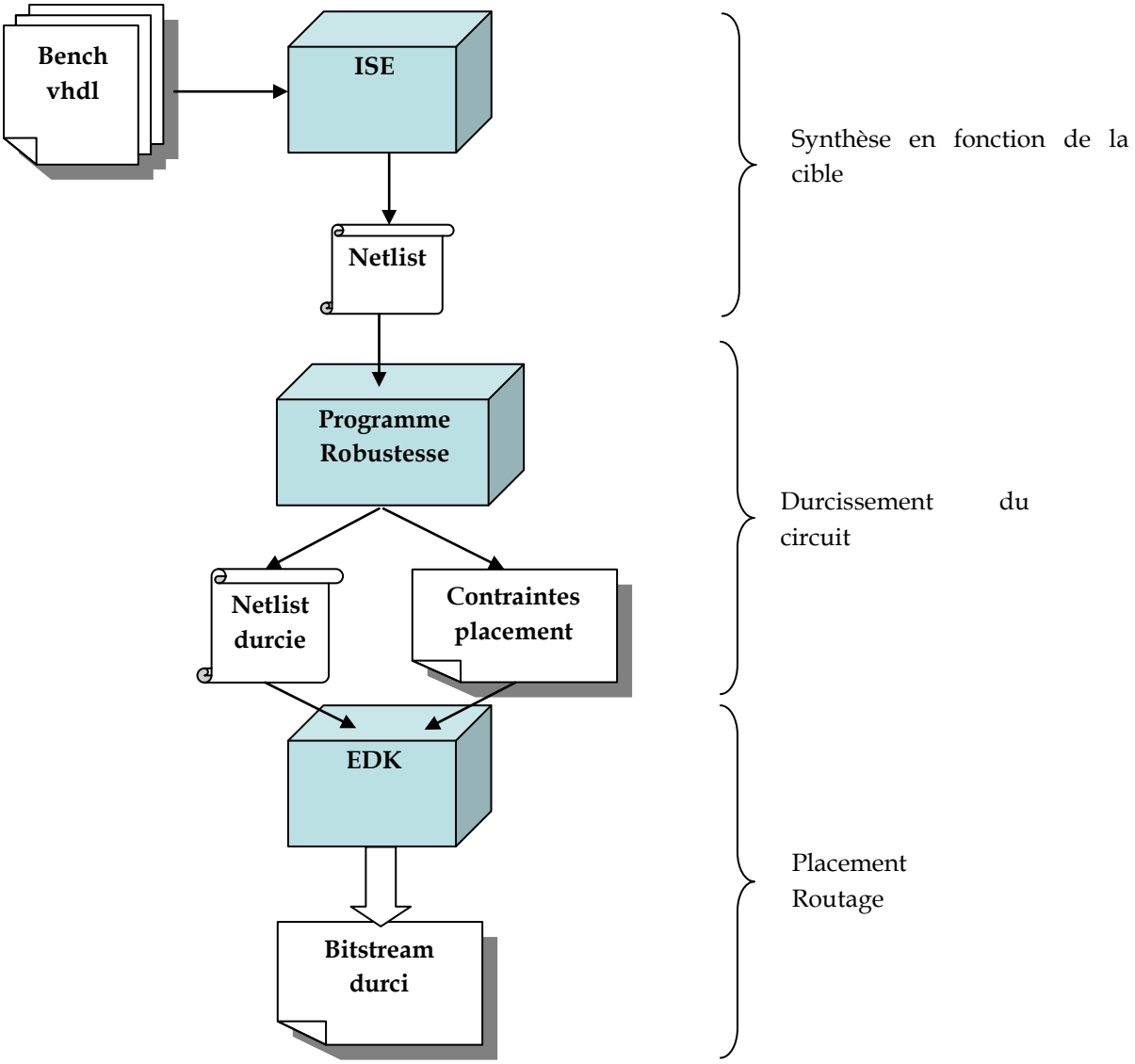


Figure 4-4. Flot de durcissement pour la famille Virtex5

Pour qu'une netlist soit correctement durcie par notre programme, il est à noter que quelques précautions doivent être prises. En effet, nous devons nous assurer qu'au moment de la génération de la netlist originale, celle-ci est créée en ignorant la hiérarchisation des entités de l'application, afin avoir une seule entité au lieu de plusieurs entités hiérarchisées.

Nous avons appliqué notre approche sur les applications énumérées dans le tableau IV-III. Par la suite, nous avons implanté la plupart de ces circuits sur notre plateforme d'injection de fautes pour valider la détection, en injectant des motifs d'erreur de multiplicités allant de 1 à 5 dans les fonctions combinatoires du circuit protégé.

Pour les campagnes d'injection dans ce chapitre nous garantissons une marge d'erreur de 10% et un taux de confiance de 95%.

Tableau IV-III Validation par injection de fautes sur V5

	Application	% LUTs protégées	% Surcoût détection	Injection de fautes (motifs dans LUT)	% Détection
ITC	**b01**	90%	40%	100	87,10%
	b02	100%	50%	16	100%
	b03	57,89%	26,32%	1444	59,40%
	b04	72,81%	29,82%	12996	70,20%
	b05	47,83%	19,57%	33856	42,90%
	b06	100%	50%	64	100%
	b07	60,61%	25,25%	9801	60,14%
	b08	47,83%	21,74%	529	52,21%
	b09	83,67%	32,65%	-	-
	b10	50%	23,68%	1444	52,40%
	b11	55%	24 %	10000	53,71%
	b12	36,86%	15,29%	65025	37,95%
	b13	71,43%	30,61%	2401	79,29%
	b14	51,85%	20,84%	234792	50,74%
	b15	41,35%	16,8%	-	-
Divers	**Multiplieur**	49,46%	21,51%	8649	50,69%
	Filtre fir	71,85%	29,41%	-	-
	Leon3	43,17%	17,30%	-	-
	Leon2	45,24%	18,16%	-	-

Outre les Benchmarks ITC, nous avons implanté notre approche sur 4 autres applications, à savoir, un Multiplieur 16bits, un filtre numérique FIR à réponse impulsionnelle finie, ainsi que les versions 2 et 3 du processeur Leon.

Le surcoût de protection lié à la logique de détection varie au niveau du bloc entre 30% et 50% du taux de couverture, en fonction de l'application protégée. Rappelons par comparaison qu'une approche classique par triplication a un surcoût plus de trois fois plus élevé que le taux de couverture maximum. Les résultats de détection obtenus lors des campagnes d'injection de fautes concordent avec le taux de couverture attendu (c'est-à-dire le pourcentage de LUT protégées, puisque les injections sont limitées aux LUTs), ce qui valide l'implantation, nous n'avons pas fait cette validation sur toutes les applications par manque de temps et parce que le nombre de circuits validés est assez significatif. Nous avons gardé les autres applications dans le tableau pour montrer le rapport couverture/surcoût de notre méthode.

Des campagnes avec des injections plus aléatoires, ciblant n'importe quel bit du bitstream d'une zone donnée, a été réalisée. Cette validation s'appuie sur l'injection de fautes dans des frames pour une zone relativement localisée. Ces campagnes d'injection de fautes ont été faites sur le processeur Leon3 exécutant le programme AES sur une population de 28237 cycles et de 921600 (bits) cibles. Les taux de confiance et marge d'erreur sélectionnés sont respectivement de 95% et 10%.

Nous avons mis en place deux catégories de campagnes, MBU et MCU :

- Injection de MBU y compris le cas particulier SEU : la frame d'un FPGA Virtex-5 est composé de 41 mots de 32 bits. Dans notre étude, un MBU consiste en plusieurs bit-flips dans un mot d'une frame donnée.
- Injection de MCU : nous considérons comme MCU les bit-flips dans des mots différents dans une ou plusieurs frames à condition que ces bit-flips ne soient pas trop éloignés sur le plan physique. Après la sélection aléatoire, dans l'ensemble de la zone implémentant le circuit à analyser, du premier bit à reconfigurer, les autres bit-flips du MCU sont sélectionnés aléatoirement mais dans une zone restreinte à 5 sous-frames de 3 mots chacune comme l'illustre la figure 4-5. La zone choisie est autant que possible conforme à la base de données de motifs d'erreur provoqués par un tir laser avec un spot de 8 µm.

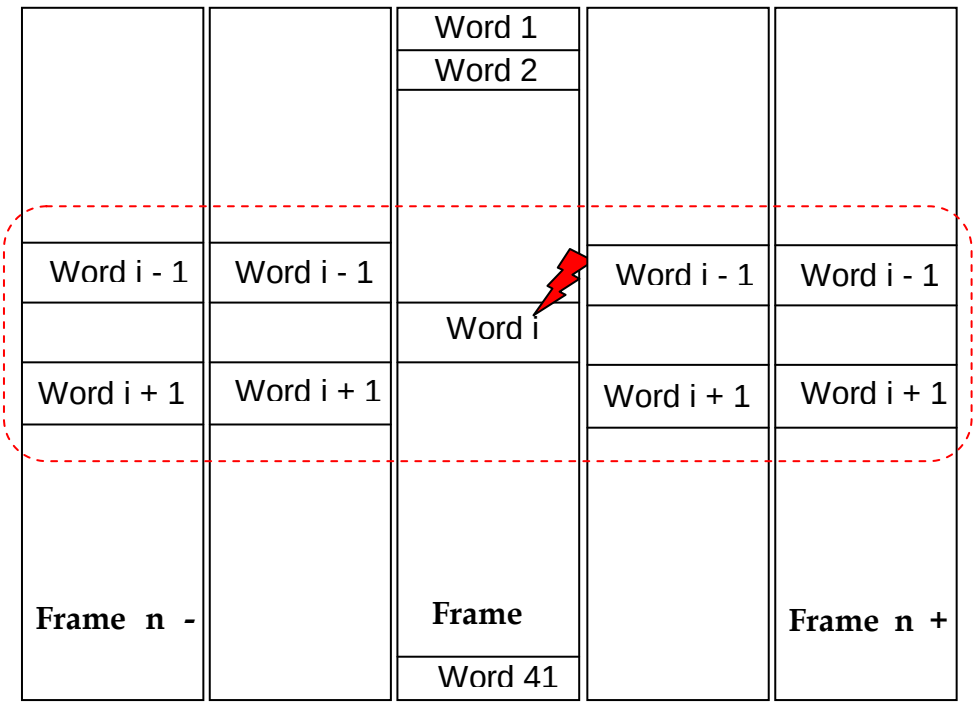


Figure 4-5. Sélection des bits à modifier par les MCUs

Dans les deux cas, notre choix d'injecter des MBF de multiplicités variant entre 2 et 8 bits est motivé par l'étude des campagnes de tirs Laser de 8 µm sur la famille Virtex 2 Pro, dont la répartition est présentée dans la figure 4-6. Pour la famille Virtex 5, de technologie plus fine, on peut supposer que le même genre d'effet, avec une multiplicité plus élevée, serait obtenu avec un spot d'attaque également réduit.

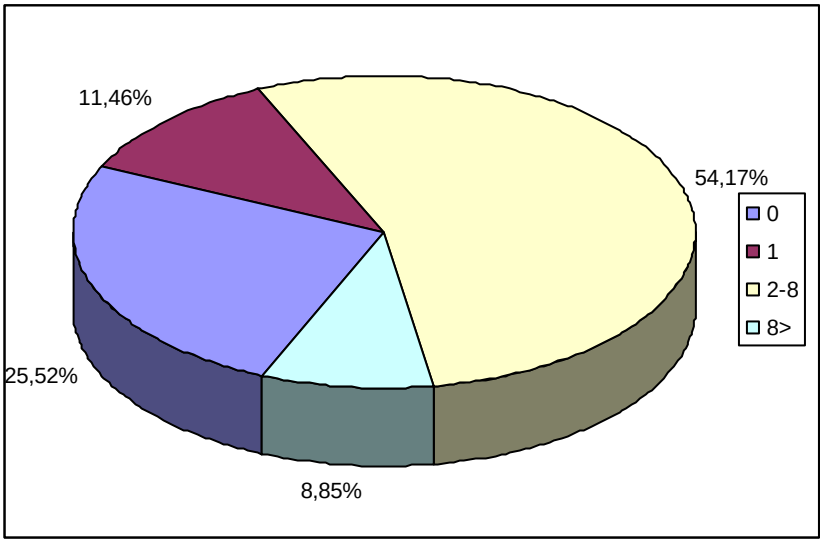


Figure 4-6. Répartition des tirs laser en fonction du nombre de bit-flips (Virtex II Pro, spot 8 µm)

Les résultats de ces campagnes d'injection de fautes sont présentés dans les tableaux IV-IV et IV-V.

Tableau IV-IV Résultats d'injections de MBU aléatoires dans tout le bitstream

	SEU	MBU : 2	MBU : 4	MBU : 6	MBU : 8
Nb de Fautes injectées	9192	9191	9189	9187	8985
% Silencieuse	88,22%	85,00%	80,74%	79,34%	79,03%
% Faux positif	1,01%	0,99%	1,40%	2,33%	2,67%
% Erreur de donnée	0,00%	0,00%	0,03%	0,13%	0,36%
% Détection Erreur de donnée	0,00%	0,00%	0,00%	0,00%	0,00%
% Défaillance	9,89%	12,59%	15,33%	14,91%	12,51%
% Détection Défaillance	0,88%	1,43%	2,49%	3,29%	5,43%

Tableau IV-V Résultats d'injections de MCU aléatoires dans tout le bitstream

	MCU : 2	MCU : 4	MCU : 6	MCU : 8
Nb de Fautes injectées	9193	9193	9193	9193
% Silencieuse	86,09%	71,89%	60,86%	60,72%
% Faux positif	1,08%	1,62%	2,56%	2,83%
% Erreur de donnée	0,00%	0,00%	0,12%	0,00%
% Détection Erreur de donnée	0,00%	0,00%	0,00%	0,00%
% Défaillance	12,00%	23,87%	32,86%	32,24%
% Détection Défaillance	0,84%	2,62%	3,60%	4,20%

Les résultats de ces campagnes nous confirment que le taux des fautes silencieuses pour les injections aléatoires dans les bits de configuration, même dans des frames configurant le circuit à analyser, reste élevé (allant de 60% à 86%) pour des MCUs de 2 à 8 bits et cela pour la raison que nous avons évoqué dans le chapitre II : la grande proportion des bits d'interconnexion qui n'altère pas nécessairement l'application. Nous allons par la suite nous concentrer donc sur les injections non silencieuses.

Une fois les injections silencieuses écartées, nous remarquons que plus de 10% des injections ayant conduit à une défaillance de l'application ont été détectées, ce pourcentage augmentant proportionnellement avec le nombre de bits inversés. En effet, plus le nombre de bits modifiés est grand plus nous avons de chances d'inclure une cellule protégée ou un élément de détection. Le porcentage est même nettement supérieur dans le cas des MBUs (jusqu'à 43% de défaillances détectées pour les MBUs de multiplicité 8). Les injections conduisant uniquement à des erreurs de données, c'est-à-dire à un résultat de chiffrement erroné, sont en revanche non détectées, mais très peu nombreuses. Le taux de faux positif, qui reste assez petit, est dû à l'injection d'erreurs soit dans la logique de détection et de propagation du signal d'erreur, soit dans les bits configurant les réplicas. Globalement, il apparaît donc que la méthode proposée, malgré un coût pouvant être nul en ressources, peut apporter une augmentation très sensible de robustesse pour l'ensemble des erreurs de configuration, et pas seulement pour les erreurs de configuration des LUTs. Ceci est notamment valable lorsque les erreurs sont multiples, cas qui n'est pas géré par les protections implantées par les fabricants directement dans les composants.

IV.3. Adaptation de la méthode sur une architecture Stratix

IV.3.1. Présentation des caractéristiques Stratix favorables à l'approche

La clé de la haute performance des FPGA Stratix IV réside dans les blocs ALMs, dont une présentation détaillée a fait l'objet du paragraphe II.3.1 du deuxième chapitre.

Comme nous l'avons présenté dans le paragraphe II.3.1, les ALMs disposent de 8 entrées avec une LUT fracturable qui peut être divisée en deux LUT adaptatives (ALUTs). Chaque ALM est capable de configurer une fonction à 7 entrées ou une combinaison de deux fonctions de 1 à 6 entrées chacune.

L'outil d'ALTERA Quartus II intègre cette option de fracturabilité pour optimiser les performances, l'efficacité, la puissance ainsi que la surface. Cependant, d'après une étude que nous avons menée, et résumée dans le tableau IV-VI, il y a en moyenne plus de 35% des ressources combinatoires disponibles mais initialisées et par conséquent gaspillées, ce taux

atteignant les deux tiers pour le bench b08. Ceci vient du fait que pour rajouter une deuxième fonction dans ces ALMs là, il faut rajouter des fonctions qui ont de 1 à 4 entrées communes avec la première fonction implantée. C'est cet aspect-là que nous allons tenter d'exploiter afin d'implanter la duplication de la fonction en question, de façon similaire à ce qui a été réalisé sur Virtex. Egalement de façon similaire, des ALMs supplémentaires sont utilisées pour implanter les comparateurs prenant en compte 3 paires de fonction/réplica et les détecteurs assurant la combinaison des signaux d'alarme.

Tableau IV-VI Evaluation initiale de l'approche sur Stratix4

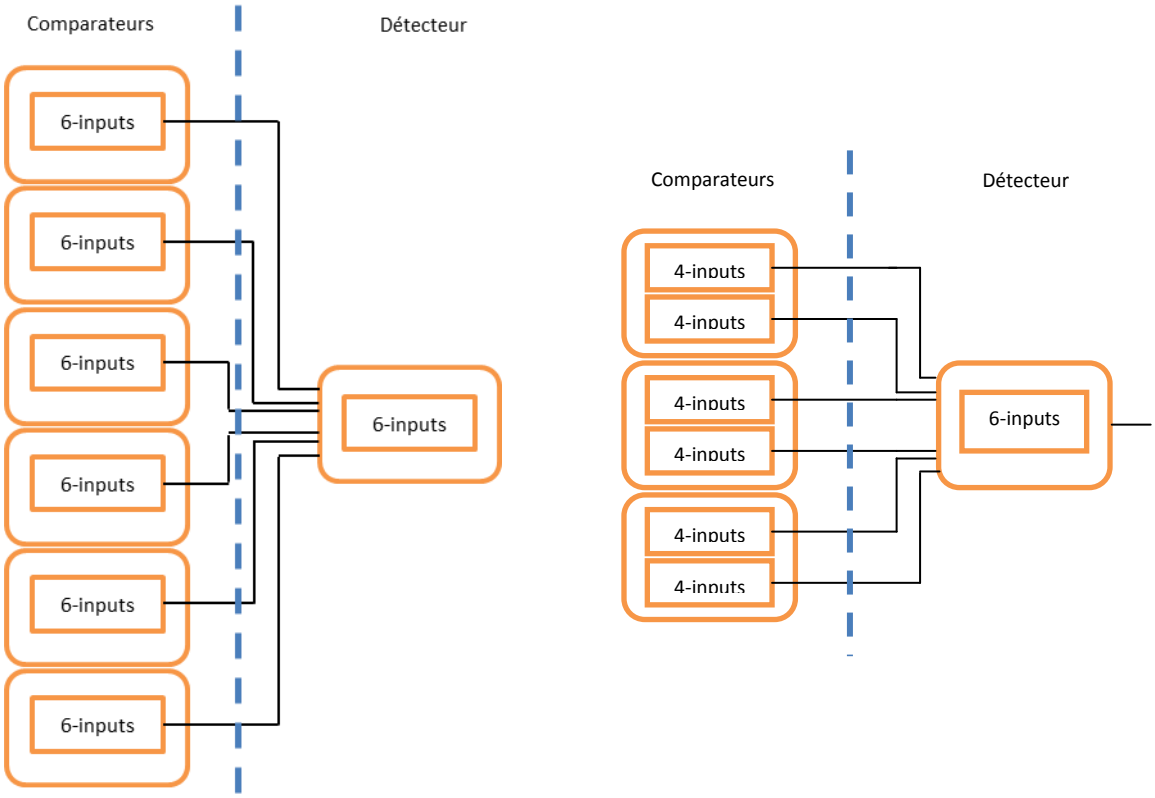
Application	Nb ALUT	Nb ALM	% ALUT inutilisée
b01	5	4	60,00%
b02	4	3	50,00%
b03	20	15	50,00%
b04	130	68	4,62%
b05	141	84	19,15%
b06	8	5	25,00%
b07	45	36	60,00%
b08	18	15	66,67%
b09	22	12	9,09%
b10	35	24	37,14%
b11	160	92	15,00%
b12	261	183	40,23%
b13	52	30	15,38%
b14	711	464	30,52%
b15	1403	806	14,90%
Moyenne			33,18%

Remarques sur la stratégie de détection adoptée :

Nous aurions pu implanter sur chaque ALM deux comparateurs faisant la comparaison entre 4 paires de fonction/réplica au lieu de 3. Mais cela aurait impliqué que nous aurions 2 sorties de comparaison pour 4 fonctions/réplicas et par conséquent 2 entrées dans un détecteur au lieu de 1 pour 3 fonctions/réplicas (voir figure 4-7).

- Cas 1 : Un comparateur à 6 entrées : 3 fonctions/réplicas par ALM ; une sortie pour cette ALM implique que les sorties de 6 ALM de comparateurs seront routées vers le détecteur => 6*3 = 18 fonctions protégées par un détecteur
- Cas 2 : Deux comparateurs à 4 entrée par ALM : 4 fonctions/réplicas par ALM ; deux sorties pour cette ALM implique que les sorties de seulement 3 ALM de comparateurs seront routées vers le détecteur => 4*3 = 12 fonctions protégées par un détecteur

➔ Ce que nous gagnons en comparateurs nous le perdons en détecteurs



(Cas 1) (Cas 2)

Figure 4-7. Comparaison des stratégies de détection

Par ailleurs, nous aurions pu procéder autrement pour l'implantation des détecteurs. En effet, nous aurions pu implanter ces détecteurs dans des extended ALUT (une LUT à 7 entrées) et de cette manière regrouper les signaux de 7 comparateurs au lieu de 6. Cependant, cette méthode possède l'inconvénient qu'avec ces extended LUT, l'ALM ne pourrait contenir qu'une bascule en plus de cette extended LUT, alors qu'avec des ALUT simples une seconde bascule ou une ALUT à deux entrées peut être placée dans la même ALM.

IV.3.2. Résultats de l'application de la méthode sur une architecture Stratix

Tout comme nous l'avons fait pour Virtex-V, la première étape dans notre démarche de durcissement sur Stratix nécessite la génération d'une netlist modifiable. Or l'outil Quartus II version 9.0 de Altera ne permet pas de générer ce genre de netlist ni de convertir les netlists binaires. Par conséquent, nous avons tenté deux approches :

❖ Utilisation de fonctionnalités de l'outil Quartus :

Nous avons essayé dans un premier temps d'utiliser l'outil Quartus pour appliquer tout le flot de durcissement. Pour cela nous nous sommes appuyés sur les outils propres à Quartus se basant sur des commandes TCL :

- L'éditeur d'assignements offre la commande « Manual Logic Duplication » permettant de dupliquer les composants du circuit, une fois identifiés ;
- « Resource Property Editor » permet d'inclure des composants supplémentaires au circuit existant. Néanmoins, rajouter des composants après la synthèse nécessite de passer en mode ECO (Engineering Change Orders). Dans ce mode, le concepteur peut apporter des modifications au circuit après la phase de placement et routage. Cette commande pourrait donc permettre de rajouter les comparateurs et les détecteurs, indispensables à notre approche ;
- Pour finir une troisième commande « set location assignment » permet de placer les nouveaux éléments.

L'idée est donc d'écrire un script TCL qui permet de gérer les contraintes du projet. Ce script identifie les composants à protéger, placés dans une ALM partiellement exploitée, pour y dupliquer la fonction originale. Bien évidemment, il ne faut pas oublier de rajouter la logique de détection, et ce en fonction du nombre de LUTs protégées. Toutefois, nous nous sommes heurtés à une contrainte qui nous a fait changer de piste : la commande « Manual Logic Duplication » nécessite la spécification des nœuds destination, en l'occurrence les composants de comparaison. Or, à ce stade, ces cellules de comparaison ne sont pas encore créées, car l'outil qui nous permet de les rajouter nécessite l'utilisation du mode ECO et ce dernier entre en jeu dans une étape ultérieure de la phase de placement routage. Par conséquent, la duplication échoue et l'outil de placement routage estime qu'il n'y a pas de chemin physique entre le nœud source et le nœud de destination.

Les détails des tentatives de la mise en place de la méthode de durcissement du circuit en mode ECO sont présentés dans l'Annexe C.

❖ Utilisation d'un outil Tiers : Précision

Nous avons donc décidé de procéder de la même manière que pour les circuits que nous durcissons sur Virtex V, mais en utilisant un autre outil de synthèse : Précision, de Mentor Graphics. Le flot de durcissement adopté est illustré par la figure 4-8

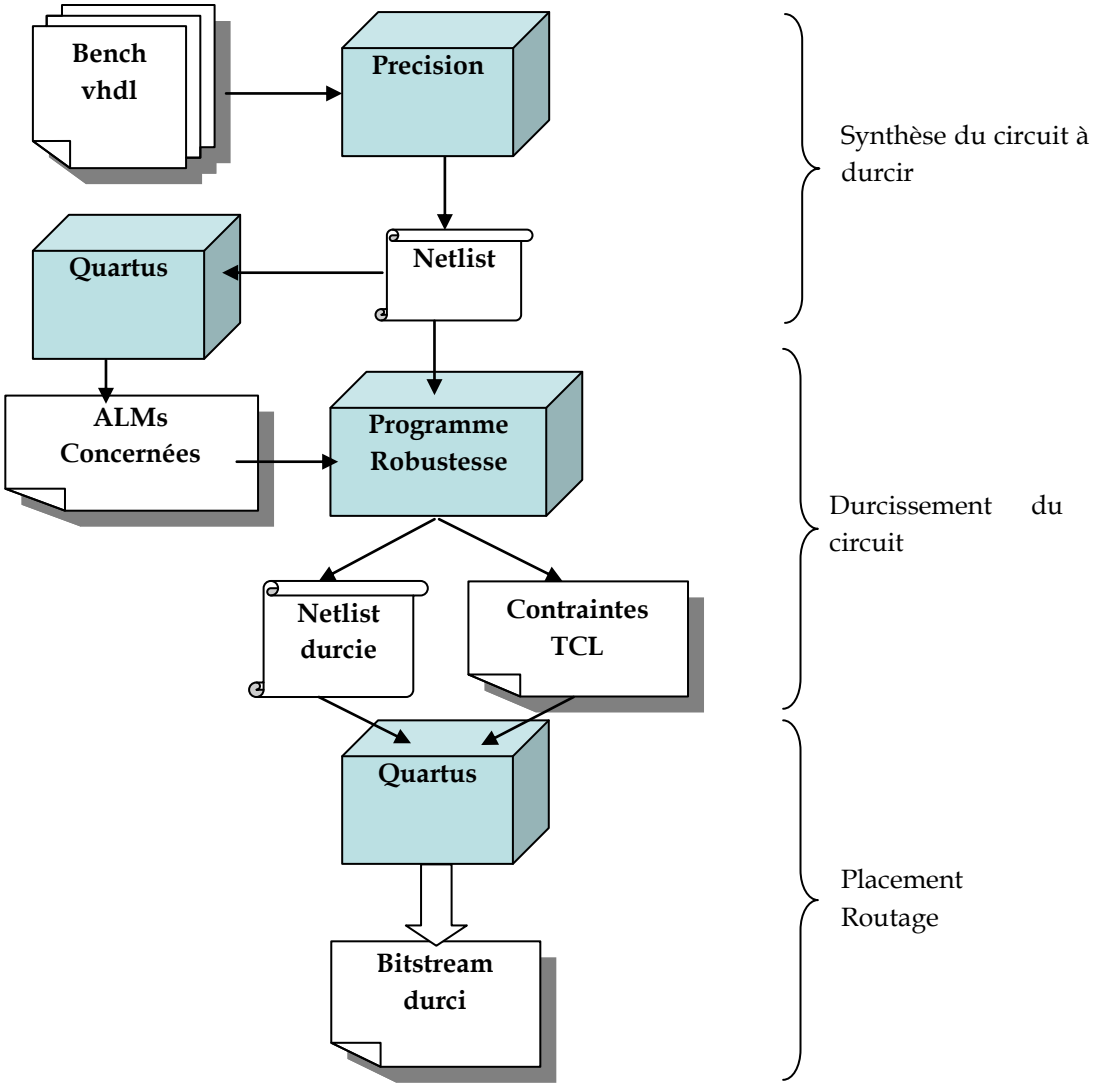


Figure 4-8. Flot de durcissement pour la famille Stratix4

Le tableau IV-VII met en évidence la différence entre les outils Quartus II 9.0 et Précision 2011a, à travers les résultats de synthèse des benchmark ITC'99 avec ces deux outils. Nous avons également fait un calcul approximatif des LUTs que nous pouvons dupliquer à coût nul (hors coût des LUTs de comparaison et de propagation du signal d'erreur) dans chaque benchmark.

Tableau IV-VII Coûts en ressources des netlists en fonction de l'outil de synthèse

Benchmark	Quartus directement			Precision puis Quartus		
	ALUT	ALM	Taux couverture estimé	ALUT	ALM	Taux couverture estimée
b01	5	4	60%	10	9	80%
b02	4	3	50%	8	7	75%
b03	20	15	50%	11	17	209,09%
b04	130	68	4,62%	79	53	34,18%
b05	141	84	19,15%	119	74	24,37%
b06	8	5	25%	11	8	45,45%
b07	45	36	60%	57	36	26,32%
b08	18	15	66,67%	20	14	40%
b09	22	12	54,55%	19	20	110,53%
b10	35	24	37,14%	31	19	22,58%
b11	160	92	15%	68	59	73,53%
b12	261	183	40,23%	264	163	23,48%
b13	52	30	15,38%	48	33	37,50%
b14	711	464	30,52%	811	528	30,21%
b15	1403	806	14,90%	1497	1020	36%

Toutes les ALMs utilisées par le circuit sont prises en compte par ce calcul préliminaire sans distinction sur leur contenu (bascule, LUT ou LUT + bascule…). Ceci conduit à des taux de couverture des blocs combinatoires, à coût de duplication nul, allant jusqu'à 209,9%. Cela signifie en fait qu'il serait possible pour ce circuit d'implanter de la logique de détection dans des ALMs utilisées exclusivement par une ou deux bascules dans la version non protégée.

D'après ce tableau deux autres observations peuvent être relevées:

- Le nombre de LUTs des circuits varie entre Quartus et Précision. Pour ces 15 exemples, nous avons presque autant de d'exemples plus coûteux en termes de LUTs utilisées dans les deux cas. Le résultat est quasiment le même en terme de coût d'ALM à une exception près, le bench « b07 », dans lequel nous avons 36 ALM utilisées par les deux netlists. Il n'y a donc pas de raison majeure de préférer un outil à l'autre de manière générale ; notons toutefois que l'utilisation de Precision conduit en général à réduire le nombre d'ALM pour les exemples les plus gros.
- Les taux de couverture que nous avons évalués pour la duplication à coût nul sont dans 10 cas sur 14 plus intéressants avec une netlist générée par Précision plutôt que par Quartus.

A partir de là nous pouvons conclure que l'utilisation d'une netlist générée avec Précision est plutôt préférable pour l'implantation de la technique proposée, même sans tenir compte des contraintes liées à l'outil Quartus.
Dans l'adaptation de notre méthode à l'architecture Xilinx, seules les fonctions à moins de 6 entrées étaient prises en compte. On se propose ici d'ajouter une nouvelle catégorie de fonctions lors de la génération de la netlist durcie, à savoir les fonctions à 6 entrées, puisqu'avec l'architecture du Stratix IV dans une ALM nous pouvons placer deux fonctions à 6 entrées.

Nous reprenons donc la méthode proposée pour l'architecture Virtex pour protéger les fonctions à moins de 7 entrées. Pour une optimisation maximale, ces fonctions doivent être sur une ALM partiellement utilisée, mais dans le paragraphe IV.3.2.1 nous allons tout d'abord ignorer cette condition afin d'établir un plafond des surcoûts de durcissement sur le Stratix IV.

IV.3.2.1. Duplication sans optimisation

Dans ce paragraphe, nous nous proposons d'ignorer la condition impliquant l'utilisation des ALM partiellement utilisées. Nous étendons toutefois la protection à toutes les fonctions combinatoires à moins de 7 paramètres.

IV.3.2.1.1. Exploitation de la stratégie de partage automatique des ressources Stratix

L'architecture Stratix et l'outil Quartus offrent une stratégie de partage automatique des ressources entre les LUTs. Cette stratégie s'illustre par le fait que sur une ALM nous pouvons implanter deux LUTs, avec jusqu'à 4 entrées communes. Cette optimisation permet de réduire le taux d'ALM partiellement utilisées, plus particulièrement pour les circuits composés essentiellement de fonctions logiques utilisant des paramètres en commun, comme par exemple les additionneurs (pour 3 paramètres communs, dans une ALM, une LUT calcule l'addition tandis que l'autre LUT calcule la retenue).

Par ailleurs, l'architecture des ALMs permet d'implanter facilement deux LUTs à petit nombre d'entrées.

En ce qui concerne les grandes fonctions (à plus de 4 paramètres) qui sont en moyenne les plus utilisées sur le panel des applications ITC '99, il faut avoir des fonctions ayant des paramètres communs pour que ces dernières soient implantées dans une seule ALM. L'automatisation du partage de ressources de Quartus permet en théorie de les placer dans une même ALM et donc devrait permettre à nos réplicas de partager automatiquement les ressources pouvant être partagées.

Nous avons appliqué notre approche de durcissement sur les benchmarks ITC en les plaçant avec des contraintes concernant uniquement les fonctions à protéger ainsi que leurs réplicas. En partant des netlists modifiées et des fichiers de contraintes générées avec notre programme de durcissement, nous avons obtenu des surcoûts de protection trop élevés par rapport à l'estimation faite initialement. Nous avons donc ignoré les contraintes de placement qui nous avaient été bénéfiques pour les circuits durcis sur Virtex V, pour pouvoir comparer.
Le tableau IV-VIII confronte les résultats obtenus avec et sans contraintes de placement routage en appliquant l'approche et en répliquant les fonctions ayant moins de 7 paramètres. Les résultats de la deuxième catégorie «Sans contrainte» semblent être meilleurs dans 100% des cas.

Tableau IV-VIII Durcissement de toutes les fonctions à moins de 7 entrées avec et sans contrainte de placement

Application	Nb ALM original	Nb ALUT			Couverture	Netlist sans contrainte		Netlist avec contrainte	
		Original	Protégées	Circuit durci		ALM	Surcoût	ALM	Surcoût
b01	9	10	10	23	100%	14	55,56%	16	77,78%
b02	7	8	8	18	100%	10	42,86%	12	71,43%
b03	17	11	11	24	100%	17	0%	26	52,94%
b04	53	79	51	151	64,56%	97	83,02%	103	94,34%
b05	74	119	67	216	56,30%	126	70,27%	128	72,97%
b06	8	11	11	26	100%	15	87,50%	17	112,50%
b07	36	57	30	100	52,63%	56	55,56%	60	66,67%
b08	14	20	20	49	100%	29	107,14%	33	135,71%
b09	20	19	18	44	94,74%	30	50%	39	95%
b10	19	31	31	74	100%	44	131,58%	47	147,37%
b11	59	68	45	132	66,18%	79	33,90%	84	42,37%
b12	163	264	241	604	91,29%	363	122,70%	371	127,61%
b13	33	48	45	101	93,75%	63	90,91%	71	115,15%
b14	528	811	482	1487	59,43%	871	64,96%	903	71,02%
b15	1020	1497	986	1819	65,86%	1645	61,27%	1719	68,53%

Nous avons noté que dans le cas d'application de contraintes, l'outil de synthèse délègue toutes les optimisations possibles à l'utilisateur et ne fait plus de gros efforts dans le placement routage. Par conséquent, si nous imposons des contraintes de placement sur la partie du circuit que nous avons répliqué en délaissant le reste du circuit, Quartus n'optimise pas le reste du circuit et le place d'une manière aléatoire. Autrement dit, les ALUTs sur lesquelles nous n'avons pas appliqué de contraintes de placement routage seront désormais placées aléatoirement sans que l'outil ne cherche à les optimiser, en les plaçant par exemple dans une même ALM que des bascules. Ceci nous amène dans quelques cas à avoir des ALMs entièrement consacrées à

l'implantation d'une bascule alors qu'elle peut contenir un autre registre et/ou deux fonctions logiques.

Au vu de ces résultats, nous avons décidé de revoir notre technique de génération des contraintes. Cette nouvelle stratégie sera présentée dans le paragraphe IV.3.2.2.1.

IV.3.2.1.2. Séparation des ressources des LUTs Stratix

Les résultats précédents nous ont montré que le partage automatique des ressources n'aide pas systématiquement le placement du réplica avec la fonction originale dans la même ALM, et ceci même dans le cas où cette fonction est placée dans une ALM partiellement utilisée. Nous avons donc décidé de renoncer au partage automatique des ressources et d'améliorer notre stratégie de placement routage. La nouvelle stratégie doit permettre non seulement de placer chaque paire fonction/réplica dans une même ALM mais en plus d'utiliser le maximum de signaux indépendants entre une fonction et son réplica. Ceci nous permet de diminuer les fautes non détectées dans les cas où il y a une faute au niveau des interconnexions. La figure 4-9 représente une fonction et sa réplique implémentées sur un ALM et chacune de ces fonctions utilise des signaux différents.

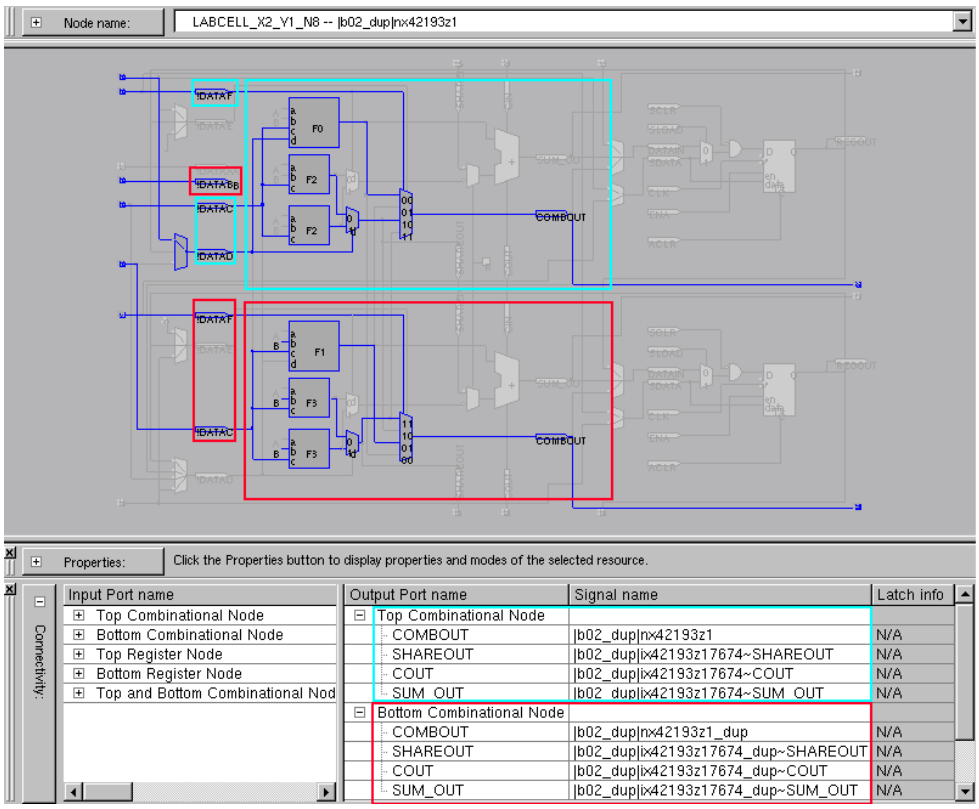


Figure 4-9. Duplication dans une ALM

Il est possible d'avoir des signaux indépendants pour les deux ALUT de l'ALM sous condition d'avoir des fonctions à moins de 5 entrées. Pour le reste, nous allons essayer de varier le plus possible les signaux utilisés par les deux ALUTs :

- Concernant les fonctions à 5 entrées, nous allons utiliser 3 entrées indépendantes pour chaque fonction ainsi que deux entrées communes. Cette configuration utilise la totalité des entrées de l'ALM ce qui ne permet pas à celle-ci d'implanter des bascules sauf si cette bascule est la sortie de la fonction elle-même.
- Pour les fonctions à 6 entrées nous sommes dans l'obligation d'avoir 4 entrées communes entre la fonction à protéger et sa réplique.

IV.3.2.2. Duplication sélective

Dans les résultats précédents, les taux de surcoût étaient parfois supérieurs aux taux de couverture car nous avions aussi dupliqué les fonctions logiques qui sont implantées par des ALMs complètement utilisées, ce qui amène à dupliquer entièrement l'ALM, à condition qu'elle ne comporte pas de LUT à 7entrée, et à avoir un surcoût allant jusqu'à 100% sur la logique combinatoire concernée. Dans la suite, nous ne dupliquerons que la logique combinatoire implémentée sur une ALM partiellement utilisée.

IV.3.2.2.1. Les stratégies de placement

Pour réaliser une duplication sélective, nous devons réaliser plusieurs synthèses :

- Une première permet d'avoir la netlist éditable avec Précision
- Une seconde permet d'avoir avoir une image précise de l'implantation du circuit par Quartus
- Après la deuxième synthèse, nous spécifions à l'outil que nous voulons garder le même placement. Ensuite, nous re-synthétisons afin de générer en plus du bitstream un fichier de contraintes décrivant le placement choisi.

Le fichier de contraintes permet au programme de durcissement non seulement de répliquer les ALMs partiellement utilisées mais aussi de prendre en compte le placement du circuit non durci dans la génération du fichier de contraintes, et nous obtenons un deuxième fichier de contraintes comprenant l'ensemble du circuit durci (IOB et bascules compris).

Or, comme illustré dans la figure 4-10, un placement routage avec ce deuxième fichier de contraintes génère dans certains cas (quand la fonction à protéger est une fonction à 6 paramètres et que dans cette ALM il existe déjà une bascule) des conflits entre les réplicas et les bascules. Dans ce cas l'outil de synthèse ne peut pas router à la fois la 6ème entrée du réplica et l'entrée de la bascule.

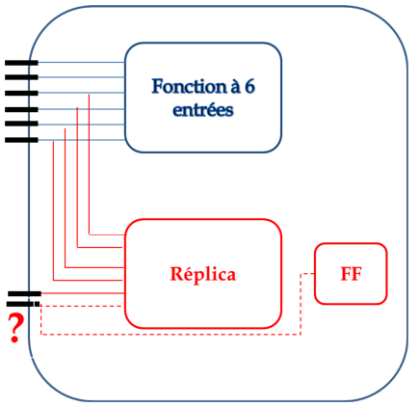


Figure 4-10. Conflit de routage entre réplica et FF

Nous avons alors effectué 5 implantations différentes pour chaque benchmark :

1. Avec contraintes sur l'ensemble du circuit durci, en supprimant les contraintes sur des bascules quand il y a des problèmes au niveau du placement routage.
2. Avec contraintes sur l'ensemble du circuit durci, en supprimant cette fois ci les contraintes sur les réplicas qui sont en concurrence avec la bascule au lieu d'enlever la contrainte sur cette dernière.
3. Avec contraintes uniquement sur les cellules du circuit original, l'idée consistant à vérifier si l'outil place instinctivement les réplicas dans les cellules partiellement utilisées.
4. Avec contraintes uniquement sur les réplicas, en supposant que le circuit durci va être replacé comme l'a été le circuit original.
5. Sans contrainte en léguant tout le travail de placement à l'outil.

La figure 4-11 illustre les surcoûts des différentes stratégies de placement, elle montre que la stratégie de placement numéro 1, « contraintes sur tout sauf les bascules problématiques », présente les meilleurs résultats sur presque tous les benchmarks. Elle est même plus avantageuse que le placement avec optimisation sur la surface fait par Quartus.

Cette figure montre également que la stratégie adoptée dans le paragraphe IV.3.2.1 est la pire stratégie. En effet, appliquer des contraintes uniquement sur les fonctions et leurs réplicas génère des surcoûts très importants, et cela même en générant ces contraintes en fonction des contraintes appliquées sur le circuit non durci.

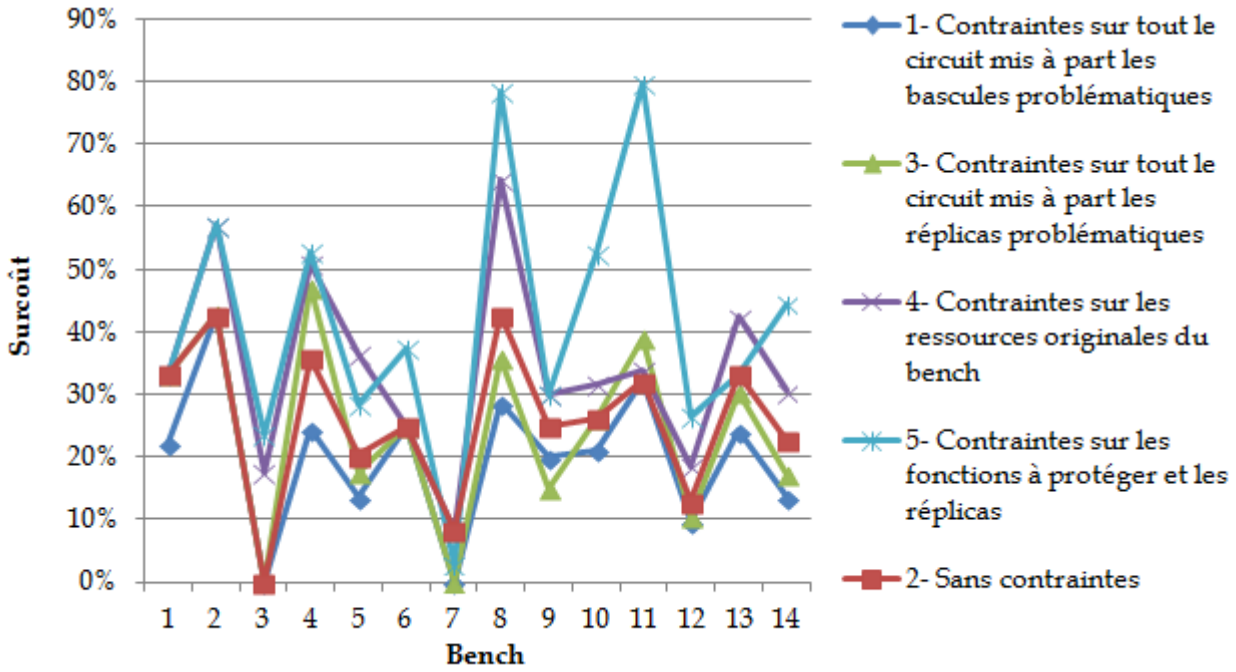


Figure 4-11. Comparaison entre les différentes stratégies de placement

Le tableau IV-IX représente les résultats chiffrés obtenus avec la meilleure stratégie (n°1) en comparaison avec une implantation sans contraintes.

Tableau IV-IX Durcissement avec contrainte de placement sur le circuit original et les réplicas

Application	Nb ALM original	Nb ALUT			Couverture	Netlist sans contrainte		Netlist avec contrainte	
		Original	Protégées	Circuit durci		ALM	Surcoût	ALM	Surcoût
b01	9	10	7	20	70%	12	33,33%	11	22,22%
b02	7	8	7	18	87,50%	10	42,86%	10	42,86%
b03	17	11	9	24	81,82%	17	0%	17	0%
b04	53	79	21	109	26,58%	72	35,85%	66	24,53%
b05	74	119	23	152	19,33%	89	20,27%	84	13,51%
b06	8	11	4	17	36,36%	10	25%	10	25%
b07	36	57	3	61	5,26%	39	8,33%	36	0%
b08	14	20	8	32	40%	20	42,86%	18	28,57%
b09	20	19	10	34	52,63%	25	25%	24	20%
b10	19	31	8	43	25,81%	24	26,32%	23	21,05%
b11	59	68	45	131	66,18%	78	32,20%	78	32,20%
b12	163	264	39	319	14,77%	184	12,88%	179	9,82%
b13	33	48	15	69	31,25%	44	33,33%	41	24,24%
b14	528	811	180	1063	22,19%	648	22,73%	600	13,64%
b15	1020	1497	229	1819	15,30%	1167	14,41%	1125	10,29%

Les meilleurs taux de couverture dépassent les 80% avec les benchmarks b02 et b03. Ce dernier présente un surcoût nul malgré son taux de couverture exceptionnel. Ceci résulte essentiellement des caractéristiques intrinsèques de cet exemple qui nous ont permis d'ailleurs

d'avoir, dans de précédentes expérimentations, un taux de couverture estimé de 209,9% ce qui résulte en un surcoût nul. Ceci s'explique par le fait que nous pouvons, pour ce circuit, implanter la logique de détection dans des ALMs utilisées exclusivement par une ou deux bascules. Le pire cas de couverture avoisine les 5% pour le benchmark b07, et ce pour un surcoût nul dans le cas d'une netlist implantée avec contrainte et allant jusqu'à 8% pour la netlist implantée sans contraintes. Ce circuit, avant durcissement, est optimisé de fait par le placement routage. Ceci implique le nombre élevé d'ALMs complètement utilisées, seules 3 LUTs étant placées dans des cellules partiellement utilisées.

Le tableau IV-X nous permet de vérifier que le nombre de LUTs implantés par Quartus correspond bien au nombre théorique de LUTs auquel nous nous attendons.

Tableau IV-X Vérification du nombre théorique de LUTs des circuits durcis

Application	Vérification Nb ALUT théorique					
	Original	**Réplicas**	**Comparateurs**	**Détecteur Niveau 1**	**Détecteur Niveau 2**	**Total**
b01	10	7	3	0	0	20
b02	8	7	3	0	0	18
b03	11	9	3	1	0	24
b04	79	21	7	2	0	109
b05	119	23	8	2	0	152
b06	11	4	2	0	0	17
b07	57	3	1	0	0	61
b08	20	8	3	1	0	32
b09	19	10	4	1	0	34
b10	31	8	3	1	0	43
b11	68	45	15	3	0	131
b12	264	39	13	3	0	319
b13	48	15	5	1	0	69
b14	811	180	60	10	2	1063
b15	1497	229	77	13	3	1819

IV.3.2.2.3. Optimisation de la stratégie n°1

Selon les résultats obtenus, la meilleure façon d'implanter notre approche sur la Stratix IV est le placement de tout le circuit en enlevant les contraintes problématiques au niveau des bascules mais l'outil de synthèse n'optimise pas toujours cela au niveau du placement. Il faut donc refaire le placement de telle sorte à placer ces bascules-là, soit dans les ALMs contenant la logique de détection (une ou deux bascules par ALM), soit dans les ALMs implantant une « extented LUT » à 7 entrées, auquel cas cette ALM peut potentiellement contenir une bascule.

La figure 4-12 représente l'état du placement routage après la suppression des contraintes sur les bascules qui entrent en conflit avec les LUTs réplicas, l'outil se chargeant du placement de

ces dernières. Nous pouvons distinguer les cellules ayant des contraintes de placement des autres cellules : la première catégorie est colorée en gris alors que la seconde est en orange pour les LUTs et en rouge pour les bascules.

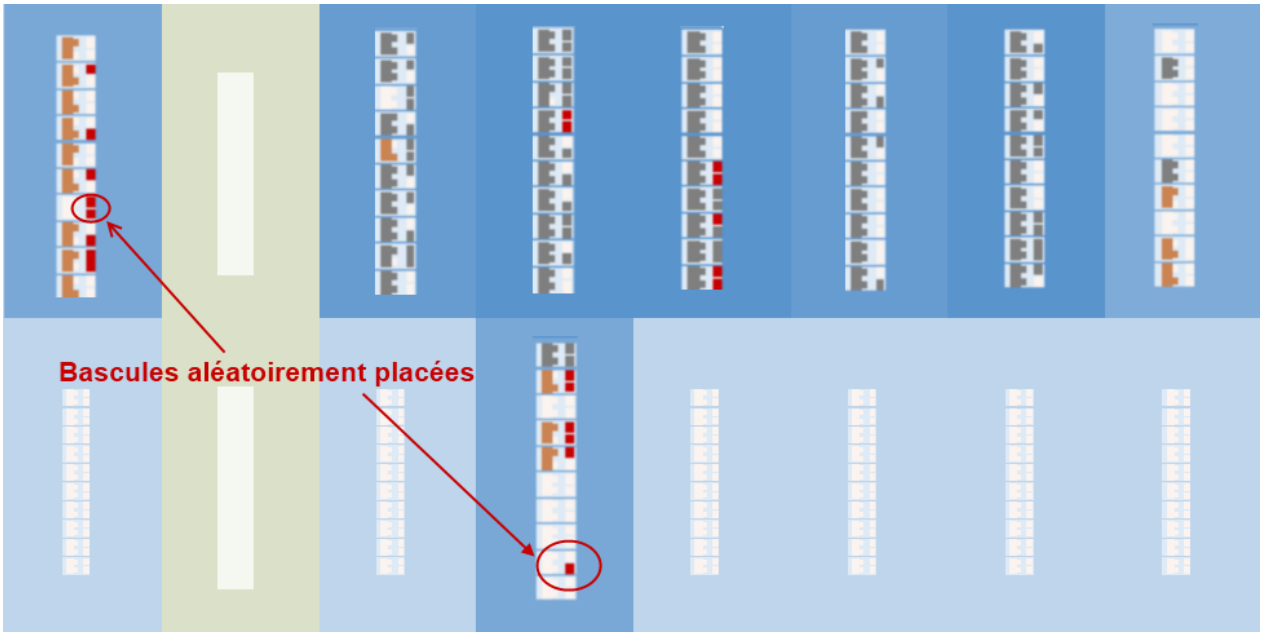


Figure 4-12. Implantation du benchmark b04 sans contrainte sur les bascules problématiques

La figure 4-13 représente une implantation après application de quelques contraintes sur les bascules qui avaient été placées d'une manière aléatoire ; en appliquant des contraintes sur 3 bascules nous avons pu gagner 3 ALMs.

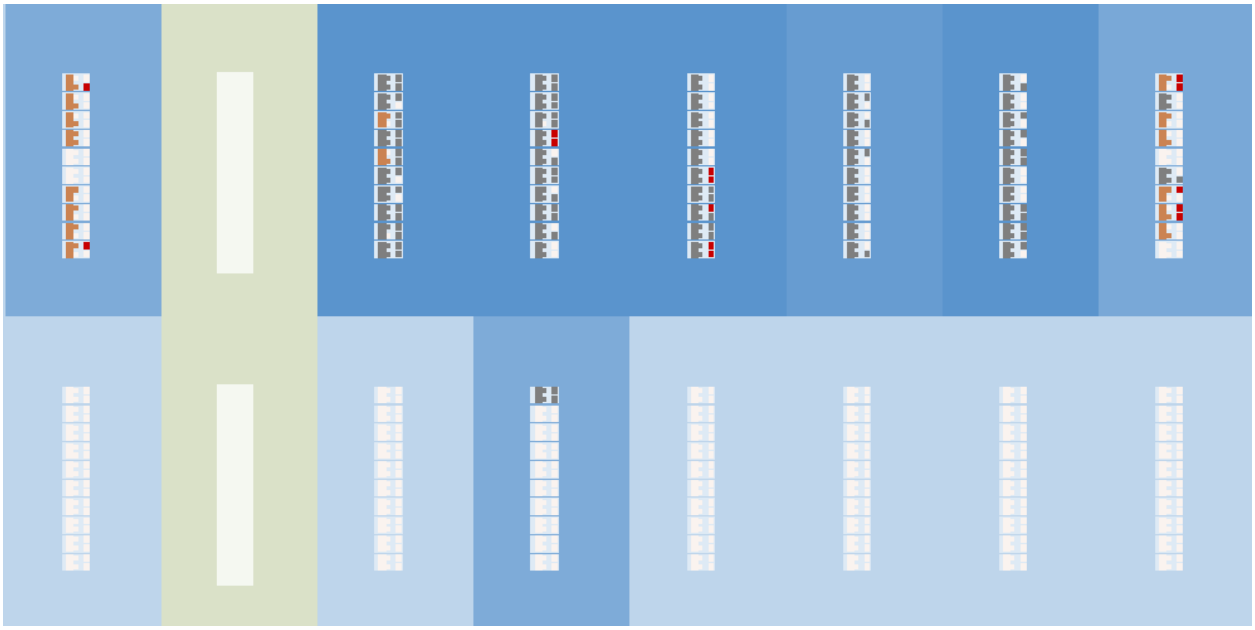


Figure 4-13. Implantation du benchmark b04 avec placement de 3 bascules dans des ALMs partiellement utilisées

Le tableau IV-XI représente le nombre de cellules potentiellement utilisables par les bascules problématiques pour les différents benchmarks.

Tableau IV-XI Cellules potentiellement utilisables par les bascules avec conflit de routage

Application	Nb LUT de la logique de détection			Nb LUT à 7 entrées	Cellules potentielles pour FF problématique
	Comparateurs	Détecteur Niveau 1	Détecteur Niveau 2	Total	
b01	3	0	0	0	6
b02	3	0	0	0	6
b03	3	1	0	0	8
b04	7	2	0	2	20
b05	8	2	0	7	27
b06	2	0	0	0	4
b07	1	0	0	1	3
b08	3	1	0	0	8
b09	4	1	0	1	11
b10	3	1	0	0	8
b11	15	3	0	0	36
b12	13	3	0	22	54
b13	5	1	0	0	12
b14	60	10	2	53	197
b15	77	13	3	78	264

L'implantation de certaines bascules dans des ALMs fausse toutefois le recensement des ALUTs utilisées. En effet l'outil de visualisation du placement routage utilisé dans les figures 4-12 et 4-13 indique l'utilisation de la LUT associée à la bascule dans l'ALM alors qu'en réalité la logique combinatoire est toujours disponible.

Malgré les fortes contraintes d'optimisation du placement, le surcoût de la protection est souvent proportionnellement élevé par rapport au taux de couverture, qui est souvent assez petit, surtout pour les plus gros exemples. L'approche semble donc moins efficace que sur Virtex, même si le ratio entre taux de couverture et surcoût est nettement plus petit que celui des approches classiques.

Le surcoût est dû exclusivement à la logique de comparaison et à la propagation du signal d'erreur. Dans la section suivante, nous allons discuter d'éventuelles optimisations pour l'implantation de cette logique.

IV.4. Optimisations supplémentaires envisagées

La logique de comparaison et de propagation du signal d'erreur est à source du surcoût de la protection. Pour remédier à cela, nous avons regardé de plus près les ressources logiques existant sur les deux plateformes Xilinx V5 et Stratix IV et nous avons remarqué que la chaîne

de propagation de retenue n'est utilisée que rarement (environ 10% des cas sur un échantillon de 10 benchmarks ITC). Nous avons donc cherché à exploiter ces ressources. Nous avons aussi cherché à évaluer le coût d'autres approches permettant d'augmenter le taux de couverture ou de tolérer certaines erreurs.

IV.4.1. Chaîne de propagation de retenue sur VirtexV

La chaîne de propagation de retenue sur Virtex V accélère le chemin de propagation critique et permet d'atteindre des vitesses très élevées pour les calculs arithmétiques. Les chaînes de propagation de retenue sont basées sur des multiplexeurs (MUXCY), comme illustré par la figure 4-14. Les sorties sont calculées en utilisant la fonction XOR.

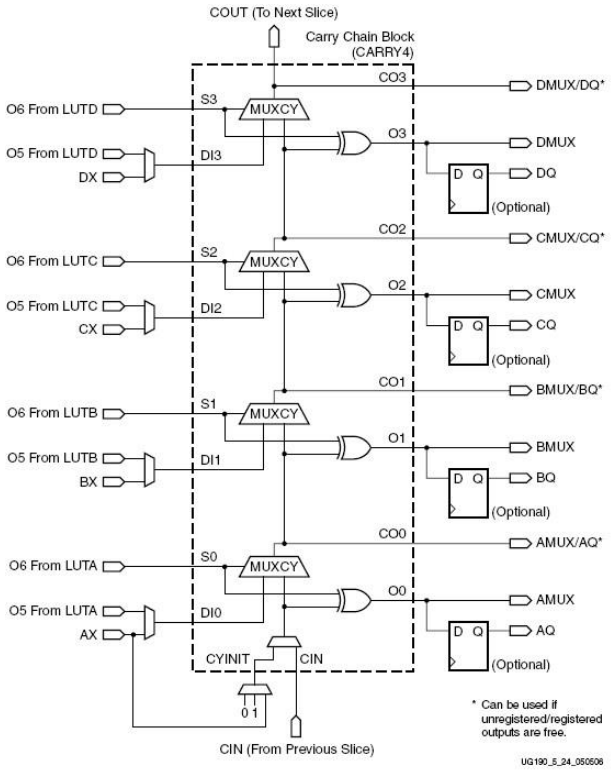


Figure 4-14. Chaîne de propagation de retenue V5

Cette fonction XOR pourrait faire office de comparateur pour une fonction et son réplica. Nous avons donc essayé d'exploiter ces ressources inutilisées afin de réduire le nombre de LUTs, utilisées comme comparateur, et de ce fait avoir aussi un signal d'erreur par fonction protégée et non pas un signal par bloc de trois fonctions (voir figure 4-15).

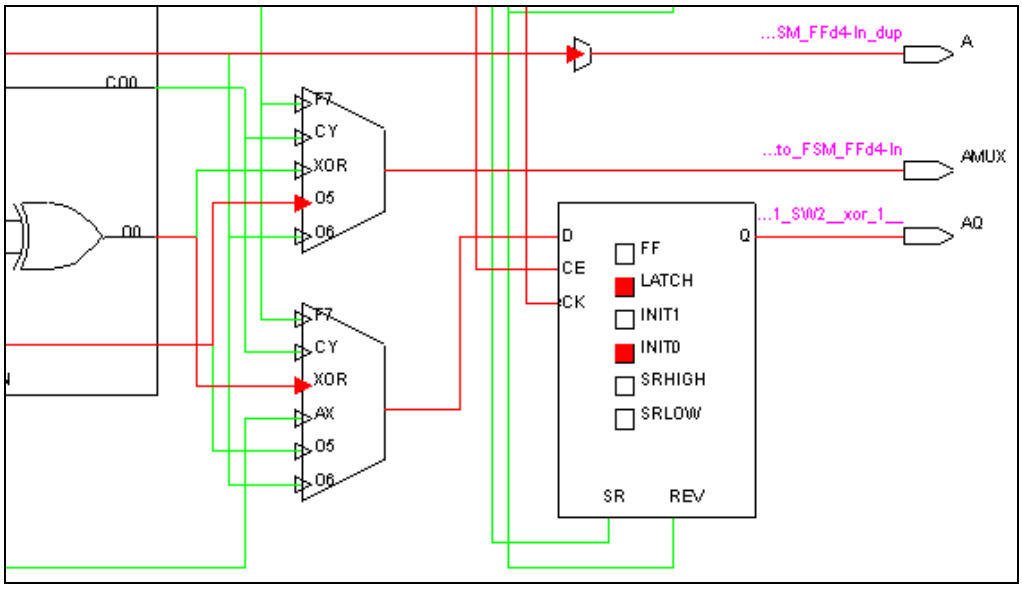


Figure 4-15. Comparaison implantée avec la porte XOR du V5

Dans la pratique nous nous sommes heurtés à deux problèmes :

- La comparaison ne se fait pas directement à la sortie des signaux des LUTs : la première LUT est certes directement accessible par le XOR mais la seconde doit sortir de la slice et par la suite entrer comme un nouveau signal d'entrée, ce qui pénalise fortement le temps de calcul.
- Lorsque nous utilisons cette porte XOR et que le signal doit sortir puis ré-entrer nous perdons une des deux sorties de la LUT que nous comptions utiliser comme sortie de comparaison. Par conséquent le signal de comparaison est obligatoirement stocké dans une bascule avant d'atteindre la LUT de détection, ce qui ne permet plus une détection d'erreur dans le même cycle.

Par ailleurs, nous avons tenté d'exploiter des portes XOR existant à l'intérieur des LUTs du Virtex5. Cependant, ces portes ne sont pas visibles au niveau de la netlist et sont seulement exploitées par la LUT pour optimiser certains calculs. Dans l'état où en sont les choses du point de vu accessibilité, nous ne sommes pas parvenus à les utiliser pour optimiser le coût de détection (sans devoir ajouter une bascule supplémentaire par comparaison).

IV.4.2. Chaîne de propagation de retenue sur StratixIV

L'architecture Stratix IV est complètement différente de celle des FPGAs Virtex5 mais nous y trouvons tout de même un système de propagation de retenue semblable à celui de son concurrent et qui parait exploitable : la chaîne de retenue sur cette architecture est composée d'additionneurs (2 additionneurs par ALM). Nous avons donc tenté d'exploiter ces composants pour faire la comparaison entre la fonction et son réplica.

L'idée est de placer la fonction à protéger dans la partie supérieure de l'ALM et de forcer le routage pour que sa sortie soit directement connectée à l'entrée de l'additionneur qui se trouve dans la partie inférieure de l'ALM. Cet additionneur aurait comme deuxième entrée la sortie de la réplique qui est placée aussi dans la partie inférieure de la même ALM. La sortie principale se comporterait comme une porte XOR pour comparer le résultat des deux fonctions et déclencher la détection en cas d'erreur. Les multiplexeurs doivent être commandés pour connecter en sortie le signal de la fonction dans le premier cas et la sortie de l'addition dans le deuxième, comme schématisé dans la figure 4-16.

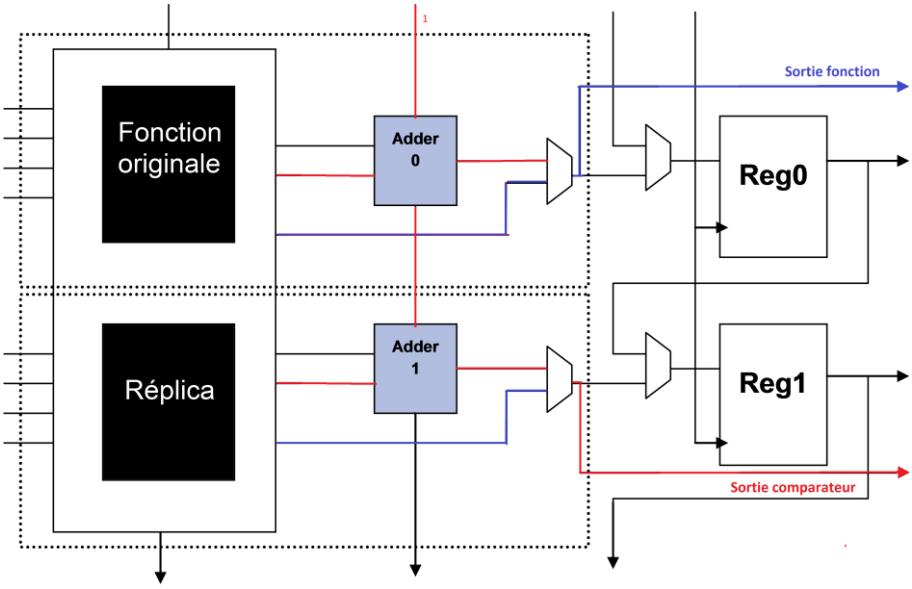


Figure 4-16. Exploitation des additionneurs pour la comparaison

Toutefois, compte tenu des contraintes imposées par l'outil Quartus, il s'est avéré que cette manipulation n'est pas possible : en cas de tentative de forçage du routage du circuit, présenté ci-dessus, le placement-routage n'aboutit pas correctement.

IV.4.3. Duplication totale sur StratixIV

Nous avons remarqué que l'outil Quartus gère plutôt bien le placement des ressources, en comparaison à des stratégies de routage développées dans le paragraphe IV.3.2.2.1. En effet, seule la stratégie n°1 présente de meilleurs résultats que le placement routage de Quartus. Dans cette optique nous avons pensé appliquer à nos benchmark une DMR totale comprenant la duplication des registres en laissant Quartus gérer le placement routage de toutes les ressources.

Le tableau IV-XII représente une estimation du coût de duplication totale des circuits, au meilleur des cas, en nous basant sur les surcoûts des duplications partielles que nous avons obtenu en prenant en compte les bascules à dupliquer en plus des LUTs non protégées par notre durcissement partiel.

Tableau IV-XII Estimation de coût de la duplication totale des circuits

Application	Nb ALM original	Netlist avec DMR	
		ALM	Surcoût
b01	9	15	66,67%
b02	7	11	57,14%
b03	17	25	47%
b04	53	114	115,09%
b05	74	165	122,97%
b06	8	16	100%
b07	36	69	91,67%
b08	14	28	100%
b09	20	34	70%
b10	19	39	105,26%

Les estimations pour une duplication totale de ces benchmarks sont dans certains cas encourageantes ; pour les benchmarks b01, b02 et b03 par exemple le surcoût ne dépasse pas les deux tiers. Mais quand le circuit est plus grand, le coût de la protection augmente, jusqu'à 122% de surcoût pour le benchmark b05.

IV.4.3. Triplication sélective sur Stratix IV

Nous avons aussi cherché à exploiter l'architecture Stratix au mieux, pour implanter une TMR sélective. Nous allons nous appuyer sur les coûts que nous avons obtenus pour la duplication sélective pour faire une estimation du coût d'une triplication visant les mêmes cellules logiques.

Nous allons remplacer la notion de détecteurs et de comparateurs par une 3ème instance de la cellule à protéger et par des voteurs. Pour la 3ème instance nous n'aurons pas d'autre choix que d'utiliser une cellule logique supplémentaire pour l'implanter. Par contre, concernant les voteurs, nous allons procéder comme suit. Les ALMs qui implantent les comparateurs utilisés dans la duplication vont désormais nous servir de voteur. Au lieu de comparer 3 couples fonction/réplicas, une ALM servira à surveiller le bon déroulement de 2 fonctions en implantant deux voteurs avec des entrées indépendantes les unes des autres.

Le Tableau IV-XIII représente une estimation des surcoûts que l'on obtiendrait pour une TMR sélective.

Tableau IV-XIII Estimation de coût d'une TMR sélective des circuits

Application	NB LUT		Couverture	Nb ALM				Surcoût	Rapport Surcoût / Couverture
	Original	Réplicas		TMR			Original		
				Second réplica	ALM Voteur	ALM TMR			
b01	10	7	70,00%	7	4	11	9	122,22%	**1,75**
b02	8	7	87,50%	7	4	11	7	157,14%	**1,80**
b03	11	9	81,82%	9	5	14	17	82,35%	**1,01**
b04	79	21	26,58%	21	11	32	53	60,38%	**2,27**
b05	119	23	19,33%	23	12	35	74	47,30%	**2,45**
b06	11	4	36,36%	4	2	6	8	75,00%	**2,06**
b07	57	3	5,26%	3	2	5	36	13,89%	**2,64**
b08	20	8	40,00%	8	4	12	14	85,71%	**2,14**
b09	19	10	52,63%	10	5	15	20	75,00%	**1,43**
b10	31	8	25,81%	8	4	12	19	63,16%	**2,45**
b11	68	45	66,18%	45	23	68	59	115,25%	**1,74**
b12	264	39	14,77%	39	20	59	163	36,20%	**2,45**
b13	48	15	31,25%	15	8	23	33	69,70%	**2,23**
b14	811	180	22,19%	180	90	270	528	51,14%	**2,30**

D'après nos estimations, cette triplication sélective de la logique combinatoire aura un coût souvent bien inférieur à 3 fois la logique protégée. Le taux de couverture dépasse 80% pour certaines applications, mais ce taux reste très variable selon le benchmark. Dans le cas de l'application b07, par exemple, le taux de couverture ne dépasse pas 5%. D'un autre côté, les surcoûts oscillent également beaucoup selon l'application, entre 13% et 157%, ce qui nous donne un rapport surcoût/couverture variant entre 1,01 pour b03 et 2,64 pour b07. Cette approche n'a pas été automatisée, mais pourrait donc être envisagée dans certains cas pour réduire le coût des approches classiques tout en augmentant la tolérance aux erreurs.

IV.5. Conclusion

Nous avons automatisé notre approche de durcissement de circuits pour deux familles de FPGA à mémoire SRAM, en exploitant au mieux les ressources initialement inutilisées par l'application. En faisant de la redondance matérielle sélective, il est possible d'augmenter sensiblement la robustesse des applications pour un coût matériel réel limité (en moyenne de l'ordre de 50% ou moins des ressources protégées). Le coût réel peut être nul si suffisamment de ressources du FPGA sont inutilisées pour l'application initiale. L'objectif n'étant pas d'atteindre le même niveau de robustesse qu'une implantation TMR, mais plutôt d'améliorer la robustesse à moindre coût; cette méthode peut être considérée comme un bon compromis, avec des rapports coût/protection pouvant être adaptés par le concepteur selon ses contraintes, par une sélection judicieuse des fonctions à protéger. Nous avons aussi évalué sur Stratix IV la duplication totale et la triplication partielle en optimisant au mieux les ressources disponibles. La triplication sélective pourrait donner lieu à de futurs développements.

Conclusion générale et perspectives

La conception des circuits intégrés exige de plus en plus de techniques et d'outils pour l'évaluation de leur niveau de robustesse. En effet, les circuits sont désormais plus sensibles aux perturbations naturelles même au niveau de la mer, à cause de leurs dimensions de plus en plus petites et de leur plus grande quantité de logique et de mémoire. A ces attaques naturelles, nous pouvons ajouter les attaques intentionnelles liées à la sécurité, qui visent, par exemple, à obtenir la clé secrète utilisée par un algorithme de chiffrement. L'analyse de la robustesse des circuits est donc devenue souvent indispensable, même lorsqu'ils ne sont pas utilisés dans des applications dites "critiques". Nous nous sommes principalement focalisés dans cette thèse sur le cas d'applications implantées sur FPGA configurable par mémoire SRAM.

L'étude de l'état de l'art des techniques d'injection de fautes a mis en lumière les avantages et les inconvénients de chacune d'elles. Les approches à base de simulation RTL induisent des temps d'expérience très importants. L'émulation accélère les campagnes d'injection par rapport à la simulation RTL, et permet une analyse à partir d'une description de haut niveau. Pour ces raisons, nous avons opté pour l'émulation. Cependant, l'exécution de certaines tâches au niveau matériel s'accompagne d'une diminution de la flexibilité. C'est pour cette raison que nous avons choisi d'utiliser un processeur embarqué comme niveau intermédiaire entre l'ordinateur hôte et la logique programmable. Nous conservons ainsi un bon niveau de flexibilité, tout en limitant les transferts entre le PC et la plateforme de prototypage et en étant capables d'exécuter des tâches complexes pour la commande des injections et l'analyse des effets.

L'environnement d'injection de fautes à été mis en place initialement sur la plateforme Virtex II Pro. Cette plateforme a été couplée à une base de données de motifs d'erreur réels, obtenus en analysant le résultat d'attaques physiques. Ceci nous a permis de faire des campagnes d'injection les plus réalistes possibles et d'évaluer la robustesse d'un certain nombre de circuits, notamment par rapport à des erreurs dans la mémoire de configuration du FPGA. Nous avons par la suite fait évoluer notre environnement d'injection de fautes pour une plateforme Virtex V sur laquelle nous avons, d'une part, testé la robustesse du processeur Leon3, et d'autre part validé une technique de protection se basant sur la vérification du flot de contrôle. Nous avons aussi montré l'intérêt, du point de vue du temps d'injection, d'une plateforme reconfigurable utilisant un accès direct à la mémoire de configuration.

Grâce à nos études d'évaluation de robustesse, nous avons conclu que les injections dans les bits de configuration sont plus néfastes pour le bon fonctionnement d'une application que les erreurs dans les bascules utilisateur, et que la multiplicité des erreurs accroît d'une manière significative l'impact sur les applications. Nous avons aussi montré que les techniques de protection fonctionnelles ou par redondance d'information ne permettent pas d'obtenir des résultats satisfaisant face aux erreurs dans la mémoire de configuration. Partant de ce principe, nous avons proposé et développé une technique de durcissement pragmatique exploitant les ressources du FPGA non exploitées par l'application considérée. Cette technique a été automatisée et évaluée pour deux familles de composants, à savoir les Virtex V et Stratix IV. Il a été montré que le rapport entre surcoûts et taux de couverture dépend de l'application, mais aussi de la plateforme choisie. Toutefois, dans les deux cas, une augmentation sensible de robustesse peut être obtenue avec un coût matériel réel pouvant être nul. De plus, le rapport peut être adapté par un concepteur en fonction de ses contraintes, en sélectionnant judicieusement les zones à protéger.
Des campagnes d'injection de fautes nous ont permis d'avoir des résultats de validation encourageants pour les applications durcies sur Virtex V. Des injections similaires ne sont pas réalisables de manière fiable sur Stratix IV. Pour cette plateforme, une campagne de validation par tirs laser a donc été mise en place ; elle est actuellement en cours en partenariat avec EADS-IW.

Ce travail de thèse offre plusieurs autres perspectives, d'un point de vue évaluation de robustesse d'une part, et en termes d'amélioration de robustesse d'autre part.

L'environnement d'injection de fautes que nous avons développé permet l'analyse de systèmes complexes. Nous avons fait cette analyse avec des injections, soit aléatoires, soit ciblées sur des éléments précis des circuits évalués. Notre environnement permettrait également l'évaluation des circuits bloc par bloc. Ceci permettrait d'identifier les relations entre la robustesse intrinsèque de chaque bloc et le niveau de robustesse du circuit complet. L'objectif à terme serait de pouvoir limiter les injections à une précaractérisation des blocs (ou IPs), ré-utilisable pour évaluer la robustesse de différents circuits utilisant l'interconnexion de plusieurs de ces blocs. Des travaux dans le sens d'une telle analyse hiérarchique sont d'ores et déjà entamés par notre équipe.

Une autre perspective pour les injections est la possibilité d'étendre l'utilisation d'une base de données de motifs d'erreurs réalistes, en la couplant avec une modélisation des erreurs induites par certaines sources de perturbation. Une telle étude est entamée dans le cas d'attaques par faisceau laser sur circuits sécurisés.

La méthode de protection que nous avons proposée peut aussi être étendue. La détection d'une erreur au cycle même de son occurrence permet en principe de rajouter un mécanisme de recouvrement qui masquerait l'erreur avec un petit délai par rapport à une exécution normale. Plusieurs méthodes de recouvrement peuvent être envisagées, utilisant par exemple une endo-reconfiguration partielle avec éventuelle relocalisation de bloc dans le circuit, en cas de fautes permanentes. Ceci permettrait d'aller vers un système "auto-cicatrisant" (ou "self-healing" en anglais).

Nous avons par ailleurs évalué le coût d'une TMR sélective en exploitant l'architecture du FPGA pour diminuer le coût de la protection. Cette évaluation peut conduire à la mise en place automatisée d'une telle méthode de masquage partiel.

Enfin, nous pouvons aussi proposer comme perspective à long terme la définition d'un algorithme permettant d'optimiser le placement-routage des circuits à implanter en prenant en considération simultanément les caractéristiques intrinsèques des FPGAs cibles et les contraintes de sûreté de fonctionnement. Ceci permettrait de maximiser l'utilisation des ressources en incluant les contraintes de redondance.

Bibliographie

[Alte.-11] http://www.altera.com/literature/hb/stratix-iv/stx4_siv51002.pdf

[Angh.-00] L. Anghel, "Les Limites Technologiques du Silicium et Tolérance aux Fautes", Thèse de Doctorat, Laboratoire TIMA, INPG Grenoble, Décembre 2000.

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

[Anto.-03-1] L. Antoni, R. Leveugle, B. Feher, "Using Run-Time Reconfiguration for Fault Injection Applications", IEEE Trans. on Instrumentation and Measurement, vol 52, issue 5, pp. 1468-1473, Octobre 2003.

[Anto.-03-2] L. Antoni, "Injection de Fautes par Reconfiguration Dynamique de Réseaux Programmables", Doctorat de l'Institut National Polytechnique de Grenoble (INPG), 2003.

[Atme.-01-1] www.zmitac.aei.polsl.pl/ui/instrukcje/FPSLIC/Presentations/FPGA.ppt

[Atme.-01-2] www.zmitac.aei.polsl.pl/ui/instrukcje/FPSLIC/Presentations/config40K.ppt

[BarE.-06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, "The sorcerer's apprentice guide to fault attacks", Proceedings of the IEEE, vol. 94, no. 2, February 2006, pp. 370-382.

[Baum.-01] R. Baumann, "Soft Errors in Advanced Semiconductor Devices – Part 1 : The Three Radiation Sources", IEEE Trans. on Device and Materials Reliability, vol. 1, issue 1, pp. 17-22, Mars 2001.

[Berg.-09] S. Bergaoui, R. Leveugle, 'IDSM: An improved control flow checking approach with disjoint signature monitoring', Conference on Design of Circuits and Integrated Systems (DCIS), Zaragoza, Spain, November 18-20, 2009

[Bick.-10] R. Bickham. "An Analysis of Error Detection Techniques for Arithmetic Logic Units", PhD thesis, Vanderbilt University, 2010.

[Blod.-04] B. J. Blodget, S. P. McMillan, P. B. James-Roxby, "Reconfiguration of a programmable logic device using internal control", Pub. No US 2004/0117755, Pub. Date Jun. 17/2004 http://www.patentlens.net/imageserver/getimage/US_2004_0117755_A1.pdf?id=603136&page=all

[Boue-98] J. Boue, P. Petillon, Y. Crouzet, "MEFISTO-L: a VHDL-based Fault Injection Tool for the Experimental Assessment of Fault Tolerance",28th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-28), Munich, Allemagne, pp. 168-173, Juin 1998

[Burg.-96] L. Burgun, F. Reblewski, G. Fenelon, J. Barbier, O. Lepape, "Serial Fault Emulation", 33rd Conf. on Design Automation (DAC'96), Las Vegas, USA, pp. 801-806, Juin 1996

[Cani.-08] Canivet, G.; Clediere, J.; Ferron, J.B.; Valette, F.; Renaudin, M.; Leveugle, R. "Detailed Analyses of Single Laser Shot Effects in the Configuration of a Virtex-II FPGA", International On-Line Testing Symposium (IOLTS'08), Rhodes, Grèce, Juillet 2008.

[Carm.-01] C. Carmichael. Triple module redundancy design techniques for virtex FPGAs. Xilinx Application Note XAPP197, 1, 2001.

[Carm.-09] C. Carmichael and C. W. Tseng, "Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory," 2009, http://www.xilinx.com/support/documentation/application_notes/xapp1088.pdf

[Chen.-99] K.T. Cheng, S.Y. Huang, W.J. Dai, "Fault Emulation: A new Methodology for Fault Grading", IEEE Trans. on Computer Assisted Design, vol.18, issue 10, pp. 1487-1495, Octobre 1999.

[Corn.-00] F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero, "RT-Level Fault Simulation Techniques based on Simulation Command Scripts", Proc. XV Conf. on Design of Circuits and Integrated Systems (DCIS'00), Montpellier, France, Novembre 2000.

[Dodd-96] P. Dodd et al., "Impact of technology trends on SEU in CMOS SRAMs", IEEE Trans. Nucl. Sci., vol. 43, Dec. 1996, pp. 2797–2804.

[Duze.-00] S. Duzellier, D. Falguere, L. Guibert, V. Pouget, P. Fouillat, R. Ecoffet, "Application of Laser Testing in Study of SEE Mechanisms in 16-Mbit DRAMs", IEEE Trans. On Nuclear Science, vol. 47, n° 6, Decembre 2000.

[Emme.-07] J. M Emmert, C. E. Stroud and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks", IEEE Trans. on VLSI Systems, v. 15, n. 2, pp. 216 - 226, Feb 2007.

[Ferr.-12-1] J. B. Ferron, L. Anghel, R. Leveugle, "Towards low-cost soft error mitigation in SRAM-based FPGAs: a case study on AT40K", 3rd IEEE Latin Amerian Symposium on Circuits and Systems (LASCAS), Playa del Carmen, Mexico, 2012

[Ferr.-12-2] J.B. Ferron, « Analyse statique de l'effet des erreurs de configuration dans des FPGA configurés par SRAM et amélioration de robustesse » Doctorat de l'université de Grenoble, 2012.

[Gasi.-01] G. Gasiot, "Etude de la Sensibilité des Technologies CMOS/SoI et CMOS bulk aux rayonnements radiatifs terrestres", Rapport technique de doctorat, Juin 2001.

[Habi.-65] H. D. Habing, "The use of lasers to simulate radiation induced transients in semiconductor devices and circuits", IEEE Trans. Nucl. Sci., vol. NS-12, Dec. 1965, pp. 91-100.

[Hadj.-05] K. Hadjiat, «Evaluation Prédictive de la Sûreté de Fonctionnement d'un Circuit intégré Numérique» Doctorat de l'Institut National Polytechnique de Grenoble (INPG), 2005.

[Hwan.-98] S-A. Hwang, J.H. Hong, C-W Wu, "Sequential Circuit Fault Simulation Using Logic Emulation", IEEE. Trans. on Computer-Aided Design of Circuits and Systems, vol. 78, issue 8, pp. 724-736, Août 1998.

[ITRS-02] www.itrs.net/Links/2002Update/2002Update.pdf, "International Technology Roadmap for Semiconductors (ITRS) 2002 update"

[Jaul.-09] P. Jaulent, « Etude des effets singuliers transitoires dans les amplificateurs opérationnels linéaires par photo-génération impulsionnelle non linéaire », Doctorat de l'université Bordeux I (2009)

[JEDE.-07] JEDEC Standard, "Test Method for Alpha Source Accelerated Soft Error Rate", JESD89-2A, Octobre 2007.

[Jenn-94] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson, "Fault Injection into VHDL models: the MEFISTO Tool", Proc. 24th Int. Symp. on Fault-Tolerant Computing (FTCS-24), pp. 66-75, Juin 1994.

[John.-08] J. Johnson, W. Howes, M. Wirthlin, D. L McMurtrey, M. Caffrey, P. Graham, and K. Morgan. Using duplication with compare for on-line error detection in FPGA-based designs. In Proceedings of IEEE Aerospace Conference, 2008, page 1–11, 2008.

[JuYu.-10-1] Ju-Yueh Lee; Yu Hu; Majumdar, R.; Lei He; Minming Li, "Fault-tolerant resynthesis with dual-output LUTs," Design Automation Conference (ASP-DAC), 2010

[JuYu.-10-2] Ju-Yueh Lee; Zhe Feng; Lei He, "In-place decomposition for robustness in FPGA," Computer-Aided Design (ICCAD), 2010

[Kore.-07] I. Koren, C. M Krishna, and Inc Books24x7. Fault-tolerant systems. Elsevier/Morgan Kaufmann, 2007.

[Lapr.-04] J.-C. Laprie, "Sûreté de fonctionnement informatique: concepts, défis, directions", ACI Sécurité et Informatique, Toulouse, Novembre 2004.

[Leve.-99] R. Leveugle, "Towards modeling for dependability of complex integrated circuits", 5th IEEE Int. On-Line Testing Workshop (IOLTW'99), Rhodes, Grèce, pp. 194-198, Juillet 1999.

[Leve.-09] R. Leveugle, A. Calvez, P. Maistri, P. Vanhauwaert, "Statistical Fault Injection: Quantified Error and Confidence", Design, Automation and Test in Europe, Date 2009, pp 502-506

[Lewi.-05] D. Lewis, V. Pouget, F. Beaudoin, G. Haller, P. Perdu, P. Fouillat, "Implementing Laser-Based Failure Analysis Methodologies Using Test Vehicles", IEEE Trans. On Semiconductor Manufacturing, vol. 18, n° 2, Mai 2005.

[Lima-03] F. Lima, L. Carro, and R. Reis, "Designing fault tolerant systems into sram-based fpgas", Design Automation Conference, 2003. Proceedings, June 2003, pp. 650 - 655.

[More.-95] Y. Moreau, S. Duzellier, and J. Gasiot, "Evaluation of the upset risk in CMOS SRAM trough full three dimensional simulation", IEEE Trans. Nucl. Sci., vol. 42, Dec. 1995, pp. 1789–1796.

[Naza.-12] G. L. Nazar, L. Carro, "Fast error detection through efficient use of hardwired resources in FPGAs". 17th IEEE European Test Symposium (ETS), 2012, Annecy.

[Nikn.-11] M. Niknahad, O. Sander and J. Becker, "A study on fine granular fault tolerance methodologies for FPGAs," in 6th Int. Workshop on Reconfig. Communication-centric Systems-on-Chip (ReCoSoC), 2011 .

[Pies.-06] S.J. Piestrak. Dependable computing : Problems, techniques and their applications. In First Winter School on Self-Organization in Embedded Systems, Schloss Dagstuhl, Germany, 2006.

[Port.-06] M. Portolan, R. Leveugle, "A highly flexible hardened RTL processor core based on LEON2", IEEE Transactions on Nuclear Science, vol. 53, no. 4, part 1, August 2006, pp. 2069- 2075

[Poug.-99] V. Pouget, D. Lewis, H. Lapuyade, R. Briand, P. Fouillat, L. Sarger and M.-C. Calvet, "Validation of radiation hardened designs by pulsed laser testing and SPICE analysis", Microelectron. Reliab., vol. 39, 1999, pp. 931–935.

[Samp.-97] J.R. Sampson, W. Moreno, F. Falquez, "Validating Fault Tolerant Designs using Laser Fault Injection (LFI)", IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'97), Paris, France, pp. 175-183, Octobre 1997.

[Seda.-98] R. Sedaghat-Maman, E. Barke, "Real Time Fault Injection Using Logic Emulators", 3rd Asian And South Pacific Design Automation Conf., Yokohama, Japon, pp. 475-479, 1998.

[Shni.-98] N. Shnidman, W. Mangione-Smith, and M. Potkonjak, "On-Line fault detection for bus-based field programmable gate arrays," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 6, no. 4, pp. 656–666, Apr.1998.

[Ster.-07] L. Sterpone and M. Violante « A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs », IEEE Transactions on Nuclear Science, Vol 54, #4, pp 965 - 970, August 2007

[Upeg.-06] A. Upegui, E. Sanchez, «Evolving Hardware with Self-reconfigurable connectivity in Xilinx FPGAs », Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA, pp. 153 – 162, 15-18 June 2006

[Vanh.-08] P. Vanhauwaert, « Analyse de Sûreté par Injection de Fautes dans un Environnement de Prototypage à base de FPGA » Doctorat de l'Institut National Polytechnique de Grenoble (INPG), 2008.

[Vela.-00] R. Velazco, S. Rezgui, R. Ecoffet, «Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection», IEEE Transactions on Nuclear Science (TNS), Vol 47 , Issue 6, pp. 2405 – 2411, December 2000.

[Xili.-04-1] http://www.xilinx.com/support/documentation/ip_documentation/opb_hwicap.pdf

[Xili.-04-2] http://www.xilinx.com/support/documentation/application_notes/xapp662.pdf

[Xili.-04-3] Xilinx_Corp, « XAPP 290: Two flows for partial reconfiguration: Module Based and difference based » Sept. 2004

[Xili.-07-1] http://www.xilinx.com/support/documentation/application_notes/xapp290.pdf

[Xili.-07-2] http://www.xilinx.com/support/documentation/user_guides/ug002.pdf

[Xili.-08] www.dte.us.es/ing_inf/dise_comp/11_basic_fpga_arch_8.ppt

[Zout.-87] J. A. Zoutendik, L. S. Smith, G. A. Soli, and R. Y. Lo, "Experimental evidence for a new SEU model in a CMOS SRAM obtained from model verification", IEEE Trans. Nucl. Sci., vol. 34, Dec. 1987, pp. 358–366. tel-00009485, version 1 - 14 Jun 2005

Publications de l'auteur

Conférences internationales :

- R. Leveugle, M. Ben Jrad, "A New Methodology for Accurate Predictive Robustness Analysis of Designs Implemented in SRAM-based FPGAs", IEEE 17th International Conference on Electronics, Circuits, and System (ICECS), Athens, Greece, 12-15 December, 2010

- R. Leveugle, M. Ben Jrad, P. Maistri "Towards Virtual Fault-based Attacks for Security Validation", 4th International Conference on Dependability (DEPEND), French Riviera,Nice/Saint Laurent du Var, France August 21-27, 2011

- M. Ben Jrad, R. Leveugle "Pattern-based injections in processors implemented on SRAM-based FPGAs", 13th Latin-American Test Workshop (LATW), Quito, Ecuador, April 11-13, 2012

- M. Ben Jrad , R. Leveugle "Comparison of FPGA Platforms for Emulation-based Fault Injections using Run-Time Reconfiguration" 27th Design of Circuits and Integrated Systems (DCIS), Avignon ,France, November 28-30, 2012

- M. Ben Jrad , R. Leveugle "Evaluating a Low Cost Robustness Improvement in SRAM-based FPGAs" 19th International On-Line Testing (IOLT), Crete, Greece, July 8-10, 2013 (Poster)

- R. Leveugle , M. Ben JRAD "On Improving at no Cost the Quality of Products built with SRAM-based FPGAs " The 5th Asia Symposium on Quality Electronic Design (ASQED), Penang, Malaysia, August 26-28, 2013

Manifestations nationales :

- M. BEN JRAD, R. Leveugle, " Injection de fautes par reconfiguration partielle Application à un FPGA Virtex II Pro", Journées Nationales du Réseau Doctoral en Microélectronique (JNRDM), Montpellier, France, 7-9 Juin, 2010

Glossaire

AES : Advanced Encryption Standard

ALM : Adaptive Logic Module

Bit-flip : Inversion de la valeur mémorisée dans une cellule mémoire

BRAM : Block RAM

CLB : Conbinantial Logic Block

CTR : Compile-Time Reconfiguration

DDR : Double Data Rate

DMR : Dual modular redundancy

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

FSM : Finite State Machine

FPGA : Field Programmable Gate Array

IP : Intellectual Property

IOB : Input Output block

LAB : Logic Array Blocks

LUT : Look-Up-Table

MBF : Multiple Bit Flip

MBU : Multiple Bit Upset

MCU : Multiple Cell Upset

Netlist : Description niveau éléments composants le circuit ainsi que leurs interconnexions

Particule α : Particule hautement ionisée

Pipeline : Architecture qui découpe l'exécution d'une tâche en plusieurs étages qui sont opérés en parallèle, chacun sur une instruction différente

PLB : programmable Logic Block

PROM : Programmable Read Only Memory

RAM : Random Access Memory

RTL : Register Transfer Level

RTR : Run-Time Reconfiguration

SEE : Single Event Effects

SEFI: Single Event Functional Interrupts

SER : Soft Error Rate

SET : Single Event Transient

SEU : Single Event Upset

Soft Error : Erreur transitoire caractérisée par une modification de donnée réversible ou un état erroné temporaire

SoPC : System on Programmable Chip

SRAM : Static Random Access Memory

TMR : Triple modular redundancy

XOR : Porte logique réalisant un Ou-exclusif

Annexe A – Détails sur l'architecture Virtex

Le FPGA conventionnel de la famille Virtex, représenté dans la figure A-1, inclut des blocs d'entrée/sortie (I/O) localisés autour du périmètre du FPGA, des interfaces série multi-gigabit (MGT) intercalés avec les blocs I/O, des blocs logiques configurables (CLB), des blocs RAM (BRAM) intercalés entre les CLBs, de la logique de configuration, une interface de configuration, un processeur (dans notre cas un PowerPC) et un port d'accès à la configuration interne ***internal configuration access port (ICAP)***.

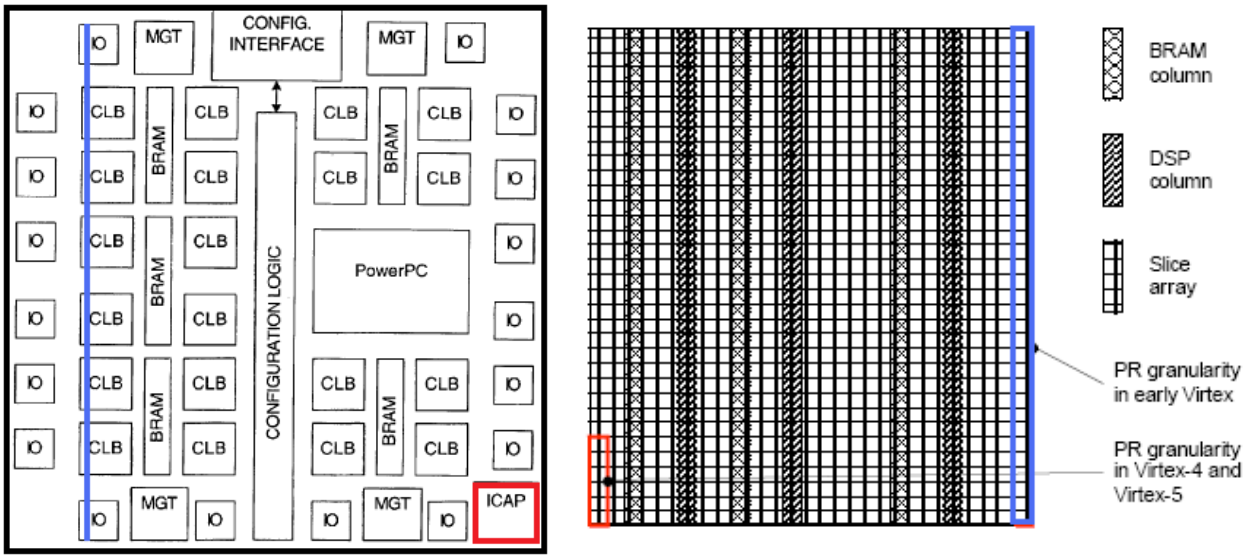


Figure A-1. Schéma synoptique d'un FPGA conventionnel

En général, le FPGA est configuré en fonction d'un ensemble de données de configuration, qui sont téléchargées à partir d'une mémoire externe telle qu'une mémoire ROM, vers le tableau de la mémoire de configuration du FPGA et à travers l'interface de configuration et la logique de configuration.

L'interface de configuration peut être par exemple une interface SELECTMAP ou l'interface JTAG. Le tableau de la mémoire de configuration peut être perçu comme un tableau rectangulaire de bits. Les bits peuvent appartenir à des CLBs, BRAMs, MGTs ou I/Os. Ils sont regroupés dans des frames qui ont une largeur d'un seul bit et la longueur du tableau (un exemple de frame est représenté dans la figure A-1 par un rectangle bleu). Une frame représente la plus petite granularité qui peut être lue ou écrite.

Les frames sont regroupées en colonnes qui correspondent plus ou moins aux ressources physiques du FPGA. La figure A-1 montre la relation entre ces ressources et la mémoire de configuration. Les valeurs des données de configuration sont chargées dans le tableau de la mémoire de configuration frame par frame à partir de la mémoire externe via l'interface de configuration.

Le nombre de frame et leur taille dépendent du FPGA ; le tableau A-I met en évidence la différence entre un FPGA V2P et un V5.

Tableau A-I Caractéristiques du bitstream pour Virtex2P et Virtex5

Device	**XCV2P30**	**V5LX110T**
Nombre de bits	11.589.920	31.886.848
Bits de configuration	11,575,552	31,110,144
Nombre de frame	1.756	23.712
Taille d'une Frame (en mot de 32 bits)	206	41
Frames par colonne CLB	22	36
CLBs par Frame	82	20

La différence notable au niveau d'une frame V5 et V2P, c'est que dans l'ancienne architecture, la frame couvre une colonne de 82 CLBs (traversant tout le long du FPGA) ; alors que sur le Virtex-5 que nous avons étudié, le FPGA est divisé en 8 lignes (voir la figure A-2)

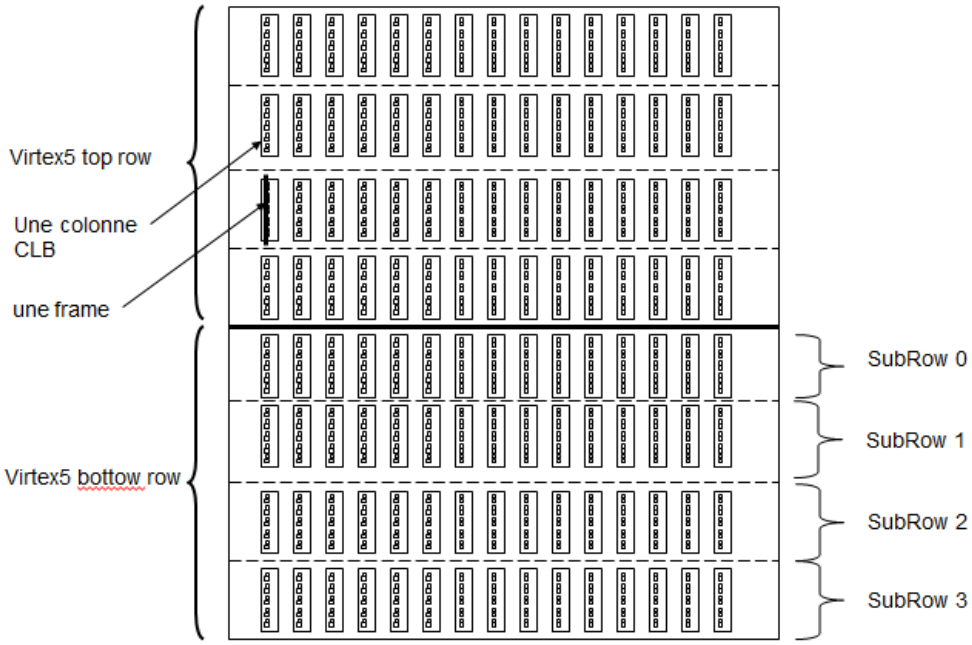


Figure A-2. Architecture du Virtex-5

Dans la V5 les frames couvrent des colonnes de 20 CLBs, et donc une décomposition plus fine du FPGA. Par conséquent, il faut 8 frames (de 41 mots) au lieu d'une seule (de 206 mots) pour la V2P, pour couvrir toute la longueur du FPGA, d'où la grande augmentation du nombre de frame.

Le nombre de frames par colonne varie en fonction du type de la colonne. Le tableau A-I représente la répartition des frames en fonction du type de la colonne et du FPGA.

Tableau A-II Répartition des frames de configuration en fonction des ressources

Type de colonne	**IOB**		**CLB**		**BRAM**		**BRAM Int**		**GCLK**	
Device	C/D	F/C	C/D	F/C	C/D	F/C	C/D	F/C	C/D	F/C
XC2VP30	2	26	46	22	8	64	8	22	1	4
XC5VLX110T	2*8	54	54*8	36	4*8	128	4*8	30	1*8	4

C/D : colonne par dispositif

F/C : frame par colonne

La multiplication par 8 correspond à la compartimentation du Virtex-V en 8 parties.

Chaque colonne de frames configure un type de ressources :

- Colonne GCLK : Configure toutes les ressources de gestion d'horloge dans le FPGA. Une seule colonne par dispositif.
- Colonne IOB : Cette colonne configure la tension des ports d'entrée/sortie de chaque côté de la puce. Pour la famille V2P les IOBs présents en haut et en bas du FPGA sont configurés dans les colonnes CLB.
- Colonne CLB : Cette colonne configure les ressources logiques et d'interconnexions des CLBs.
- Colonne BRAM : Configure tout l'espace mémoire utilisateur présent dans les BRAMS.
- Colonne Interconnexions BRAM : Configure les ressources d'interconnexions des BRAMs et des multiplieurs

Chaque dispositif contient une colonne de centre qui inclut la configuration des 4 horloges globales. Il existe deux colonnes IOB qui représentent la configuration de tous les IOBs aux extrémités gauche et droite du dispositif. La majorité des colonnes sont des colonnes CLB qui contiennent chacune une colonne CLB.

Il existe deux autres types de colonnes qui sont le contenu de la RAM et des interconnexions.

L'espace d'adressage total est divisé en blocs. Il existe deux types de blocs: RAM et CLB.

Le bloc RAM contient uniquement la colonne contenue du SelectRAM. Les autres colonnes appartiennent au bloc CLB.

Les deux espaces d'adressage RAM et CLB sont divisés en adresses minor et major. Chaque colonne de configuration a une adresse major unique dans l'espace RAM ou CLB. Chaque frame de configuration possède une adresse minor unique dans la colonne.

L'espace d'adressage CLB commence à 0 à partir de la colonne du centre ensuite alterne entre les moitiés gauche et droite du dispositif pour toutes les colonnes CLB, ensuite les colonnes IOB et enfin les colonnes interconnect SelectRAM.
Pour l'espace d'adressage RAM, la colonne contenu SelectRAM de gauche correspond à la colonne 0 alors que la colonne SelectRAM de droite correspond à la colonne 4 (pour la V5 et 8 pour la V2P).

Annexe B – Détails sur le composant ICAP

Afin de remédier aux problèmes d'endo-reconfiguration du FPGA, l'ICAP a été ajouté. Ce dernier permet d'accéder via la logique interne du FPGA au tableau de la mémoire de configuration (CLBs, BRAMs…). En d'autres termes, une partie de la configuration du FPGA peut reconfigurer une autre partie du FPGA.

La figure B-1 représente l'ICAP, qui peut être décrit comme suit:

- Deux bus entrée et sortie de largeur 8 bits. La figure B-2 illustre le format de paquet de données envoyé vers le bus d'entrée de l'ICAP.
- Signal d'entrée write qui indique quand une opération de lecture ou d'écriture va être effectuée sur le module ICAP.
- Signal Chip Enable CE.
- Signal d'horloge.
- Signal de sortie busy indique quand des données peuvent être reçues par l'ICAP.

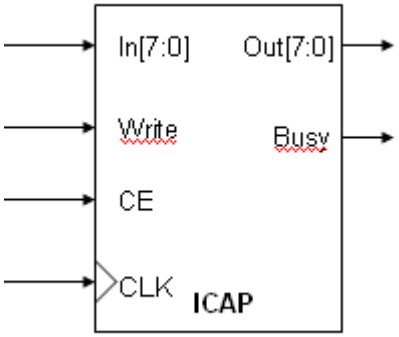


Figure B-1. Schéma synoptique de l'ICAP [Xili.-07-2]

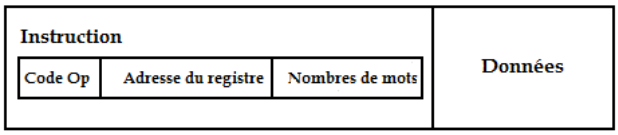


Figure B-2. Format d'un paquet de données

L'interface ICAP se situe dans le coin inférieur droit du FPGA, et peut être manipulée via les outils Xilinx EDK 8.2 et XPART (Xilinx Partial Reconfiguration Toolkit).

Présentation détaillée de l'ICAP

La figure B-3 est un schéma simplifié d'un système d'endo-reconfiguration. Le circuit inclut un processeur, un module de contrôle de BRAM, une BRAM, un module de contrôle de l'ICAP, un module ICAP, et un module de logique de configuration de l'ICAP.

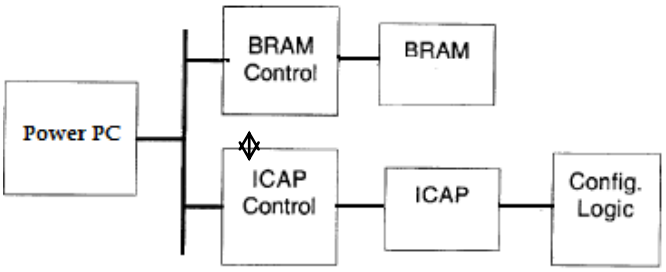


Figure B-3. Schéma simplifié d'un système d'endo-reconfiguration [Blod.-04]

Le module de contrôle de l'ICAP a un registre de 32 bits dont le contenu peut être mappé un à un avec les données correspondantes et les signaux de contrôle de l'ICAP. Le processeur lit et écrit sur ce registre via le bus OPB (comme illustré dans la figure B-4).

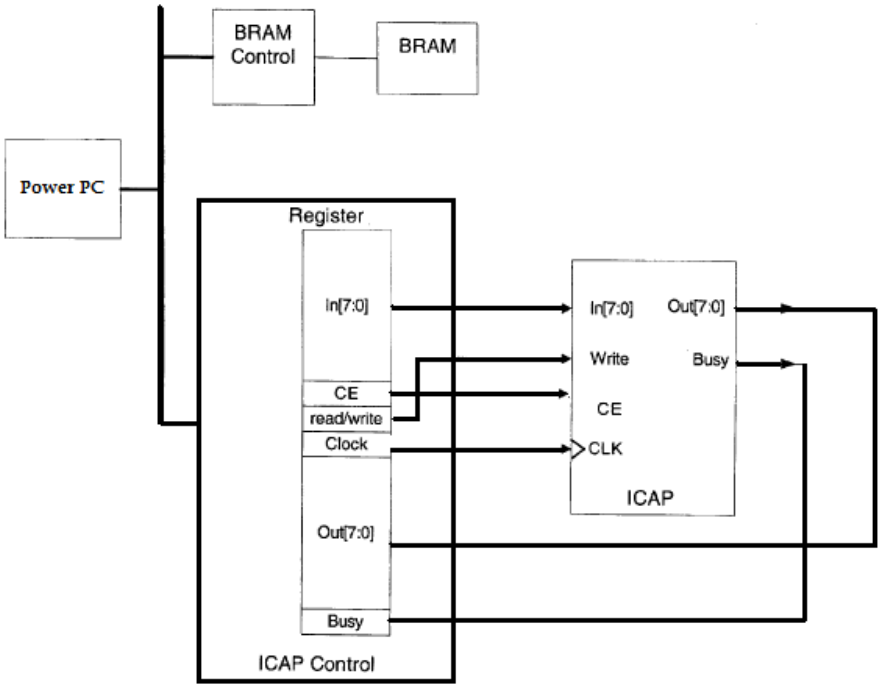


Figure B-4. Liaison entre le contrôle ICAP et l'ICAP [Blod.-04]

Le processeur est connecté au module mémoire (tel que la BRAM) et au module de contrôle de l'ICAP via le bus PLB. Le module de contrôle de l'ICAP est également lié à la BRAM à travers un bus dédié pour faciliter le transfert de données pour que les transferts soient réduits sur le bus système. Optionnellement, deux liaisons dédiées entre le processeur et la BRAM et entre le processeur et l'ICAP (voir figure B-5).

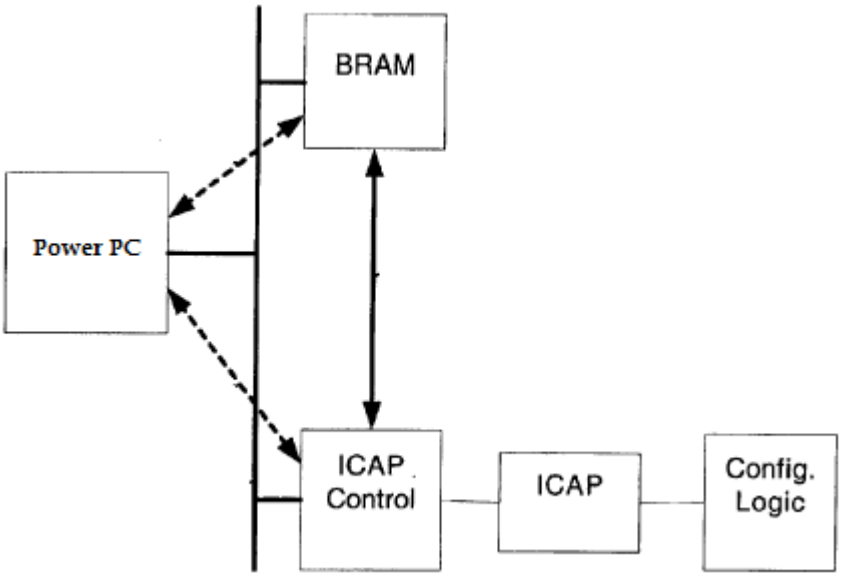


Figure B-5. Schéma d'un circuit intégré utilisant l'endo-reconfiguration [Blod.-04]

Le contrôle de l'ICAP comprend un moteur d'accès direct à la mémoire (DMA) et un périphérique de contrôle de registres (DCR), illustré dans la figure B-6. Ces éléments sont formés par les CLBs. Les commandes sont acheminées au moteur DMA à travers le DCR.

Le DCR est un registre de 32 bits qui stocke une entrée de port d'identification sur 4 bits (PORT_ID), une entrée write enable (WR) sur un bit, une entrée read-back enable (RB) sur un bit, un flag instruction done (DONE), un flag reconfiguration done (CONFIG_DONE), une adresse de début sur 11 bits (START_ADDR), une adresse de fin sur 11 bits (END_ADDR), et deux autres bits non utilisés.

Port ID	WR	RB	Done	Config_Done	Start_Addr	End_Addr

Figure B-6. Outil de contrôle de registre DCR

Les ports de l'ICAP sont accessibles à la logique du FPGA à travers la grille d'interconnexion générale.

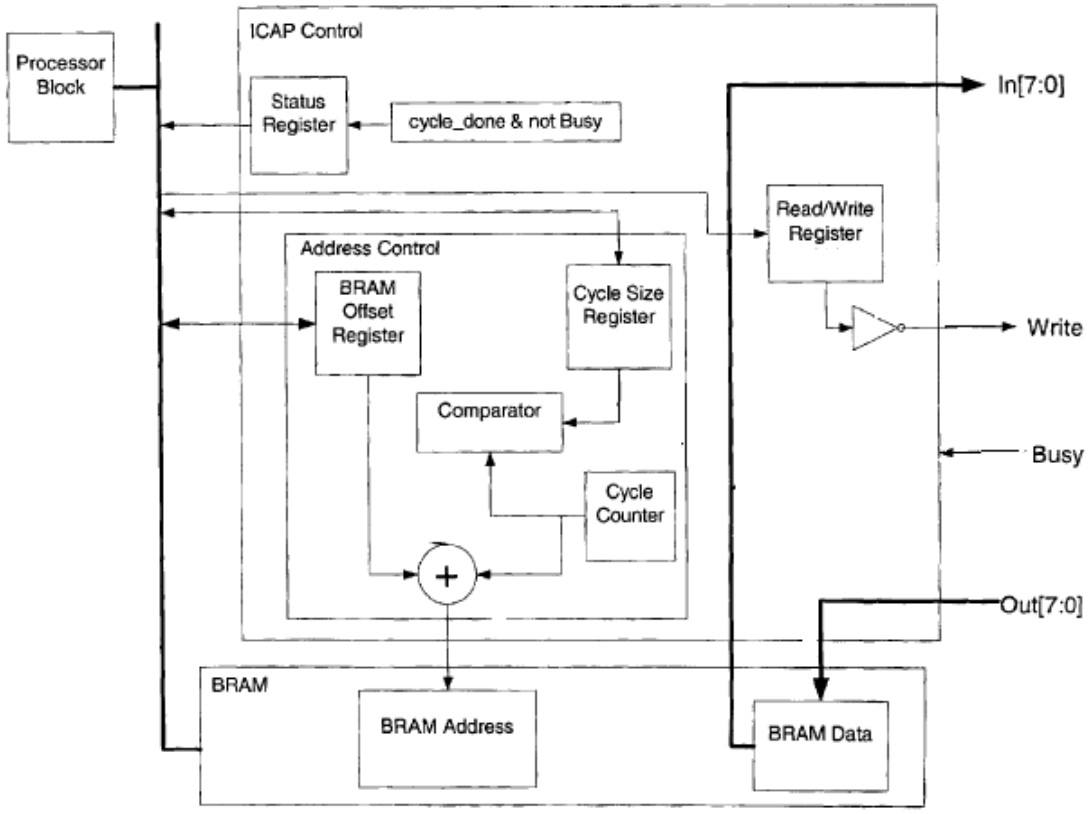


Figure B-7. Module de contrôle de l'ICAP [Blod.-04]

BRAM Status Register cycle_done, le module de contrôle de l'ICAP illustré dans la figure B-7 correspond à l'exemple du module de contrôle représenté dans la figure B-5. Ici, le bus de données bidirectionnel qui relie le module de contrôle de l'ICAP avec la BRAM est éclaté en deux bus unidirectionnels.

Le module de contrôle de l'ICAP est traversé par les bus de données in et out de l'ICAP. En d'autres termes, les bus in et out de l'ICAP sont directement reliés à la BRAM.

Le module de contrôle de l'ICAP inclut un registre d'état, un module de contrôle d'adresses et un registre de lecture/écriture :

- Le registre d'état est d'un seul bit de longueur ; dans le cas où ce registre est égal à 1, la fonction logique ((cycle_done 526) AND (NOT (Busy 220))), indique au processeur que le transfert de lecture/écriture du cycle est terminé. Après la lecture de ce registre il est immédiatement remis à zéro.
- Le module de contrôle d'adresse comprend un registre d'offset de la BRAM, un registre de taille de cycle, un comparateur, un compteur de cycles et un additionneur. Il génère les adresses mémoire (adresses BRAM) pour les données de la BRAM en cours de lecture/écriture par l'ICAP.

Mise en place d'un projet EDK avec le HwICAP

Il existe deux outils qui peuvent être utilisés pour l'écriture de programmes exécutés sur le PowerPC : l'outil traditionnel EDK 8.2 XPS peut être utilisé pour le matériel et les logiciels de conception, ou l'outil SDK peut être utilisé pour la conception de logiciels. Ce dernier est une version adaptée d'Eclipse et fournit à l'utilisateur un environnement de développement convivial et les fonctionnalités d'Eclipse. Quelque soit l'outil utilisé, des bibliothèques et fichiers d'en-tête vont être inclus pour l'utilisation du matériel ICAP. Heureusement, les périphériques associés à l'ICAP englobent les fichiers d'entête et des fonctions de haut-niveau en vue d'interfacer avec l'ICAP pour l'auto-reconfiguration. Le fichier en-tête qui doit être inclus dans tous les programmes faisant appel à l'ICAP est "xhwicap.h".

Nous avons choisi de développer notre programme d'auto-reconfiguration à l'aide d'EDK XPS.

Lors de la création d'un système d'auto-reconfiguration partielle, il y a beaucoup de spécificités à prendre en considération. En vue d'intégrer l'ICAP et le powerPC nous devons procéder comme suit :

1) Créer un projet EDK 8.2 avec l'assistant de projet. Préciser les paramètres du Virtex 2 Pro.

2) Rajouter le module de reconfiguration partielle « opb_hwicap »puis le connecter au reste du système avec le bus OPB.

3) Génération du bitstream

4) Dans l'onglet application cliquer sur « Add software application Project » et créer un projet en C qui tournera sur le PPC405_0. Génération du fichier exécutable de type .elf.

5) Connecter la carte au PC et charger le bitstream avec le port JTAG.

Passons maintenant au chargement de l'exécutable : pour faciliter l'opération nous avons créé un script « load » faisant appel à un fichier .tcl permettant de charger et de lancer le fichier « executable.elf » sur le PowerPC.

Pour la communication avec la carte nous avons utilisé l'Hyper Terminal.

Programme de reconfiguration partielle

Le programme que nous avons conçu s'exécute sur le power PC afin de lui permettre de faire de la reconfiguration partielle. Notre programme permet à l'utilisateur de :

- Créer une instance HwICAP qui nous permettra de récupérer toutes les caractéristiques du FPGA (tels que le nombre de colonnes de bram, de mots par frame…)
- Lire et reconfigurer le contenu d'une frame en précisant dans quelle colonne elle se trouve et son emplacement dans cette dernière.

Les frames sont lues séquentiellement avec les adresses ascendantes pour chaque opération.
Plusieurs frames consécutives peuvent être lues ou écrites avec une seule commande de configuration.
La reconfiguration est accomplie en utilisant un mécanisme qui devra respecter un ordre de lecture, modification et écriture des frames. Pour modifier le circuit FPGA, le périphérique détermine la configuration des *frames* qui doivent être modifiées, ensuite on lit chaque *frame*, une à la fois dans le bloc mémoire.
Le contenu de chaque *frame* est modifié avant d'être écrit à l'ICAP. L'OPB HWICAP prévoit également la relecture de la configuration des ressources d'états. Dans ce cas, les *frames* sont lues dans le bloc RAM et les bits correspondants sont extraits du bloc RAM.
Nous allons utiliser les API de HWICAP pour accéder au bloc mémoire et à l'ICAP et par la suite lire et écrire sur les *frames.*

Pour les colonnes CLBs du V2P, les 18 premiers bits contrôlent les 2 IOBs en haut de la colonne suivis des 18 bits alloués pour la colonne, et enfin les 18 bits suivants contrôlent les 2 IOB en bas de la colonne. Tandis que pour la V5 ces colonnes sont exclusivement réservées aux CLBs.
Traditionnellement, l'endo-reconfiguration est réalisée par le chargement de frames de reconfiguration pré-générées dans la BRAM, en utilisant une logique personnalisée, et en écrasant les frames ciblées du tableau de la mémoire de configuration par ces frames pré-générées. Cette approche présente les inconvénients suivants:

- Flexibilité limitée (frames pré-générées…)
- Surcoût en efficacité, puisque la reconfiguration modifiée doit être pré-chargée.
- Le chargement de toute la frame alors qu'une partie de la frame doit être reconfigurée.

L'API HWICAP

Les fonctions que nous avons développées ainsi que les fonctions les plus importantes de l'API HwIcap :

La récupération des caractéristiques du FPGA :

Pour l'utilisation de l'IP OPB_HwIcap dans notre environnement nous devons tout d'abord l'instancier avec notre fonction Init_Icap. Celle-ci permet par ailleurs de récupérer et d'afficher les différentes caractéristiques du FPGA (nombre de blocs bram, blocs clb par frame…). Cette fonction fait appel à la méthode *XHwIcap_Initialize* de Xilinx. Il faut noter qu'on ne doit avoir qu'une seule instance de XHwIcap à la fois.

Lecture d'une frame :

Read_Frame permet la lecture de la frame choisie et la met dans une zone de stockage tampon avec la méthode xilinx *XHwIcap_DeviceReadFrame*. Cette dernière sera lue grâce à la méthode *HwIcap_StorageBufferRead*

Injection de fautes dans une frame :

Reconfigure_Frame est la fonction de notre environnement qui permet la reconfiguration d'une frame choisie en modifiant son contenu et cela se fait de la manière suivante: lecture de la frame (XHwIcap_DeviceReadFrame met la frame choisie dans le buffer) suivie d'une modification (XHwIcap_DeviceWriteFrame) et enfin l'écriture de la nouvelle configuration (XHwIcap_StorageBufferWrite).

Injection de fautes dans une LUT et bascule

Reconfigure_FF et Reconfigure_LUT sont les fonctions pour la reconfiguration de bascule et de LUT. Pour choisir la bascule ou la LUT il faut savoir dans quelle colonne (XHwIcap_mSliceX2Col), dans quelle ligne (XHwIcap_mSliceY2Row) elle se situe et sa slice (XHwIcap_mSliceXY2Slice). Les valeurs récupérées seront des paramètres pour les méthodes de lecture et d'écriture dans les CLB FF et LUT (XHwIcap_GetClbBits et XHwIcap_SetClbBits)

NB : La seule différence lors de la reconfiguration (ou la lecture) d'une bascule ou d'une LUT est un paramètre que prennent les méthodes GetClbBits et SetClbBits (XHI_CLB_Lut pour les LUT et XHI_CLB_FF pour les bascules).

Injection de fautes par bitstream partiel

Reconfig est la méthode pour la reconfiguration du FPGA avec le bitstream partiel via la méthode *XHwIcap_SetConfiguration* celle-ci prend en paramètre le fichier .bit contenant le bitstream partiel ainsi que la taille de ce dernier et l'hwIcap instancié.

Le fichier .bit sera chargé à la SDRAM et récupéré par des méthodes de la bibliothéque « sysace_stdio.h »

Injection de fautes dans la Bram

- Write_Bram est la méthode pour la modification du contenu d'une adresse dans la bram et cela en s'appuyant sur la méthode Xilinx XHwIcap_mSetBram et on peut bien évidemment récupérer le contenu d'une adresse avec XHwIcap_mGetBram.

Mémoires de stockage

❖ Pour les données et instructions du PowerPC :

Dans l'outil de développement fourni par Xilinx (EDK 8.2), lors de la mise en œuvre du PowerPC sur le FPGA Virtex-II Pro, les blocs de RAM embarqués (BRAMs) sont automatiquement initialisés avec les données et les instructions du programme lors de la génération du bitstream. Le contenu des BRAMs est précisé dans le fichier Block Memory Map (BMM), mis à jour suite au placement et routage.
Les données et les instructions du programme peuvent également être stockées dans la SDRAM externe. Les deux avantages de cette solution sont que la SDRAM est beaucoup plus grande que la BRAM disponible à l'intérieur du FPGA et que la recharge du bitstream sur la carte n'est pas obligatoire, à chaque modification du programme exécuté par le PowerPC car la SDRAM est configurée séparément.
Donc nous avons choisi de mettre les données et les instructions du programme du PowerPC dans la SDRAM.

❖ Pour les bitsteams partiels :

Comme pour le stockage des données et des instructions du PowerPC, une décision doit être prise concernant le stockage des bitstreams partiels dans le système. La taille des bitstreams partiels est généralement semblable à celle du code de l'application C gérant les injections. Le code pour la reconfiguration du FPGA demeure de taille relativement constante, indépendamment de la taille des bitstreams partiels. Aussi, lors de la configuration du système, 128Ko de bloc RAM sont réservés pour le code du programme et 64Ko sont réservés pour les données du programme. Toutefois, plusieurs bitstreams partiels devront être stockés dans la mémoire par la suite, par conséquent l'espace mémoire nécessaire pour le stockage des bitstreams sera en général nettement plus grand que celui nécessaire pour l'application.

Les bitstreams sont donc chargés dans un premier temps dans la SDRAM avec le Xilinx Microprocessor Debugger (XMD). Ils sont ensuite temporairement et individuellement stockés, par l'application gérant l'injection, dans la RAM du PowerPC pendant leur utilisation par l'ICAP. Un système de liste est géré pour pouvoir remplacer dans la RAM un bitstream partiel lu par l'ICAP par le bitstream partiel nécessaire pour l'injection suivante.

Annexe C – Mise en place de la méthode de durcissement du circuit en mode ECO

Le mode ECO (Engineering Change Orders) de Quartus est un mode dans lequel le concepteur peut faire des modifications au circuit après la phase de placement et routage. Dans notre cas, l'utilisation de cette approche vise à implanter les comparateurs et les détecteurs. Concernant la duplication des fonctions du circuit, elle sera faite par option d'assignement spécifique aux nœuds qu'on veut dupliquer, et ce, grâce à l'éditeur d'assignement.

Duplication et réorganisation des cellules du circuit

On détermine tout d'abord les fonctions du circuit à dupliquer. Après la compilation du projet, nous pouvons avoir une vue de notre puce FPGA avec le circuit implanté, et ce à travers l'outil « Chip Planner » tel que le montre la figure C-1.

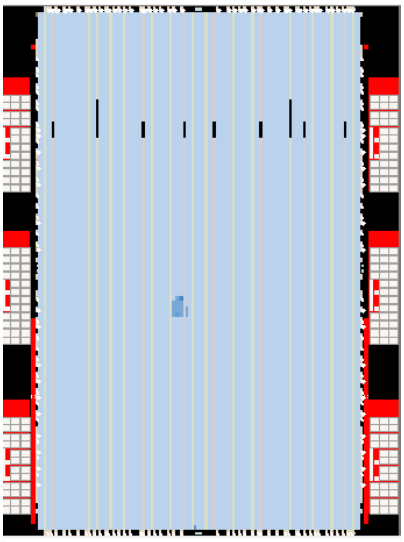


Figure C-1. Chip Planner de Quartus II

Une fois que les nœuds sont déterminés, nous allons exploiter une option d'assignement dans l'éditeur d'assignement des nœuds, nommé « Manual Logic Duplication ». En fait, cette option est proposée par Quartus pour un autre motif : si la sortie d'un nœud (registre ou cellule combinatoire) a pour destination deux autres nœuds dont l'un est loin de lui, alors, grâce à cette option, on guide le compilateur à dupliquer le nœud source et à mettre le réplica proche de ce nœud distant afin de réduire le chemin critique entre les deux.

On va donc chercher à exploiter cette option et on va l'assigner à chaque fonction du circuit en choisissant comme destination de la fonction réplica les comparateurs qu'on rajoute dans l'étape de détection d'erreurs. C'est ainsi qu'on peut dupliquer toutes les fonctions du circuit au dehors du code RTL. Un autre avantage de cette méthode, c'est le fait que chaque fonction et son réplica sont mis automatiquement dans le même ALM.

Ajout de comparateurs et des détecteurs

Pour faire des modifications au niveau de circuit compilé, nous devons choisir le mode ECO dans Chip Planner. Ce dernier nous permet de créer de nouveaux nœuds dans notre projet compilé, en choisissant l'ALM correspondant tel que le montre la figure C-2:

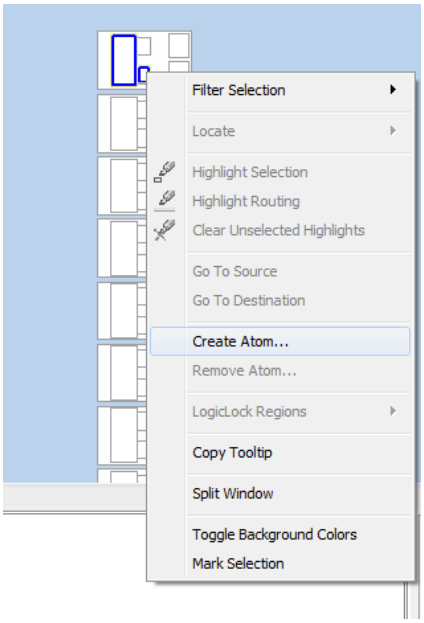


Figure C-2. Création d'un nouveau nœud dans le circuit

Le nœud est donc rajouté à notre circuit. Il faut maintenant spécifier ses entrées et ses sorties ainsi que sa fonction dans l'éditeur « Resource Property Editor » (voir figure C-3).

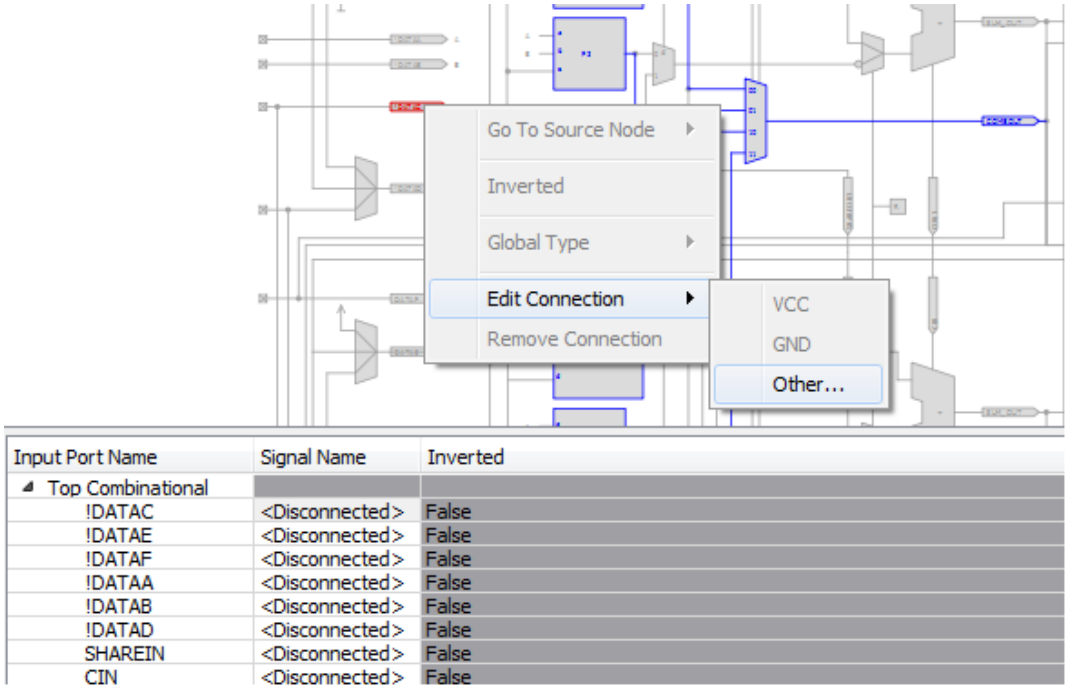


Figure C-3. Spécification des entrées du nœud dans l'éditeur « Resource Property Editor »

Pour les comparateurs, les entrées des nœuds seront les signaux des fonctions et leurs réplicas qui se trouvent dans la liste des signaux de « Node Finder ». La sortie sera reliée au détecteur.

Pour la fonction réalisée par le comparateur, il s'agit d'une fonction combinatoire introduisant ses 6 entrées.

L'éditeur « Resource Property Editor » ne nous permet pas d'écrire explicitement la fonction réalisée. On peut agir uniquement sur les fonctions des 4 LUTs constituant la partie combinatoire de l'ALM (F0, F1, F2 et F3).

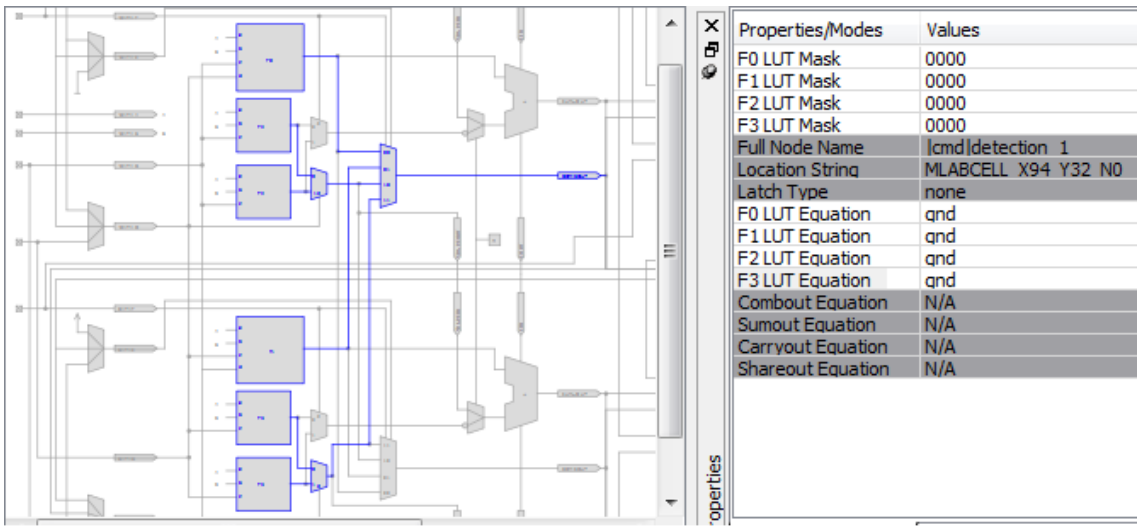
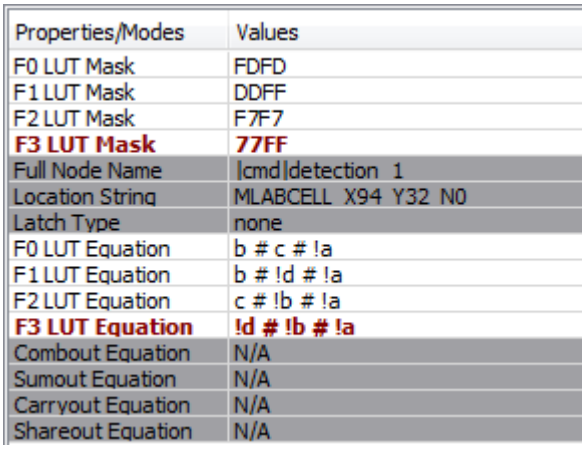


Figure C-4. Equations des LUTs de l'ALM

Pour réaliser la fonction souhaitée du comparateur, nous avons développé un exemple simple effectuant cette fonction et nous avons dégagé les équations des LUTs correspondantes qui dépendent des entrées des nœuds. Pour faciliter l'écriture de ces fonctions, il suffit d'écrire la valeur des masques des LUTs au lieu d'écrire la fonction assurée par chacune des LUTs.



Properties/Modes	Values
F0 LUT Mask	FDFD
F1 LUT Mask	DDFF
F2 LUT Mask	F7F7
F3 LUT Mask	**77FF**
Full Node Name	|cmd|detection 1
Location String	MLABCELL X94 Y32 N0
Latch Type	none
F0 LUT Equation	b # c # !a
F1 LUT Equation	b # !d # !a
F2 LUT Equation	c # !b # !a
F3 LUT Equation	**!d # !b # !a**
Combout Equation	N/A
Sumout Equation	N/A
Carryout Equation	N/A
Shareout Equation	N/A

Figure C-5. Valeurs des MASKs des comparateurs

A ce stade, les nœuds sont placés dans le circuit par le concepteur mais rien n'assure que les nouveaux changements sont corrects et que le routage physique, du et vers, ces nœuds sera effectué. Pour ce faire, il faut cliquer sur le bouton « Check and Save All Netlist Changes » dans l'éditeur.

Finalement, si les changements sont corrects, l'outil « Fitter » met à jour la netlist et lance de nouveau la compilation à partir de la phase « Fitter » en conservant les autres résultats obtenus suite à l'étape de synthèse.
Le rapport de compilation est mis à jour et tient compte des nouveaux nœuds ajoutés durant la phase « Post-compilation ».

Contraintes et limite de l'approche

Malheureusement, nous sommes face à un problème : l'affectation de l'option « Manual Logic Duplication » aux nœuds du circuit sera enregistrée dans le fichier QSF du projet. Ce fichier de contraintes sert comme un fichier d'entrée à l'outil « Fitter » de Quartus. A ce stade, les cellules de détection d'erreurs ne sont pas encore créées, car elles le sont dans une phase ultérieure « Post compilation ». Par conséquent, la duplication échoue et l'outil « Fitter » estime qu'il n'y a pas un chemin physique entre le nœud source et le nœud de destination !

Can't duplicate source node « Mux 0~0 » -- no path exists between source node and destination node « detection »

Nous avons essayé plusieurs méthodes mais aucune n'a contourné le problème:

- **Essai 1 :**

Nous avons essayé d'agir sur les fichiers d'entrée de l'outil « Fitter » en introduisant le code Tcl de l'ajout des cellules de détection d'erreurs dans le fichier QSF mais un message d'erreur s'affiche indiquant que les modifications faites dans ce fichier sont fausses.

The Quartus II Settings File changed outside of Quartus II Software and contains errors. Rewriting the Quartus II Settings File with the current state of assignments in the Quartus software

- **Essai 2 :**

Nous avons inséré les nouvelles cellules dans le circuit après la compilation du projet, puis nous avons enregistré ces changements dans la netlist. Ensuite, nous avons exporté ces changements en fichier .csv (Comma Separated Value Files). Par la suite, on a spécifié ce fichier comme un des fichiers de contraintes d'assignation.

➔ Aucun résultat

- **Essai 3 :**

Nous avons essayé d'exporter ces changements dans un fichier .qxp servant comme une entrée à l'outil de synthèse.

Le fichier .qxp est une netlist précompilée que nous pouvons utiliser comme fichier source du projet. Il contient toutes les modifications faites dans le circuit lors de son partitionnement en plusieurs entités « Design Partition ». Dans ce cas, il s'agit de compilation incrémentale, car nous pouvons compiler chaque entité avec des paramètres différents. Ce fichier peut être exporté aussi dans le cas où le circuit contient une seule entité.

Nous avons donc fait nos changements après compilation et nous avons exporté le fichier correspondant. Pour ce faire, dans la barre Menu on choisit l'onglet « Project » puis on clique sur « Export Design Partition ».

L'étape suivante consiste à le rajouter comme fichier source du projet puis on compile notre projet. Malheureusement, une erreur se produit et le message d'erreur suivant s'affiche :

Incremental Compilation export does not support exporting a project after applying ECO changes

- **Essai 4 :**

Nous avons exécuté l'étape « Analysis and Synthesis » et nous avons voulu insérer les cellules de détection avant l'étape « Fitter » qui fait appel au fichier de contraintes .QSF ». Ceci n'était pas faisable et une erreur s'affiche :

ECO cannot be done before fitting

TITRE : Robustesse par conception de circuits implantés sur FPGA SRAM et validation par injection de fautes

RESUME Cette thèse s'intéresse en premier lieu à l'évaluation des effets fonctionnels des erreurs survenant dans la mémoire SRAM de configuration de certains FPGAs. La famille Virtex II Pro de Xilinx est utilisée comme premier cas pratique d'expérimentation. Des expérimentations sous faisceau laser nous ont permis d'avoir une bonne vue d'ensemble sur les motifs d'erreurs réalistes qui sont obtenus par des sources de perturbations réelles.

Une méthodologie adaptée d'injection de fautes a donc été définie pour permettre une meilleure évaluation, en phase de conception, de la robustesse d'un circuit implanté sur ce type de technologie. Cette méthodologie est basée sur de la reconfiguration dynamique. Le même type d'approche a ensuite été évalué sur plusieurs cibles technologiques, ce qui a nécessité le développement de plusieurs environnements d'injection de fautes. L'étude a pour la première fois inclus la famille AT40K de ATMEL, qui permet un type de reconfiguration unique et efficace.

Le second type de contribution concerne l'augmentation à faible coût de la robustesse de circuits implantés sur des plateformes FPGA SRAM. Nous proposons une approche de protection sélective exploitant les ressources du FPGA inutilisées par l'application. L'approche a été automatisée sur plusieurs cibles technologiques (Xilinx, Altera) et l'efficacité est analysée en utilisant les méthodes d'injection de fautes précédemment développées.

Mots clés : FPGAs SRAM, mémoire de configuration, injection de fautes, robustesse détection d'erreur.

TITLE: Robust design of circuits implemented on SRAM-based FPGAs and validation by fault injection

ABSTRACT This thesis focuses primarily on the evaluation of the functional effects of errors occurring in the SRAM configuration memory of some FPGAs. Xilinx Virtex II Pro family is used as a first case study. Experiments under laser beam allowed us to have a good overview of realistic error patterns, related to real disturbance sources.

A suited fault injection methodology has thus been defined to improve design-time robustness evaluations of a circuit implemented on this type of technology. This methodology is based on run-time reconfiguration. The approach has then been evaluated on several technological targets, requiring the development of several fault injection environments. The study included for the first time the ATMEL AT40K family, with a unique and efficient reconfiguration mode.

The second type of contribution is focused on the improvement at low cost of the robustness of designs implemented on SRAM-based FPGA platforms. We propose a selective protection approach exploiting resources unused by the application. The approach has been automated on several technological targets (Xilinx, Altera) and the efficiency has been analyzed by taking advantage of the fault injection techniques previously developed.

Keywords: FPGAs SRAM, configuration memory, fault injection, robustness, error detection.

INTITULE ET ADRESSE DU LABORATOIRE
Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.