Analyse de sûreté par injection de fautes dans un environnement de prototypage à base de FPGA

Pierre Vanhauwaert

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

T H E S E

pour obtenir le grade de

DOCTEUR DE l'IP Grenoble

Spécialité : Micro et Nano Electronique

préparée au laboratoire **TIMA** dans le cadre de
l'Ecole Doctorale d'« Electronique, Electrotechnique, Automatique, Télécommunications, Signal »

présentée et soutenue publiquement

par

Pierre VANHAUWAERT

le 04 avril 2008

TITRE :

ANALYSE DE SURETE PAR INJECTION DE FAUTES DANS UN ENVIRONNEMENT DE PROTOTYPAGE A BASE DE FPGA

DIRECTEUR DE THESE :
Régis LEVEUGLE

JURY

M.	Frédéric PETROT	, Président
M.	Fabrice MONTEIRO	, Rapporteur
M.	Habib MEHREZ	, Rapporteur
M.	Régis LEVEUGLE	, Directeur de thèse
M.	Philippe ROCHE	, Co-encadrant
M.	Laurent HILI	, Examinateur

Remerciements

Je remercie Monsieur Bernard Courtois, directeur du laboratoire TIMA, pour m'y avoir accueilli et Madame Dominique Borrione, qui a repris la direction, pour m'y avoir conservé.

Je remercie Monsieur Frédéric Petrot, Professeur à Grenoble INP, pour m'avoir fait l'honneur d'être le président de mon jury de thèse.

J'adresse mes remerciements à Messieurs Habib Mehrez, Professeur à l'Université Pierre et Marie Curie (Paris VI), et Fabrice Monteiro, Professeur à l'Université de Metz, pour l'intérêt qu'ils ont porté à mon travail en tant que rapporteurs.

Je remercie Monsieur Philippe Roche, Docteur-Ingénieur chez STMicroelectronics, pour m'avoir permis d'effectuer cette thèse, pour l'intérêt qu'il a porté à mon travail durant ces trois années (et demi) et pour son encadrement.

Je remercie Laurent Hili, Ingénieur à l'Agence Spatiale Européenne, d'avoir apporté son expérience dans le Jury.

Je souhaite également remercier Robin Rolland pour son aide, Alexandre Chagoya pour sa disponibilité, Abdelaziz Ammari, Karim Hadjiat, Michele Portolan, Paolo Maistri et tous ceux avec qui j'ai travaillé pendant cette thèse.

Enfin je tiens tout particulièrement à remercier Monsieur Régis Leveugle, Professeur à Grenoble INP, pour m'avoir proposé ce sujet de thèse, pour son encadrement, pour ses conseils et pour la confiance qu'il m'a apportée d'abord en stage puis en thèse.

Merci à toutes les personnes du TIMA que j'ai eu la chance de rencontrer et de cotoyer en particulier les membres du groupe ~~QLF~~ ARIS : Karim, Abdelaziz, Michele, Paolo, JB, Vincent et Gaëtan.

Merci aux membres de mon groupe d'adoption avec qui j'ai passé de très bons moments depuis mon stage en 2003 jusqu'à aujoud'hui, au TIMA et en dehors.

Merci aux ergiens Grenoblois et Toulousains pour leur enthousiasme.

Merci à Pascale et Jérôme pour leur amitié.

Merci à la famille Foltête, petits et grands, pour leur soutien. Merci à la (grande) famille Dupré La Tour pour leur acceuil et en particulier à Mathieu pour tout ce qu'il m'a fait découvrir.

Merci beaucoup à Solenne pour sa compréhension, son soutien, ses encouragements lors de la rédaction et pour tout ce qu'elle m'apporte dans la vie.

Merci énormément à ma frangine (qui a quand même réussi à soutenir sa thèse un an avant moi malgré ses quatre ans d'avance). Merci pour tout.

Table des matières

Liste des figures

Liste des tableaux

Liste des équations

Introduction

L'évolution des technologies microélectroniques augmente la sensibilité des circuits intégrés face à leur environnement. Leurs dimensions plus petites et leurs tensions d'alimentation plus faibles, entre autres, les rendent moins résistants face aux « attaques » naturelles extérieures comme par exemple les impacts de particules. Ces phénomènes sont observés dans un environnement spatial depuis de nombreuses années mais doivent désormais être considérés dans l'environnement terrestre. Par exemple, le basculement d'un point mémoire peut être le résultat de l'impact d'un neutron atmosphérique. Ces « attaques » peuvent engendrer des fautes transitoires dans les circuits et en conséquence modifier les données qu'ils traitent ou leur comportement. Avec les dimensions actuelles de circuits, les problèmes liés à la perte de l'intégrité du signal doivent également être considérés comme des sources de fautes.

Les fautes initiales peuvent éventuellement, suivant plusieurs conditions, conduire à des erreurs. Or le comportement erroné d'un circuit peut être tout à fait inacceptable si celui-ci est mis en œuvre dans une application dite « critique ». C'est la raison pour laquelle les fabricants ont besoin d'outils efficaces pour analyser le comportement de leurs circuits lorsqu'une ou plusieurs erreurs se produisent lors de son fonctionnement.

La découverte par le concepteur d'un comportement inacceptable le pousserait à modifier sa description initiale pour supprimer cette vulnérabilité. En conséquence, pour minimiser un surcoût éventuel, il est nécessaire de réaliser une analyse tôt dans le flot de conception, particulièrement avant que le circuit ne soit fabriqué. Une analyse à haut niveau d'abstraction permet de choisir les protections les plus adaptées. Si un système n'est pas suffisamment robuste il est possible soit de modifier la description originale (ajout de mécanismes architecturaux de protection par exemple) soit de définir des contraintes à plus bas niveau (choix d'une technologie spécifique ou de cellules de bibliothèque plus robustes), et cela à moindre coût.

L'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, injectées 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 haut niveau. La simulation, plus coûteuse en temps, peut cependant permettre des analyses plus complètes que l'émulation.

L'objectif de ce travail de thèse est le développement d'une méthodologie et d'un environnement améliorant l'étude de la robustesse de circuits intégrés. L'environnement mettra en œuvre un prototype matériel du circuit à analyser. L'analyse devra en outre être la plus automatisée possible, la plus optimisée possible du point de vue des durées d'expérimentation et être portable sur de nombreuses plateformes de prototypage. Elle devra être également très flexible du point de vue des types d'analyse

possible. Nous souhaitons répondre aux attentes d'une majorité de concepteurs et donc permettre l'analyse du plus grand nombre de types de circuits dans un environnement unique.

Notre travail se focalisera sur l'analyse de circuits numériques synchrones. Indépendamment de la technologie cible pour le circuit fabriqué (CMOS massif, CMOS SoI…), les descriptions considérées seront les descriptions synthétisables parce que seules celles-ci ont pour finalité une implantation matérielle et parce que l'utilisation d'un prototype le requiert. Les descriptions traitées seront de niveau transfert de registres (RTL) mais l'approche pourrait être étendue à des descriptions synthétisables de plus haut niveau.

Du fait de l'approche à haut niveau certains phénomènes ne seront pas étudiés ou pris en compte dans notre recherche d'un environnement et d'une méthodologie optimisés. En effet certains phénomènes physiques, comme par exemple la propagation d'un pic de courant dans de la logique combinatoire, ne peuvent être analysés qu'après caractérisation du circuit fini. En conséquence notre attention se portera sur les conséquences des erreurs induites dans des éléments mémoires (registres, bancs de mémoires). Les phénomènes se produisant dans un bloc de logique combinatoire seront considérés uniquement en partant d'hypothèses sur leur mémorisation.

Enfin l'analyse de sûreté ne saurait faire abstraction de la charge de travail des circuits considérés. L'analyse réalisée est une analyse dynamique et fonctionnelle. Cependant il est parfois impossible de connaître quelles sont exactement les tâches que devra effectuer un circuit dans son environnement d'utilisation (confidentialité, clients multiples, applications multiples…). Dans ce cas la mise en œuvre d'une application représentative pour chaque circuit sera nécessaire.

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

Le chapitre 2 est l'état de l'art des techniques d'analyse par 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 aux objectifs décrits ci-dessus.

Le chapitre 3 a pour but de présenter l'environnement d'analyse que nous proposons. A partir du flot d'analyse existant au TIMA nous présentons comment intégrer et tirer profit d'un processeur dans un environnement FPGA afin d'obtenir un environnement flexible et performant. Les possibilités qui sont offertes à l'utilisateur pour obtenir la configuration de l'environnement la plus performante y sont décrites.

Afin d'améliorer les performances de l'environnement nous proposons plusieurs techniques innovantes en terme d'instrumentation et de contrôle des injections. Ces techniques sont détaillées et évaluées par rapport aux techniques existantes dans le chapitre 4.

Le chapitre 5 porte sur l'évaluation des performances attendues lors de la mise en œuvre de l'environnement et des techniques proposés dans les deux précédents chapitres. Cette évaluation a pour objectif de comparer quantitativement certaines techniques et de guider l'utilisateur dans ses choix pour l'analyse.

Le chapitre 6 présente tout d'abord un outil développé pour automatiser la configuration de l'environnement d'analyse. Pour terminer il valide la méthodologie, les techniques et l'environnement grâce à l'analyse de deux circuits significatifs. Ces analyses mettent également en lumière les avantages les plus marquants de l'environnement.

1 Sûreté de fonctionnement et modèles de fautes

1.1 Contexte et terminologie

Les effets des radiations dans l'espace sont connus depuis 1958 après la découverte des ceintures de Van Allen à partir des expériences effectuées par des compteurs Geiger embarqués dans les satellites Explorer. Des particules de haute énergie sont piégées dans ces ceintures de Van Allen à cause du champ magnétique terrestre qui force ces particules à décrire des boucles autour de la Terre.

La première observation des effets néfastes de ces particules sur les applications spatiales fut la perte du satellite Telstar en 1962, suite à une trop forte exposition aux radiations. L'orbite de Telstar, premier satellite de télécommunications, se situait dans les ceintures de radiations et son exposition fut encore aggravée par des tests nucléaires (Starfish…).

La réduction des dimensions mises en œuvre dans les procédés microélectroniques a depuis accentué la sensibilité des circuits. Ceux-ci sont désormais vulnérables au niveau avionique et même au niveau de la mer où l'exposition est cependant réduite par rapport à l'environnement spatial. En 1996, ces problèmes sont clairement identifiés comme le montrent les exemples cités par Normand [NORM-96].

Normand présente également quelques faits importants pour l'analyse des défaillances des circuits. On peut tout d'abord différencier les particules qui produisent des fautes : ions lourds, neutrons à haute énergie et particules α. En ce qui concerne les applications mises en oeuvre au niveau de la mer, les neutrons atmosphériques sont présentés comme la principale source de fautes. En plus de l'environnement on notera aussi l'influence de certains paramètres physiques des circuits sur l'occurrence et les conséquences des fautes.

Outre les applications critiques au niveau terrestre telles que l'automobile ou les applications médicales, il faut désormais prendre en considération les applications qui peuvent subir des attaques non plus naturelles et aléatoires mais intentionnelles. Il est en effet possible des générer des fautes dans un circuit en modifiant son environnement (température, tension d'alimentation) ou bien à l'aide d'un laser par exemple. Or le comportement erroné de circuits spécifiques comme les circuits de cryptage permet d'obtenir des informations que l'utilisateur souhaite cacher [BONE-98].

Les préoccupations liées à l'analyse des erreurs dans les circuits intégrés se sont donc accrues ces dernières années. Dans ce contexte il est nécessaire pour la compréhension de ce qui suit de présenter certaines notions importantes et de préciser la terminologie.

La charge Q_{col} est la charge collectée après un impact de particule ou après tout autre type de perturbation. La charge critique Q_{crit} est la charge collectée minimum pour créer une erreur comme définie dans la norme JESD89 [JEDE-01]. La charge collectée après un impact de particule dépend de la densité du matériau, de l'énergie nécessaire à la création de paire électron-trou (E_i = 3.6eV pour le silicium) mais aussi de la longueur de pénétration de la particule (L) et du LET_{th}. Le LET (*Linear Energy Transfer*) est l'énergie moyenne perdue (ΔEt) par unité de longueur de trace (Δx) normalisée à

la densité du matériau. Le seuil LET_{th} est l'énergie à partir de laquelle l'impact provoque une faute. Comme toute la charge générée par une particule incidente n'est pas collectée l'efficacité de la collection de charge (Q_s ou CCE pour *Charge Collection Efficiency*) est le rapport entre la charge collectée sur la charge générée. Enfin la notion de section efficace est très souvent utilisée pour caractériser la sensibilité d'un circuit. La section efficace σ est le rapport entre le nombre de fautes et le nombre de particules incidentes par cm².

On définit la notion de sûreté de fonctionnement (*dependability*) d'un système 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 oeuvre, de se comporter de façon nominale, de rendre un service conformément à une référence prédéterminée. La propriété de sûreté de fonctionnement est associée à plusieurs attributs, les principaux étant :

- fiabilité (*reliability*) : la continuité de service est assurée pendant une durée minimale,
- disponibilité (*availability*) : le pourcentage du temps pendant lequel le système est prêt à l'utilisation est supérieur à un certain seuil,
- innocuité (*safety*) : la probabilité d'occurrence de défaillances jugées critiques est inférieure à un certain seuil,
- confidentialité (*security*) : la probabilité que la confidentialité et l'intégrité des données traitées soient maintenues.

Le terme « fonctionnement » est parfois omis, on parle alors simplement de sûreté, mais il faut garder en mémoire qu'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, 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.

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 d'un nœud de celui-ci. Cet enchaînement est très souvent illustré par un schéma comme celui de la Figure 1-1. Une faute engendre une erreur lorsqu'elle est activée. Or une faute peut également rester latente si elle se trouve dans une partie non utilisée du circuit. On distingue deux grandes catégories d'erreur: les erreurs permanentes et les erreurs transitoires.

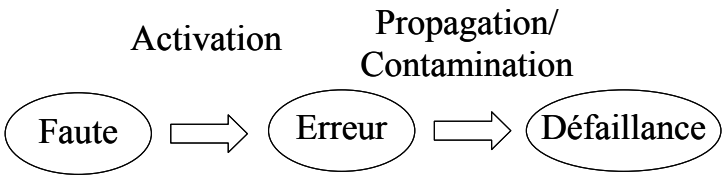

Figure 1-1 : Schéma Faute-Erreur-Défaillance

Une erreur permanente (*hard-error*) correspond à la destruction partielle du circuit. Ce type d'erreur peut être causé par une certaine catégorie d'événements singuliers (*Single Event Effects* - SEE) notamment suite à l'impact d'une particule ayant une très forte énergie. Un tel impact peut conduire à des phénomènes de *latchup* (*Single-Event Latchup* - SEL) ou de claquage de grille (*Single-Event Burnout* - SEB) et donc, potentiellement, à la destruction du circuit. Notre étude s'est portée uniquement sur les erreurs transitoires car la gestion des erreurs permanentes se situe davantage au niveau technologique ou au niveau de la mise sous boîtier pour éviter la destruction de transistors lors d'un impact. Nous ne traiterons également pas les phénomènes d'effets cumulatifs comme le déplacement atomique.

Une erreur transitoire (*soft-error*) est caractérisée par une modification de donnée réversible ou un état erroné temporaire. Elle n'engendre pas de destruction et le circuit fonctionne à nouveau normalement après disparition de cette erreur. Par exemple, une erreur transitoire dans un registre disparaît après réécriture de ce registre. Une erreur transitoire peut-être la conséquence soit de certains événements singuliers (impact d'une particule ayant une énergie relativement faible) soit de la perte de l'intégrité du signal. Les événements singuliers qu'il faut considérer pour ces erreurs sont notamment ceux qui font basculer un point mémoire (*Single Event Upset* - SEU) ou qui créent un pic de courant dans une partie de logique combinatoire (*Single Event Transient* - SET) ou dans un bloc analogique. Dans cette thèse nous nous focaliserons cependant sur les parties numériques des circuits.

La probabilité que l'impact d'une particule touche plusieurs nœuds d'un même circuit était auparavant négligeable. Or désormais les effets multiples doivent également être pris en compte. C'est le cas lorsqu'une faute transitoire affecte simultanément plusieurs bits d'un circuit (*Multiple Bit Upset* - MBU ou *Multiple Cell Upset* – MCU [JEDE-05]). De plus un pic de courant en logique combinatoire (SET) peut se propager et être mémorisé dans plusieurs éléments mémoires distincts ce qui correspond à une erreur transitoire multiple.

Pour mesurer de façon quantitative la vulnérabilité d'un système on calcule ou on mesure le nombre d'erreurs transitoires qui se produisent en un temps donné (*Soft Error Rate* – SER). L'unité de mesure est alors le nombre d'erreurs par heure (erreur.heure^{-1}). Une autre unité très utilisée est le FIT (*Failure In Time*) qui correspond au nombre de défaillances, ou d'erreurs suivant les cas, en 10^9 heures. Notre étude ne porte pas sur le taux d'erreurs calculé ou mesuré de façon statique. Nous verrons néanmoins qu'il peut être intéressant de coupler le calcul du SER à l'analyse de sûreté par injection de fautes.

Le principe de l'injection de fautes est la génération volontaire de fautes et peut être appliqué à plusieurs niveaux. Des fautes peuvent, par exemple, être injectées au niveau transistors, au niveau transferts de registres (*Register Transfer Level* - RTL), et jusque dans le code exécuté par le système. Dans tous les cas il faut définir un modèle de faute qui doit être le plus proche possible des phénomènes réels que nous venons de résumer, et qui seront davantage détaillés dans la section suivante.

1.2 Fautes transitoires

1.2.1 Causes

1.2.1.1 Environnement du système

Il y a dans l'environnement d'un circuit trois sources principales de radiations qui peuvent produire des fautes transitoires : les impuretés radioactives dans le circuit lui-même, les rayons cosmiques et les interactions Neutron/Boron 10 [BAUM-01].

- Impuretés radioactives

Une particule alpha (α) est une particule ionisante composée de deux neutrons et de deux protons. Une telle particule est émise quand le noyau d'un isotope instable présent dans le matériau décroît à un état de plus basse énergie.

Les particules alpha proviennent majoritairement des matériaux qui composent le boîtier (moule, matériaux composites, soudures). Les matériaux qui composent le circuit lui-même sont de faibles sources de particules alpha. Le Tableau 1-1, tiré de [BAUM-01], présente les émissivités de matériaux utilisés pour les circuits et leurs boîtiers. Ces particules α ont une énergie d'environ 3 à 10 MeV, ce qui correspond à une charge de plus de 100 fC. Pour comparaison, le nœud électrique dans un circuit intégré en technologie CMOS 90nm a une charge de 1 à 10fC [KARN-04].

Tableau 1-1 : Emissivité de particules α de certains matériaux

Matériau	Emissivité ($\alpha.h^{-1}.cm^{-2}$)
Wafer complètement processé	< 0,0009
Métal Cuivre épais	< 0,0014
Moule Composant	0,002 < - < 0,024
Boîtier Flip Chip	0,0009 < - < 0,002
Soudures à base de Plomb	0,002 < - < 7,2

- Rayons cosmiques

Les rayons cosmiques primaires sont les rayons cosmiques qui atteignent la Terre. Ce flux contient approximativement 1000 particules.$m^{-2}.s^{-1}$ (360 particules.$cm^{-2}.h^{-1}$), principalement des protons (92%) avec des énergies supérieures à 1 GeV. Il contient également des particules alpha (6%) et des ions lourds (2%).

Un ion lourd est le noyau d'un atome avec plus ou moins d'électrons que normal ou sans électron du tout. Les 2% d'ions lourds ci-dessus couvrent tout le spectre des éléments, le carbone et le fer sont

parmi les plus importants. Ils sont en partie rejetés par le champ magnétique terrestre et sont plus nombreux au niveau des pôles qu'au niveau de l'équateur.

A l'altitude de 50km, les rayons cosmiques primaires commencent à interagir avec l'atmosphère terrestre pour produire une cascade de particules secondaires qui elles aussi interagissent. Seulement 1% des rayons cosmiques primaires atteignent le niveau de la mer sans interagir. Ces réactions créent un large spectre de particules, même au niveau de la mer où le flux est de 10000 particules.m^{-2}.s^{-1} (3600 particules.cm^{-2}.h^{-1}). Il faut noter que les ions lourds sont une cause d'erreurs transitoires principalement pour les applications avioniques et peuvent être négligés au niveau de la mer.

Le Tableau 1-2 présente les flux de particules secondaires à deux altitudes caractéristiques : la surface de la Terre et l'altitude moyenne des vols commerciaux. Les valeurs sont tirées de [GASI-01].

Tableau 1-2 : Comparaison de flux ($cm^{-2}.s^{-1}$) de particules

Type de particule	Altitude = 12000 mètres	Niveau de la mer	Rapport
Neutrons	9	$6\ 10^{-3}$	$1,5\ 10^{3}$
Protons	$2\ 10^{-1}$	$2\ 10^{-4}$	10^{3}
Electrons	3	$4\ 10^{-3}$	$7,5\ 10^{2}$
Muons	10^{-1}	$2\ 10^{-2}$	5
Pions chargés	$5\ 10^{-3}$	10^{-5}	$5\ 10^{2}$

Les particules α issues des rayons cosmiques interagissent très rapidement avec les molécules dans l'atmosphère et donc aucune n'atteint la surface de la Terre. Les protons et les électrons n'interagissent pas avec le Silicium.

Pour les technologies actuelles les pions et les muons ont une influence négligeable sur le SER d'un système [GASI-01]. A cause de leur flux important et de leur stabilité, les neutrons à haute énergie (supérieure à 1MeV) sont la principale source de fautes transitoires dues aux rayons cosmiques, notamment au niveau de la mer.

❖ Interactions Neutron/Boron 10

La troisième source de particules ionisantes est due aux interactions entre des neutrons et le Boron 10. Le Boron 10 (^{10}B) est utilisé comme dopant de type P et dans la formation de couches diélectriques BPSG (Boro Phospho Silicate Glass). L'interaction de celui-ci avec un neutron thermique (E<15eV), provenant des rayons cosmiques secondaires, produit une particule alpha et un noyau de Lithium, tous deux capables de créer une faute transitoire.

❖ Comparaison des SER induits

Comme notre étude cible principalement les applications terrestres, il faut considérer les flux et les émissivités au niveau de la mer. D'autre part les neutrons et les particules alpha n'ont pas la même influence sur le taux de fautes.

Le Tableau 1-3 présente la comparaison entre le flux de neutrons et l'émissivité en particules alpha de boîtiers de circuit au niveau de la mer pour une cellule SRAM en technologie 0,18µm.

On note que le flux de neutrons est 20000 fois plus important que l'émissivité en particules alpha. Or lorsque l'on étudie les SER induit, on remarque que les proportions sont très différentes. Bien que la part des neutrons dans le SER induit soit prédominante, on ne peut négliger l'impact de particules α sur le taux d'erreurs.

L'interaction entre un neutron et du Boron 10 engendre une charge de 25 fC/µm pour le noyau de Lithium (dans 94% des cas) et une charge maximum de 16 fC/µm pour la particule alpha, sur une distance inférieure à 3µm. On obtient donc les mêmes ordres de grandeurs que pour une particule alpha provenant d'impuretés dans le matériel.

Tableau 1-3 : Comparaison neutrons/particules alpha

Particules	Flux/Emissivité (alt. 0m) ($cm^{-2}.s^{-1}$)	SER induit
Neutrons (rayons cosmiques)	$6\ 10^{-3}$	82%
α (impuretés)	$3\ 10^{-7}$	18%

Les données présentées dans le Tableau 1-3 doivent cependant être considérées pour une technologie précise (CMOS sur silicium massif en 0,18µm). Nous verrons dans la partie 1.2.3 que l'influence de chaque type de particule sur le SER total varie en fonction de la technologie.

L'environnement d'un circuit (boîtier inclus) est la « source originale » de fautes transitoires. C'est la plus étudiée et elle reste la plus importante en quantité. Mais malgré le nombre important d'études visant à définir avec précision les flux de particules sur Terre, il reste impossible de prédire exactement l'impact d'une particule sur un circuit.

L'autre source de fautes transitoires est la perte de l'intégrité du signal. Ces phénomènes sont croissants, notamment à cause de la réduction des dimensions.

1.2.1.2 Perte de l'intégrité du signal

L'intégrité du signal (*signal integrity*) désigne « la qualité des signaux engendrés dans un circuit et qui pourrait éventuellement être altérés par des interférences dues aux autres éléments du circuit ou du système » [ANGH-00].

Les problèmes d'intégrité du signal (couplages, bruits …) sont de plus en plus pris en compte lorsque l'on étudie les phénomènes de fautes transitoires des systèmes. Plus prévisibles que les impacts de particules, ils augmentent avec la réduction des dimensions.

- Diaphonie capacitive et inductive

La diaphonie parasite (*crosstalk*) correspond au couplage capacitif (électrostatique) et inductif (magnétique) entre une interconnexion « agresseur » et une interconnexion « victime ». La ligne active (agresseur) transporte un signal correspondant à un pic de tension avec un temps de montée *tr* et une pente *dv/dt*. Les couplages capacitif et inductif causent tous deux un pic de tension sur la ligne passive (victime). Les effets de couplage peuvent alors être décrits comme la somme des effets de couplage capacitif (inverse et direct) et des effets de couplage inductif (inverse et direct). Le couplage total est proportionnel à *tr* et à *dv/dt*.

Dans [CHEN-02], Chen utilise le modèle de faute appelé *Maximum Aggressor Fault* qui est décrit dans [BAI-01] et qui représente une abstraction à haut niveau des couplages capacitifs et inductifs. Bien que l'approche n'ait pas été appliquée aux interconnexions à l'intérieur d'un circuit, le phénomène reste équivalent : une transition sur une ou plusieurs interconnexions qui entraîne un pic de tension sur une interconnexion couplée.

A cause des interconnexions longues des *System-on-Chip* (SoC) et des circuits multicouches, les phénomènes de couplage deviennent de plus en plus importants.

❖ Pollution des alimentations

D'autres problèmes de perte de l'intégrité du signal sont liés à la perturbation des alimentations. Les plus courants sont :

- les bruits de commutation sur les lignes d'alimentation (*ground bounce* ou SSN pour *Simultaneous Switching Noise*),
- les chutes de tension dues à la distribution des alimentations (*IR-drop*),
- les phénomènes de couplage de substrat (principalement pour les circuits mixtes analogique/numérique).

Les buffers d'entrée/sortie étaient auparavant les parties sensibles pour ce type de bruit. Les bruits de commutation à l'intérieur du circuit deviennent désormais un problème majeur à cause de l'évolution des performances (fréquence d'horloge plus grande …) [CHAN-97].

❖ Variations de l'horloge

Les bruits se manifestent notamment au niveau des signaux d'horloge. Dans [GUI-02], un décalage d'horloge (*clock skew*) est défini comme une variation du temps d'arrivée des fronts d'horloge entre différents points d'un circuit. Une gigue d'horloge (*clock jitter*) est d'autre part définie comme une variation temporelle de la période de l'horloge en un point donné du circuit.

Les effets des décalages d'horloge sont accrus par la complexité des arbres d'horloges dans les circuits actuels. Les principales causes sont :

- les variations de l'horloge dues à la fabrication,
- les variations des interconnexions : la finesse des diélectriques entre les couches affecte la résistance et la capacité des interconnexions,
- les variations de température et de tension d'alimentation.

Les principales causes de *clock jitter* sont :

- la génération de l'horloge (PLL),
- les capacités de couplage,
- les variations de température et de tension d'alimentation.

Parmi les sources de *jitter* et de *skew* ci-dessus, il est clair que certaines peuvent entrer en jeu au cours de l'utilisation d'un circuit. Les couplages capacitifs ou plus évidemment les variations de température et de tension d'alimentation sont des phénomènes imprévisibles. Ils peuvent d'autre part évoluer en fonction de l'utilisation (application) du circuit. En conséquence on considère que gigues d'horloge et décalages d'horloge peuvent se produire de façon aléatoire dans le temps et l'espace.

Couplages, bruits de commutations et signaux d'horloge incorrects ne sont pas les seuls phénomènes liés à la perte de l'intégrité du signal. Par exemple les interférences électromagnétiques doivent également être prises en compte si l'on souhaite être exhaustif.

Cependant l'objectif de cette partie était simplement d'introduire la perte de l'intégrité du signal comme une source de fautes transitoires. Nous verrons dans la partie 1.2.3 quel est l'impact des avancées technologiques, notamment sur ces problèmes de perte de l'intégrité du signal. Mais revenons d'abord sur les effets liés à l'environnement pour étudier plus en détail les phénomènes physiques mis en jeu.

1.2.2 Effets

❖ Ionisation

Une particule chargée qui frappe un circuit perd son énergie en créant des paires électron-trou. Les ions lourds, les particules α et les protons à faible énergie, entre autres, sont des particules ionisantes. La collection de charge, conséquence de l'impact, s'effectue par conduction (*drift*) dans la région de déplétion, par diffusion et par *funneling* (Figure 1-2)

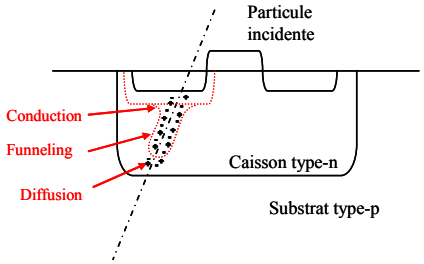

Figure 1-2 : Phénomène d'ionisation

Les charges créées dans la zone de déplétion sont séparées par le fort champ électrique qui s'y trouve, c'est la conduction. Le *funneling* correspond à une extension des lignes de champ électrique dans le substrat, au-delà de la zone de déplétion, et à la collection de charges dans cette région. Enfin au-delà de la zone de *funneling*, les charges sont collectées par diffusion.

Si la charge collectée Q_{col} est plus grande que la charge de la cellule mémoire alors la cellule mémoire bascule et la valeur logique est inversée [GASI-01][VARG-00]. Dans la logique combinatoire, le pic de courant induit génère un pic de tension. Ce pic de tension se propage alors dans la logique jusqu'à ce qu'il disparaisse (phénomène de masquage) ou jusqu'à ce qu'il atteigne une cellule mémoire. Dans ce cas, si le pic atteint le point mémoire au moment du front actif de l'horloge il est mémorisé.

❖ Recul des noyaux de silicium

La réaction induite par un impact de neutron est le recul des noyaux de silicium. Il se compose de deux phénomènes : la diffusion (principalement élastique) et l'absorption. D'une part, lors de la diffusion élastique (Figure 1-3), le neutron heurte un noyau de silicium, change de direction et de niveau d'énergie. Si le noyau de silicium gagne suffisamment d'énergie, il peut être éjecté de la matrice. Ce phénomène crée quelques paires électron-trou mais pas suffisamment pour engendrer une faute transitoire. D'autre part, le phénomène d'absorption (Figure 1-4) peut créer, dans certains cas particuliers, des particules chargées telles que des particules alpha, des protons [GASI-01] et des ions lourds. Ceux-ci peuvent alors créer des paires électron-trou et finalement une faute transitoire.

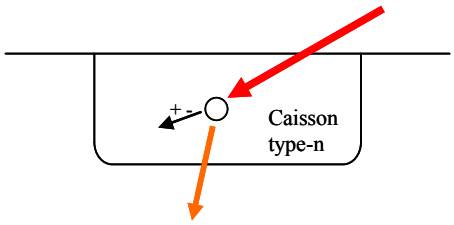

Figure 1-3 : Phénomène de diffusion

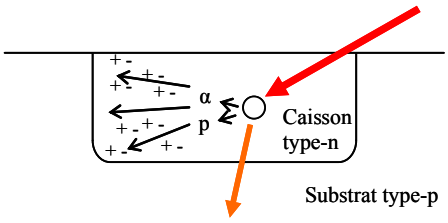

Figure 1-4 : Phénomène d'absorption

1.2.3 Impact des avancées technologiques

Les avancées technologiques tendent à augmenter la sensibilité des circuits aux fautes transitoires. Le SER par bit décroît d'une technologie à une autre mais comme le nombre de bits des circuits augmente, le SER des circuits augmente également.

D'un autre point de vue, certains procédés de conception ou certaines technologies spécifiques peuvent diminuer la sensibilité des circuits. Cette partie est consacrée aux conséquences des progrès technologiques sur le taux de fautes des circuits. Cependant prévoir les SERs pour les technologies à venir n'est pas chose facile. Certains résultats contradictoires sont proposés dans [KARN-04].

1.2.3.1 Technologie

Dans [HAZU-00], Hazucha avance que, selon les prévisions de *l'International Technology Roadmap for Semiconductors* (ITRS), on assiste à une décroissance quadratique de la charge de commutation, une croissance linéaire de la vitesse, une décroissance linéaire de la tension d'alimentation (triangles et ligne pleine Figure 1-5) mais également une croissance quadratique du nombre de transistors (ligne pointillée Figure 1-5). Le Tableau 1-4 présente quelques paramètres clés et leurs évolutions avec la réduction des dimensions (sources : ITRS roadmap 2002 update).

Le besoin de performances est l'une des principales causes de la réduction des dimensions. Des fréquences d'horloge pour des calculs rapides et des tensions d'alimentation plus faibles pour les applications mobiles sont, par exemple, ce qui peut définir les avancées technologiques.

Tableau 1-4 : Evolution des paramètres technologiques (ITRS)

	2004	2005	2007	2010	2016
DRAM 1/2 pitch (nm)	90	80	65	45	22
MPU longueur de grille masque (nm)	53	45	35	25	13
MPU longueur de grille physique (nm)	37	32	25	18	9
Vdd - hautes performances (V)	1	0.9	0.7	0.6	0.4
Densité de transistors - SRAM (10^9.cm^{-2})	393	504	827	1718	7208
Densité de transistors - logique (10^9.cm^{-2})	77.2	97.2	154.3	309	1235
Gbits par cm^2 (en production)	1.15	1.46	2.35	4.75	28.85

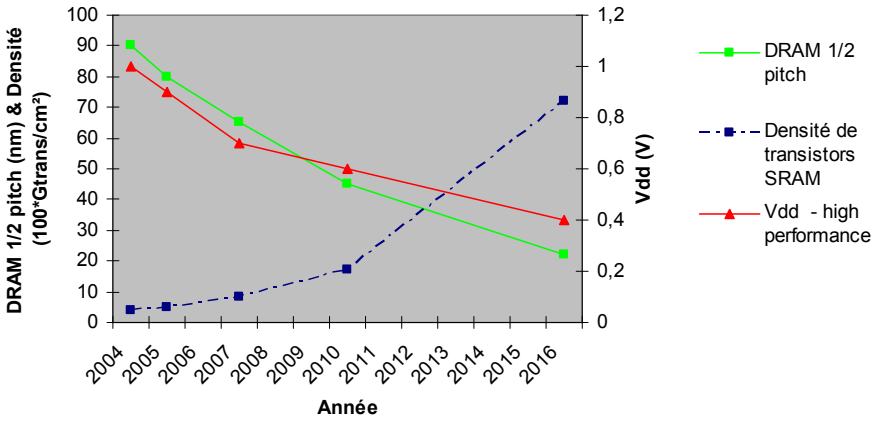

Figure 1-5 : Avancées technologiques à venir (ITRS roadmap)

L'impact de l'augmentation de la fréquence d'horloge sur les pertes d'intégrité du signal est clair. Une fréquence plus grande entraîne des variations de courant plus rapides (di/dt), ce qui augmente les bruits de commutation.

La réduction de la tension d'alimentation induit une diminution de la charge critique Q_{crit} et donc une sensibilité plus grande. Certains résultats présentés dans [KARN-04] montrent que, pour une technologie 90nm CMOS, une réduction de 10% de la tension d'alimentation entraîne une augmentation de 18% du taux de fautes dues aux neutrons.

Les nouvelles technologies sont également caractérisées par une efficacité de collection de charge Q_s réduite. L'impact des paramètres Q_{crit} et Q_s sur le SER d'un circuit semble difficile à définir précisément [SEIF-02]. Le modèle défini dans [HAZU-00] permet néanmoins d'évaluer l'impact de plusieurs paramètres sur le SER relatif aux neutrons :

$$SER \propto F \times A \times \exp(-\frac{Qcrit}{Qs})$$

Équation 1-1 : Modèle de SER

Le taux de fautes dépend du flux (F) de neutrons ayant une énergie supérieure à 1MeV (particules.cm^{-2}.s^{-1}), de la taille de la zone sensible (A) du circuit (cm^2), de Q_{crit} et de Q_s.

Comme nous l'avons vu précédemment les neutrons sont la plus grande cause de SEU pour les technologies « anciennes ». Or pour les technologies avancées (charge stockée comprise entre 10fC et 40fC) l'importance des particules α augmente exponentiellement. En dessous de 10fC par nœud le

nombre de particules alphas capables de générer un SEU sature, et le taux d'erreur devient dépendant de la tension d'alimentation et décroit exponentiellement avec l'augmentation de celle-ci [HARE-01].

Dans le même esprit 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 forte charge critique (petites cellules mémoires, amplificateurs) les particules α sont prédominantes dans le calcul du SER.

En ce qui concerne la fission du ^{10}B, il est possible de remplacer ce type d'élément par un autre matériau comme par exemple le ^{11}B. Le Tableau 1-5, tiré de [GASI-01], montre l'importance des particules alphas, des neutrons et du ^{10}B, dans le SER pour deux technologies différentes (avec et sans ^{10}B).

Tableau 1-5 : Importance des sources sur le SER

	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'évolution des technologies se caractérise par la réduction des dimensions des technologies CMOS sur silicium massif (*bulk*) mais aussi par l'utilisation de nouvelles technologies comme la technologie silicium sur isolant (SoI pour *Silicium on Insulator*). Les avantages (réduction de la jonction parasite, immunité au *latch-up*) et les inconvénients (sensibilité aux décharges électrostatiques) intrinsèques de la technologie SoI ne sont pas discutés ici mais il est intéressant d'évaluer les caractéristiques de cette technologie face aux perturbations.

En comparaison à la technologie *bulk*, certaines caractéristiques du SoI diminuent sa sensibilité aux *soft-errors* :

- grâce à l'oxyde enterré la longueur de collection est plus courte. Ceci réduit l'efficacité de collection de charge Q_s et donc la sensibilité du circuit. Comme la zone de collection est plus petite que pour la technologie *bulk* le SER est réduit,

alors que d'autres au contraire l'augmentent :

- la capacité réduite et la tension d'alimentation (a priori) plus faible d'un dispositif SoI diminuent sa charge critique Q_{crit}. De plus le transistor bipolaire parasite amplifie le pic de courant causé par un impact de particule. Des procédés de conception spécifiques peuvent limiter ce phénomène mais pas le supprimer.

Des résultats expérimentaux ont montré que la technologie SoI est moins sensible aux SEU en 0,25µm [HARE-01] [GASI-02]. Mais pour la technologie suivante (0,18µm) l'avantage du SoI (volume de collection plus petit) est contrarié par l'effet de substrat flottant et les taux d'erreurs sont équivalents pour les technologies *bulk* et SoI.

La technologie SoI voit également ses dimensions diminuer. Les résultats publiés dans [HADD-05] montrent que les SER d'un même microprocesseur PowerPC en technologie SoI 0,18µm et 0,13µm sont comparables. L'influence de la tension d'alimentation du cœur de microprocesseur PowerPC sur son taux de fautes a également été étudié et les résultats publiés dans [IROM-05]. Ceux-ci ne montrent pas de différences entre les tensions d'alimentation 1,3V et 1,6V en technologie 0,18µm. En revanche le SER du processeur en technologie 0,13µm avec des alimentations de 1,3V et 1,1V révèle des différences importantes et une augmentation de la sensibilité avec la diminution de la tension d'alimentation.

Pour les technologies CMOS-bulk et CMOS-SoI plus les dimensions sont petites plus la taille de la zone sensible pour chaque bit diminue. Ce qui entraîne une réduction du SER/bit.

D'autre part il semble que les effets multiples auront une importance prépondérante pour les technologies futures. Bien que seulement 2% de tous les événements dans les cellules mémoires d'un cache SRAM en technologie 90nm impactent plusieurs bits [KARN-04], l'ITRS *roadmap* prévoit une augmentation régulière du pourcentage de MBU dans le SER.

1.2.3.2 Conception et encapsulation

Sans considérer l'utilisation de cellules durcies dont les caractéristiques limitent la sensibilité aux SEU, la façon dont un circuit est conçu influence le SER au niveau système. Cette partie liste de façon non exhaustive quelques points où la conception a un impact sur le SER.

Si l'on considère les éléments mémoires inclus dans la logique d'un circuit, un exemple de l'influence de la conception sur le taux de fautes est présenté dans [SEIF-02]. Deux processeurs avec des cellules mémoires différentes sont étudiés : un processeur est conçu avec des verrous (sensibles au niveau d'horloge), l'autre est conçu avec des bascules (sensibles aux fronts d'horloge). Le fait que la sortie d'un verrou est un noeud flottant rend le processeur correspondant plus sensible.

Nous avons vu l'importance de la tension d'alimentation sur le SER d'un circuit. Or le schéma d'alimentation a également un effet sur la sensibilité d'un circuit face aux problèmes de perte de l'intégrité du signal.

Des schémas d'alimentation particuliers sont en général mis en œuvre pour des applications « faible consommation », l'idée étant de jouer sur la tension de seuil V_{th} pour réduire la consommation. Trois schémas d'alimentation différents sont étudiés dans [ABBA-04] :

- double tension : tension haute pour les chemins critiques, tension basse ailleurs,
- double tension de seuil : tension de seuil haute pour les chemins critiques, tension de seuil basse ailleurs,
- tensions de seuil multiples : tension de seuil basse pour les transistors actifs, tension de seuil haute pour les transistors en mode repos.

Des simulations avec des niveaux de bruit importants ont été effectuées avec ces trois schémas d'alimentation, plus des schémas « classiques » avec V_{th} haute et V_{th} basse. Les résultats montrent que les différents circuits n'ont pas la même sensibilité. Les schémas double tension de seuil et tension de seuil haute sont ceux pour lesquels le nombre de fautes est la plus faible.

En ce qui concerne l'influence de l'évolution technologique sur le SER d'un circuit nous pouvons encore citer le choix des mémoires et du boîtier.

Aucun SEU n'a été observé dans les cellules mémoires non volatiles EPROM ni dans l'espace, ni sous flux de particules [MCNU-99]. Les mémoires volatiles (SRAM, DRAM) sont au contraire sensibles aux radiations. Les mémoires non volatiles Flash sont également sensibles aux SEUs mais l'ordre de grandeur du nombre de défaillances en 10^9 heures (FIT) est 3,5 à 5 fois inférieur à celui des mémoires SRAM [FOGL-04]. L'augmentation de l'intégration et la complexification des systèmes augmentent la taille des mémoires embarquées sur les puces. Ceci entraîne logiquement une augmentation du taux d'erreurs des SoCs.

Enfin les boîtiers puce-retournée (*flip-chip*) peuvent remplacer les boîtiers avec câblage par fil (*wire-bonding*) afin de réduire la taille et le poids du boîtier. Or la technologie puce-retournée augmente la sensibilité du système et en particulier le poids des particules alphas dans le SER du système [SEIF-02].

Ces quelques exemples montrent que les phénomènes de erreurs transitoires ont une importance croissante pour les nouvelles technologies et que les concepteurs ont un rôle important sur le SER du système final et que leur travail doit prendre en compte ces aspects.

1.2.4 Modélisation

Un modèle de faute est une abstraction de l'état incorrect d'un circuit. Les impacts de particules et la perte de l'intégrité du signal sont les causes de fautes transitoires non intentionnelles. Ces fautes peuvent être également générées volontairement, par exemple pour découvrir des informations secrètes. Dans les deux cas l'injection de fautes permet d'évaluer les conséquences de ces fautes. Il est cependant nécessaire d'avoir un ou plusieurs modèles de fautes qui représentent le plus fidèlement possible les phénomènes physiques. De plus les modèles de fautes doivent être applicables dans le cadre de l'approche choisie pour l'injection.

Les modèles de fautes de base sont les intermédiaires entre les phénomènes physiques présentés ci-dessus et les modèles de fautes à plus haut niveau nécessaire dans le cadre d'injection de fautes.

1.2.4.1 Modèles de faute de base

- Single Event Transient

Un *Single Event Transient* (SET) se produit lorsqu'une particule ionisante frappe un noeud sensible de la logique combinatoire. Le pic de tension induit peut être mémorisé dans un point mémoire s'il l'atteint au moment de la fenêtre de capture (Figure 1-6).

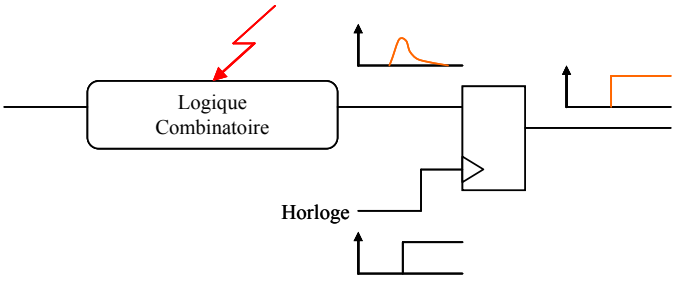

Figure 1-6 : Single Event Transient (SET)

Un SET n'engendre pas toujours une erreur. En effet trois types de masquage peuvent empêcher la propagation et la mémorisation du pic de tension :

- le masquage électrique lorsque le pic de tension est atténué par les portes logiques (cependant, comme la plupart des portes sont non linéaires, les circuits avec des gains en tension conséquents vont voir une amplification des pics des tensions [SUBH-05]),
- le masquage logique quand le pic de tension se propage sur une entrée de porte qui n'influence pas à cet instant la valeur de la sortie,
- le masquage temporel lorsque le pic se propage mais n'est mémorisé par aucune cellule mémoire parce qu'il n'est pas présent au moment de la fenêtre de capture [SHIV-02].

Le masquage est une des raisons pour lesquelles le SER des parties combinatoires est moins élevé que celui des parties séquentielles.

❖ Single Event Upset

Un *Single Event Upset* (SEU) se produit lorsqu'une particule ionisante frappe un noeud sensible d'une cellule mémoire et entraîne le basculement de la valeur logique mémorisée. Physiquement, si la charge collectée Q_{col} est plus importante que la charge critique Q_{crit} alors il y a inversion de l'état logique, de '0' vers '1' ou de '1' vers '0'. La Figure 1-7 illustre l'exemple d'une cellule SRAM à 6 transistors .Si une particule fait basculer un noeud la perturbation se propage à travers l'inverseur et fait basculer l'autre nœud. Comme le second nœud commande le premier, les deux nœuds basculent. Le seul moyen de supprimer cette erreur est la réécriture de la cellule mémoire.

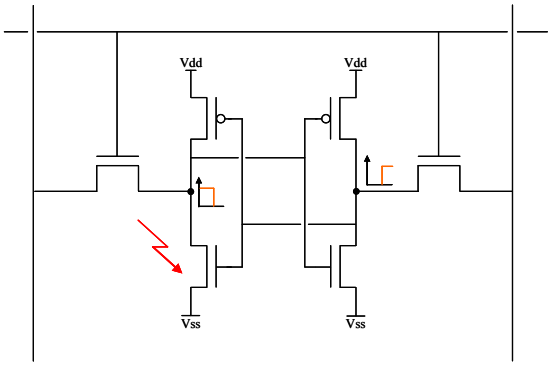

Figure 1-7 : Single Event Upset (SEU)

Les éléments sensibles aux SEU sont tous les éléments mémoires volatiles d'un système : bascules (*Flip-Flops*) et verrous (*latches*) individuels, registres et RAM (SRAM, DRAM…).

❖ Multiple Cell Upset et Multiple Bit Upset

Les *Multiple Cell Upset* (MCU) et les *Multiple Bit Upset* (MBU) sont la conséquence des phénomènes de diffusion et de collection de charge dans plusieurs cellules mémoires. Ils sont causés par un unique impact de particule. Les cellules appartiennent au même mot dans le cas des MBUs. La faute ne peut alors pas être corrigée par un code correcteur simple (un bit). Dans le cas des MCUs les bits n'appartiennent pas au même mot mémoire [JEDE-07].

Le nombre de cellules affectées dépend de l'énergie de la particule, de son angle d'incidence et de la façon dont est implantée la cellule mémoire. En conséquence, prévoir l'occurrence ou la probabilité d'occurrence des MBUs avant placement/routage est impossible.

❖ Faute due au bruit

Les modèles SET et SEU ont été largement utilisés pour caractériser l'incidence des impacts de particules. Cependant, à notre connaissance, la modélisation des phénomènes liés à la perte de l'intégrité du signal n'a pas encore été étudiée. Bien que des modèles de fautes pour les phénomènes de couplage aient été définis dans [BAI-01] ils ne concernent que les interconnexions entre les blocs d'un système.

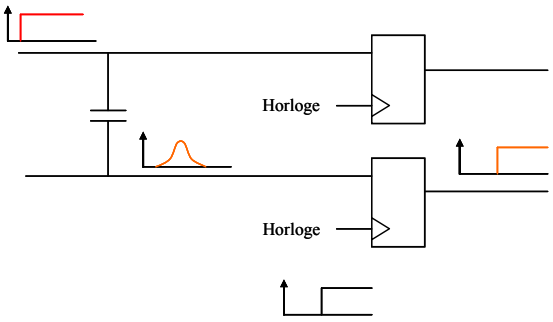

Figure 1-8 : Faute due au couplage

Le couplage parasite et le bruit de commutation produisent des pics de courant dans les parties de logique combinatoire et dans les interconnexions (Figure 1-8). Ils peuvent donc être modélisés avec le modèle de faute SET. En effet un pic créé sur une ligne victime peut se propager à travers la logique combinatoire et être mémorisé par une cellule mémoire, exactement comme un pic de courant engendré par un impact de particule, i.e. un SET.

Les effets de la variation de l'horloge sont plus proches du modèle SEU. Considérons un décalage d'horloge (*clock skew*) pour une cellule mémoire comme représenté sur la Figure 1-9. δ_1 est le temps de calcul de la logique combinatoire. Si le décalage de l'horloge est δ_2 et si l'entrée change à $T+\delta_1$ ($\delta_2 > \delta_1$), alors la nouvelle valeur d'entrée sera mémorisée à $T+\delta_2$. Dans ce cas le basculement de la valeur mémorisée se produit entre deux fronts d'horloge « idéaux ». Cette faute peut être considérée équivalente à une faute transitoire de type SEU.

Nous considérons qu'un *jitter* d'horloge aura approximativement les mêmes effets. Les effets de la perte de l'intégrité du signal peuvent alors être modélisés avec les mêmes modèles SEU et SET. Une des différences est la fréquence d'apparition de ces fautes.

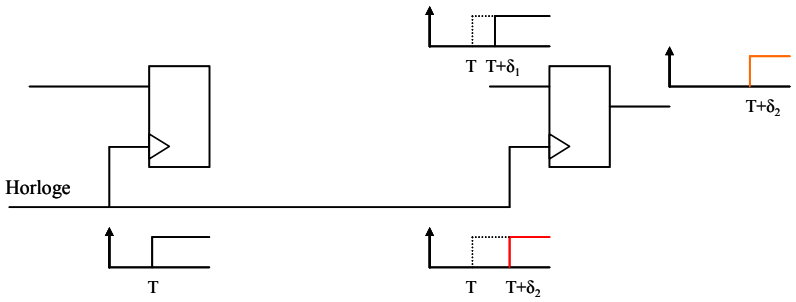

Figure 1-9 : Faute due aux variations de l'horloge

❖ Faute par attaque volontaire

Alors que la sécurité est une propriété de plus en plus recherchée pour de nombreuses applications (cartes de crédit, télécommunications…) il devient indispensable de considérer les attaques intentionnelles comme une source de fautes. Par exemple, les attaques par fautes sont utilisées pour modifier le comportement d'un système et ainsi récupérer des données sensées rester secrètes. Cette technique est appelée *Differential Fault Analysis* (DFA).

Des fautes transitoires peuvent être injectées en modifiant l'environnement du circuit de différentes façons. Variation de la tension d'alimentation, variation de l'horloge ou du reset, injections optiques (flashes, laser) font partie des techniques d'injection possibles.

En fonction de la précision de la technique employée, ces attaques visent à générer des fautes transitoires équivalentes à celles issues, soit des impacts de particule ionisante, soit de la perte de l'intégrité du signal. En conséquence les mêmes modèles de faute sont applicables. Une différence importante concerne toutefois les valeurs des multiplicités spatiales et temporelles.

1.2.4.2 Modèles de fautes aux niveaux RTL et portes

Ce paragraphe définit les modèles de fautes au niveau transfert de registres et au niveau portes qui dérivent des modèles de fautes présentés ci-dessus. On remarquera qu'il existe d'autres modèles de fautes utilisables au niveau RTL que ceux présentés ici.

Un SET devrait en théorie être appliqué au niveau transistors et non pas au niveau comportemental ou au niveau portes. Les caractéristiques temporelles du circuit implanté sont indispensables pour savoir si un SET sera mémorisé ou non. De plus, dans le cadre du prototypage, une limitation est qu'un prototype ne fonctionne pas à la fréquence nominale du circuit et surtout que les portes logiques ne sont pas les mêmes. En conséquence, un SET pourrait être mémorisé lors du fonctionnement du circuit alors qu'un SET injecté dans le prototype ne l'est pas, et inversement.

Cependant un SET peut être vu comme un nœud forcé à un certain niveau logique, pendant une courte durée et sans égard au niveau logique normal (sans faute) de ce nœud. Dans ce cas un SET peut-être assimilé à une faute de collage transitoire (*transient stuck-at*).

Une faute de collage (*Stuck-At - SA*) est un modèle de faute qui correspond usuellement à un défaut physique : un nœud « collé » à la tension 0 (collage à '0') ou à la tension V_{dd} (collage à '1') à cause d'une mauvaise fabrication.

Ce modèle de faute peut-être étendu et être utilisé à haut niveau d'abstraction. Un nœud peut-être forcé à une certaine valeur logique (0 ou 1), durant un temps long voir infini (collage permanent) ou durant une période très courte (collage transitoire). L'application du modèle de collage est cependant difficile à haut niveau car l'implantation exacte du circuit reste inconnue. La synthèse et les optimisations qui lui sont associées peuvent en effet supprimer certains nœuds présents dans la description à haut niveau. Cela dépend par exemple du fait de conserver ou non la hiérarchie lors de la synthèse.

Le modèle de collage peut donc être appliqué à haut niveau aux entrées (et aux sorties) du système, aux entrées ou aux sorties des cellules mémoires et aux signaux entre différents blocs si la hiérarchie est conservée lors de la synthèse. Nous avons vu que le même modèle de faute peut être utilisé pour les SET et pour les effets de diaphonie parasite.

A haut niveau d'abstraction SEU, MBU et MCU mettent en œuvre le même modèle de fautes : l'inversion de la valeur mémorisée dans une ou plusieurs cellules mémoires aussi appelé *bit-flip*. La différentiation se fait par le nombre (multiplicité spatiale) et la localisation des cibles. On considère l'inversion singulière (*single bit-flip*) pour le SEU et l'inversion multiple (*multiple bit-flip*) pour MBU et MCU.

Un SET peut également se propager dans la logique combinatoire et être mémorisé dans une ou plusieurs cellules mémoires et ainsi engendrer une erreur transitoire. Dans ce cas on fait abstraction des éventuels masquages et on utilise les modèles d'inversion singulière ou multiple.

Enfin nous avons vu que les fautes induites par les variations de l'horloge peuvent être assimilées à des SEU et donc modélisées par des inversions singulières ou multiples.

La Figure 1-10 présente les différentes sources de fautes et les modèles de fautes pouvant les représenter. En toute rigueur un modèle de faute à haut niveau peut également être considéré comme un modèle d'erreur puisqu'il correspond obligatoirement à une modification de l'état du circuit. Dans la suite de ce document nous conservons le terme modèle de faute.

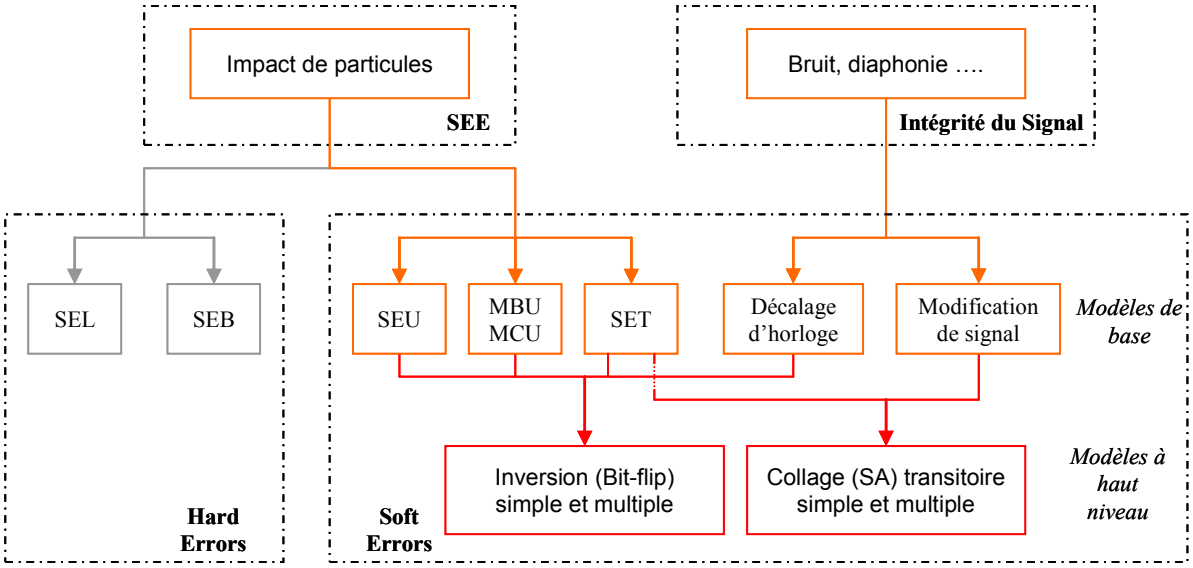

Figure 1-10 : Modèles de fautes

1.3 Conclusion

Nous avons vu dans ce chapitre que les problèmes liés à la sûreté de fonctionnement sont croissants et que les sources d'erreurs augmentent avec la réduction des dimensions des procédés microélectronique. Les fautes transitoires doivent désormais être pris en compte au niveau du sol et plus seulement pour les applications spatiales. Le caractère multiple des fautes ne peut plus être ignoré comme par le passé. Enfin certains phénomènes liés à l'intégrité du signal sont maintenant potentiellement à l'origine de fautes transitoires à cause des dimensions de plus en plus petites.

Ceci nous montre que l'analyse de sûreté est plus que jamais nécessaire pour s'assurer de la robustesse ou au contraire de la vulnérabilité des circuits intégrés. Dans cette optique et à la vue des phénomènes divers à considérer nous avons présenté les modèles de fautes qui nous paraissent les plus probants dans le cadre de l'analyse de sûreté par injection de fautes.

Dans le chapitre suivant nous présentons l'état de l'art des techniques d'analyse basées sur l'injection de fautes. Cet état de l'art nous permettra ensuite de définir un environnement répondant le plus possible aux problématiques actuelles dans le domaine de l'analyse de sûreté par injection de fautes.

2 Etat de l'Art sur les techniques d'injection de fautes

Depuis la fin des années 80, l'injection de fautes s'est avérée être une technique efficace pour l'analyse de sûreté. De nombreuses approches d'injection s'appliquent lorsque le circuit à analyser est déjà fabriqué ; au niveau des broches [ARLA-90][MADE-94][MART-99], par corruption de la mémoire [MICH-94], par exposition à un flux d'ions lourds [GUNN-89], par perturbations de l'alimentation [KARL-91], par laser [SAMP-97], ou au niveau logiciel [KANA-92][CARR-98][BENS-98].

D'autres techniques d'injection permettent une analyse plus tôt dans le flot de conception, typiquement au niveau transferts de registres ou au niveau portes. Les deux types d'approche possibles sont la simulation et l'émulation matérielle ou prototypage. Nous allons présenter dans ce chapitre une vue d'ensemble des approches employées.

Afin de pouvoir réaliser l'injection de fautes il peut être, selon la méthode, nécessaire d'instrumenter la description originale. L'instrumentation se fait soit en insérant un ou plusieurs blocs (saboteurs) entre les blocs du circuit original, soit en modifiant un ou plusieurs blocs de ce circuit (mutants). 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 noeuds pour une instrumentation niveau portes.

2.1 Approches utilisant des circuits physiques

- Injection au niveau des broches d'entrée/sortie

Des nombreux outils ont été développés dans les années 90 pour injecter des fautes via les broches d'un circuit. Le premier doit être MESSALINE [ARLA-90], développé au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS). Cette technique a été améliorée et les injecteurs de l'outil RIFLE [MADE-94] permettent également de détecter si les fautes injectées ont produit des erreurs. On peut également citer AFIT [MART-99] pour ses performances (fréquence d'injection).

L'avantage de ces outils est qu'ils ne sont pas intrusifs et qu'ils ne modifient pas le fonctionnement temporel des circuits. Un problème majeur de l'injection physique par les broches est la définition de modèles représentatifs des fautes internes d'un circuit. Il est proposé dans [ARLA-90] de réaliser des simulations afin de déterminer les séquences de fautes à injecter. La gestion des bruits induits par la ou les sondes, rapportés dans [ARLA-90] et [MART-99], limite également l'utilisation de cette technique.

- Corruption de la mémoire

L'approche présentée dans [MICH-94] utilise un injecteur de fautes commercial : l'appareil DEFI. Celui-ci permet d'injecter des fautes (permanentes ou transitoires – simples ou multiples) dans la mémoire programme d'un processeur.

Cette solution a été mise en œuvre pour valider des mécanismes de détection d'erreurs. Elle est trop limitée pour l'analyse de circuits actuels car les fautes ne peuvent être injectées que dans la mémoire programme.

❖ Perturbation de l'alimentation

Une approche d'injection de fautes via la perturbation de l'alimentation a été proposée dans [KARL-91]. Elle est basée sur l'ajout d'un transistor MOS entre l'alimentation et le port V_{cc} du circuit à analyser. Elle permet d'injecter des fautes en modifiant la tension de grille de ce transistor.

Cette approche se rapproche des phénomènes de chutes de tension (*IR drop*) qui peuvent provoquer des fautes transitoires comme présenté dans la section 1.2.1.2. Son inconvénient est que les effets de la perturbation sont difficilement estimables et qu'ils varient selon le circuit étudié. Par exemple la perturbation peut se propager d'abord aux sorties du circuit avant d'affecter un nœud interne. Les caractéristiques des perturbations doivent donc être soigneusement définies.

❖ Interférence Electromagnétique

L'injection de fautes par interférences électromagnétiques (*Electro Magnetic Interferences* - EMI) dérive de l'environnement industriel. Un injecteur produit des rafales représentatives de charges inductives lors de commutations à l'aide de relais ou de disjoncteurs [KARL-95]. Les fautes sont injectées sur les signaux d'entrée/sortie et peuvent se propager dans le circuit.

Le nombre de fautes injectées dépend de la modulation du signal radio-fréquence et de la valeur du champ magnétique comme présenté dans [VARG-05]. Vargas et. al propose une nouvelle approche EMI basée sur l'utilisation des outils de certification de compatibilité électromagnétique.

❖ Laser

Le laser permet d'injecter des fautes représentatives des *soft-errors* 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. Certains paramètres comme la fréquence d'opération ou la précision peuvent cependant être des facteurs limitant.

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 (anciennement IXL) ont des caractéristiques similaires : précision spatiale de 0,1µm, taille du spot de 1µm mais une fréquence de fonctionnement beaucoup plus élevée (80MHz). Deux limitations rapportées dans [DUZE-00] sont les pulsations de fuites qui peuvent entraîner 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.

❖ Flux de particules

L'analyse sous radiations permet d'obtenir le taux de fautes accéléré d'un système (*Accelerated SER* - ASER). C'est évidemment l'approche qui permet d'injecter les fautes les plus représentatives

des fautes dues aux radiations lors du fonctionnement du circuit. Les particules mises en œuvres (ions lourds, protons, neutrons) [KOGA-96] sont les mêmes que dans l'environnement et leurs énergies sont contrôlables.

C'est sans doute l'approche la plus utilisée par les industriels pour caractériser leurs circuits. Cette technique souffre néanmoins de contrôlabilités spatiale et temporelle réduites. En outre l'accès aux installations est un paramètre important du fait de leur nombre limité et de leur utilisation intensive. Une campagne d'injection demande également une phase de préparation (*test set-up*) importante.

❖ Méthodes logicielles

Du fait du coût en temps et en argent des techniques précédentes, de nombreuses approches d'injection basées sur le logiciel (*SoftWare Implemented Fault Injection* - SWIFI) ont été développées.

Une des premières approches logicielles, FIAT [SEGA-88], propose de lier le programme principal à des programmes agissant comme des chevaux de Troie. FERRARI [KANA-92] offre une plus grande contrôlabilité en modifiant les instructions au moment de leur exécution. Xception [CARR-98] utilise les exceptions et les mécanismes de débug des processeurs modernes. Il est également possible d'injecter des *bit-flips* dans la mémoire d'un processeur en exécutant des séquences de code comme présenté dans [VELA-00].

❖ Comparaison des approches existantes

Le Tableau 2-1, extension de celui tiré de [KARL-95], résume les caractéristiques des approches ci-dessus.

Les approches d'injection de fautes par perturbation de l'alimentation et interférences électromagnétiques présentent une reproductibilité et une contrôlabilité faibles. L'injection au niveau des broches coûte peu en temps et en argent mais souffre d'une accessibilité faible. Ces techniques sont peu employées.

Les approches par laser, par radiation et logicielle sont les plus mises en œuvre.

L'injection par laser offre une bonne contrôlabilité mais la taille du spot laser reste un défi pour les technologies actuelles et futures. De plus une campagne d'injection par laser demande une mise en œuvre relativement lourde et un équipement qui reste coûteux.

Les approches logicielles (SWIFI) sont les plus simples et les moins onéreuses. Le problème essentiel est que le procédé d'injection interfère avec l'application et donc en modifie les caractéristiques (temps d'exécution…). De plus ces approches ne ciblent que les systèmes à base de processeur(s) et l'accessibilité est faible (impossibilité d'injecter des fautes dans le pipeline d'un processeur par exemple).

Soumettre un circuit à un flux de particules a l'avantage de mettre en jeu exactement les mêmes phénomènes que ceux auxquels ce circuit peut être confronté lors de son utilisation. Comme le laser cela permet une grande accessibilité puisque chaque point mémoire du circuit analysé peut être ciblé.

Cependant une faible contrôlabilité, une approche statique et des coûts d'expérimentation importants sont les points faibles de l'approche.

Tableau 2-1 : Comparaison des techniques d'injection de fautes dites « physiques »

	Contrôlabilité spatiale	Contrôlabilité Temporelle	Flexibilité[1]	Reproductibilité[2]
Niveau broches	Haute	Haute	Moyenne	Haute
Corruption Mémoire	Moyenne	Moyenne	Basse	Haute
Radiation	Basse	Aucune	Basse	Moyenne
Perturbation alimentation	Basse	Basse	Haute	Basse
EMI	Basse	Basse	Haute	Basse
Laser	Basse à Moyenne	Haute	Haute	Moyenne à Haute
Logiciel	Moyenne	Moyenne	Basse	Haute

	Accessibilité[3]	Durée	Coûts
Niveau broches	Basse	Moyenne	Basse
Corruption Mémoire	Moyenne	Moyenne	Basse
Radiation	Haute	Moyenne	Haute
Perturbation alimentation	Moyenne	Basse	Basse
EMI	Basse	Basse	Moyenne
Laser	Haute	Basse	Moyenne
Logiciel	Moyenne	Moyenne	Basse

[1] Flexibilité = Possibilité et facilité de changer de circuit cible dans l'environnement d'analyse.

[2] Reproductibilité = Capacité à reproduire les résultats de façon statistique pour une configuration donnée.

[3] Accessibilité = Capacité à atteindre une cible d'injection spécifique.

Mais au final la plus grande limitation de toutes ces approches est qu'elles nécessitent un circuit « physique » pour pouvoir effectuer les injections. Or il est essentiel d'effectuer l'analyse au plus tôt dans le flot de conception afin de limiter au maximum d'éventuelles modifications.

2.2 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.

Des travaux ont récemment été menés afin d'effectuer l'analyse de sûreté au niveau système (description SystemC ou utilisation d'un modèle comportemental d'architecture (ISA)).

La simulation à haut niveau de description (*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. Le contre poids est que les simulations niveau portes sont approximativement dix fois plus lentes que les simulations RTL voire davantage.

Les simulateurs analogiques tels SPICE 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 RTL ou niveau portes car leur mise en œuvre est la plus réaliste.

2.2.1 Simulation niveau système

Des techniques et des outils ont été développés afin de permettre la modélisation au niveau système et de prendre en compte simultanément les parties logicielles et les parties matérielles d'un système. Plusieurs approches et langages (SystemC, Esterel, POLIS…) peuvent être utilisés dans cette perspective.

D'un point de vue analyse de sûreté, des études ont récemment porté sur l'injection de fautes au niveau système dans un environnement de conception conjointe matériel/logiciel (co-design) [LAJO-00] [BOLC-01] [ROTH-05]. L'objectif est de réaliser l'analyse très tôt dans le flot de conception et éventuellement d'ajouter des mécanismes de protection contre les erreurs transitoires en minimisant le temps de reprise.

L'approche niveau système cible particulièrement les *System on Chip* (SoC). Les avantages d'une telle approche sont moindres si l'on s'intéresse à un bloc pris indépendamment. Surtout l'inconvénient d'une approche à aussi haut niveau est le manque de précision des modèles de faute utilisables. Par exemple, le modèle de faute choisi dans [BOLC-01] est le « *single functional failure* » qui correspond au dysfonctionnement d'un bloc à la suite d'une ou plusieurs fautes physiques. Un tel modèle est trop éloigné des modèles de base comme le SEU.

2.2.2 Simulation niveau instructions

Un ISS (*Instruction Set Simulator*) est un programme qui simule un processeur en interprétant ses instructions (une instruction à la fois). Cela permet à l'utilisateur d'observer l'état interne de la machine ciblée comme par exemple le contenu des registres utilisateur.

Au niveau instructions deux approches sont possibles : la modélisation ISS (*Instruction Set Simulation*) et la modélisation ISA (*Instruction Set Architecture*). L'approche ISA considère les transferts de registres. Elle est donc plus détaillée que l'approche ISS qui considère les transactions.

Dans [DELO-96] Delong et al. introduisent l'injection de fautes au niveau ISA où un modèle comportemental (VHDL) d'un processeur exécute du code machine. Au niveau ISA, le logiciel est décrit en assembleur ciblant spécifiquement une architecture matérielle du système (e.g. processeur, mémoire…). L'architecture interne du processeur n'est cependant pas prise en compte. Un modèle ISA exécute tout code destiné au processeur qu'il modélise et donne les mêmes résultats que ce processeur au cycle près, mais en général sans lien direct avec la microarchitecture du processeur.

Dans [DELO-96] le principe général est donc la simulation haut niveau. L'injection est effectuée grâce aux capacités du langage VHDL. La méthode est basée sur les « *Bus Resolution Functions* » (BRF). Une BRF est associée à un type de signal : lorsque deux sources différentes tentent d'actualiser un signal de ce type, la BRF associée affecte la bonne valeur au signal. La corruption de la nouvelle

valeur est possible grâce à la possibilité de communiquer avec la BRF. L'ajout d'un nouveau type et d'alias est nécessaire. Deux *process* doivent être rajoutés à chaque module matériel pour réaliser les injections.

Les défauts de cette technique sont, comme présenté dans [DELO-96], des temps de simulation trop importants et le fait que les modèles de fautes niveau portes (collage à 1 ou collage à 0) sont appliqués au niveau comportemental. En outre cette technique est difficilement applicable aux architectures qui n'utilisent pas des bus 3 états.

2.2.3 Simulation RTL

L'injection de fautes lors de la simulation du modèle RTL (VHDL ou verilog) du circuit à analyser peut se faire de deux façons ; soit en utilisant les commandes du simulateur pour forcer certains signaux internes, soit en utilisant une description modifiée du circuit permettant l'injection de fautes.

- Exploitation des commandes du simulateur

Lors de la simulation RTL de la description du circuit à analyser il est possible d'utiliser les commandes du simulateur ou les routines d'un langage spécifique (TCL par exemple) afin d'injecter une ou plusieurs fautes.

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 simulation se déroule normalement jusqu'à ce que le cycle d'injection soit atteint et que tous les *process* soient arrêtés au niveau d'une instruction *wait*. Pour l'injection d'une faute dans un signal (au sens VHDL), celui-ci est déconnecté de sa source et forcé à une nouvelle valeur jusqu'à la fin de la durée de la faute. Si la faute est transitoire le signal est reconnecté à sa source lorsque la durée de la faute est écoulée. Pour les variables qui conservent leurs valeurs lorsqu'un *wait* est atteint, la méthode est identique. Pour les variables dites atemporelles qui ne conservent pas leurs valeurs, un *breakpoint* est utilisé pour stopper la simulation et injecter une faute dans ces variables. La manipulation de signaux cible l'injection de fautes de collage permanente et transitoire dans une description structurelle alors que la manipulation de variables vise l'injection de *bit-flips* au niveau comportemental.

L'université de Turin a également développé un outil d'injection basé sur les commandes d'un simulateur commercial. Cet outil exploite les mécanismes de *debug* du simulateur, en l'occurrence ModelSim, et l'interface proposée par le langage TCL (*Tool Command Language*) [CORN-00].

Une liste de fautes est générée par analyse de la description comportementale (VHDL RTL) en fonction du modèle de fautes sélectionné. Le « simulateur de fautes » se compose d'un ensemble de routines qui interagissent avec le simulateur. La commande utilisée est le *breakpoint* ; simple pour les variables et double pour les signaux.

L'approche ci-dessus a été étendue pour l'injection de *bit-flips* dans les éléments mémoires [BERR-02]. Les commandes *checkpoint* et *restore* sont alors utilisées pour sauvegarder et charger les états de simulation. A noter aussi que le nombre de fautes à injecter est réduit à l'aide d'un prétraitement (*preprocessing*). Pour un exemple d'application spatiale le nombre de fautes à injecter est ramené à 13% du nombre total de fautes.

Une approche identique, basée sur le langage Perl, le langage TCL et la commande *force* de Modelsim, est utilisée par l'Agence Spatiale Européenne comme présenté dans [GUTI-04].

- Simulation avec description instrumentée

Le laboratoire LAAS a développé une approche à base d'instrumentation, MEFISTO-L, parallèlement à une approche utilisant les commandes du simulateur [BOUE-98].

L'instrumentation se traduit 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. Les mutants ne sont pas supportés par MEFISTO-L. La description originale est d'abord analysée pour générer les modèles de fautes, les cibles potentielles et les signaux à observer. L'outil se base aussi sur un simulateur commercial pour la compilation et la simulation de la description. Des efforts importants ont été réalisés pour la mise en place des expériences et sur l'analyse de mécanismes de protection présents dans les circuits ciblés.

De la même façon, Gracia et al. proposent dans [GRAC-01] trois approches basées sur l'utilisation des commandes du simulateur (voir partie précédente), l'utilisation de saboteurs, et l'utilisation de mutants.

Un grand nombre de saboteurs est considéré, du saboteur « série simple » (modification d'un signal entre deux blocs) jusqu'au saboteur « n bits bidirectionnel ». Chaque saboteur inclut une entrée de contrôle d'injection, et potentiellement une entrée lecture/écriture pour spécifier le sens des données. Les signaux ajoutés sont commandés à partir du simulateur.

L'implémentation de mutants peut se faire de manière statique et dynamique. De façon statique le mécanisme de configuration du VHDL permet de compiler une configuration spécifique qui reste la même durant toute la simulation. De façon dynamique, l'utilisation de blocs gardés est présentée comme une technique pour effectuer des instanciations dynamiques. Si une expression gardée est vraie alors l'architecture originale (non instrumentée) est chargée, sinon l'architecture instrumentée est chargée. Au cycle d'injection l'architecture mutée est chargée et ceci pendant toute le durée de la faute. L'architecture originale est ensuite rechargée. Un mutant correspond à un seul modèle de faute.

Cette approche permet d'injecter des fautes permanentes et des fautes transitoires. Les mutants permettent logiquement l'utilisation de modèles de fautes plus complexes mais les temps de simulation augmentent alors considérablement.

La génération de mutants permettant des injections avec des modèles de faute plus complexes a également été étudiée au TIMA [LEVE-03b]. Les mutants présentés permettent notamment l'injection

de transitions erronées dans une machine à états finis ou un organigramme de contrôle. Le ou les mutants générés sont ensuite simulés lors des campagnes d'injection.

2.2.4 Simulation niveau portes

Parallèlement aux approches à haut niveau, des techniques d'injection ont été développées pour des descriptions au niveau portes (après synthèse). Parmi celles-ci nous pouvons citer FAST, VERIFY et Roban.

En 1996 Cha et al. ont proposé un outil appelé FAST pour *Fault Simulator for Transients* [CHA-96]. Il se compose en fait de deux simulateurs distincts. Le premier simule le circuit jusqu'au moment où la faute injectée est capturée par une cellule mémoire. La propagation des signaux et les caractéristiques temporelles des portes ont donc une certaine importance lors de cette première phase. Le second simulateur (extension de l'outil PROOFS) est utilisé à partir de ce point afin d'observer les manifestations de la faute sur les sorties du circuit. L'intérêt de l'approche à deux simulateurs est d'accélérer la simulation après que la faute ait été mémorisée puisqu'à partir de ce point il n'est plus nécessaire d'avoir une grande précision temporelle.

Afin d'obtenir le modèle de faute, au niveau portes, le plus proche possible d'un SEU, des simulations SPICE ont été menées. Elles ont permis d'obtenir des données précises comme la durée nécessaire de la faute, les délais des portes, et plus généralement le comportement des bascules (*Flip-Flops*) et des verrous (*latches*).

L'outil VERIFY (*VHDL-based Evaluation of Reliability by Injecting Faults efficientlY*) est basé sur une extension du langage VHDL et permet l'injection de fautes au niveau portes mais également au niveau RTL.

Comme présenté dans [SIEH-97] un nouveau type de signal interne est ajouté aux types VHDL existants. Il est composé de deux champs supplémentaires : le temps moyen entre chaque faute et la durée moyenne de chaque faute. Chaque faute est associée à un composant et reste donc transparente pour les autres composants. Elle est activée via l'interface du simulateur.

Cette approche nécessite un compilateur dédié et un simulateur dédié afin de gérer l'extension du VHDL. Les deux ont été développés au sein de l'outil VERIFY. Le compilateur extrait les signaux d'injection et les lie à l'exécutable pour la simulation. La description niveau portes est alors obtenue après synthèse de la description RTL. L'approche supporte deux modèles de faute : le collage et le *bit-flip*.

Roban est un outil qui a été développé par iRoC Technologies et qui permet l'injection de fautes dans les blocs combinatoires. Son objectif est d'évaluer la probabilité d'occurrence d'erreurs transitoires dans les différents registres du circuit. Il a été intégré avec les simulateurs commercialisés tels ModelSim et NCSim, ce qui offre un simulateur et un injecteur de fautes via une unique interface.

Un environnement d'injection basé sur Roban et sur un modèle de faute transitoire (SET) a été développé et présenté dans [ALEX-04] . L'environnement de test (génération de stimuli, capture des

sorties …) est implémenté au niveau comportemental pour être indépendant du circuit analysé. Cet environnement permet d'injecter des fautes simples ou multiples dans tous les nœuds d'un circuit. Seule l'injection de fautes lors d'un état stable a été considérée afin de réduire le temps de simulation.

La forme exacte du pic de tension est fonction du type de particule considéré, du matériel et du lieu d'impact supposé. Cependant si la durée T durant laquelle le pic de tension initial est supérieur à la tension de seuil de la porte est connue, alors le pic de tension est rectangulaire et sa durée est égale à T. Deux phases de simulation du circuit sont nécessaires ; une première pour référence et une seconde pour l'injection de fautes. Les valeurs des registres internes sont comparées pour détecter d'éventuelles erreurs.

2.2.5 Simulation niveau transistors

Le niveau transistors (*switch level* ou *transistor level*) est le niveau intermédiaire entre le niveau portes et le niveau physique. Les descriptions au niveau transistors sont donc plus détaillées que celles au niveau portes et les simulations permettent d'observer certains phénomènes comme le partage de charge. L'outil développé à l'université de Téhéran nommé INJECT [ZARA-03] permet l'injection de fautes à plusieurs niveaux d'abstractions et plus particulièrement au niveau transistors. L'approche utilise des descriptions en verilog.

La description originale est mutée afin de permettre l'injection de fautes puis simulée avec ModelSim. L'avantage d'INJECT est sa capacité à injecter des fautes à différents niveaux d'abstraction. L'outil partage cependant le principal défaut des autres approches utilisant des simulations : un temps CPU élevé.

2.3 Approches à base d'émulation

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 apparaît avec les FPGAs à antifusibles de Actel puis elle est généralisée.

Un FPGA est un circuit intégré constitué d'éléments logiques programmables, d'éléments mémoires et d'interconnexions programmables. Les plus utilisés sont basés sur des cellules de configuration SRAM. Ces derniers sont fabriqués avec les technologies CMOS les plus avancées et sont re-configurables à volonté. Leurs deux principaux inconvénients sont probablement qu'ils doivent être re-configurés après chaque coupure de l'alimentation (mémoire SRAM volatile) et qu'ils sont sensibles aux radiations.

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 à une approche logicielle). L'émulation est également plus

rapide que la simulation, notamment pour des circuits analysés avec un grand nombre de vecteurs d'entrée.

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…) (Figure 2-1). 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*).

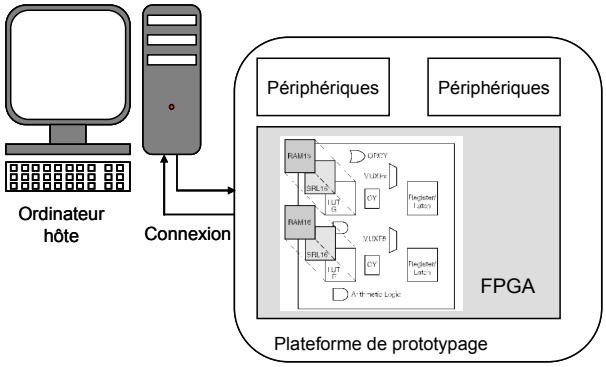

Figure 2-1 : Environnement de prototypage type

L'émulation matérielle a d'abord été mise en œuvre dans le cadre de la validation de vecteurs de tests. Cela correspond à vérifier que ces vecteurs détectent un maximum de défauts de fabrication, i.e. qu'ils présentent la meilleure couverture de fautes possible. L'objectif était alors de proposer une alternative aux approches ATPG (*Automatic Test Pattern Generation*) logicielles et ces approches préfigurent les approches d'analyse de sûreté par injection de fautes basées sur l'émulation. Parmi les techniques développées 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].

D'un point de vue analyse de sûreté, l'émulation représente un gain de temps important par rapport à la simulation. Autre avantage : la reconfiguration dynamique est aussi une caractéristique qui peut être exploitée pour l'injection de fautes. L'utilisation du prototypage s'est accélérée avec les améliorations des circuits programmables récents (vitesse, taille de la logique programmable…).

Deux approches se distinguent pour l'injection de fautes dans un environnement FPGA : l'instrumentation du circuit prototypé et la manipulation du fichier de configuration du FPGA.

2.3.1 Injection de fautes par reconfiguration

Pour les FPGAs basés sur de la SRAM, le fichier de configuration contient les données de configuration et 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 de

façon sérielle depuis l'ordinateur hôte sur le FPGA. Ces données peuvent ensuite être relues à partir du FPGA (*readback*), ainsi que le contenu de toutes les cellules mémoires (*capture*).

Une fois qu'un FPGA a été configuré il existe deux manières de le re-configurer : 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 le 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 dynamique se fait par modification directe des données de configuration.

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 récupérées 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 mécanismes de reset (*Global Set/Reset*) de ces points mémoires sont alors utilisés pour en inverser la valeur.

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].

L'université de Séville et l'Agence Spatiale Européenne ont conjointement développé le système FT-UNSHADES [TOMB-04]. Celui-ci met en œuvre deux copies du circuit à analyser ; une de référence et une pour l'injection. La comparaison des états des deux copies permet de détecter l'activation ou non de la faute injectée. L'observation de l'état des bascules est réalisée à l'aide des mécanismes de *readback* et de *capture* disponibles sur les FPGA Xilinx de la famille Virtex. Cette technique permet l'injection de *bit-flips*.

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

Pour réduire les temps de reconfiguration induits par la reconfiguration partielle, il est proposé dans [KAFK-06] d'utiliser un processeur embarqué pour effectuer les reconfigurations. Les données nécessaires à la sélection des cibles d'injection (LUTs, multiplexeurs…) sont récupérées grâce à l'outil de placement/routage. Toutes les configurations nécessaires sont préparées avant la campagne et téléchargées au début de celle-ci. Durant la campagne il n'y a aucun transfert de données entre l'hôte et le circuit programmable. Les résultats (classification des fautes) sont générés par un comparateur matériel et renvoyés par le FPGA à la fin de la campagne.

Les expériences menées démontrent non seulement que l'approche est faisable mais également qu'elle est approximativement 30 fois plus rapide qu'une approche à base de simulation. Elle peut cependant nécessiter beaucoup d'espace mémoire si un grand nombre de configurations prédéfinies doit être stocké. De plus, comme pour l'approche précédente, la mise en œuvre des deux copies du circuit analysé (référence et avec fautes) réduit de moitié la taille possible pour celui-ci.

2.3.2 Injection de fautes avec instrumentation

Le principe de l'instrumentation pour l'émulation est le même que pour la simulation. Elle correspond à la modification de la description initiale afin de permettre l'injection de fautes. Le circuit instrumenté est ensuite synthétisé puis émulé.

La pertinence du prototypage pour l'analyse de sûreté a été mise en avant au TIMA en 1999 [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.

Une approche d'injection de fautes à base de FPGA a été développée par le Politecnico de Torino comme présenté dans [CIVE-01a]. Elle permet l'injection de *bit-flips* dans les éléments mémoires d'une version instrumentée du circuit à analyser. Dans [CIVE-01b], l'outil FIFA (*Fault Injection by means of FPGA*) est amélioré et met en oeuvre des techniques dédiées aux microprocesseurs.

L'outil est composé de deux modules purement logiciels (exécutés sur l'ordinateur hôte) et d'un module mixte logiciel/matériel. Les deux modules logiciels sont le module qui génère la liste de fautes à partir d'une *netlist* du circuit et des vecteurs d'entrée et l'analyseur de résultats. Le module mixte inclut l'outil d'instrumentation, l'outil de contrôle des injections et une interface matérielle émulée sur le FPGA qui exécute les commandes du contrôleur.

L'instrumentation correspond à l'ajout d'un registre et de logique combinatoire pour effectuer l'injection (Figure 2-2). Cette logique combinatoire ne comprend que deux portes logiques et donc induit un délai faible. Le registre ajouté (*mask-register*) peut se charger de façon sérielle ou parallèle.

Il permet d'injecter les fautes et d'observer les effets de ces injections. Lorsque les données d'injection sont chargées, ce registre se comporte tel un registre à décalage. Ce qui signifie que les éléments mémoires sont chargés en série et que pour charger le registre complet il faut un cycle d'horloge par élément.

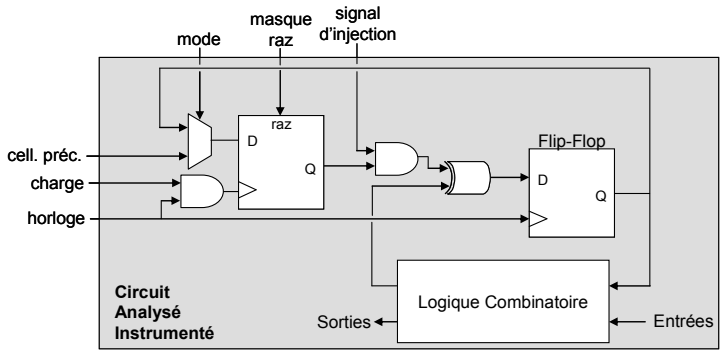

Figure 2-2 : Instrumentation à base de registre masque

Lorsque le circuit analysé est un microprocesseur, l'évolution proposée dans [CIVE-01b] et [CIVE-03] permet l'injection de fautes dans la mémoire de celui-ci. L'interface matérielle est alors capable de le contrôler à l'aide d'un jeu de commandes (*step*, *run*, *evaluate*, *inject* et *tick*). Des saboteurs sont ajoutés au microprocesseur pour isoler et contrôler les blocs qui le constituent (bus, mémoire cache, mémoire…). Un bus dédié est également ajouté pour accéder à la logique combinatoire ajoutée. Au cycle d'injection l'élément mémoire ciblé est lu et sa valeur est envoyée au module logiciel *Fault Injection Manager*. Celui-ci calcule la valeur erronée correspondant à l'injection de faute et cette valeur est réécrite en mémoire.

Le groupe Microélectronique de l'Université de Madrid a récemment développé un environnement FPGA pouvant mettre en oeuvre de façon distincte deux approches possibles : une première avec connexion des éléments mémoires en chaîne de *scan* appelée *state-scan* et une seconde avec un registre-masque appelé *mask-scan* [PORTe-04].

La chaîne de *scan* (ou registre à décalage) est utilisée pour télécharger l'état des éléments mémoires du circuit au moment de l'injection. L'instrumentation n'ajoute que très peu de logique puisque un seul multiplexeur par bascule est nécessaire. Les états erronés, chargés en série, doivent être définis avant le début de la campagne et sont stockés en mémoire RAM.

Le masque est appliqué aux éléments mémoires du circuit pour inverser le contenu de ceux qui sont ciblés. Cette dernière technique est reprise de [CIVE-01a].

Pour les deux approches l'architecture globale du système est la même : circuit à analyser, contrôleur d'injection et analyseur de résultats sont émulés par la logique programmable et des bancs de mémoires RAM sont utilisés pour stocker des données. La partie logicielle correspond à la génération du circuit instrumenté, à la génération de la liste de fautes, à l'accès aux bancs mémoires et à la configuration du FPGA. L'instrumentation se fait au niveau *netlist*, après synthèse.

Un effort important a été mené pour intégrer au maximum la partie injection en matériel. L'approche à base de chaîne de *scan* est la plus gourmande en temps mais permet l'analyse de résultats (classification des fautes).

Les travaux ci-dessus ont été poursuivis et trois approches d'injections sont proposés dans [LOPE-05] : l'approche *state-scan*, l'approche *mask-scan* et une approche dite multiplexée temporellement (*time-multiplexed*). La nouveauté est que la plupart des tâches (contrôle des injections, injection, classification des fautes) est exécutée par des blocs matériels afin de limiter les transferts entre le prototype et l'ordinateur hôte qui sont très coûteux en temps. L'environnement ainsi développé est qualifié d'autonome.

L'approche avec masque implémente la même instrumentation et les mêmes caractéristiques (registre à décalage…) que celle proposé dans [CIVE-01a] et décrite ci-dessus. La seule différence se situe au niveau du contrôleur d'injection qui ici est implémenté en matériel.

Le but de la technique *time-multiplexed* (Figure 2-3) est de réduire les temps de campagne en arrêtant chaque expérience lorsque les effets de la faute injectée disparaissent. Pour cela les émulations avec et sans injection de faute sont exécutées au même moment et leurs sorties sont comparées directement par le matériel. En conséquence la logique supplémentaire nécessaire est assez importante. La logique combinatoire originale n'est pas dupliquée mais chaque bascule (FF) est répliquée trois fois : une FF pour le masque, une FF pour l'injection et une FF pour le chargement et la restauration de l'état. C'est une extension de l'approche *mask-scan* ; les sorties ne sont plus observables mais un signal d'erreur est généré lorsqu'une erreur est détectée.

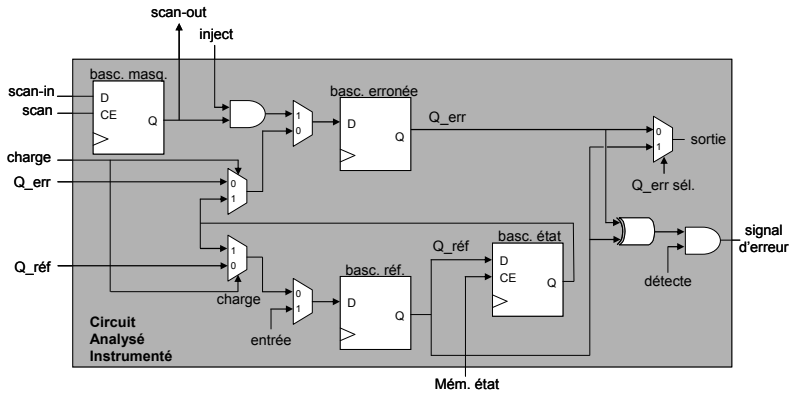

Figure 2-3 : Instrumentation « time-multiplexed »

Dans [GARC-06] l'approche d'injection de fautes autonome a été étendue aux circuits avec mémoire embarquée, typiquement les microprocesseurs. L'idée est de dupliquer la mémoire (mémoire de référence et mémoire erronée) et de vérifier les accès en mémoire plutôt que le contenu des mémoires.

Une description d'un circuit au niveau transistors (voir paragraphe 2.2.5) est plus détaillée qu'au niveau portes mais n'est pas implantable sur FPGA. Comme présenté dans [EJLA-04] il est cependant possible d'utiliser les portes logiques d'un circuit programmable pour modéliser, sans perte

d'information, une description niveau transistors. Les transistors et les nœuds sont modélisés par des *pseudo-transistors* et de *pseudo-nœuds* composés de portes logiques et de cellules mémoires.

Modéliser un circuit complet avec des *pseudo-transistors* et des *pseudo-nœuds* nécessiterait cependant trop de ressources. C'est la raison pour laquelle uniquement la ou les parties supposées sensibles sont modélisées de la sorte. Le reste du circuit est émulé directement à partir de la description niveau portes ce qui donne une émulation dite mixte. Des expériences ont par exemple été menées sur un processeur RISC 32 bits avec uniquement l'unité arithmétique et logique (ALU) décrite au niveau transistors.

Les fautes sont injectées sur la grille de chaque transistor à l'aide d'un multiplexeur qui valide ou non l'injection. Une chaîne de *scan* peut aussi être configurée pour activer les fautes et observer les signaux erronés. Le modèle de fautes utilisé est un modèle correspondant au niveau d'abstraction : le collage de transistor.

2.4 Bilan

La simulation d'un circuit pour l'analyse de sûreté peut se faire à différents niveaux au cours du développement : au niveau système, au niveau instructions, au niveau HDL (comportemental ou structurel) ou après synthèse. Les deux points importants sont que plus on descend dans les niveaux d'abstraction plus les modèles de fautes seront proches des phénomènes physiques mais plus les campagnes d'injection seront longues.

La simulation au niveau instructions permet l'injection de fautes via la corruption de la valeur de certains signaux. Les temps de simulation sont cependant déjà importants et les modèles de fautes ne sont pas assez représentatifs.

L'injection de fautes durant la simulation de descriptions HDL peut se faire suivant deux approches : l'utilisation des commandes du simulateur ou l'instrumentation du circuit analysé. Les approches proposées dans [JENN-94] et [CORN-00] mettent en œuvre les capacités de debug incluses dans les simulateurs. Elles sont en conséquence dépendantes du simulateur utilisé et potentiellement des langages qu'il supporte. L'instrumentation offre des approches indépendantes du simulateur et avec des modèles de fautes plus raffinés (transition erronée par exemple). La contre-partie est la nécessité d'intervenir sur la description originale avant simulation. L'instrumentation peut être automatisée mais dépend alors du langage utilisé et de la façon dont le circuit est écrit.

Pour l'analyse d'un circuit décrit au niveau portes les modèles de fautes sont plus précis et la structure exacte du circuit est connue. En effet il est possible que certains éléments de la description haut niveau soient supprimés lors de la synthèse (optimisation, codage des états…), ce qui peut rendre une analyse à haut niveau caduque. Au niveau portes FAST et ROBAN donnent de bons résultats avec des modèles de fautes précisément définis. FAST et VERIFY dépendent du simulateur contrairement à ROBAN qui peut-être intégré à plusieurs simulateurs commerciaux. Le principal inconvénient de la simulation niveau portes est son coût en temps CPU.

Des techniques ont été développées pour accélérer les campagnes d'injection par simulation. Il est par exemple proposé dans [BERR-02] de réduire la liste de fautes en analysant soit uniquement la structure du circuit analysé soit la structure du circuit et l'application qui lui est destinée. Mais de manière générale la simulation reste très gourmande en temps, principalement au niveau portes.

L'utilisation du prototypage pour l'analyse de sûreté est favorisée par l'augmentation des performances des circuits programmable et permet de réaliser des campagnes d'injection plus rapides. On notera que la taille et la fréquence de travail des FPGA augmentent mais également qu'ils intègrent parfois des cœurs de processeurs (ex. PowerPC).

Le prototypage a d'abord été mis en œuvre pour la validation des vecteurs de test pour la détection des défauts de fabrication. Les approches développées n'ont été utilisées que pour émuler des fautes de collage mais ont sans doute servi de base lors du développement d'approches pour l'analyse de sûreté. Les principales différences sont la prise en considération de l'application dans le cas de l'analyse de sûreté et les modèles de fautes employés.

Les mécanismes de reconfiguration dispensent d'instrumenter le circuit à analyser. Ils limitent donc les temps de préparation des campagnes d'injection et ignorent les contraintes d'écriture des descriptions (qui doivent néanmoins rester synthétisables). Mais la mise en œuvre et les protocoles liés à ces mécanismes (mécanismes de lecture de la configuration, de reconfiguration…) dépendent du FPGA utilisé, ce qui limite la portabilité des approches développées. De plus les avantages apportés dépendent beaucoup des caractéristiques du circuit programmable (vitesse de lecture de la configuration, vitesse de reconfiguration, contrôle des signaux de set et reset…) et de la connectique entre l'ordinateur hôte et le FPGA [LEVE-03a].

Utiliser une version instrumentée du circuit à analyser permet de rester indépendant du type de FPGA. Pour une instrumentation au niveau HDL il suffit que la description instrumentée soit synthétisable. Pour une instrumentation niveau portes il suffit que la description HDL originale ait été synthétisée en ciblant la technologie du FPGA utilisé. Il est également nécessaire que le ou les blocs matériels utilisés pour l'analyse soient synthétisables. Pour certaines approches ces blocs matériels se limitent à une interface entre le circuit à analyser et le lien physique avec l'ordinateur hôte [CIVE-01a]. Il est cependant possible d'effecteur un certain nombre de tâches nécessaires à l'analyse de sûreté (contrôle des injections, analyse des sorties…) directement par ces blocs matériels. C'est l'approche proposée dans [LOPE-05]. Ceci permet d'une part d'effectuer ces tâches plus rapidement et d'autre part de limiter les transferts de données entre l'ordinateur et le FPGA. La vitesse de transfert est un paramètre important des approches FPGA et peut devenir un facteur limitant en fonction du type de connexion et de la quantité d'informations à échanger.

Contrairement à la simulation, l'émulation nécessite de prendre en compte les ressources matérielles disponibles. Il est théoriquement possible de simuler tout système indépendamment de sa taille. La fréquence de travail du processeur influe sur la durée des simulations mais ne les empêche pas de s'exécuter. La quantité de mémoire disponible peut bloquer la simulation si elle est trop faible

mais les ordinateurs actuels disposent en général de suffisamment de ressources. La quantité de logique programmable et le nombre d'entrées/sorties du FPGA peuvent quant à eux être des facteurs limitant si le système à prototyper est trop grand. Bien que les ressources disponibles dans les FPGA les plus récents soient importantes, ceci peut poser un problème pour les approches basées sur l'émulation de deux copies (référence et avec faute) du circuit à analyser. Et cela est particulièrement le cas lorsque le circuit à analyser est un véritable SoC (processeur, co-processeur, périphériques …).

2.5 Conclusion

Le choix du prototypage est basé sur la possibilité d'effectuer l'analyse tôt dans le flot de conception et sur l'accélération conséquente apportée par rapport à la simulation. De plus le prototype peut éventuellement être mis en œuvre dans un environnement (périphériques, mémoires) proche de son environnement fonctionnel final. L'émulation permet alors d'analyser le comportement global (matériel et logiciel) du système en présence de fautes. Les approches actuelles ont cependant certaines limitations que nous allons chercher à pallier avec les propositions présentées dans les chapitres suivants.

3 Analyse à base de prototypage avec processeur embarqué

L'analyse de sûreté a pour objectif d'analyser le comportement d'un circuit lorsqu'une faute, singulière ou multiple, intentionnelle ou non, se produit dans ce celui-ci. Deux grands types d'analyse sont possibles. La plus couramment effectuée est la classification. Celle-ci consiste à définir des classes de fautes en fonction de leurs conséquences sur le comportement du système. Par exemple une faute qui n'est pas activée est classifiée «latente» (*latent*). Si elle est activée et entraîne un dysfonctionnement du circuit elle est classifiée « défaillance ». Nous verrons qu'il est également possible d'analyser plus finement le comportement erroné d'un système en étudiant la propagation d'une faute après injection. Un système qui possède un niveau de sûreté requis est dit robuste, sinon on le considère comme vulnérable.

Pour analyser finement un système il faut définir un ensemble de fautes (cibles, multiplicité, cycles d'injection…) à injecter. Une expérience correspond à l'exécution d'une application type et à l'injection d'un sous-ensemble précis de fautes pendant le déroulement de cette application. L'ensemble des expériences représente une campagne d'injection.

Comme indiqué dans le chapitre précédent, l'analyse de sûreté est un domaine étudié depuis quelques années maintenant et nous disposons aujourd'hui d'un certain recul sur les méthodes mises en œuvre et sur ce qu'il est possible d'améliorer. Ce chapitre 3 porte tout d'abord sur ce qui nous a motivé à considérer un nouvel environnement d'analyse, nous y expliquons certains choix et présentons le flot d'analyse correspondant. Une fois cette base établie nous présentons plus en détail l'environnement et ce qu'il apporte d'un point de vue qualitatif.

3.1 Motivations et flot d'analyse

Comme nous l'avons vu dans le chapitre précédent les premières approches d'analyse de sûreté étaient basées sur la simulation du circuit à analyser. Le laboratoire TIMA ne fait pas exception et une approche à base de simulation RTL avec instrumentation a été développée et mise en œuvre. La Figure 3-1 présente succinctement le flot d'analyse alors suivi comme cela est présenté dans [HADJ-05].

Cependant le prototypage matériel s'est révélé être une technique pour une analyse de sûreté rapide et assez légère à mettre en œuvre. C'est pour cela qu'il a été décidé de développer un environnement et un flot d'analyse à base de prototypage, en parallèle de celui basé sur la simulation RTL.

Notre objectif est de développer un environnement le plus portable possible et donc le moins dépendant du prototype utilisé. Si l'on se limite aux FPGAs les approches basées sur la reconfiguration dynamique ont montré leur intérêt mais les procédés mis en œuvre (modes de configuration, de relecture de la configuration…) sont trop spécifiques au prototype. De plus nous ne souhaitons pas nous limiter aux FPGAs qui incluent des mécanismes de reconfiguration dynamique.

L'instrumentation du circuit à analyser permet au contraire de rester davantage indépendant de la plateforme de prototypage et même du flot de conception puisque le circuit instrumenté est

simplement considéré comme un circuit à implanter en logique programmable. Indépendamment du niveau d'instrumentation, niveau RTL ou niveau portes, le circuit instrumenté reste une entrée dans le flot d'analyse.

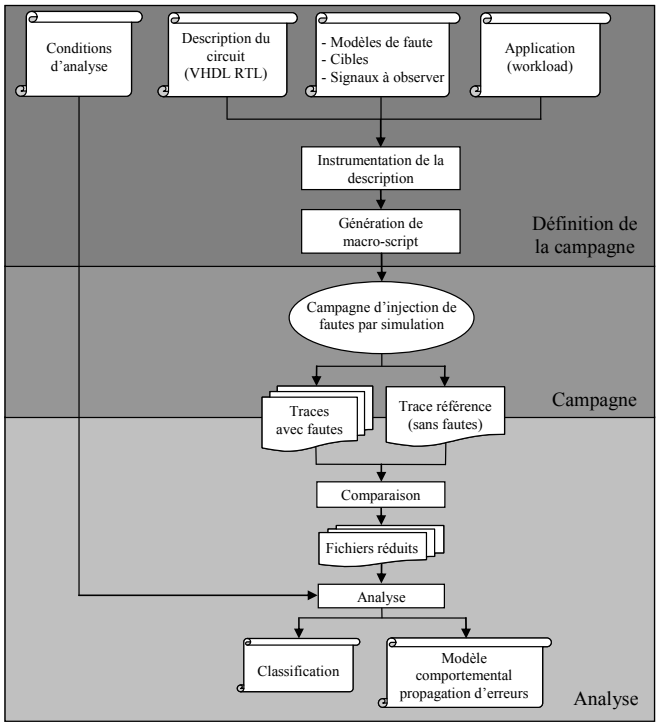

Figure 3-1 : Flot d'analyse à base de simulation

Lorsque l'on considère les ressources disponibles il est indéniable que l'instrumentation augmente le nombre d'entrées/sorties du circuit et la quantité de logique programmable nécessaire, contrairement à la reconfiguration. Mais nous verrons que les entrées/sorties supplémentaires ne sont pas une limitation lorsque l'on utilise un *wrapper* pour le circuit à analyser et nous verrons aussi que la taille de la logique supplémentaire reste faible.

Néanmoins il nous semble nécessaire de prendre en considération ce paramètre de logique programmable limitée. Ainsi nous décidons de ne mettre en œuvre qu'une seule copie du circuit à analyser. Certaines approches mettent en œuvre une copie de référence et une copie pour l'injection de fautes afin de comparer les sorties des deux copies en ligne. Ne pas implanter une copie référence du circuit permet d'avoir le double de logique programmable disponible pour le circuit à analyser. En outre implanter deux copies du circuit pose d'avantages de problèmes lorsque le circuit est connecté à des périphériques car ceux-ci doivent alors être partagés entre les deux copies.

Un des objectifs de l'environnement proposé est d'offrir la possibilité d'analyser des *System-On-Chip*. Le circuit analysé est dans ce cas connecté à un ou plusieurs périphériques tels de la mémoire ou un co-processeur. Seul le circuit ou le bloc à analyser est la cible d'injection de fautes mais il est mis en œuvre dans un contexte le plus proche possible du contexte de son application finale. Ceci a pour but de permettre une analyse dynamique plus simple et plus rapide. L'analyse est plus simple car il n'est pas nécessaire de modéliser le fonctionnement exact du ou des périphériques. L'analyse est plus

rapide car les entrées correspondant aux périphériques ne doivent pas être téléchargées depuis l'ordinateur hôte. Les périphériques peuvent être émulés à l'aide de la logique programmable (ex : co-processeur cryptographique) ou être disponibles sur la plateforme de prototypage (ex : banc de mémoire RAM).

Comme nous l'avons vu dans le chapitre précédent il existe déjà des approches d'analyse de sûreté à base de prototypage avec instrumentation du circuit à analyser. Celles-ci sont basées sur une plateforme de prototypage connectée à un ordinateur hôte qui est l'interface avec l'utilisateur. Les modules nécessaires à l'analyse (génération de la liste de fautes, analyse des sorties…) sont soit des modules logiciels exécutés sur l'hôte [CIVE-01a] soit des modules matériels implantés dans la logique programmable [LOPE-05].

Dans le cas où la majorité des modules sont des modules logiciels la partie matérielle se limite à une interface entre le circuit à analyser et la connectique avec l'ordinateur hôte. Cette interface ne dépend donc que du type de connectique (série, ethernet…) et des entrées/sorties du circuit. Si l'on considère que le circuit est instrumenté en conséquence, cette interface matérielle peut être utilisée pour toutes les campagnes sur un même circuit. Du côté de l'ordinateur hôte il est possible de modifier assez facilement les paramètres pour l'analyse (modèle de fautes, multiplicité, type d'analyse…) ou bien les modules logiciels eux-mêmes. Il est par exemple assez simple de rajouter une classe de faute dans le module d'analyse des sorties. Là où les traces peuvent également être sauvegardées sur l'ordinateur hôte afin d'effectuer plusieurs analyses différentes de ces traces.

L'inconvénient de cette approche est le grand nombre de transferts nécessaires entre le prototype et l'ordinateur hôte. Or ces transferts sont très gourmands en temps même à haut débit. Et c'est justement afin de réduire les transferts de données que certains proposent d'implanter un maximum de fonctionnalités en matériel. Il est en effet possible d'effectuer certaines tâches comme la génération de la liste de fautes ou l'analyse des sorties du circuit directement par des blocs matériels. Ceux-ci doivent être développés lors de la phase de définition de la campagne d'injection et être inclus dans le flot de développement du prototype. La faiblesse est alors qu'il faut reprendre le flot de conception matériel si l'on souhaite modifier certains paramètres. Par ailleurs, certaines tâches comme la génération de la liste de fautes peuvent être délicates à implanter en matériel (citons par exemple une génération avec distribution réellement aléatoire).

Ce que nous proposons est donc un environnement non pas à deux mais à trois niveaux : l'ordinateur hôte, un microprocesseur embarqué et les modules matériels. Ces derniers sont implémentés avec la logique programmable ce qui inclut l'émulation du circuit analysé et de l'interface matérielle, comme pour les approches existantes.

Nous tirons partie du microprocesseur embarqué afin de conserver un niveau de flexibilité important tout en transférant un maximum de tâches sur la carte et donc en limitant le nombre des transferts entre l'ordinateur hôte et la plateforme de prototypage. L'utilisation d'un processeur embarqué est grandement facilitée par l'existence de processeurs intégrés dans certains FPGA récents,

en plus de la logique programmable (exemple : famille Virtex2Pro chez Xilinx). Il est cependant possible de mettre en œuvre un cœur de processeur synthétisable et d'implanter ce processeur dans la logique programmable disponible.

La Figure 3-2 présente l'environnement d'analyse avec microprocesseur embarqué (en dur) dans le FPGA.

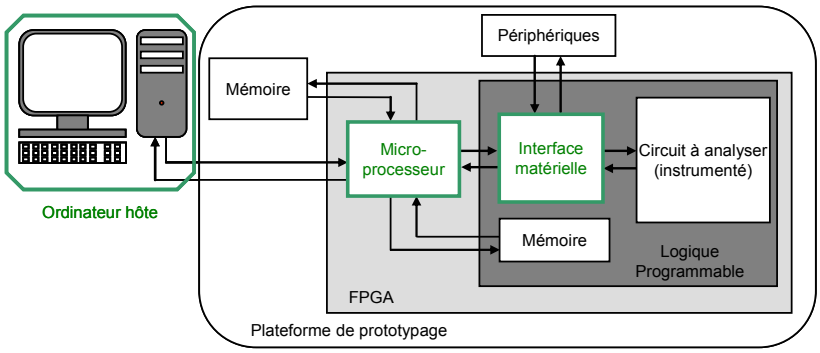

Figure 3-2 : Environnement d'analyse type avec processeur embarqué en dur

Avec cet environnement à trois niveaux il est alors possible de choisir à quel niveau doivent être exécutées les tâches nécessaires à une campagne d'injection. Le but pour l'utilisateur est d'obtenir le meilleur compromis entre complexité et vitesse des campagnes d'injection. Par analogie avec le flot de d'analyse par simulation, le flot d'analyse correspondant à l'environnement ci-dessus est présenté sur la Figure 3-3.

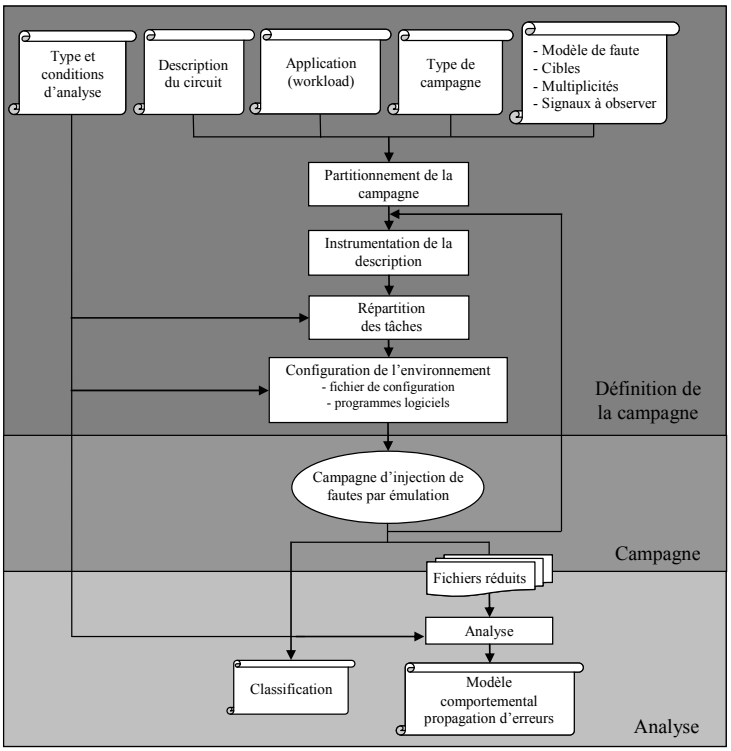

Figure 3-3 : Flot d'analyse à base d'émulation

Avant de rentrer dans les détails on notera l'étape « Répartition des tâches », représentation des choix qui s'offrent à l'utilisateur. Bien sûr l'étape « Campagne d'injection de fautes » présentée est simplifiée et cela pour deux raisons. La première vient du fait que l'on considère, comme pour la simulation, que cette étape s'exécute automatiquement, sans intervention de l'utilisateur. La seconde tient de cet environnement à trois niveaux où rien n'est figé ; présenter tous les cas possibles ne donnerait qu'un résultat peu lisible.

Dans la suite de ce chapitre nous étudions dans un premier temps la définition de la campagne de façon globale (partie 3.2). Nous présentons les trois niveaux d'exécution des tâches indépendamment dans la partie 3.3 puis nous décrivons comment se manifeste la flexibilité de l'environnement proposé dans la partie 3.4. Enfin nous introduisons la façon dont les tâches peuvent être réparties entre les niveaux de l'environnement (partie 3.5).

3.2 Définition des campagnes : principes de base

Définir une campagne d'injection pour l'environnement proposé correspond à sélectionner un cas d'étude, à générer les modules (matériels et logiciels) qui seront mis en œuvre lors de la campagne et à leur assigner une (ou plusieurs) tâche en fonction des souhaits du concepteur.

Typiquement la définition d'une campagne se base sur cinq entrées (voir Figure 3-3) :

1. la description du circuit à analyser,
2. l'application à exécuter,
3. les données relatives aux fautes à injecter (modèle, cibles, multiplicité, signaux à observer),
4. le type de campagne,
5. les conditions d'analyse.

Il faut tout d'abord sélectionner le circuit à analyser et fournir la description de celui-ci. Il peut s'agir d'une description au niveau RTL ou bien d'une description au niveau portes. Pour une description au niveau RTL, celle-ci doit impérativement être synthétisable. Pour une description niveau portes il est nécessaire qu'elle ait été synthétisée en ciblant la technologie correspondant au circuit de prototypage utilisé.

Parallèlement au circuit à analyser l'utilisateur doit disposer d'une application la plus significative possible. En effet le but de l'analyse est d'étudier le comportement du circuit en présence de fautes. Ce comportement dépend très souvent de l'application que le circuit est en train d'exécuter au moment où la faute se produit. Il est parfois difficile, voire impossible, de disposer de la ou des applications que devra exécuter le circuit lors de son utilisation finale. Il faut dans ce cas utiliser une application qui mettra en œuvre soit toutes les parties du circuit, soit celles qui seront mises en œuvre pour l'application finale.

Cette application ou charge de travail peut avoir différentes formes possible : liste des vecteurs d'entrées pour chaque cycle d'horloge, programme logiciel compilé à sauvegarder en mémoire embarquée, programme en ROM émulé avec la logique programmable. Pour les circuits simples il est

également possible de générer de manière aléatoire ou pseudo-aléatoire les vecteurs d'entrée du circuit.

Une bonne connaissance du circuit et de l'application qu'il doit effectuer permet de définir efficacement les points relatifs à l'injection de fautes proprement dite. Le premier choix se fait sur le modèle de faute utilisé : *bit-flip* ou collage transitoire. Ce point se rapporte aux modèles de fautes dont la pertinence a été mise en perspective dans le premier chapitre (partie 1.2.4.2). Du modèle de fautes dépend tout particulièrement la sélection des cibles d'injection et la sélection des points d'observation. Jusqu'à présent l'occurrence des fautes transitoires était considérée comme peu fréquente et avec des conséquences très localisées. En d'autres termes injecter une seule faute simple lors de l'exécution complète de l'application était suffisamment significatif. On parle dans ce cas de multiplicité spatiale égale à 1 (faute simple) et de multiplicité temporelle égale à 1 (une seule faute pour toute l'application). Or il devient de plus en plus pertinent d'injecter des fautes multiples et éventuellement plusieurs fois lors d'une expérience. La sélection des multiplicités spatiales et temporelles demeure à l'appréciation de l'utilisateur.

En ce qui concerne la partie injection de fautes, l'utilisateur doit définir le type de campagne qu'il souhaite effectuer. Il existe trois types de campagne possibles : déterministe, aléatoire (ou pseudo-aléaoire) et exhaustive. Le type de campagne détermine la nature de la liste de fautes et la façon dont celle-ci peut être générée. Pour une campagne déterministe l'utilisateur définit, pour chaque faute, la cible et le moment de l'injection. La liste de fautes peut également être générée de façon aléatoire (ou au moins pseudo-alétoire). Lors d'une campagne exhaustive, des fautes sont injectées dans chaque cible et à tous les cycles d'horloge de l'application.

Les conditions d'analyse doivent également être définies lors de la définition de la campagne. Elles ont un impact sur le choix du niveau auquel l'analyse peut être effectuée et donc sur la génération, ou non, de blocs matériels et logiciels. Deux grands types d'analyse sont possibles : la classification et l'analyse des chemins de propagation d'erreurs. Il y a pour la classification plusieurs options possibles afin d'effectuer une analyse la plus raffinée et rapide possible.

Lorsque ces cinq points ont été complètement spécifiés par l'utilisateur, la définition de la campagne se déroule suivant quatre étapes :

- le partitionnement de la campagne,
- l'instrumentation de la description à analyser,
- la répartition des tâches,
- la configuration de l'environnement.

En raison du nombre de ressources limité mais également par soucis de simplicité il est parfois nécessaire de partitionner une campagne d'injection en plusieurs sous-campagnes. Une sous-campagne correspond alors à l'injection d'un nombre restreint de fautes regroupées avec cohérence.

Chaque sous-campagne est caractérisée par ses données relatives aux fautes (cibles, multiplicité…), par son type de campagne et par ses conditions d'analyse. Le concepteur peut par exemple choisir d'analyser séparément et différemment deux blocs distincts d'un même circuit.

L'instrumentation est ensuite effectuée en fonction des cibles sélectionnées et des signaux à observer pour la campagne ou les sous-campagnes. Le flot de la Figure 3-3 suppose une instrumentation par sous-campagne mais il est tout à fait possible, si les ressources disponibles le permettent, de n'effectuer qu'une seule instrumentation. La même description instrumentée est alors utilisée pour chaque sous-campagne. Nous verrons dans le chapitre 4 à quel niveau d'abstraction l'instrumentation est, nous semble-t-il, la plus probante.

La répartition des tâches est le processus qui permet à l'utilisateur d'effectuer l'analyse la plus proche de ses attentes. Une répartition des tâches cohérente dépend des données d'entrée de l'analyse (type d'analyse …) mais également des caractéristiques de chaque niveau d'exécution. Cette étape est donc présentée dans la partie 3.5.

Lorsque l'analyse est effectuée par simulation RTL la mise en place de l'environnement d'analyse est relativement rapide. Par exemple pour l'injection de fautes via les commandes du simulateur lors de la simulation de descriptions HDL comportementales il « suffit » de générer les fichiers de commandes et de compiler la description à l'aide du simulateur HDL.

La configuration de l'environnement pour une approche basée sur le prototypage est plus complexe. En effet il faut effectivement réaliser un prototype du circuit à analyser et plus largement configurer l'environnement avant de pouvoir lancer la première campagne d'injection. Il faut d'une part concevoir la partie matérielle et générer le fichier de configuration (voir partie 2.3) qui sera téléchargé sur la carte de prototypage et d'autre part développer les programmes logiciels qui seront exécutés sur le PC hôte et sur le processeur embarqué. On notera que ce dernier point est une nouveauté par rapport aux approches d'analyse existantes. La génération du fichier de configuration est l'étape la plus gourmande en temps. Bien qu'il existe des environnements logiciels de conception complets (Xilinx ISE et EDK, Altera Nios II Development Kit…), certaines étapes comme la synthèse ou le placement/routage restent incompressibles.

Lorsque l'environnement est complètement configuré, le lancement d'une campagne correspond au téléchargement des données (application, liste de fautes) et au lancement des exécutables en parallèle.

Les trois étapes – instrumentation, répartition des tâches, configuration – doivent être effectuées complètement ou en partie seulement pour chaque sous-campagne si plusieurs sous-campagnes sont nécessaires. Prenons l'exemple d'un concepteur qui, après une campagne de classification exhaustive, souhaite raffiner son analyse en terme de cycles d'injection et de type d'analyse des résultats. Il est alors, dans certains cas, possible de ne re-développer que les programmes logiciels tout en gardant la même architecture matérielle. Nous verrons dans les parties 3.3 et 3.4 que l'environnement à trois niveaux offre ainsi une certaine flexibilité dont le concepteur peut tirer avantage.

3.3 Les trois niveaux de l'environnement

3.3.1 PC hôte

L'ordinateur hôte ne sert pas uniquement pour la configuration de l'environnement ou comme interface mais c'est une partie active de l'environnement d'analyse. C'est une interface dans le sens où il sert à lancer les campagnes d'injection et à récupérer les résultats générés. Mais le PC hôte peut également inclure certains modules logiciels effectuant des tâches lors des campagnes d'injection.

Ainsi la pertinence de l'ordinateur hôte comme niveau d'exécution prend réellement son sens lorsque l'on considère la possibilité qu'il offre de générer certaines données nécessaires à la campagne d'injection, l'application et la liste de fautes, et d'effectuer une partie de l'analyse des résultats obtenus.

Dans l'environnement à trois niveaux proposé, et indépendamment de la tâche à exécuter, le PC hôte est caractérisé par :

- une fréquence d'horloge supérieure au processeur embarqué,
- une grande facilité de développement et de modification (logiciels relativement simples),
- une grande précision grâce aux informations disponibles (testbenchs, résultats de campagnes précédentes…),
- une possibilité de stockage plus importante.

Mais également:

- la nécessité de télécharger/récupérer les données sur la carte.

3.3.2 Processeur embarqué

Le processeur embarqué est le niveau intermédiaire entre l'ordinateur hôte et la logique programmable. C'est un niveau intermédiaire de par sa place dans l'environnement et de par ses performances par rapport aux deux autres niveaux d'exécution. Son utilisation est ce qui différencie notre approche des approches à base de prototypage existantes.

Nous verrons dans la partie 3.4.5 que ce processeur peut-être un processeur implanté en dur ou bien un processeur émulé à l'aide de la logique programmable disponible.

Le microprocesseur embarqué est dévoué à l'exécution de plusieurs tâches, certaines optionnelles, d'autres obligatoires. Les tâches potentiellement réalisables sont celles cités plus haut pour l'ordinateur hôte : génération de données et analyse des résultats. La gestion de la campagne et le contrôle de l'interface matérielle sont assignés au processeur embarqué. Le contrôle consiste à :

- récupérer les données générées par le PC,
- gérer l'exécution des expériences qui constituent chaque campagne ou sous-campagne,
- remonter les résultats à la fin de la campagne.

On rappelle qu'une expérience correspond à l'exécution d'une application type et à l'injection d'un sous-ensemble précis de fautes pendant le déroulement de cette application. Le programme embarqué doit par exemple compter le nombre d'expériences effectuées et, le cas échéant, synchroniser les phases d'expérience et d'analyse.

En tant que niveau intermédiaire le processeur est également chargé du contrôle de l'interface matérielle. Via le contrôle de l'interface il commande, en fonction de la répartition des tâches, la façon dont sont gérés l'horloge, le reset, les entrées et les sorties du circuit analysé.

L'utilisation d'un processeur permet de tirer avantage des mécanismes d'interruption afin d'accélérer la réponse de celui-ci à un événement survenant au niveau de l'interface matérielle. A la fin d'une expérience l'interface matérielle peut par exemple activer un signal d'interruption qui sera capté par le processeur. Celui-ci peut alors traiter les données obtenues et passer à l'expérience suivante et ainsi de suite. Le contrôle de l'interface d'injection est étudié plus en détails dans le chapitre 4.

Dans l'environnement proposé le processeur embarqué est caractérisé par :

- des transferts rapides avec l'interface matérielle et le circuit analysé,
- une grande facilité de développement et de modification (programmes logiciels relativement simples et rapidement transférables),
- la possibilité des générer des données en cours de campagne, voire à chaque cycle.

Mais également:

- une fréquence d'horloge inférieure au PC hôte,
- une faible précision si les informations disponibles (application, liste de fautes…) ne sont pas intégrées aux programmes,
- une capacité de stockage moyenne.

3.3.3 Interface et modules matériels

La partie matérielle, troisième et dernier niveau d'exécution de l'environnement, inclut une interface entre le processeur embarqué et le prototype instrumenté du circuit à analyser. L'interface matérielle, au sens large du terme, a deux objectifs : se connecter à l'environnement et contrôler le circuit analysé.

L'environnement que nous proposons est véritablement un *System on Programmable Chip* (SoPC) qui comprend un processeur et des périphériques (mémoire, communication…) connectés par un ou plusieurs bus comme présenté sur la Figure 3-4.

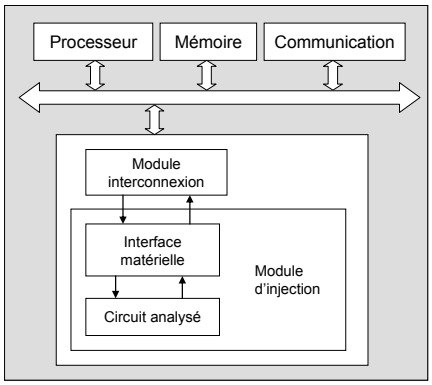

Figure 3-4 : Architecture SoPC avec périphérique d'injection

Le circuit à analyser est donc connecté au processeur embarqué via une interface qui est connectée à un bus par un module d'interconnexion. L'ensemble module d'interconnexion et module d'injection forme un périphérique d'injection vu par le processeur comme tout autre périphérique. L'un de nos objectifs est de développer un environnement le plus portable possible. A ce titre les protocoles (propriétaires ou non) mis en œuvre lors de l'implémentation de l'environnement ne doivent pas avoir de conséquence sur l'architecture globale de l'environnement. Nous limitons donc volontairement le terme « interface matérielle » à la partie dépendante de la méthode et indépendante de l'implémentation, i.e. l'interface entre le module d'interconnexion et le circuit analysé.

On remarquera que l'utilisation d'un *wrapper* autour du circuit analysé entraîne une absence de contraintes sur le nombre d'entrées/sorties de celui-ci. Les entrées/sorties du circuit ne sont connectées ni aux plots du FPGA ni à un bus interne.

L'interface matérielle est donc chargée du contrôle du circuit ce qui correspond à l'exécution des commandes provenant du processeur embarqué. Elle comprend un bloc de contrôle et un certain nombre de modules matériels (Figure 3-5) dont l'implémentation ou non dépend de la répartition des tâches choisie par l'utilisateur. L'utilisation de blocs matériels pour effectuer certaines tâches permet d'accélérer les expériences puisque cela réduit les transferts de données entre l'interface et le processeur. Il est ainsi possible, dans certains cas, de générer les vecteurs d'entrées, de générer la liste de fautes et d'analyser en ligne les sorties du circuit.

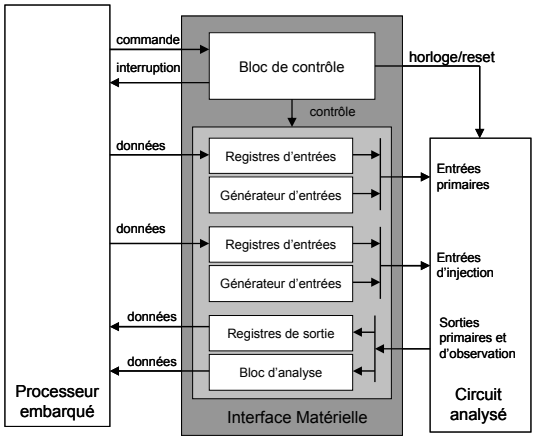

Figure 3-5 : Interface matérielle

Le circuit à analyser est instancié par l'interface matérielle. Les entrées et les sorties primaires ainsi que les entrées d'injection et les sorties d'observation, ajoutées lors du processus d'instrumentation, sont donc disponibles au niveau de son interface. L'horloge et le reset sont générés par l'interface sur commande du processeur. Les entrées primaires et les entrées d'injection sont soit générées par des blocs matériels, soit temporairement mémorisées dans des registres d'entrée dont l'écriture est également commandée par le processeur. Il en va de même pour les sorties primaires et les sorties d'observation qui, soit servent d'entrées à des blocs matériels (analyse), soit sont temporairement stockées dans des registres de sortie qui sont lus par le processeur.

Dans l'environnement proposé la réalisation de tâches par des blocs matériels est caractérisée par :

- très peu de transferts avec le processeur embarqué,
- la possibilité de générer des données en cours de campagne, voire à chaque cycle.

Mais également:

- un développement plus complexe et surtout des modifications coûteuses en temps,
- une faible précision si les informations disponibles (application, liste de fautes…) ne sont pas intégrées dans les blocs matériels,
- une possibilité de stockage assez faible et plus complexe à mettre en œuvre.

3.3.4 Bilan

Nous venons de voir la présentation des trois niveaux d'exécution que nous proposons de mettre en œuvre dans notre environnement d'analyse ainsi que leur utilisation possible pour l'exécution des tâches nécessaires à l'analyse de sûreté d'un circuit. Leurs principales caractéristiques sont récapitulées dans le Tableau 3-1.

La mise en œuvre d'un processeur embarqué est ce qui différencie notre approche des approches existantes. Le sous-chapitre suivant montre que s'appuyer sur trois niveaux d'exécution participe à la flexibilité de l'environnement d'analyse proposé.

Tableau 3-1 : Caractéristiques des 3 niveaux d'exécution

	Flexibilité[1]	**Précision[2]**	**Mémoire[3]**	**Temps d'exécution**	**Rapidité des transferts**
Ordinateur hôte	++	++	+	+	--
Processeur embarqué	+	-	-	-	-
Modules matériels	-	-	--	++	++

[1] Facilité et rapidité de modification

[2] Aptitude à générer des données déterministes

[3] Inclut l'espace disponible et la facilité de mise en œuvre

3.4 Un environnement flexible et performant

Au moment où l'analyse de sûreté devient nécessaire pour un nombre croissant d'applications (avionique, automobile, médical…) nous avons cherché à définir un environnement d'analyse le plus flexible possible. Ce sous-chapitre montre que l'environnement s'adapte un maximum aux souhaits de l'utilisateur en terme d'analyse, aux techniques qu'il désire utiliser et aux moyens à sa disposition.

3.4.1 Circuits cibles

Tout d'abord il semble indispensable qu'un environnement d'analyse puisse permettre l'analyse de différents types de circuit. Cette propriété s'adresse aussi bien au concepteur qui conçoit déjà plusieurs types de circuits qu'à celui qui ne souhaite pas être limité dans le futur.

Nous avons réparti les circuits en quatre catégories :

- les circuits avec entrées externes,
- les circuits autonomes,
- les périphériques de processeur,
- les processeurs.

Pour les circuits avec entrées externes il est nécessaire de définir les vecteurs d'entrées primaires à chaque cycle d'horloge. Comme cela a été présenté précédemment ces vecteurs d'entrées peuvent être générés à l'un des trois niveaux d'exécution. Le contrôle du circuit dépend du niveau auquel les entrées sont générées. Un cas typique est un circuit dont l'application est générée par le PC hôte puis sauvegardé en RAM dans l'environnement. Les vecteurs d'entrée sont alors lus en RAM puis écrits dans les registres d'entrée de l'interface par le processeur embarqué. Indépendamment du type d'analyse, le processeur commande donc l'interface cycle par cycle.

Nous considérons qu'un circuit est « autonome » lorsque ses entrées primaires sont réduites à l'horloge et au reset. C'est par exemple le cas des circuits dont l'application se trouve en ROM (exemple : microcontrôleur i8051) qui peut être émulée avec de la logique programmable. Le contrôle de tels circuits dépend alors de la gestion des sorties (i.e. du type d'analyse). Si l'analyse ne se fait qu'à la fin de l'application alors le processeur peut commander à l'interface de générer l'horloge du circuit pour toute la durée de l'application.

Il est également possible de mettre en œuvre un second processeur afin d'étudier le fonctionnement d'un périphérique. Ce second processeur, comme celui chargé des injections, peut être un processeur en dur - certains FPGA, comme les Xilinx Virtex2Pro, incluent deux cœurs de processeur en dur - ou émulé via la logique programmable. Il exécute une application type alors que le premier processeur reste chargé du contrôle des injections. La mise en œuvre d'un tel environnement d'analyse n'est cependant pas aisée puisqu'il faut alors veiller à la bonne synchronisation des deux processeurs notamment au moment de l'injection de fautes.

Enfin l'environnement permet l'analyse de processeurs, que l'on peut considérer comme un cas particulier, certes plus complexe, de circuit avec entrées externes. En effet un processeur est généralement connecté à des périphériques et notamment à des bancs mémoires externes qui contiennent ses instructions et ses données.

Le problème est que les mémoires disponibles dans la plupart des SoPCs sont connectées à un unique bus dans le but de réduire le nombre de connexions. Le processeur embarqué a besoin de mémoire pour son exécution et cette mémoire n'est alors plus disponible pour le processeur analysé. De plus si le processeur analysé est directement connecté à une mémoire externe cela supprime l'accès à ses sorties à partir du processeur embarqué. Il y a alors des conflits dans l'utilisation des ressources.

Une solution proposée dans [PORTo-07] est d'implanter la mémoire du processeur analysé dans la mémoire du processeur embarqué. Il suffit alors de rajouter un module logiciel sans nécessité de modifier les parties matérielles. Pour chaque cycle du circuit analysé ses sorties sont mémorisées dans des registres de sortie au niveau de l'interface matérielle. Le processeur embarqué récupère les signaux de contrôle, les données sortantes et l'adresse demandée par le circuit analysé et il agit en conséquence : soit il va chercher les données à lire pour les renvoyer à l'interface, soit il va mettre à jour l'image de la RAM.

Une autre solution est d'implémenter un accès direct en mémoire (DMA - Direct Memory Access) au niveau de l'interface matérielle. Deux espaces mémoires distincts sont dans ce cas réservés : une pour le processeur embarqué et une pour le processeur analysé.

La première solution nécessite un fonctionnement en mode « pas à pas » alors que celle avec accès direct en mémoire permet aussi un fonctionnement en mode « interface autonome ». Cela a l'avantage de remplacer les transferts « RAM - processeur embarqué – interface » par des transferts « RAM – interface » moins gourmands en temps. Les deux modes de fonctionnement sont détaillés dans la section 4.2.1 et l'accélération apportée par le DMA est illustrée dans la partie 4.2.3.

Pour les cas où le périphérique lit directement les données en RAM, il faut que celui-ci soit capable d'accéder à un bloc mémoire de taille suffisante et éventuellement qu'il n'y ait pas de conflit entre les éléments ayant accès à ce bloc mémoire.

3.4.2 Types d'instrumentation

Nous avons vu qu'il existe plusieurs approches en ce qui concerne l'instrumentation du circuit à analyser, aussi bien pour la simulation que pour l'émulation. On rappelle que l'instrumentation correspond à la modification de la description originale afin de permettre l'injection de fautes.

Deux grandes approches coexistent : l'instrumentation au niveau RTL et l'instrumentation au niveau portes. Pour chaque niveau d'instrumentation il existe plusieurs techniques dont les propriétés diffèrent. Succinctement nous pouvons citer [LEVE-03b] pour l'instrumentation au niveau RTL et [PORTe-04] ou [LOPE-05] pour l'instrumentation au niveau portes.

Pour notre environnement, si l'on considère que la logique programmable nécessaire est effectivement disponible, il n'y a pas de contraintes sur le type d'instrumentation ; ni sur le niveau ni sur la technique. Seule l'instrumentation au niveau transistor n'est pas supportée mais le modèle de faute correspondant (collage de transistor) ne correspond à aucun modèle de faute considéré ici (voir partie 1.2.4.2).

Le circuit à analyser est instancié par l'interface matérielle (niveau hiérarchique inférieur). Il est donc tout à fait possible d'instancier une description HDL qui est synthétisée avec l'interface. De même il est possible d'instancier une *netlist* qui est insérée lors du processus de placement/routage. Bien sûr le type d'instrumentation a un impact sur la façon dont l'interface est implémentée et notamment sur le chargement des données d'injection. Il suffit de prendre en compte le chargement parallèle ou sériel des données d'injection lors de la conception de l'interface.

3.4.3 Méthode de génération des données

La génération des données pour la campagne a été introduite lors de la présentation des trois niveaux d'exécution dans le sous-chapitre 3.3. Nous l'étudions ici plus en détails parce que cet aspect de la flexibilité est très important, notamment en ce qui concerne la génération de la liste de fautes.

3.4.3.1 Génération des entrées primaires

L'analyse de sûreté est une analyse dynamique et fonctionnelle. La génération de l'application dépend donc du circuit à analyser et on peut considérer que dans la plupart des cas elle est connue préalablement et précisément par le concepteur. On parle alors de génération déterministe de l'application au niveau de l'ordinateur hôte.

La Figure 3-6 présente le cas où les vecteurs d'entrée d'un circuit avec entrées externes sont obtenus pour chaque cycle d'horloge à partir d'un *testbench* utilisé pour la simulation. Cette traduction peut être obtenue par une simulation pendant laquelle on récupère les entrées du circuit à chaque cycle d'horloge. Si certains signaux pour la simulation sont indéfinis (valeurs 'U' ou 'X'), une solution est de leur assigner la valeur '0' comme cela est présenté sur la Figure 3-6.

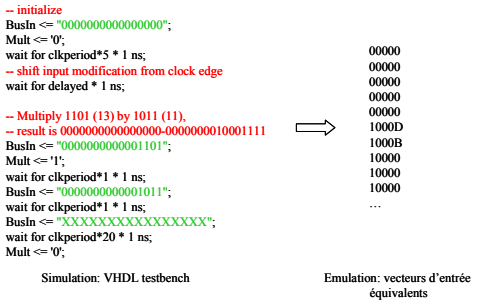

Figure 3-6 : Traduction d'un testbench VHDL en suite de vecteurs d'entrées

Dans le cas d'une génération déterministe, les vecteurs d'entrées sont téléchargés sur la carte et sauvegardés en mémoire au démarrage de la campagne. Une fois les données sauvegardées en RAM, elles sont accessibles au processeur et au périphérique d'injection.

La génération de données au niveau du processeur embarqué peut être aléatoire, pseudo-aléatoire ou exhaustive. Si les entrées du circuit doivent être définies à chaque cycle d'horloge il est peu probable qu'une génération aléatoire représente une application significative. En revanche il est possible de générer aléatoirement des données prises en compte au départ de l'application. Ceci peut être le cas pour les circuits effectuant des calculs du type multiplication ou cryptage. On peut imaginer générer de manière exhaustive les données prises en compte au départ de l'application, comme par exemple effectuer un calcul n fois (n expériences) avec toutes les valeurs possibles en entrée. Cependant ceci induirait un nombre d'expériences considérables pour un circuit un tant soit peu important.

L'avantage du processeur embarqué est que les vecteurs d'entrée peuvent être générés pour chaque cycle d'horloge au moment de la commande de l'interface par le processeur. Le processeur envoie alors les vecteurs d'entrées à l'interface puis commande la génération de l'horloge. Ceci permet de s'affranchir de la sauvegarde en mémoire si l'espace mémoire est trop faible. Il est possible de générer la même suite pseudo-aléatoire pour chaque expérience pour que le circuit opère les mêmes actions à chaque expérience. Se baser sur l'heure pour initialiser l'algorithme donne au contraire une suite aléatoire différente pour chaque expérience.

Au niveau de l'interface matérielle il est possible de générer des vecteurs d'entrées de façon pseudo-aléatoire ou de façon exhaustive simplement avec un compteur. Le schéma de la Figure 3-7 présente un exemple de générateur pseudo-alétoire du type registre à décalage avec rétroaction linéaire (LFSR - *Linear Feedback Shift Register*) qui peut être implémenté en matériel. Les sorties des bascules sont lues en parallèle ou en série et l'initialisation peut se faire avec les set et reset (non représentés). Dans notre cas les sorties seraient directement connectées aux entrées du circuit à analyser et l'horloge serait la même. L'inconvénient d'un tel générateur est que les suites générées ne sont pas suffisamment aléatoires puisque l'initialisation est toujours la même. Si nécessaire il est cependant possible d'implémenter un chargement sériel de la chaîne de FFs pour l'initialisation. L'avantage par rapport aux autres niveaux d'exécution est qu'aucun accès mémoire n'est nécessaire.

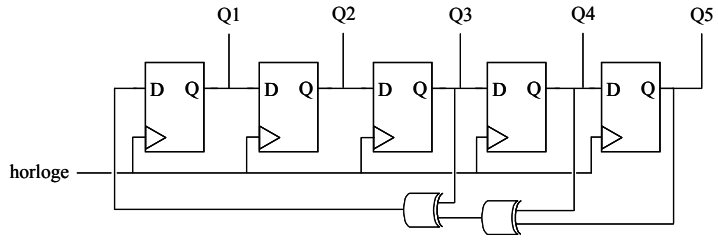

Figure 3-7 : Exemple de bloc pour la génération pseudo-aléatoire de données (LFSR)

La pertinence de la génération des vecteurs d'entrées au niveau matérielle rejoint celle au niveau processeur embarqué. Cela permet d'obtenir des vecteurs d'entrée significatifs uniquement pour certains cas particuliers du type cryptage où un vecteur d'entrée est la valeur à crypter par exemple.

Mais utiliser un module matériel permet alors d'accélérer la génération par rapport à la génération au niveau processeur, en utilisant une petite partie de la logique programmable disponible.

3.4.3.2 Génération de la liste de fautes

Une liste de fautes comprend une suite de couples « mot cible(s) »/« mot cycle » correspondant chacun à une injection. La taille du « mot cible » dépend de l'instrumentation et du contrôle des injections. Sans codage cette taille est égale au nombre de bascules potentiellement ciblées, que le chargement soit sériel ou parallèle. Nous verrons dans le chapitre 4, qu'il est parfois possible de réduire la taille du « mot cible » en le codant. Si l'injection est basée sur l'inversion de la valeur mémorisée (e.g. approche avec masque) la multiplicité spatiale, i.e. le nombre de bascules effectivement ciblées, correspond au nombre de bits à '1' dans ce « mot cible ». Le « mot cycle » contient évidemment le cycle d'horloge de l'injection. La multiplicité temporelle, i.e. le nombre de cycles par expérience pour lesquels on injecte des fautes, doit être définie a priori. Il indique le nombre de couples dans la liste de fautes à considérer par expérience.

La génération de la liste de fautes dépend du type de campagne que l'utilisateur souhaite réaliser. Néanmoins l'utilisation du PC hôte pour cette tâche peut se justifier par les données déjà disponibles par le concepteur. Plus il a d'informations sur le circuit qu'il doit étudier plus il lui est facile de déterminer où et quand il souhaite injecter des fautes. Si par exemple il n'a besoin d'injecter que quelques fautes mais cela de manière très localisée dans le temps et l'espace, il peut définir sa liste de fautes à partir de l'ordinateur hôte avant de lancer sa campagne.

La Figure 3-8 présente un exemple de liste de fautes (8 fautes) telle qu'elle est générée par le PC hôte. La cible (à gauche) et le cycle d'horloge (à droite) sont deux mots de 32 bits.

```
0001 0100
0001 0101
0001 0102
00C0 0100
00C0 0101
80A0 0209
80A0 020A
```

Figure 3-8 : Exemple de liste de fautes déterministe

S'il possède des informations sur la localisation probable des cellules dans le circuit final il peut choisir d'injecter des fautes multiples (MBU) en considérant ces informations. Comme pour l'application on parle de génération déterministe de la liste de fautes.

Dans le cas d'une génération déterministe, la liste de fautes est téléchargée sur la carte et sauvegardée en mémoire au démarrage de la campagne.

La mise en œuvre du processeur embarqué est très pertinente pour la génération de la liste de fautes. Les gros circuits (nombre de cibles important) sont souvent accompagnés d'applications assez longues (nombre de cycles d'horloge important) et dans ce cas une liste réduite de faute aléatoire permet une analyse significative réalisée en un temps acceptable.

Le pseudo-code de la Figure 3-9 présente un exemple de génération aléatoire du « mot cible » avec l'algorithme de *Mersenne-Twister* au niveau du processeur embarqué. L'algorithme de *Mersenne-Twister* permet d'obtenir une distribution uniforme et l'initialisation par la fonction init_genrand() avec la date courante comme paramètre permet d'obtenir une liste de fautes aléatoire. Cet exemple porte sur la génération d'un mot dont un seul bit est à '1' soit l'injection d'un SEU. Ce type de génération aléatoire serait très coûteux à implanter avec une approche purement matérielle.

```
/*=========================*/
XTime_GetTime(Start_Camp_Ptr);
init_genrand((unsigned long)Start_Camp);
…
/*== SEU ==*/
InjectionData=0x80000000;
shift_float=genrand_int32()/(1+(double)RAND_MAX);
/*== shift between 0 and 31 (<32) ==*/
shift_float=32*shift_float;
shift=(int)shift_float;
InjectionData=InjectionData>>shift;
/*=========================*/
```

Figure 3-9 : Génération aléatoire du mot d'injection par le processeur embarqué

Pour les circuits relativement petits ou si l'on n'étudie qu'un sous ensemble d'un gros circuit, il est envisageable de générer une liste de faute exhaustive afin d'obtenir une analyse plus précise.

Enfin des blocs matériels de génération pseudo-aléatoire comme celui de la Figure 3-7 peuvent être implantés pour la cible d'injection d'une part et le cycle d'injection d'autre part. Contrairement aux vecteurs d'entrée la liste de fautes doit être définie avant le lancement de chaque expérience et ne peut être générée en parallèle de l'exécution de l'application. Pour chaque expérience la liste de fautes doit être mémorisée afin que le bloc de contrôle puisse comparer le cycle d'horloge courant au cycle d'injection et, le cas échéant, positionner les entrées d'injection en fonction de la ou des cibles sélectionnées. L'horloge du bloc de génération n'est pas la même que celle du circuit analysé.

Avec de tels blocs on ne peut garantir une multiplicité spatiale constante. Une solution envisageable est alors de générer non pas le « mot cible » mais l'index des bits qui doivent être positionnés dans le mot cible. Pour l'injection de *bit-flips* simples un seul générateur suffit. Pour l'injection de *bit-flips* multiples (multiplicité spatiale supérieure à 1) on utilisera un nombre de générateurs égal à la multiplicité spatiale.

Les quelques remarques ci-dessus illustrent la complexité et le manque de flexibilité induits par l'utilisation de modules matériels. On voit que passer d'un modèle de faute simple à un modèle de faute multiple nécessite de reprendre la phase de développement matériel et peut s'avérer compliqué. En outre la mise en œuvre de générateurs suivant un algorithme précis comme l'algorithme de *Mersenne-Twister* s'avère plus complexe. Utiliser des modules matériels pour la génération des fautes entraîne un gain de temps dû à la diminution des transferts avec le processeur mais également une perte de flexibilité.

On notera que la génération pseudo-aléatoire ou aléatoire ne peut pas être utilisée avec tous les types d'instrumentation et de chargement des données d'injection. En effet la technique où les bascules sont reliées en chaîne de *scan* avec chargement sériel des états erronés (State-scan) ne permet pas de générer aléatoirement ces états. Chaque état erroné doit prendre en compte l'état des bascules au cycle d'injection ce qui dépend de l'application. Les états erronés doivent donc être définis a priori avant le lancement de la campagne. Dans [LOPE-05] les états erronés sont générés au niveau du PC hôte.

3.4.4 Analyse

3.4.4.1 Niveau d'analyse

L'analyse des résultats obtenus est la troisième et dernière tâche qu'il peut être pertinent d'assigner à l'ordinateur hôte. Ceci est principalement le cas pour une analyse raffinée telle que l'analyse de chemins de propagation des erreurs. Pour ce type d'analyse il faut effectuer un nombre conséquent d'expériences et potentiellement stocker beaucoup d'informations pour chaque expérience. Malgré la taille de la mémoire disponible sur certaines plateformes de prototypage, souvent cela ne suffit pas pour des circuits analysés significatifs. En outre ce type d'analyse nécessite une certaine mise en forme qu'il est plus aisé d'effectuer au niveau du PC.

Le processeur embarqué peut effectuer tout ou partie de l'analyse des résultats. Lors de chaque expérience le processeur peut récupérer les sorties du circuit à analyser et les comparer avec les résultats attendus, ceci à chaque cycle d'horloge ou uniquement à la fin de l'application (e.g. vérification de l'exactitude d'un calcul). Il peut alors tirer avantage des bancs de mémoire disponibles pour stocker soit des données intermédiaires comme les résultats attendus soit le résultat de l'analyse avant transfert vers le PC hôte à la fin de la campagne. Des bancs de mémoires peuvent être émulés sur la logique programmable ou être présents sur la carte de prototypage, ces derniers pouvant être de taille conséquente.

Enfin un bloc matériel d'analyse peut effectuer l'analyse en ligne des sorties du circuit. Les sorties primaires et d'observation sont alors directement connectées aux entrées de l'analyseur. L'analyse est effectuée en fonction des conditions d'analyse définies par l'utilisateur et qui auront été intégrées à la description du bloc d'analyse. Une nouvelle fois cela présente un gain en temps d'autant plus qu'il faut ajouter aux sorties primaires les sorties d'observation dont le nombre peut être important pour une analyse raffinée. L'analyse au niveau matériel convient bien aux analyses simples où la transposition des conditions d'analyse en description matérielle n'est pas trop complexe. En revanche, elle est mal adaptée par exemple pour l'analyse des chemins de propagation d'erreurs.

3.4.4.2 Type d'analyse

La plupart des outils décrits dans la littérature visent à classifier les fautes, c'est-à-dire identifier les ensembles de fautes associés aux principaux modes de défaillance ou de détection pouvant être

activés. L'environnement présenté permet en plus de la simple classification, l'analyse des chemins de propagation d'erreurs.

La classification correspond à la génération d'un modèle simple (Figure 3-10) représentant le comportement en présence de fautes. Une telle classification permet surtout à un concepteur d'évaluer l'efficacité des mécanismes de détection d'erreurs et le taux de défaillance critique du circuit.

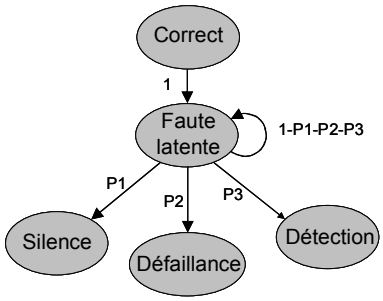

Figure 3-10 : Exemple de modèle obtenu par classification des fautes

Si la classification est logicielle (PC hôte ou processeur embarqué) il est nécessaire d'exécuter une fois l'application sans injection de fautes (*golden run* ou de *fault-free run*) afin de récupérer une trace de référence. On appelle trace l'évolution des signaux observés à chaque front active de l'horloge. Si la classification est matérielle le bloc de classification se charge de classifier chaque faute en ligne lors de chaque expérience. Dans ce cas la trace de référence est en général obtenue par simulation puis intégrée, intégralement ou en partie seulement, dans le bloc de classification.

La classification par le PC hôte permet de conserver les traces (avec et sans fautes) après la campagne d'injection. En outre la comparaison de la trace de chaque expérience avec la trace de référence est très rapide. Mais procéder de cette façon signifie transférer, de la plateforme au PC hôte, la trace pour chaque expérience et ceci peut littéralement exploser les temps de campagnes. Cette solution n'est donc pas efficace.

Le processeur embarqué permet d'implémenter les mêmes algorithmes logiciels de comparaison. Dans le cas de la classification par le processeur embarqué la trace de référence est conservée en mémoire embarquée tout au long de la campagne d'injection. Ensuite pour chaque expérience avec injection, deux solutions sont envisageables :

- la trace complète est sauvegardée en mémoire puis comparée à la trace de référence à la fin de l'expérience ; après comparaison et classification la trace peut être écrasée,
- les sorties du circuit analysé sont comparées à la trace de référence à chaque cycle d'horloge, seule la trace de référence est alors mémorisée et il est possible de stopper l'exécution de l'expérience lorsqu'une erreur ou une défaillance est détectée.

On utilise un compteur logiciel pour chaque classe de faute. Lorsqu'une faute est classifiée par le processeur, le compteur correspondant est incrémenté. A la fin de la campagne, seuls les résultats de la classification sont remontés au PC hôte, ce qui correspond à seulement quelques octets par classe de fautes.

Il est possible d'implémenter en matériel un bloc de sauvegarde et de comparaison de trace mais le gain en vitesse, par rapport à une approche équivalente via le processeur, peut-être faible comparé aux développements nécessaires. L'intérêt d'un bloc de classification matériel augmente surtout s'il permet de comparer les sorties du circuit à des valeurs prédéfinies par le concepteur et intégrés dans ce bloc lors de la configuration de l'environnement. Nous pouvons prendre pour exemples la comparaison des sorties avec des valeurs prédéfinies à chaque cycle ou encore la comparaison de l'évolution de sorties spécifiques avec une suite de valeurs prédéfinies sans considération de l'horloge etc. La complexité du bloc de classification dépend alors du concepteur, de ses besoins et du temps à sa disposition.

La classification que l'on qualifie de traditionnelle a pour objectif la génération d'un modèle comme celui de la Figure 3-10. Ce type de classification considère les fautes injectées dans leur ensemble et permet d'analyser le comportement global du circuit. Nous proposons également une classification dite « raffinée » qui mémorise la classe d'appartenance de chaque faute et qui permet ainsi d'analyser précisément l'impact de chaque faute injectée. Ce raffinement n'entraîne pas de surcoût matériel mais nécessite un contrôle des injections différent et induit des temps de campagne plus grands à cause de la quantité de résultats qui est plus importante.

La Figure 3-11 illustre la structure générale du modèle comportemental complexe correspondant à un modèle fonctionnel de haut niveau du comportement du circuit en présence de fautes. Les sommets du graphe correspondent aux états atteints par le circuit durant les expériences. Chaque état est défini par une liste de signaux erronés parmi ceux observés. Chaque état est aussi étiqueté avec un type fonctionnel par exemples : correct, erreur, sans erreur, détection, tolérance, défaillance, etc.

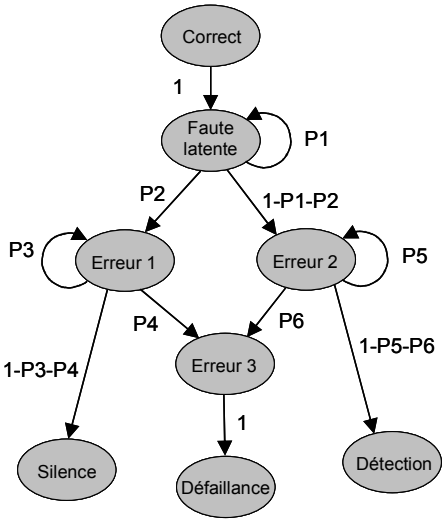

Figure 3-11 : Exemple de modèle obtenu par analyse des chemins de propagation d'erreurs

Les transitions observées entre les états sont également enregistrées. Le graphe de propagation d'erreurs commence par l'état « Correct» et se termine par une configuration d'erreur ou bien un état de « Détection » par exemple. Chaque transition est associée à une probabilité de passage de l'état

« Si » à l'état « Sj » calculée en fonction de l'activité du système au moment des expériences. Cette probabilité est obtenue à partir du pourcentage de cas où l'état « Sj » est atteint juste après « Si », durant l'ensemble de la campagne d'injection.

L'analyse des résultats d'une expérience donnée peut s'interrompre sous certaines conditions définissant des états terminaux (généralement liés à une détection ou à une défaillance). Les conditions sont spécifiées par l'utilisateur et peuvent être très précises, incluant l'état exact de certains signaux ou des propriétés d'évolution temporelle.

Un tel graphe est généré à partir de fichiers réduits qui dérivent des traces obtenues lors de la campagne d'injection. Les fichiers réduits ne contiennent que les différences des signaux observés avec ceux de l'expérience de référence.

La génération des graphes de propagation à partir des fichiers réduits est dérivée des travaux de Karim Hadjiat [LEVE-03b]. Nous nous sommes uniquement intéressés à la génération des fichiers réduits dans un contexte d'analyse par prototypage.

La génération des fichiers réduits peut-être effectuée par un module logiciel sur le PC hôte. Mais de la même manière que pour la classification il faut dans ce cas transférer les traces correspondantes à toutes les expériences. Pour réduire la taille des données à transférer il est possible de générer les fichiers réduits directement par le processeur embarqué. Un exemple d'algorithme est présenté en Figure 3-12.

```
cycle d'horloge = 0
offset mémoire = 0
taille mémoire = nombre de mots de 32-bits pour les sorties * nombre de cycles total
faire{
        pour(nombre de mots de 32-bits pour les sorties) {
                Lecture en mémoire du "mot attendu" suivant
                Lecture en mémoire du "mot avec faute" suivant
                si("mot attendu" et "mot avec faute" différents) {
                        si (cycle d'horloge non envoyé) {
                                Envoi d'un octet de synchronisation au PC hôte
                                Envoi du cycle d'horloge au PC hôte
                                Mise à 1 du drapeau "horloge envoyée"
                        }
                        pour(index 1 à 4) {
                                si("octet attendu" et "octet avec faute" différents) {
                                        Envoi de l'index
                                        Envoi de l'"octet attendu"
                                        Envoi de l'"octet avec faute"
                                }
                        }
                }
                Incrément de l'offset mémoire
        }
        Incrément du cycle d'horloge
        si ("horloge envoyée" = 1)
                Envoi d'un octet de synchronisation au PC hôte

}tant que (offset mémoire < taille mémoire)

Envoi d'un octet de synchronisation au PC hôte
```

Figure 3-12 : Pseudo-algorithme pour la génération des fichiers réduits

L'implémentation de l'algorithme ci-dessus par un bloc matériel nécessiterait un accès à un bloc mémoire très important pour le périphérique d'injection. Il n'est pas envisageable d'utiliser la logique programmable pour émuler un bloc mémoire suffisant car la quantité de données à stocker peut être très importante (trace sans faute, trace avec faute, différences entre les deux traces). Dans la pratique une implantation de ce type d'analyse dans l'interface matérielle est peu envisageable.

3.4.5 Implantation

La large gamme de prototypes offre désormais le choix entre l'utilisation d'un cœur de processeur implanté en dur *(hard processor core*) ou l'implémentation d'un processeur sur la logique programmable disponible (*soft processor core*).

En effet certains FPGAs incluent un cœur de processeur implanté en dur au centre de la logique programmable. On peut par exemple citer les familles Virtex2Pro développée par Xilinx ou FPSLIC produite par Atmel. L'environnement correspond alors à celui présenté Figure 3-13.

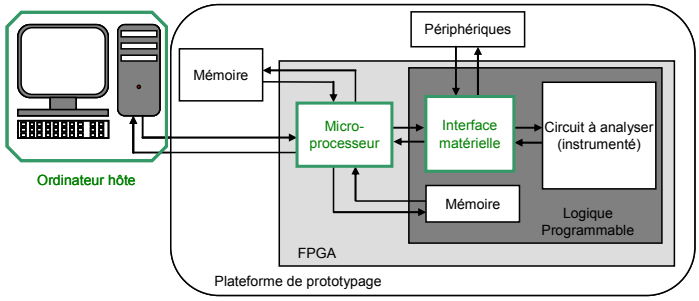

Figure 3-13 : Environnement avec coeur de processeur implanté en dur

Lorsqu'il n'y a pas de processeur en dur il est possible d'utiliser une partie de la logique programmable afin d'en implanter un. Certains vendeurs proposent leurs propres cœurs de processeur synthétisable tels MicroBlaze et PicoBlaze (Xilinx) ou Nios et Nios II (Altera). Si l'on ne souhaite pas utiliser un processeur « propriétaire » il est possible d'implanter un processeur « libre » comme par exemple le processeur Leon développé par Gaisler Research. L'environnement correspond alors à celui présenté Figure 3-14.

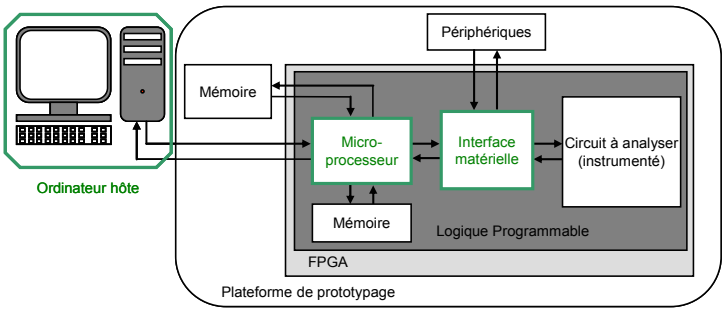

Figure 3-14 : Environnement avec cœur de processeur synthétisable

L'inconvénient de l'implémentation d'un processeur à l'aide de la logique programmable est que cela réduit d'autant la logique programmable disponible pour le périphérique d'injection et pour le

circuit à analyser. C'est donc une contrainte pour l'implémentation sur un FPGA dont la logique programmable disponible est peu conséquente.

L'environnement proposé convient donc particulièrement aux plateformes de prototypage qui incluent un FPGA avec processeur en dur ou un FPGA de taille importante, ce qui est souvent le cas pour les FPGAs récents. Enfin on peut envisager d'implémenter l'environnement à l'aide de machines d'émulation plus complexes du type Cadence Incisive Palladium ou Mentor Graphics Veloce. Les investissements requis sont toutefois d'un ordre de grandeur très différent. Il peut être économiquement plus rentable de disposer d'un certain nombre de cartes FPGA réalisant des expériences d'injection en parallèle, plutôt que d'investir dans une machine d'émulation aussi complexe. L'utilisation de cette dernière se justifie essentiellement pour des systèmes intégrés de très grande complexité ou si elle est disponible et inutilisée (cas peu probable en pratique).

3.5 Répartition des tâches

Comme nous l'avons vu dans la partie 3.1 la répartition des tâches dépend directement du type de circuit analysé, du type de campagne, du type et des conditions d'analyse. Pour une tâche donnée il est parfois possible de l'effectuer à différents niveaux. Par exemple si le type d'analyse souhaité est la classification alors celle-ci peut être effectuée au niveau du processeur embarqué ou au niveau de l'interface matérielle. Dans le premier cas le programme de comparaison des sorties doit être généré, dans le second cas il faut concevoir le bloc matériel pour l'analyse en ligne. En revanche nous avons vu qu'effectuer la classification au niveau du PC hôte n'a pas de sens à cause des transferts de données entre la carte et le PC. Le choix du concepteur se base donc sur les caractéristiques de chaque niveau d'exécution présentées précédemment. Le Tableau 3-2 propose pour chaque tâche les niveaux d'exécution possible afin de tirer profit au maximum des performances de chaque niveau.

Tableau 3-2 : Répartition des tâches

Tâche	Niveau d'exécution		
	PC hôte	Microprocesseur embarqué	Interface matérielle
Génération des entrées primaires	Déterministe	Exhaustive / Aléatoire	Exhaustive / Pseudo-aléatoire
Génération de la liste de fautes	Déterministe	Exhaustive / Aléatoire	Exhaustive / Pseudo-aléatoire
Analyse des résultats	Chemins de propagation d'erreur	Chemins de propagation d'erreur / Classification	Classification

3.6 Conclusion

Au vu des problématiques actuelles en terme de sûreté de fonctionnement et des techniques d'analyse existantes, un environnement d'injection de fautes basé sur un prototype du circuit analysé est un outil efficace. Les approches existantes sont cependant marquées par deux inconvénients : des temps de transferts importants et une certaine rigidité pour les approches effectuant un maximum de tâches au niveau matériel.

L'idée est donc d'intégrer dans l'environnement un niveau intermédiaire entre le PC hôte et le matériel et d'obtenir un environnement à trois niveaux : PC hôte - microprocesseur embarqué – blocs matériels. Un des objectifs est une plus grande flexibilité caractérisée notamment par une étape de répartition des tâches entre les niveaux d'exécution au cours du flot d'analyse.

Les trois niveaux qui composent l'environnement présentent chacun des caractéristiques différentes qui permettent d'obtenir le meilleur compromis entre la complexité de la tâche à effectuer et les performances obtenues lors de son exécution.

Plusieurs possibilités sont donc offertes au concepteur pour la génération des vecteurs d'entrée du circuit, pour la génération des listes de fautes et pour l'analyse des résultats obtenus. Mais l'environnement supporte également plusieurs types de circuit, plusieurs techniques d'instrumentation et il peut être implémenté sur différentes plateformes de prototypage.

Afin d'avoir un environnement d'analyse plus rapide et moins coûteux en logique programmable nous proposons plusieurs techniques. Le chapitre suivant présente ainsi deux modes de contrôle possibles, deux techniques d'instrumentation et un contrôle optimisé des injections pour ces types d'instrumentation.

4 Optimisation des techniques pour l'analyse de sûreté

Comme nous venons de le voir l'approche que nous proposons offre une grande flexibilité et supporte notamment plusieurs types de circuit et plusieurs techniques d'instrumentation.

De plus la mise en œuvre de techniques d'instrumentation au niveau RTL [LEVE-03b] ou au niveau netlist [PORTe-04][LOPE-05] sont envisageables. Ces approches ont chacune leurs avantages mais il nous a semblé que des améliorations peuvent être apportées afin de réduire les coûts en surface et en temps d'expérience. Nous proposons donc une technique d'instrumentation réduite, son extension en fonction des cibles d'injection et un contrôle optimisé des injections.

Le contrôle du circuit analysé dépend de son niveau d'autonomie et de l'analyse effectuée. Ainsi l'étude des circuits autonomes avec analyse effectuée par un bloc matériel ne requiert un contrôle qu'au début et à la fin de chaque expérience. Nous formalisons dans ce chapitre les deux modes de contrôle possibles.

4.1 Instrumentation du circuit cible

L'instrumentation est la modification de la description originale afin de rendre possible l'injection de fautes. Elle doit permettre d'accéder aux ressources internes (nœuds, cellules mémoires) du circuit analysé. L'instrumentation dépend donc du modèle de faute choisi et de l'architecture du prototype mis en oeuvre.

4.1.1 Modèles de fautes dans une approche FPGA

On rappelle les deux modèles de fautes identifiés (voir 1.2.4.2) pour des descriptions au niveau RTL et au niveau portes :

- l'inversion de bits mémorisés (bit-flip) simple et multiple,
- le collage (SA) transitoire simple et multiple.

L'architecture d'une cellule configurable dans un FPGA basé sur des cellules SRAM est constituée de trois grands groupes d'éléments : les portes logiques (multiplexeurs, OU-exclusif), les LUTs et les cellules mémoires (bascules ou verrous). Une LUT est configurée pour réaliser une certaine fonction logique entre ses entrées. Les connexions entre les éléments sont établies également en fonction de la configuration du FPGA. L'architecture exacte (portes logiques disponibles…) dépend du type de FPGA.

L'injection d'un *bit-flip* correspond au basculement d'un élément mémoire. Pour cela deux approches sont envisageables : soit ajouter des éléments (combinatoires et/ou séquentiels) pour contrôler la donnée en entrée qui est mémorisée soit contrôler les entrées set, reset, clear ou preset de la cellule mémoire.

L'injection d'un collage peut être effectuée via les connexions entre les éléments du circuit analysé. L'ajout de logique combinatoire (ex : multiplexeur) permet alors de « déconnecter » un signal et de le connecter momentanément à la masse ou au Vcc.

Les attributs d'une faute à injecter sont principalement la cible (simple ou multiple), l'instant d'injection et la durée de la faute. Or dans un environnement FPGA et avec l'utilisation d'un *wrapper*, seul le contrôle des entrées primaires (horloge et reset inclus) et des entrées d'injection ajoutées lors de l'instrumentation est possible.

Le contrôle de la cible d'injection se fait via la ou les entrées d'injection, que le chargement soit sériel ou parallèle. En ce qui concerne les attributs temporels de la faute il faut dans tous les cas prendre en compte l'horloge du circuit. Il est en effet inutile de définir des instants et des durées d'injection puisque les caractéristiques temporelles du prototype sont différentes du circuit final. On définit donc l'instant et la durée d'une faute en nombre de cycles d'horloge.

Les attributs pour chaque modèle de faute sont donc :

- bit-flip : cible et cycle d'horloge,
- collage transitoire : cible, cycle d'horloge initial, nombre de cycles de la faute.

Nous n'avons conservé par la suite que le modèle de faute *bit-flip* parce que celui-ci modélise le mieux les phénomènes liés aux impacts de particule. Les techniques présentées peuvent facilement être adaptées au cas des collages transitoires.

4.1.2 Niveau netlist : avantages et inconvénients

La mise en œuvre d'un périphérique d'injection dans un environnement de prototypage FPGA permet de choisir le niveau de description du circuit analysé. Il est possible d'instancier un circuit décrit au niveau RTL ou au niveau portes. Cette liberté nous a permis de choisir le niveau d'instrumentation puisque celle-ci est préalable à l'instanciation.

Tout d'abord nous considérons que l'environnement de développement (Xilinx XPS, Altera Nios II Development Kit…) supporte les langages VHDL et verilog ce qui lève une possible restriction dépendante de l'environnement de développement.

Malgré cela le langage avec lequel la description initiale est effectuée a son importance puisque les modifications dépendent de la syntaxe du langage. De plus pour un langage donné l'instrumentation est dépendante de la manière dont le concepteur a décrit le circuit surtout si l'instrumentation est faite sur une description de haut niveau. Il faut alors également prendre garde à ce que les modifications apportées à la description initiale ne soient pas supprimées lors de la phase de synthèse. Le développement d'un outil d'instrumentation est alors complexe. L'avantage d'une instrumentation au niveau RTL est cependant qu'elle permet d'utiliser des modèles de fautes complexes. Par exemple il est possible d'injecter des fautes correspondant à des transitions erronées [LEVE-03b], ce qui devient impossible si l'instrumentation est réalisée après synthèse.

Choisir une instrumentation au niveau portes c'est tout d'abord dépasser les contraintes dues à la description initiale. D'ailleurs celle-ci peut être dans certains cas disponible uniquement au niveau portes. Pour une description RTL l'unique contrainte est qu'elle soit synthétisable mais ceci est nécessairement le cas pour du prototypage.

L'instrumentation au niveau portes sous-entend que la description à modifier a été synthétisée pour une certaine architecture de FPGA. La contrainte porte alors sur le format de la netlist et sur les éléments qui peuvent être mis en œuvre lors de la modification du circuit. Les éléments logiques ou séquentiels employés doivent être disponibles dans l'architecture ciblée.

Contrairement à l'instrumentation au niveau RTL, il faut se contenter au niveau portes des modèles de fautes de base *bit-flip* et collage.

Nous avons cependant opté pour une instrumentation au niveau portes pour plusieurs raisons :

- indépendance par rapport au langage de description initial,
- indépendance par rapport aux optimisations dues à la synthèse,
- pertinence du modèle de fautes *bit-flip*,
- rapidité et simplicité du développement d'un outil d'instrumentation automatique,
- possibilité d'utiliser une bibliothèque d'éléments à mettre en oeuvre en fonction du FPGA ciblé.

4.1.3 Choix de l'injection synchrone

Pour une injection asynchrone le basculement du point mémoire se propage à travers la logique combinatoire à sa sortie jusqu'à atteindre le ou les points mémoires suivant. Si la faute est injectée entre les fronts d'horloge des cycles N-1 et N, la propagation de la faute est éventuellement observable au cycle N au niveau des points mémoire suivants.

Pour une injection synchrone l'effet de la faute se propage au cycle suivant le cycle d'injection. Si la faute est injectée au front d'horloge du cycle N elle se propage éventuellement au cycle N+1 au niveau des points mémoire suivants.

Pour injecter un SEU de manière asynchrone il faut soit contrôler les entrées set et reset soit contrôler l'horloge de la bascule ciblée. Nous choisissons d'injecter les fautes de façon synchrone en modifiant la logique combinatoire à l'entrée de la bascule.

Il faut remarquer que la synchronisation de l'injection avec le front d'horloge n'est pas une limitation car l'analyse réalisée est au niveau purement fonctionnel (sans prise en compte des temps de propagation). Tous les SEUs pouvant survenir dans une même bascule pendant un cycle d'horloge donné sont donc équivalents, et peuvent être assimilés à une inversion du contenu de la bascule au début du cycle.

4.1.4 Technique d'instrumentation développée

4.1.4.1 Principe d'instrumentation « réduite »

❖ Injection

Nous nous sommes tout d'abord intéressé à la cellule mémoire la plus simple, la bascule D sans entrée de validation ni set/reset. Celle-ci recopie la valeur de l'entrée D sur la sortie Q à chaque front actif de l'horloge. L'entrée D est connectée à la sortie d'un bloc de logique combinatoire dont la fonction n'a pas d'importance ici. La sortie Q est également connectée à un bloc combinatoire.

L'injection d'un *bit-flip* correspond à l'inversion de la valeur mémorisée par la cellule mémoire. Pour ce type de bascule la valeur mémorisée dépend uniquement de la valeur de l'entrée D au moment du front actif de l'horloge. Pour que la valeur mémorisée soit inversée il suffit donc d'inverser la valeur de l'entrée D.

Pour cela la technique la plus simple est d'utiliser une porte réalisant la fonction OU-exclusif (XOR) comme cela est présenté sur la Figure 4-1. Pour plus de clarté, les blocs de logique combinatoire sont regroupés dans un seul bloc.

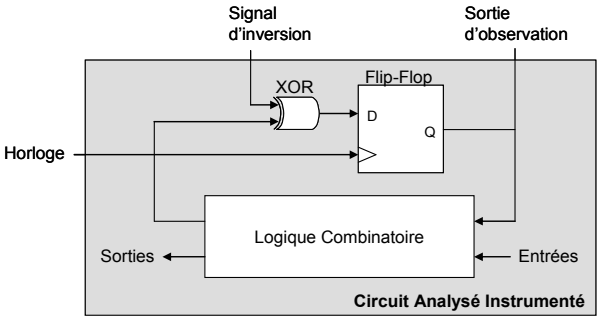

Figure 4-1 : Instrumentation réduite

Le Tableau 4-1 représente la table de vérité obtenue. Nous avons bien une inversion de la valeur mémorisée lorsque l'entrée d'injection est positionnée à '1'. Lorsque celle-ci est à '0' la cellule XOR est transparente et l'entrée D a la même valeur que la sortie de la logique combinatoire.

Tableau 4-1 : Table de vérité d'injection pour une bascule D simple

Sortie logique combinatoire	Signal d'inversion	Entrée D	Sortie Q
0	0	0	0
1	0	1	1
0	1	1	0
1	1	0	1

❖ Observation

On remarque sur la Figure 4-1 que la sortie de la cellule ciblée peut également être connectée à une sortie afin d'observer sa valeur. Cette sortie peut être ajoutée pour la validation de l'injection et l'observation de la valeur de la cellule au cours de l'expérience. On augmente ainsi l'observabilité à faible coût lors du processus d'instrumentation.

Il peut être également utile d'observer certaines cellules qui ne sont pas directement la cible d'injections. Cela permet d'observer la propagation de l'erreur dans le circuit au cours de chaque expérience. Nous proposons donc d'étendre l'ajout de cette sortie d'observation potentiellement à l'ensemble des bascules identifiées dans le circuit analysé.

Pour une description non hiérarchique, les entrées d'injection et les sorties d'observation apparaissent directement à l'interface du circuit. Pour une description hiérarchique, chaque entrée d'injection et chaque sortie d'observation est d'abord rajoutée à l'interface du bloc qui contient la cellule mémoire concernée. Ces entrées et sorties doivent ensuite être remontées jusqu'à l'interface du circuit complet. Elles sont ainsi disponibles à partir de l'interface d'injection.

Cette technique d'instrumentation entraîne un surcoût matériel très limité (une seule cellule XOR par bascule ciblée) et est très facilement automatisable puisqu'une seule connexion est « coupée » entre la sortie de la logique combinatoire et l'entrée de la bascule.

❖ Mise en œuvre pratique

L'instrumentation proposée cible principalement les architectures FPGA qui contiennent des portes XOR comme l'architecture Xilinx Virtex2. Il est alors très simple de rajouter l'instanciation de ces portes dans la description niveau portes.

Pour les architectures qui ne comprennent pas de cellules XOR (Altera StratixII, Atmel FPSLIC AT40K) il est possible de réaliser la fonction équivalente à l'aide d'une LUT.

4.1.4.2 Extension aux points mémoire avec entrée de validation

❖ Injection

L'instrumentation « réduite » ci-dessus ne fonctionne cependant pas pour des éléments mémoires avec une entrée de validation CE (« Chip-Enable »). Pour ces éléments mémoire la sortie recopie l'entrée au front actif de l'horloge si et seulement si l'entrée de validation est active. Lorsque le signal de validation est inactif la sortie de la cellule reste inchangée. Dans ce cas ajouter une seule porte XOR permettrait d'inverser la valeur mémorisée uniquement aux cycles d'horloge où le CE est valide. Or un SEU, par exemple, modifie la valeur mémorisée quelque soit la valeur de l'entrée de validation.

La première idée est de rajouter un multiplexeur pour forcer la valeur du CE au moment de l'injection, le signal de sélection du multiplexeur étant le signal d'inversion. Dans le cas où celui-ci est inactif le CE de l'élément mémoire prend la valeur du CE provenant du circuit original. Au cycle d'injection le signal d'inversion devient actif et le CE est forcé à la valeur '1' (Vcc). Cependant cette technique seule ne permet pas d'injecter une faute dans tous les cas. En effet si l'on considère que le signal de validation est forcé de '0' à '1' et que la valeur à l'entrée de la bascule est inversée mais que cette valeur inversée est égale à la valeur mémorisée au cycle précédent alors il n'y a pas basculement du point mémoire.

Pour les bascules avec entrée de validation il faut donc prendre en compte la valeur de la sortie Q au cycle précédent le cycle d'injection Q(T-1). Pour cela nous proposons d'ajouter un inverseur et un

multiplexeur comme présenté Figure 4-2. L'entrée de sélection du multiplexeur est le signal de validation CE. Lorsque celui-ci est actif la sortie, complémentée ou non, de la logique combinatoire est positionnée sur l'entrée de la cellule mémoire. S'il est inactif alors la sortie inversée de la cellule mémoire est prise en compte. Dans tous les cas où le signal d'inversion est positionné la valeur mémorisée est inversée par rapport à la valeur sans injection.

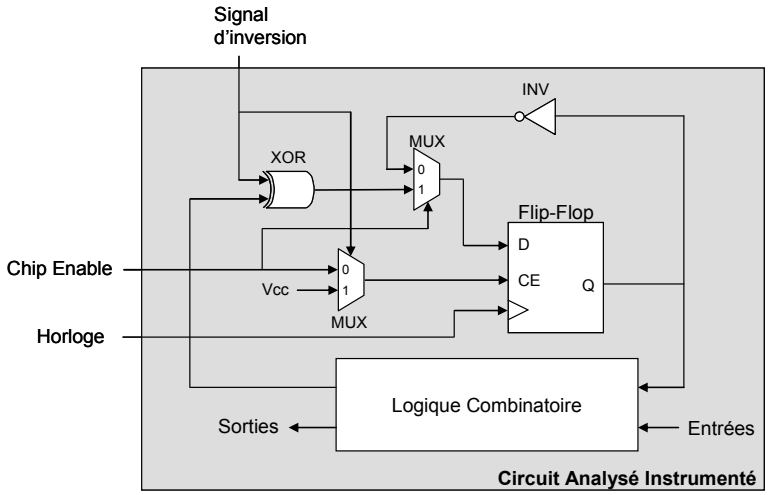

Figure 4-2 : Instrumentation avec multiplexeurs et inverseur

❖ Mise en œuvre pratique

Si l'architecture cible comprend les portes logiques mises en œuvre pour l'instrumentation de la Figure 4-2 il est tout à fait envisageable de les utiliser. C'est le cas pour l'architecture Xilinx Virtex2 qui inclut multiplexeurs, inverseurs et portes XOR.

La fonction qui est réalisée à partir du signal d'inversion, du signal d'injection, de la sortie de la logique combinatoire et de la sortie du point mémoire peut également être mise en œuvre avec une LUT (Figure 4-3). Ceci est indispensable si les portes logiques ne sont pas directement disponibles ou bien utile si l'on veut optimiser l'instrumentation. Nous verrons dans la section 4.1.5 la comparaison entre la technique à base de portes logiques et la technique à base de LUT.

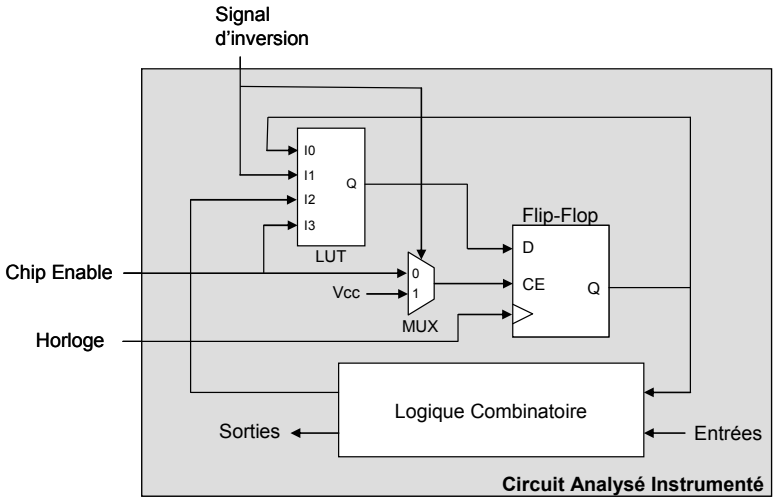

Figure 4-3 : Instrumentation LUT et multiplexeur

La fonction logique qui doit être implémentée avec la LUT est la suivante :

$$Q = (CE) * (\sim inv * CBo + inv * \sim CBo) + (\sim CE) * (\sim inv * Q + inv * \sim Q)$$
$$Q = (I3) * (\sim I1 * I2 + I1 * \sim I2) + (\sim I3) * (\sim I1 * I0 + I1 * \sim I0)$$

Équation 4-1 : Fonction logique d'injection avec LUT

4.1.4.3 Limitations

Une limitation de l'instrumentation se présente pour les bascules avec set synchrone, reset synchrone, preset asynchrone, clear asynchrone. En effet la technique d'instrumentation proposée ne permet pas de contrôler ces entrées. Pour les entrées asynchrones, qui doivent normalement être réservées à l'initialisation générale du circuit, il ne s'agit pas réellement d'un problème. En ce qui concerne les entrées synchrones, il serait possible de les contôler sur le même principe que l'entrée CE si cela était nécessaire pour un circuit donné.

Nous n'avons pas pris en compte les autres types de bascules (bascules JK, toggle Flip-Flop) car leur utilisation est moins commune et dans un environnement FPGA ces bascules sont souvent implémentées à l'aide de bascules D et de logique combinatoire.

4.1.5 Coûts

L'ajout de logique à une description originale a un coût matériel qu'il est préférable de minimiser. Cela a peu d'importance lorsque le circuit à analyser est de taille réduite mais cela peut en avoir si la logique programmable disponible est limitée. Les Tableau 4-2 et Tableau 4-3 permettent de comparer le coût d'une instrumentation à base de portes XOR avec celui d'une instrumentation à base de LUTs.

Les résultats présentés sont obtenus pour deux cas d'études (un multiplieur 16bits et le microcontrôleur i8051) implantés avec les outils Xilinx (ISE) sur FPGA Virtex2. La logique programmable de ce FPGA est composée de *Slices* qui incluent chacune deux LUTs, deux cellules mémoires, quelques cellules de logique combinatoire (ou-exclusif, multiplexeurs) et des interconnexions entre ces éléments.

Le Tableau 4-2 concerne les bascules sans entrée de validation (CE). On remarque que le coût de l'instrumentation à base de LUTs est moindre que celui de l'instrumentation à base de portes OU-exclusif. La cause principale est l'utilisation d'une LUT pour le routage seul afin d'accéder à la porte XOR pour chaque bascule ciblée. Cette LUT n'effectue aucune fonction logique, la sortie est simplement la copie de l'entrée utilisée. Le coût pour chaque bascule ciblée est donc d'une porte XOR et d'une LUT.

Pour l'instrumentation à base de LUTs les optimisations lors de la projection technologique limitent encore le coût puisqu'il y a au final moins de LUTs supplémentaires que de bascules ciblées. Il est donc préférable d'utiliser ce type d'instrumentation d'autant plus qu'elle est portable sur les autres FPGA dont l'architecture ne comprend pas de porte XOR.

Tableau 4-2 : Coûts des instrumentations à base de XORs et à base de LUTs pour bascule sans validation

Caractéristiques	i8051		
	Original	**XORs**	**LUTs**
Nombre total de FFs	1324	1324	1324
N_{FF} instrumentées	-	123 (9%)	123 (9%)
Nb 4-input LUTs	4508	4631	4595
- routage seul	17	140	17
- fonction logique	4491	4491	4578
Coût	-	+3%	+2%
Nb Slices	2484	2570	2525
Coût	-	+3%	+2%
Equivalent nb de portes	109771	110140	110293
Coût	-	+0,3%	+0,5%
Fréquence max(MHz)	38,3	34,8	34,3
Coût		-9,2%	-10,4%

Les résultats présentés dans le Tableau 4-3 concernent cette fois les bascules avec entrée de validation (CE). Les instrumentations effectuées sont celles de la Figure 4-2 pour l'instrumentation à base de OU-exclusif et de la Figure 4-3 pour l'instrumentation à base de LUTs.

Tableau 4-3 : Coûts des instrumentations à base de XORs et à base de LUTs pour bascule avec validation

Caractéristiques	Multiplieur 16bits			i8051		
	Original	**XORs**	**LUTs**	**Original**	**XORs**	**LUTs**
Nombre total de FFs	74	74	74	1324	1324	1324
N_{FF} instrumentées	-	74 (100%)	74 (100%)	-	256 (19%)	256 (19%)
Nb 4-input LUTs	179	460	323	4508	5527	5020
Coût	-	+157%	+80%	-	+23%	+11%
Nb Slices	92	362	208	2484	3384	2817
Coût	-	+293%	+126%	-	+36%	+13%
Equivalent nb de portes	1996	3052	2638	109771	113611	112075
Coût	-	+53%	+32%	-	+3.5%	+2,1%
Fréquence max(MHz)	174,1	101,9	165,4	38,3	37,4	35,0
Coût		-41,0%	-5,0%		-2,3%	-9,0%

On observe sur les Figure 4-4 et Figure 4-5 que pour les deux cas d'études et pour tous les indices l'instrumentation à base de LUTs a un coût approximativement deux fois moins important que l'instrumentation à base de XORs en terme de matériel. Le seul point pour lequel l'instrumentation

avec LUTs est désavantageuse est la fréquence maximum de fonctionnement pour le microcontrôleur. Ceci peut être expliqué par la fréquence de fonctionnement nominale qui est assez basse (<40MHz). Nous verrons dans le chapitre 6 ce qu'il en est pour d'autres cas d'étude.

Les coûts relativement importants de l'instrumentation sont à comparer à la taille des circuits et aux nombres de bascules instrumentées, notamment pour le multiplieur avec un coût proche de 300% en terme de *Slices* mais avec un taux d'instrumentation de 100% et seulement 2000 portes pour la version non instrumentée.

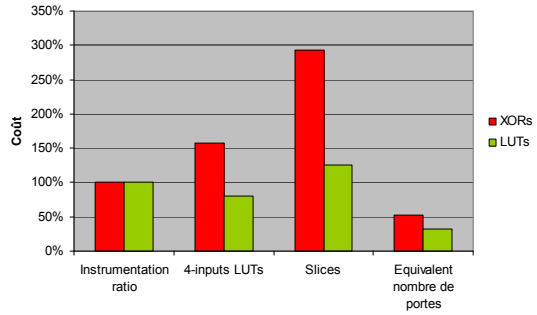

Figure 4-4 : Coût de l'instrumentation (Multiplieur 16bits)

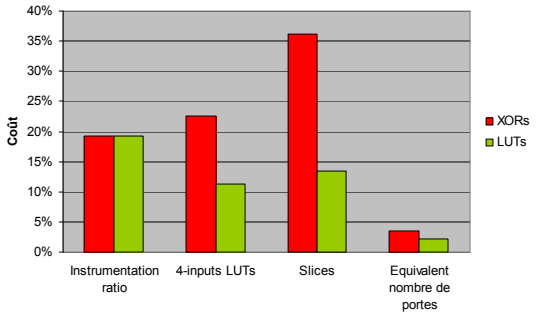

Figure 4-5 : Coût de l'instrumentation (i8051)

En comparaison des techniques d'instrumentation existantes et en terme de nombre de portes ajoutées, les performances de l'approche que nous proposons sont bonnes. Le Tableau 4-4 permet une comparaison théorique du nombre de portes ajoutées par bascule ciblée en fonction de la technique d'instrumentation (N_{FF} est le nombre de bascules ciblées).

Les trois approches d'instrumentation existantes ont été présentées au paragraphe 2.3.2. On rappelle que la technique « Mask-scan » présentée dans [CIVE-01a] correspond à l'utilisation d'un registre masque. L'approche « State-scan » est basée sur la connexion des bascules ciblées pour former un registre à décalage et l'approche « Time-multiplexed » correspond à l'émulation avec et sans injection de faute au même moment avec comparaison des sorties directement par le matériel. Ces deux approches sont détaillées dans [LOPE-05].

Tableau 4-4 : Comparaison théorique des coûts des techniques d'instrumentation existantes

Technique d'instrumentation	FF	Xor	Nor	And	Nand	Mux	Inv	LUT
Mask scan	N_{FF}	N_{FF}		$2N_{FF}$		N_{FF}		
State scan	N_{FF}					N_{FF}		
Time multiplexed	$3N_{FF}$	N_{FF}		N_{FF}		$5N_{FF}$		
Réduite	N_{FF}	N_{FF}						
Etendue (XOR)	N_{FF}	N_{FF}				$2\ N_{FF}$	N_{FF}	
Etendue (LUT)	N_{FF}					N_{FF}		N_{FF}

Pour les bascules sans entrée de validation l'instrumentation réduite est l'une des moins coûteuses avec l'approche « State-scan » puisqu'une seule porte logique est ajoutée, respectivement une porte OU-exclusif ou un multiplexeur.

Pour les bascules avec Chip-Enable, l'approche d'instrumentation étendue est certes plus coûteuse mais avec quatre portes logiques supplémentaires, elle reste équivalente à l'approche « Mask-scan ».

Les bascules ajoutées ne correspondent pas obligatoirement au procédé d'instrumentation à proprement parler. Pour les approches « State-scan », réduite et étendue, les bascules ne sont pas ajoutées lors de l'instrumentation. Elles sont néanmoins indispensables pour stocker les données correspondant aux cibles d'injection et doivent être implémentées dans l'interface d'injection.

Le coût induit par l'approche d'instrumentation que nous proposons est correct en terme de nombre de portes ajoutées et par rapport aux approches existantes. Un des avantages est qu'il est possible d'instrumenter différemment en fonction de la nature de la bascule ciblée et ainsi obtenir le coût matériel le plus faible possible. D'autre part nous avons vu que l'instrumentation à base de LUTs permet de réduire ce coût approximativement de moitié. Enfin nous verrons dans le prochain sous-chapitre qu'il est encore possible de réduire le nombre de bascules nécessaire en codant les données d'injection, en ne rajoutant que peu de logique.

4.2 Interface matérielle

4.2.1 Modes de contrôle du circuit analysé

Le circuit analysé est instancié et contrôlé par l'interface d'injection, elle-même commandée par le processeur embarqué. Le contrôle du circuit correspond à la génération de l'horloge, du reset, éventuellement au positionnement des entrées, à la récupération des sorties ou des résultats de l'analyse et au contrôle des injections.

Au vu des différences possibles entre les ensembles « type de circuit/type d'analyse » nous avons choisi de mettre en œuvre deux modes de contrôle:

- mode « pas à pas »,
- mode « interface autonome ».

En mode « pas à pas » le processeur contrôle indirectement, via l'interface, le circuit analysé. Pour chaque cycle de l'application, après avoir éventuellement généré et positionné les entrées, le processeur commande la génération d'un coup d'horloge puis il récupère les sorties du circuit. Ce mode de fonctionnement est nécessaire lorsque le processeur doit fournir les vecteurs d'entrée ou analyser les sorties à chaque cycle d'horloge.

Le chronogramme de la Figure 4-6 présente les principaux signaux au niveau de l'interface pour un fonctionnement en mode « pas à pas ». On peut distinguer trois phases :

- la commande provenant du processeur (I),
- l'exécution d'un cycle d'horloge (avec injection si les données d'injection sont positionnées) (II),
- la génération d'une interruption pour que le processeur commande le cycle suivant (III).

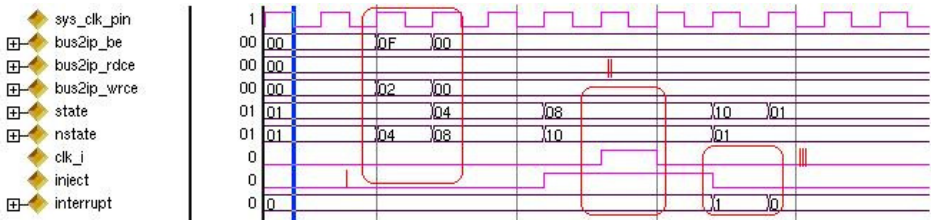

Figure 4-6 : Chronogramme de contrôle en mode « pas à pas »

En mode « interface autonome » le processeur laisse l'interface gérer l'horloge et les entrées/sorties du circuit pendant un nombre défini de cycles (le reset est dans tous les cas commandé par le processeur). Ceci est possible si les entrées du circuit analysé sont disponibles directement au niveau de l'interface (génération matérielle ou DMA) ou du circuit lui-même (ROM émulée) sans intervention du processeur embarqué. De plus les sorties doivent dans ce cas être gérées également par l'interface ; soit ignorées : soit sauvegardées en RAM, soit analysées par un module matériel dédié.

Le chronogramme de la Figure 4-7 présente le fonctionnement en mode interface autonome avec ROM émulée. Pour cet exemple les sorties sont gérées par un module matériel dédié et donc le fonctionnement continue en mode autonome après l'injection. On peut distinguer quatre phases :

- la commande provenant du processeur (I),
- l'exécution de deux cycles d'horloge (II),
- l'injection au troisième cycle d'horloge (III),
- la reprise de l'exécution de l'application pour n cycles d'horloge (IV).

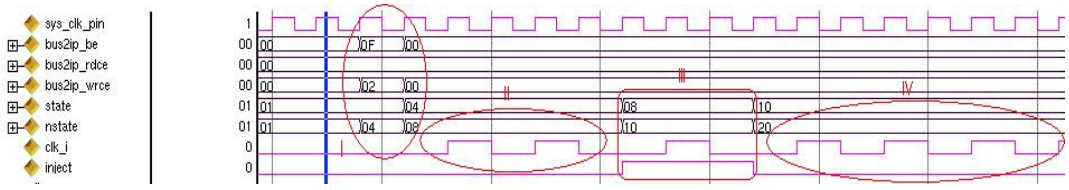

Figure 4-7 : Chronogramme de contrôle en mode autonome avec ROM émulée

Le chronogramme de la Figure 4-8 présente le fonctionnement en mode interface autonome avec une interface accédant directement en mémoire. Pour plus de simplicité le cycle d'injection n'est pas présenté. L'injection peut être effectuée directement par l'interface (comme pour le cas de la Figure

4-7) ou bien commandée par le processeur si l'observation des sorties est nécessaire au cycle d'injection. On peut distinguer quatre phases :

- la commande provenant du processeur (I),
- l'exécution de huit cycles d'horloge sans accès mémoire (les signaux *ramsn* et *romsn* sont inactifs) (II),
- la lecture en mémoire lorsqu'un signal de commande devient actif, en l'occurrence *romsn* (III),
- l'exécution du cycle d'horloge après l'accès mémoire (IV).

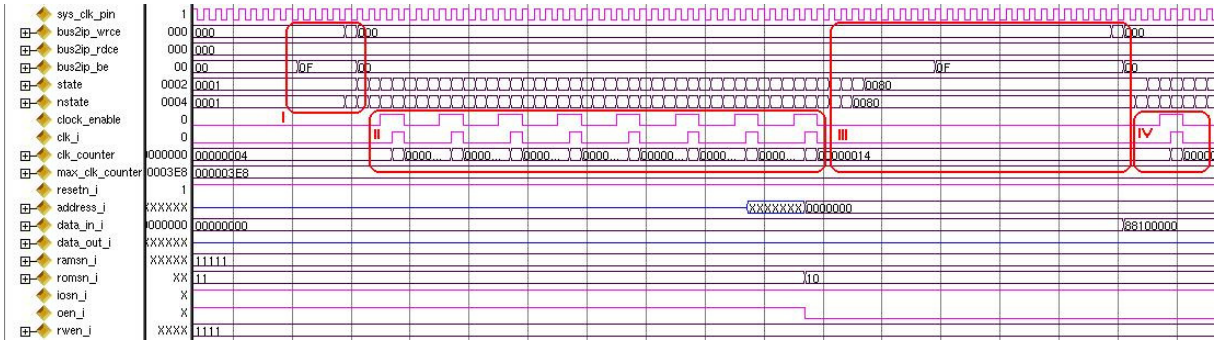

Figure 4-8 : Chronogramme de contrôle en mode autonome avec DMA

Il est évident que le mode autonome est plus rapide puisqu'il n'y a pas de transfert entre le processeur et l'interface au cours de l'expérience. La contrepartie est une contrainte plus forte sur la gestion des entrées/sorties et sur le type d'analyse possible. Afin d'accélérer les expériences pour lesquelles l'analyse est partielle il est possible de passer du mode « interface autonome » au mode « pas à pas » et inversement.

En mode autonome une interruption est générée par l'interface pour signifier que l'interface a exécuté le nombre de cycles d'horloge qui lui avait été commandé. Le processeur peut alors soit passer en mode « pas à pas » pour un ou plusieurs cycles soit lancer l'expérience suivante si celle en cours est terminée.

Si l'on s'intéresse de plus près à l'implémentation on observe que pour une interface avec DMA le nombre de cycles qui doivent être effectués en mode autonome est enregistré dans un registre qui est écrit à partir du processeur embarqué. Le mode « pas à pas » correspond dans ce cas à commander l'exécution d'un seul cycle en écrivant 1 dans le registre correspondant.

4.2.2 Contrôle optimisé des injections

❖ Registres de données d'injection avec codage

Comme nous l'avons vu sur la Figure 3-5 il est nécessaire de stocker les données relatives à chaque injection au niveau de l'interface matérielle. Ces données sont donc mémorisées à l'aide de registres avec typiquement un registre pour le cycle d'injection et un registre pour la cible d'injection.

La gestion du cycle d'injection ne diffère pas en fonction de l'approche d'injection. On peut considérer qu'un registre contient la valeur du cycle d'injection et qu'un compteur de cycles permet de contrôler le cycle d'injection.

En revanche il est possible de gérer différemment la mémorisation de la cible d'injection en fonction de la technique d'injection mise en œuvre. Pour les approches « Mask-scan » et « Time-multiplexed » le registre contenant la cible d'injection est ajouté lors de l'instrumentation et sa taille correspond au nombre de bascules ciblées. En revanche pour notre approche et pour l'approche « State-scan » ce registre est implémenté dans l'interface d'injection, ce qui offre une certaine flexibilité pour son implémentation.

Nous proposons donc de coder la cible d'injection afin de réduire le nombre de bascules nécessaires pour sa mémorisation. Par exemple pour une instrumentation avec 127 cibles, un registre de 7 bascules peut être utilisé à la place du registre de 127 bascules.

Un registre avec codage est cependant nécessaire pour chaque élément ciblé lors d'une injection. En effet après décodage un seul bit est positionné donc avec un seul registre il n'est possible d'injecter que des fautes avec une multiplicité spatiale égale à 1. Pour l'injection de MBUs avec multiplicité spatiale égale à n, n registres avec codage sont nécessaires. Il faut alors définir si le codage reste avantageux ou non. Par exemple pour 127 bascules, le codage reste avantageux en terme de cellules mémoires pour des multiplicités jusqu'à 18. Le codage devrait donc être avantageux dans la plupart des cas utiles en pratique.

Le codage de la ou des cibles d'injection requiert de la logique supplémentaire pour le décodage. Nous verrons dans le paragraphe 4.2.3 le coût total induit par codage.

- Chargement de la cible d'injection en parallèle

L'approche d'instrumentation définit la manière dont la cible d'injection est chargée, c'est-à-dire la façon d'accéder aux éléments mémoires et/ou à la logique ajoutés lors de l'instrumentation. Ce chargement peut se faire en série ou en parallèle.

Pour l'approche « State-scan » le chargement de la donnée est effectué exclusivement de manière série puisque les bascules ciblées sont connectées pour former une chaîne de scan. Bien qu'elle ne nécessite pas d'ajouter d'élément mémoire lors de l'instrumentation, cette approche est particulièrement gourmande en temps. En effet 64 cycles d'horloge sont ainsi nécessaires pour l'injection de fautes dans un registre de 64 bits.

Pour les approches « Mask-scan » et « Time-multiplexed » le registre masque peut être chargé en série ou en parallèle.

Pour notre approche le chargement se fait de façon parallèle. Lors de l'instrumentation une entrée d'injection est ajoutée à l'interface du circuit analysé pour accéder indépendamment à la logique d'injection pour chaque bascule. Ces entrées sont connectées directement au registre qui contient la cible d'injection si elle n'est pas codée ou bien aux sorties de la logique de décodage si elle est codée.

Le nombre d'entrées/sorties du FPGA n'est pas une limitation parce qu'ici les ports d'entrée pour l'injection sont intégrés à l'interface d'injection dans laquelle est instanciée le circuit analysé. En parallèle le positionnement de la cible d'injection se fait sans surcoût en terme de cycle d'horloge.

4.2.3 Coûts et bénéfices

- Surface

Le Tableau 4-5 présente les ressources requises par l'interface matérielle pour un fonctionnement « pas à pas » ou pour un fonctionnement « autonome » avec DMA. Les données présentées incluent les registres d'entrées/sorties (8), le bloc de contrôle du circuit et la partie DMA mais ne prennent pas en compte la logique pour l'injection de faute (registres, logique de décodage …). Le pourcentage d'utilisation correspond au rapport entre les ressources utilisées pour l'interface et l'ensemble des ressources disponibles dans le FPGA.

Tableau 4-5 : Coût de l'accès mémoire directe au niveau de l'interface matérielle

Fonctionnement	Pas à Pas uniquement	Autonome avec DMA	Surcoût du DMA
Nombre de FFs	242	814	+236%
Pourcentage d'utilisation	0,9%	3,0%	-
Nombre de LUTs	447	1242	+178%
Pourcentage d'utilisation	1,6%	4,5%	-
Nombre de Slices	276	827	+200%
Pourcentage d'utilisation	2,0%	6,0%	-
Fréquence max(MHz)	283,1	122,1	-56,9%

Il apparaît que l'implémentation du bloc de contrôle de la mémoire a un coût important lorsque l'on compare les ressources utilisées pour l'interface avec et sans DMA. Cependant, si l'on ramène ces valeurs à l'ensemble des ressources disponibles le surcoût parait acceptable. En ce qui concerne la fréquence maximale de fonctionnement le coût est important mais la fréquence réelle d'utilisation est celle du bus auquel est connectée l'interface (typiquement 100MHz) donc la solution avec DMA atteint une fréquence suffisante.

Le Tableau 4-6 présente le surcoût en terme de logique combinatoire et le gain en terme de cellules mémoires du codage de la cible d'injection.

Comme nous l'avions prévu le codage de la cible d'injection induit une baisse du nombre de bascules et une augmentation de la logique combinatoire donc des LUTs. Au final le nombre de *Slices* nécessaires est plus faible avec codage. On note également que le choix du codage ou non n'a pas d'impact sur la fréquence de fonctionnement de l'interface d'injection. La mise en œuvre du codage pour l'injection de SEUs est donc un avantage.

Tableau 4-6 : Comparaison des ressources nécessaires avec et sans codage

Caractéristiques	Multiplieur 16bits			i8051		
Codage	**Sans**	**Avec**	**Diff.**	**Sans**	**Avec**	**Diff.**
N_{FF} ciblées	74			256		
Nb de FFs	206	139	-32,5%	593	346	-41,7%
Pourcentage d'util.	0,8%	0,5%		2,2%	1,3%	
Nb 4-input LUTs	115	190	+65,2%	277	310	+11,9%
Pourcentage d'util.	0,4%	0,7%		1,0%	1,1%	
Nb Slices	151	147	-2,6%	446	320	-28,3%
Pourcentage d'util.	1,1%	1,1%		3,3%	2,3%	
Equivalent nb de portes	2356	2270	-3,7%	6427	6185	-3,8%
Fréquence max(MHz)	283,1	283,1	0%	283,1	283,1	0%

❖ Temps d'exécution

Il est important de signaler que l'accélération apportée par le mode autonome dépend d'un certain nombre de paramètres. En tout premier lieu l'accélération dépend du nombre d'accès mémoire effectués pendant le déroulement de l'application. Le fonctionnement autonome est d'autant plus rapide que les accès mémoire sont peu nombreux comme cela est illustré sur la Figure 4-8. En effet un cycle d'horloge du circuit étudié avec accès mémoire prend approximativement 6 fois plus de temps qu'un cycle sans accès mémoire. En revanche en mode « pas à pas » il y a communication entre l'interface et le processeur embarqué à chaque cycle de l'exécution. Donc le temps d'exécution ne dépend que du nombre de cycles. D'autre part lors des campagnes d'injection il faut prendre en compte le mode de fonctionnement tout au long de chaque expérience. Par exemple si l'expérience se déroule en mode autonome jusqu'au cycle d'injection puis en mode pas à pas ensuite, alors l'accélération est moindre.

En terme de nombre de cycles nécessaire pour l'injection d'une faute, le chargement parallèle est un avantage par rapport au chargement série puisqu'il se fait sans surcoût. Cette donnée doit cependant être nuancée si l'on considère une expérience complète et a fortiori si l'on considère une campagne de N expériences. Considérons le multiplieur 16 bits exécutant une multiplication (184 cycles), le microcontrôleur i8051 exécutant un crible d'Eratosthène (52185 cycles). Pour chaque cas, en fonction du nombre de bascules ciblées, le temps de chargement de la cible d'injection a une importance différente. Cette différence est matérialisée sur la Figure 4-9. Pour le multiplieur, lorsque le nombre de cibles augmente, le coût de l'injection augmente très vite parce qu'il n'y a pas beaucoup de bascules dans le multiplieur et parce que l'application est courte. Ce n'est plus du tout le cas pour l'autre

exemple. Si l'on injecte des fautes dans toutes les bascules du microcontrôleur, alors le temps nécessaire au chargement ne dépasse pas 3% du temps total de l'expérience.

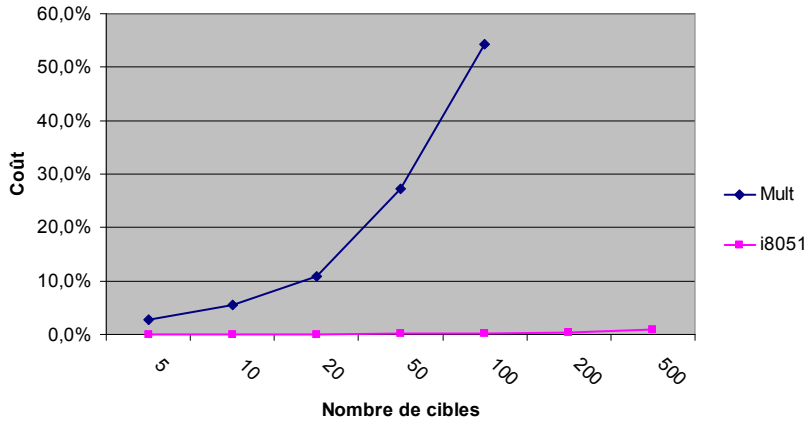

Figure 4-9 : Coût du chargement des données d'injection en série

4.3 Comparaison avec les approches existantes

Nous avons vu ci-dessus les coûts théoriques de différentes techniques d'instrumentation (Tableau 4-4) ainsi que le coût potentiel du chargement série. Différentes approches d'instrumentation ont été mises en œuvre et évaluées sur le circuit b12 (détecteur de séquence) qui est un benchmark de ITC'99 [ITC-99]. Ceci nous permet de comparer les approches existantes à celle que nous proposons pour l'analyse d'un circuit donné.

Les techniques d'instrumentation existantes sont celles présentées dans la partie 2.3.2. Ces techniques ainsi que les résultats de campagnes d'injection de fautes sont proposées dans [LOPE-05].

De notre côté nous avons en outre effectué l'instrumentation à base de LUTs du circuit b12 qui comprend 119 bascules après synthèse avec Leonardo et nous avons appliqué la technique de codage du registre de cible d'injection. Le Tableau 4-7 reprend les coûts induits par l'instrumentation pour les différentes techniques d'injection dont celle que nous proposons appelée « Etendue LUTs ». Les caractéristiques après synthèse ne sont pas reprises ici car celles-ci diffèrent légèrement en termes de LUTs à cause de l'architecture FPGA qui n'est pas la même (Virtex2Pro pour « Etendue LUTs » et Virtex-2000E pour les trois autres).

Tableau 4-7 : Comparaison pratique des coûts des techniques d'instrumentation existantes

	Etendue LUTs	Mask-scan	State-scan	Time-multiplexed
Bascules	+6%	+100%	+100%	+300%
LUTs	+69%	+86%	+72%	+389%
Slices	+92%	NC	NC	NC

Le codage de la cible d'injection réduit le coût en nombre de bascules nécessaires à +6% alors que pour les autres techniques le coût est supérieur ou égal à 100%. La logique supplémentaire pour

l'injection et pour le décodage reste légèrement moins importante par rapport aux autres techniques. En terme de matériel supplémentaire ces résultats confirment que notre approche (instrumentation + codage) entraîne un surcoût moins important que les approches existantes.

En ce qui concerne la durée de campagnes d'injection, le Tableau 4-8 présente les données obtenues avec notre environnement et reprend celles de [LOPE-05].

Tableau 4-8 : Comparaison pratique des durées des campagnes d'injection

	Etendue LUTs	**Mask-scan**	**State-scan**	**Time-multiplexed**
Fréquence horloge (MHz)	100	25	25	25
Durée de la campagne (ms)	20	108,7	142	80,80
Durée d'une expérience (µs)	1,1	5,7	7,5	4,2

Les valeurs présentées sont relatives à une campagne d'injection exhaustive où des *bit-flips* simples sont injectés dans chaque bascule du circuit et à chaque cycle d'horloge de l'application qui comprend au total 160 cycles. Comme il y a 119 bascules dans le circuit, le nombre d'expériences est de 19040. La classification est effectuée au niveau matériel par comparaison des sorties avec les sorties attendues lors de chaque expérience.

A fréquence équivalente notre approche est légèrement plus lente que la technique « Time-Multiplexed » mais plus rapide que les approches « Mask-scan » et surtout « State-scan ». Avec le b12 on se retrouve dans un cas où le nombre de cibles est important par rapport au nombre de cycles de l'application. Le chargement parallèle permet alors d'obtenir une accélération conséquente. Avec notre approche le gain de temps par expérience est approximativement de 60% ce qui est cohérent avec la tendance présentée sur la Figure 4-9.

On remarquera cependant que les temps de campagnes pour cet exemple ne sont pas réellement significatifs pour conclure qu'une technique particulière est plus intéressante qu'une autre. De la même façon l'optimisation de la répartition des tâches pour l'analyse de ce circuit permet une accélération qui au final n'est pas forcément indispensable. En effet la même campagne d'injection avec génération des vecteurs d'entrée et analyse au niveau du processeur embarqué ne prend que 33 secondes. Il est fort probable que l'accélération apportée par un bloc de classification matérielle par exemple ne compense pas le temps nécessaire à son développement si court soit-il. D'autant plus si l'on considère le temps nécessaire à la génération de l'environnement qui est proche de 20 minutes.

4.4 Conclusion

Nous avons présenté dans ce chapitre trois techniques innovantes qui ont pour objectif d'accélérer les campagnes d'injection et de réduire la logique nécessaire au contrôle des injections. Nous proposons ainsi d'optimiser l'instrumentation du circuit, le contrôle de l'interface et le chargement des données d'injection pour l'émulation de faute. En comparaison avec les approches existantes les optimisations en terme de logique consommée et de temps de campagne sont réelles.

Cependant nous avons remarqué que pour un FPGA donné les gains attendus dépendent des caractéristiques du circuit analysé. D'autre part il apparaît que les bénéfices apportés par une optimisation de la répartition des tâches ne sont pas forcément proportionnels au temps de développement potentiellement nécessaire.

Cela nous a poussé à étudier plus en détails les performances de l'environnement. L'objectif est d'évaluer a priori le temps nécessaire pour réaliser une campagne d'injection en fonction des caractéristiques du circuit, des paramètres de l'environnement et de la répartition des tâches choisie par le concepteur. Cette partie est présentée dans le chapitre suivant. Nous verrons par ailleurs au chapitre 6 des résultats sur des exemples de systèmes plus significatifs.

5 Evaluation prédictive des performances

Nous avons vu dans le chapitre 3 qu'un grand nombre de possibilités sont offertes à l'utilisateur pour son analyse. Le chapitre 4 portait sur la présentation de techniques visant à améliorer les performances d'un environnement d'injection à base de prototypage avec instrumentation. L'objectif de ce chapitre est d'évaluer les techniques proposées et les performances de l'environnement en fonction des choix de l'utilisateur.

5.1 Présentation

L'évaluation que nous proposons se base sur les différentes configurations possibles et dépend des paramètres qui caractérisent l'environnement. Les performances obtenues dépendent également du système analysé (nombre d'entrées/sorties, nombre de cycles …) et des cibles choisies (nombre de bits). Comme nous l'avons dit un des objectifs de cette évaluation est de valider les techniques que nous avons proposées dans le chapitre précédent. L'évaluation dépend donc également des techniques liées à l'injection de fautes. En résumé l'évaluation prend donc en considération 3 grands types de données :

- les paramètres de l'environnement,
- les caractéristiques du système analysé,
- les techniques employées (instrumentation, contrôle des injections).

Nous introduisons tout d'abord les paramètres $t_{proc2pc}$, $t_{proc2ram}$, $t_{proc2hw}$, t_{hw2ram}, t_{pc2ram} comme les temps de transferts entre les différents éléments de l'environnement d'analyse (Figure 5-1) soient le processeur (proc), le PC hôte (PC), la mémoire RAM sur la plateforme (ram) et le périphérique d'injection (hw).

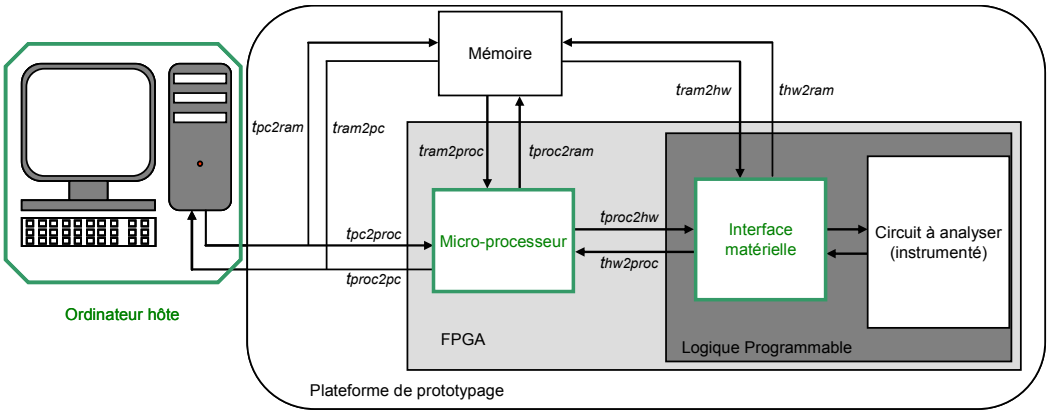

Figure 5-1 : Paramètres de l'environnement liés au transfert de données

T_{sys} est la période de l'horloge du système (processeur embarqué, bus et périphériques). L'horloge du système n'est cependant pas l'horloge qui est appliquée au circuit analysé puisque cette dernière est contrôlée par l'interface.

Les performances du processeur embarqué sont très importantes et sans doute les plus difficiles à évaluer théoriquement, notamment lorsque celui-ci est utilisé pour générer des données de façon aléatoire. Les paramètres t_{algo_pc} et $t_{algo\_\mu p}$ sont les durées requises pour générer un mot de données par le PC hôte et le microprocesseur embarqué respectivement et selon un algorithme défini. La génération d'un signal d'interruption est utilisée comme une sorte d'acquittement du périphérique d'injection à l'intention du processeur embarqué. Le périphérique génère une interruption lorsque l'exécution que le processeur lui avait commandée est terminée. Le paramètre t_{soft} est le temps que prend la routine d'interruption lorsque le processeur reçoit cette interruption. Dans le cas d'une analyse logicielle le processeur compare les sorties après injection aux sorties de référence en une durée représentée par t_{comp}.

Tout fonctionnement de l'interface intervient après une commande de la part du processeur embarqué. Celui-ci écrit alors dans un certain nombre de registres ($N_{cmd} \geq 1$) en fonction de la taille des données de commande. Le parcours complet de la machine à états inclue dans la partie contrôle de l'interface se fait en un nombre de cycles d'horloge défini par $N_{control}$. On considère également N_{load} le nombre de cycles système nécessaires au chargement d'un bit lors du chargement en série de la cible d'injection. Ceci est dû au fait que l'horloge du circuit analysé n'est pas l'horloge du système. Avec certains modes de fonctionnement (ex : mode autonome avec ROM émulé + classification matérielle) et en dehors des phases d'injection, l'interface ne gère que l'horloge du circuit. $R_{freq\_circuit}$ est le rapport entre la fréquence du système et la fréquence maximale appliquée au circuit. Une illustration est disponible sur la Figure 4-7 où l'on voit le cas où $R_{freq\_circuit} = 2$. Ces trois paramètres dépendent des détails de l'implémentation de l'interface matérielle.

Les performances obtenues dépendent également des caractéristiques du circuit analysé : N_{FF} le nombre de bascules du circuit, N_{PI} et N_{PO} les nombre d'entrées et de sorties primaires. Il faut ensuite considérer les données liées à l'instrumentation : N_{cibles} le nombre de bascules ciblées, N_{II} et N_{OO} les nombres d'entrées d'injection et de sorties d'observation. Enfin le nombre de cycles d'horloge nécessaires pour l'exécution de l'application (N_{cycles}) a son importance.

Dans certains cas il est nécessaire de prendre en compte non pas le nombre de cycles de l'application mais le nombre de cycles pour lesquels il est nécessaire de récupérer les valeurs des entrées. Ce nombre de cycles est représenté par le paramètre $N_{accès}$ qui symbolise le nombre d'accès en mémoire. Lorsque l'application est un programme à exécuter le paramètre N_{octets_prog} est utilisé pour représenter le nombre d'octets de celui-ci.

Dans la suite de ce document nous illustrons l'évaluation des performances avec quelques applications numériques. L'environnement est constitué de la plateforme de prototypage Xilinx ML310 avec un FPGA Virtex2Pro30. Le processeur embarqué est un PowerPC405 et la carte est reliée au PC hôte par une connexion série.

Nous partons du postulat que $t_{proc2pc} = t_{pc2proc}$, $t_{hw2ram} = t_{ram2hw}$ etc. Nous considérons que tous les transferts se font sur 32 bits. Ceci inclut les transferts entre l'ordinateur hôte et la plateforme et les transferts entre le processeur et les périphériques (mémoire, périphérique d'injection…).

Les paramètres t_{algo_pc} et $t_{algo\_\mu p}$ sont les durées requises pour générer un mot de 32 bits. On considère que $t_{algo\_\mu p}=10*t_{algo\_pc}$ puisque la fréquence de fonctionnement d'un processeur de PC actuelle est de l'ordre de 3 GHz et celle des derniers FPGA est voisine de 400 MHz (450 MHz pour le PowerPC embarqué dans un Virtex-4). Cette approximation est grossière mais pour être précis il faudrait prendre en compte un nombre de paramètres très important (architecture des processeurs, utilisation de caches…). Le paramètre qui doit être évalué avec le plus de précision est $t_{algo\_\mu p}$ car il est le plus mis en jeu.

Les valeurs numériques des paramètres liés à l'environnement ont été obtenues par simulation et l'aide d'un compteur logiciel. Les deux permettent de déterminer le nombre de cycles d'horloge pour l'exécution d'une action (transfert, comparaison de données etc.).

Les paramètres, leurs significations et leurs valeurs numériques sont régroupés dans le Tableau 5-1. Ce tableau comprend les paramètres liés à l'environnement et à l'implémentation de l'interface matérielle.

Tableau 5-1 : Evaluation prédictive - Paramètres de l'environnement (significations et valeurs numériques)

Paramètre	Signification	Valeur (µs)
$t_{proc2pc}$	Durée d'un transfert entre le processeur embarqué et le PC hôte	80
$t_{proc2ram}$	Durée d'un transfert entre le processeur embarqué et la mémoire embarqué	2
$t_{proc2hw}$	Durée d'un transfert entre le processeur embarqué et l'interface matérielle	2
t_{pc2ram}	Durée d'un transfert entre le PC hôte et la mémoire embarqué	60
t_{hw2ram}	Durée d'un transfert entre l'interface matérielle et la mémoire embarqué	0,3
t_{soft}	Durée d'exécution de la routine d'interruption par le processeur embarqué	10
t_{algo_pc}	Durée requise pour générer un mot de donnée par le PC hôte	5
$t_{algo\_\mu p}$	Durée requise pour générer un mot de donnée par le processeur embarqué	50
t_{comp}	Durée de comparaison de 2 mots de donnée par le processeur embarqué	1
$Tsys$	Période d'horloge du système	0,01
$N_{control}$	Nombre de cycles d'horloge pour le parcours de la partie contrôle de l'interface matérielle	6
N_{load}	Nombre de cycles d'horloge pour le chargement d'un bit lors du chargement en série de la cible d'injection	2
$R_{freq\_circuit}$	Rapport entre la fréquence du système et la fréquence maximale appliquée au circuit	2

En ce qui concerne les paramètres relatifs aux fautes, M_{temp} est le nombre de fautes injectées lors d'une même expérience. N_{bit} est le nombre de bits du « mot cible » à générer. Il dépend du nombre de cibles N_{cibles}, de la technique d'instrumentation et du codage ou non des données. Ces points ont été développés dans le chapitre 4 (section 4.2.2).

Les applications numériques sont obtenues pour deux circuits : un multiplieur 16 bits et le microcontrôleur i8051 dont les principales caractéristiques sont données dans le Tableau 5-2. La version du microcontrôleur est celle sans programme mémorisé en ROM.

Tableau 5-2 : Evaluation prédictive - Paramètres liés aux circuits et aux fautes injectées

Paramètre	Signification	Multiplieur 16 bits	Microcontrôleur i8051
N_{FF}	Nombre de bascules	71	1343
N_{PI}	Nombre d'entrées primaires	17	32
N_{PO}	Nombre de sorties primaires	18	32
N_{cycles}	Nombre de cycles de l'application	185	52185
N_{cmd}	Nombre d'écritures pour la commande de l'interface	1	1
N_{cibles}	Nombre de bascules ciblées	-	-
N_{II}	Nombre d'entrées d'injection	-	-
N_{OO}	Nombre de sorties d'observation	-	-
$N_{accès}$	Nombre d'accès en mémoire lors de l'application	-	-
N_{octets_prog}	Nombre d'octets d'un programme logiciel	-	-
M_{temp}	Multiplicité temporelle	-	-
N_{bit}	Nombre de bits du « mot cible » à générer	-	-

Les paramètres qui ne sont pas définis dans le Tableau 5-2 le sont dans la suite de ce chapitre ou dans le chapitre suivant en fonction des cas d'étude et des configurations.

5.2 Evaluation par tâche

Pour faciliter le travail nous considérons que chaque campagne d'injections comprend 7 tâches. Aux 3 tâches qui peuvent être attribuées aux différents niveaux de l'environnement s'ajoutent 4 tâches dont le niveau d'exécution est fixé. Ces 7 tâches sont définies dans le Tableau 5-3.

Toutes les évaluations sont faites pour les pires cas, c'est-à-dire que chaque expérience et chaque analyse est supposée se poursuivre jusqu'à son terme indépendamment d'une éventuelle détection en cours d'expérience qui permettrait d'y mettre un terme afin de gagner du temps.

Tableau 5-3 : Tâches à effectuer lors d'une campagne

Tâche	Commentaires
Génération des entrées primaires	Comprend la génération et les transferts jusqu'à l'écriture dans les registres d'entrée de l'interface.
Génération de la liste de fautes	Comprend la génération et les transferts jusqu'à l'écriture dans les registres d'injection de l'interface.
Déroulement de l'expérience de référence	Comprend l'exécution de l'expérience et la sauvegarde de la trace de référence. Dépend de l'implémentation d'un bloc matériel pour l'analyse des sorties.
Déroulement des expériences	Intègre le nombre de cycles système nécessaires pour effectuer un cycle circuit en ignorant tous les accès mémoires. Dépend du mode de fonctionnement
Récupération des sorties du périphérique	Comprend la récupération des sorties du périphérique d'injection (les sorties du circuit ou sorties du bloc de classification matérielle).
Analyse des résultats	Comprend l'analyse des sorties du circuit au niveau du processeur embarqué.
Récupération des résultats de l'analyse	Comprend le transfert des résultats de la carte vers le PC hôte.

5.2.1 Génération des entrées primaires et de la liste de fautes

Pour la génération de données nous appelons « configuration » le niveau auquel elle est effectuée et la façon dont sont gérées ces données au cours des expériences. Le Tableau 5-4 définit les configurations possibles.

Tableau 5-4 : Définition des configurations

Configuration	Définition
PC hôte via RAM et processeur	Données générées par le PC hôte puis mémorisées en RAM. Pour chaque expérience le processeur va lire ces données en RAM puis les envoyer au périphérique d'injection
PC hôte via RAM	Données générées par le PC hôte puis mémorisées en RAM. Pour chaque expérience le périphérique lit directement les données en RAM
Processeur embarqué via RAM et processeur	Données générées par le Processeur embarqué puis mémorisées en RAM. Pour chaque expérience le processeur va lire ces données en RAM puis les envoyer au périphérique d'injection
Processeur embarqué via RAM	Données générées par le Processeur embarqué puis mémorisées en RAM. Pour chaque expérience le périphérique lit directement les données en RAM
Processeur embarqué	Pour chaque expérience les données sont générées par le Processeur embarqué et envoyées au périphérique d'injection.
Interface matérielle	Données générées et positionnées par l'interface matérielle

Dans les Tableau 5-5 et Tableau 5-6 figurent les équations relatives au calcul prédictif des temps de génération des vecteurs d'entrée et dans le Tableau 5-7 figurent celles pour la génération de la liste de fautes. Ces temps sont fonction des caractéristiques du circuit, du niveau d'exécution et de la configuration choisie. Les équations présentées incluent la génération et l'écriture dans les registres d'entrée de l'interface.

Nous avons vu que les vecteurs d'entrées peuvent correspondre à la séquence des valeurs à appliquer aux entrées à chaque cycle (Tableau 5-5) ou bien à un programme à exécuter par le circuit analysé, typiquement s'il s'agit d'un processeur (Tableau 5-6).

Pour les quatre premières configurations du Tableau 5-5 les vecteurs d'entrée sont générés puis sauvegardés en RAM avant d'être écrits dans les registres d'entrée du périphérique d'injection lors de chaque expérience. En configuration « Processeur embarqué » celui-ci génère les vecteurs d'entrée pour chaque cycle de chaque expérience ce qui explique la mise en facteur du paramètre $t_{algo\_\mu p}$. Lorsque les vecteurs d'entrée sont générés par l'interface elle-même la génération se fait en parallèle de l'exécution mais les entrées doivent être générées pour le premier cycle d'horloge de l'application avant celui-ci donc cela prend un cycle pour chaque expérience. On majore la durée de génération des vecteurs d'entrées par $N_{control}$ en considérant que dans le pire des cas, cela prend autant de temps que de parcourir tous les états.

Tableau 5-5 : Evaluation prédictive - Génération des vecteurs d'entrée pour circuit avec entrées définies à chaque cycle

Configuration	$t_{entrées}$
PC hôte via RAM et processeur	$N_{cycles} * \frac{N_{PI}}{32} * [t_{algo\_pc} + t_{pc2ram} + (N_{expe}+1) * (t_{ram2proc} + t_{proc2hw})]$
PC hôte via RAM	$N_{cycles} * \frac{N_{PI}}{32} * [t_{algo\_pc} + t_{pc2ram} + (N_{expe}+1) * t_{ram2hw}]$
Processeur embarqué via RAM et processeur	$N_{cycles} * \frac{N_{PI}}{32} * [t_{algo\_\mu p} + t_{proc2ram} + (N_{expe}+1) * (t_{ram2proc} + t_{proc2hw})]$
Processeur embarqué via RAM	$N_{cycles} * \frac{N_{PI}}{32} * [t_{algo\_\mu p} + t_{proc2ram} + (N_{expe}+1) * t_{ram2hw}]$
Processeur embarqué	$N_{cycles} * \frac{N_{PI}}{32} * (N_{expe}+1) * [t_{algo\_\mu p} + t_{proc2hw}]$
Interface matérielle	$N_{expe} * N_{control} * T_{sys}$

Le facteur $N_{expe}+1$ provient du fait que dans la plupart des cas il faut effectuer une expérience sans injection afin de récupérer la trace de référence. Cette experience supplémentaire n'a plus lieu d'être si un bloc matériel de classification est implémenté. Son influence dans les équations reste cependant négligeable si l'on considère des campagnes un tant soit peu significatives avec un nombre d'expériences supérieur à 1000.

Le calcul du nombre de transferts se fait en divisant le nombre de bits à transférer par 32. Par soucis de simplicité on laisse dans les équations l'expression $N_x/32$ (par exemple $N_{PI}/32$) mais pour les applications numériques nous prenons en compte la valeur entière de $(N_x-1)/32+1$.

Dans le cas d'un programme à exécuter celui-ci est obligatoirement généré au niveau du PC hôte donc seules les configurations « PC hôte via RAM et processeur » et « PC hôte via RAM » subsistent dans le Tableau 5-6.

Tableau 5-6 : Evaluation prédictive - Génération des vecteurs d'entrées pour circuit exécutant un programme

Configuration	$t_{entrées}$
PC hôte via RAM et processeur	$N_{octets\_prog}/4 * t_{pc2ram} + N_{accès} * N_{PI}/32 * [(N_{expe}+1) * (t_{ram2proc}+t_{proc2hw})]$
PC hôte via RAM	$N_{octets\_prog}/4 * t_{pc2ram} + N_{accès} * N_{PI}/32 * [(N_{expe}+1) * t_{ram2hw}]$

Dans un premier temps le programme (N_{octets_prog}) est téléchargé en mémoire. Ensuite pour chaque expérience et en fonction du nombre d'accès requis ($N_{accès}$) les données sont lues en mémoire via le processeur embarqué ou directement par l'interface.

La génération de la liste de fautes, dont les équations sont présentées dans le Tableau 5-7, est basée sur le même principe que la génération des vecteurs d'entrée. L'unique différence est qu'elle dépend des données relatives aux fautes à injecter M_{temp} et N_{bit}. M_{temp} est le nombre de fautes injectées lors d'une même expérience donc le temps de génération lui est directement proportionnel. Et on rappelle que N_{bit} est le nombre de bits du « mot cible » à générer. Le « mot cycle » peut être codé dans tous les cas et le nombre de bits nécessaires pour coder le cycle d'injection est égal au nombre de bits nécessaire pour coder la valeur maximale du cycle d'injection soit N_{cycles}. Pour une génération matérielle la ou les fautes doivent être définies avant le début de l'expérience au cas où l'injection au premier cycle d'horloge est requise. On retombe alors sur le cas des vecteurs d'entrée décrit ci-dessus.

Tableau 5-7 : Evaluation prédictive - Génération de la liste de fautes

Configuration	t_{liste_fautes}
PC hôte via RAM et processeur	$N_{expe} * M_{temp} * \left(\frac{N_{bit}}{32} + \frac{\ln(N_{cycle})/\ln(2)}{32}\right) * [t_{algo\_pc} + t_{pc2ram} + t_{ram2proc} + t_{proc2hw}]$
PC hôte via RAM	$N_{expe} * M_{temp} * \left(\frac{N_{bit}}{32} + \frac{\ln(N_{cycle})/\ln(2)}{32}\right) * [t_{algo\_pc} + t_{pc2ram} + t_{ram2hw}]$
Processeur embarqué via RAM et processeur	$N_{expe} * M_{temp} * \left(\frac{N_{bit}}{32} + \frac{\ln(N_{cycle})/\ln(2)}{32}\right) * [t_{algo\_\mu p} + t_{proc2ram} + t_{ram2proc} + t_{proc2hw}]$
Processeur embarqué via RAM	$N_{expe} * M_{temp} * \left(\frac{N_{bit}}{32} + \frac{\ln(N_{cycle})/\ln(2)}{32}\right) * [t_{algo\_\mu p} + t_{proc2ram} + t_{ram2hw}]$
Processeur embarqué	$N_{expe} * M_{temp} * \left(\frac{N_{bit}}{32} + \frac{\ln(N_{cycle})/\ln(2)}{32}\right) * [t_{algo\_\mu p} + t_{proc2hw}]$
Interface matérielle	$N_{expe} * M_{temp} * N_{control} * T_{sys}$

Pour les applications numériques présentées sur les Figure 5-2 et Figure 5-3 on considère une campagne d'injection avec des multiplicités spatiale et temporelle égales à 1 i.e. injection d'un *bit-flip* simple par expérience. Nous considérons que N_{bit} est égal au nombre de cibles N_{cibles} donc sans codage - l'influence du codage est illustrée avec la Figure 5-4 - et nous prenons pour illustration l'injection de fautes dans 20% des bascules du circuit considéré, soit $N_{bit} = N_{cibles} = N_{FF} * 0,2$. Avec ces spécifications le nombre d'expériences est donc le nombre de cycles multiplié par le nombre de cibles soit $N_{expe} = N_{cycles} * N_{cibles} = N_{cycles} * N_{FF} * 0,2$. Comme le nombre d'expériences ainsi obtenu pour le cas du microcontrôleur est relativement important, il est possible de considérer une campagne pendant laquelle 1% des fautes est injecté. En revanche la campagne d'injection est exhaustive pour le multiplieur. Les valeurs obtenues sont présentées dans le Tableau 5-8.

Tableau 5-8 : Evaluation prédictive - Nombre d'expériences considéré

	Multiplieur 16 bits	**Microcontrôleur i8051**
Type de campagne	Exhaustive	1%
N_{expe}	2 590	139 855

La Figure 5-2 donne les temps de génération des vecteurs d'entrée obtenus pour les deux exemples de circuit.

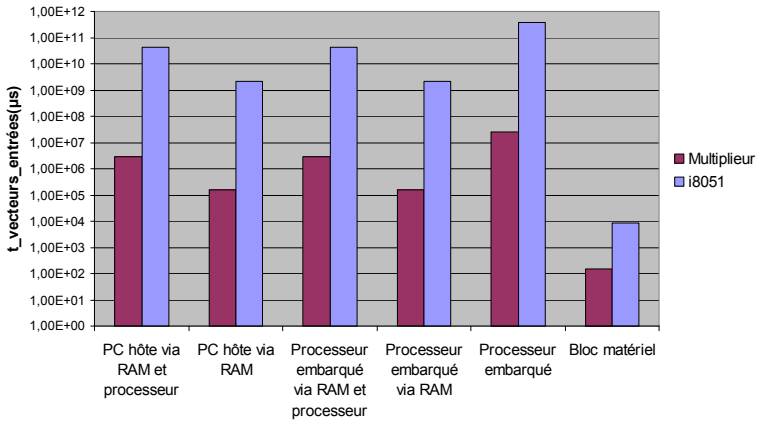

Figure 5-2 : Temps de génération des vecteurs d'entrées

On observe que les temps de génération au niveau PC et au niveau processeur embarqué sont du même ordre de grandeur. Les valeurs pour les configurations « PC hôte via RAM et processeur » et « processeur embarqué hôte via RAM et processeur », qui sont les plus simples à mettre en œuvre, sont particulièrement proches. Cela peut sembler surprenant puisque nous avons dit que les transferts du PC vers la plateforme étaient très gourmands en temps. Mais le transfert des vecteurs d'entrée n'est effectué qu'une seule fois pour toute la campagne et les nombres d'expériences et de cycles sont relativement importants ; respectivement 2590 et 185 pour le multiplieur et surtout près de 14 10^4 et 52185 pour le microcontrôleur. Dans ce cas la durée du transfert du PC hôte vers la RAM perd de son importance par rapport aux transferts de la RAM vers le périphérique d'injection à chaque cycle

d'horloge. On constate d'ailleurs qu'un périphérique d'injection avec accès mémoire direct permet d'accélérer la génération des vecteurs d'un facteur proche de 10. Enfin on remarque la rapidité de la génération matérielle même si nous avons vu que celle-ci n'est pas très pertinente pour les vecteurs d'entrée.

La Figure 5-3 illustre les temps de génération de la liste de fautes calculés pour le multiplieur et le microcontrôleur. A nouveau le processeur embarqué ne permet pas une accélération substantielle par rapport au PC hôte. Mais cette fois l'explication se situe au niveau des fréquences de fonctionnement et des temps de mémorisation des deux niveaux d'exécution. En effet le bénéfice apporté par la durée de mémorisation qui est dix fois plus courte au niveau processeur embarqué est presque complètement compensé par une fréquence de fonctionnement dix fois plus faible. La génération au niveau matériel permet toujours une accélération. Cette solution peut être envisagée même si la mise en œuvre d'algorithmes complexes est plus difficile qu'au niveau du processeur.

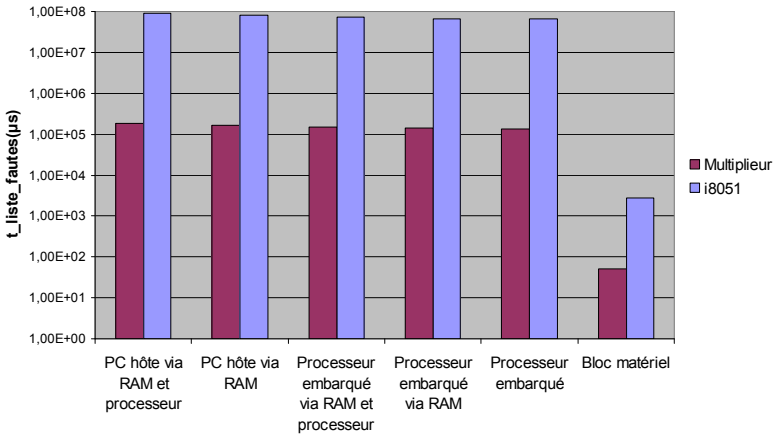

Figure 5-3 : Temps de génération de la liste de fautes

Les valeurs présentées ci-dessus ont été calculées pour une certaine valeur de N_{bit} et donc ne permettent pas d'apprécier l'utilité du codage du « mot cible ». On rappelle que le codage du « mot cible » n'est pas possible pour toutes les approches d'instrumentation. L'approche « Mask-scan » en particulier ne le permet pas puisque le masque est intégré au processus d'instrumentation et qu'il contient un bit par bascule ciblée.

Pour illustrer l'influence du codage sur le temps de génération de la liste de fautes, prenons l'exemple du microcontrôleur i8051 qui possède un nombre significatif de bascules dans sa version originale (1343 bascules). La configuration choisie est la configuration « Processeur embarqué via RAM et processeur » qui semble être la plus flexible et la plus simple à mettre en œuvre.

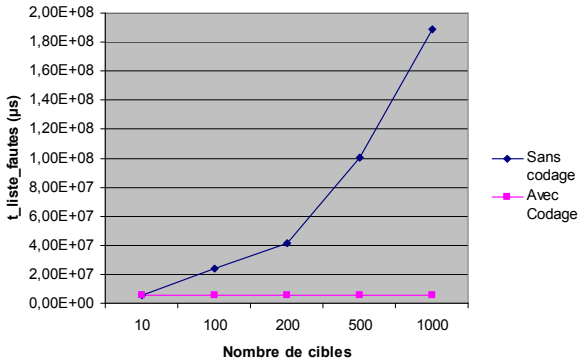

Figure 5-4 : Influence du nombre de cibles et du codage sur le temps de génération de la liste de fautes

Le graphe de la Figure 5-4 montre bien que le codage a d'autant plus d'importance que le nombre de cibles est important. Le nombre de fautes injectées est fixé à 100000 et reste constant. Lorsque seulement 10 bascules sont ciblées les deux techniques se valent et affichent un temps de génération d'environ 380 microsecondes. Pour 500 bascules ciblées, le temps de génération avec codage ne varie quasiment pas alors que celui sans codage a été multiplié par 17. Le choix du codage et de la technique d'instrumentation a donc un impact non négligeable sur la durée de la génération de la liste de fautes.

5.2.2 Déroulement de l'expérience de référence

L'expérience de référence diffère des expériences avec injection par le fait qu'il faut récupérer les sorties du circuit analysé à chaque cycle d'horloge. Le mode « pas à pas » est donc le seul mode de fonctionnement compatible avec l'expérience sans injection. Si l'analyse est effectuée par un bloc matériel alors une expérience de référence est inutile. Les équations présentées dans le Tableau 5-9 permettent de calculer la durée de cette expérience.

Tableau 5-9 : Evaluation prédictive - Expérience de référence

Type d'analyse	t_{expe_ref}
Classification avec bloc matériel	0
Classification par comparaison des sorties Génération des chemins de propagation d'erreurs	$N_{cycle} * \left(N_{cmd} * t_{proc2HW} + N_{control} * T_{sys} + t_{soft} + \frac{(N_{PO} + N_{OO})}{32} * (t_{hw2proc} + t_{proc2ram}) \right)$

La durée de l'expérience de référence est directement proportionnelle au nombre de cycles de l'application. Pour chaque cycle le processeur commande à l'interface d'exécuter un cycle (N_{cmd}*$t_{proc2HW}$), ce que l'interface réalise en un certain nombre de cycles ($N_{control}$) qui dépend de son implémentation avant d'envoyer une interruption au processeur (t_{soft}). Ce dernier lit alors les registres de sortie du périphérique d'injection, mémorise les valeurs lues puis passe au cycle suivant.

5.2.3 Déroulement des expériences

Contrairement à l'expérience de référence qui est exclusivement en mode « pas à pas », chaque expérience avec injection peut être effectuée en combinant mode « pas à pas » et mode « interface autonome ». Le nombre de combinaisons possibles est trop important pour que chacune soit étudiée ici. De plus le nombre de cycles qui sont effectués dans tel ou tel mode de fonctionnement dépend du ou des cycles d'injection de chaque expérience.

Afin de conserver une évaluation lisible et correspondant au plus grand nombre de configurations possibles nous introduisons deux paramètres nouveaux : N_{pas} et N_{auto} le nombre de cycles en mode « pas à pas » et le nombre de cycles en mode « interface autonome ». Pour chaque expérience la somme des deux est égale au nombre de cycles total : $N_{pas}+N_{auto}=N_{cycles}$. Pour le calcul il faut définir la valeur de chaque paramètre pour l'ensemble des expériences.

Une des solutions est de considérer les deux cas extrêmes et le cas médian soit :

- *Pas à Pas uniquement :* $N_{pas} = N_{cycles}$ et $N_{auto} = 0$,
- *Autonome uniquement :* $N_{pas} = 0$ et $N_{auto} = N_{cycles}$,
- *Mixte :* $N_{pas} = N_{auto} = N_{cycles}/2$.

Le premier cas correspond à un fonctionnement intégralement en mode « pas à pas » qui est nécessaire si les vecteurs d'entrée doivent être fournis par le processeur pour chaque cycle. La seconde possibilité correspond au mode « interface autonome » pour l'ensemble des expériences. L'interface doit alors avoir accès directement aux vecteurs d'entrée. L'analyse se fait à la fin de chaque expérience ou en parallèle du fonctionnement avec un bloc matériel. Enfin le cas médian est représenté par le mode mixte avec un fonctionnement en mode « interface autonome » jusqu'à l'injection puis un fonctionnement en mode « pas à pas ».

Pour les deux premiers cas le nombre d'expériences n'a pas d'importance sur la validité du choix. En revanche pour le cas médian il est nécessaire que le nombre d'expériences soit suffisant pour que l'hypothèse selon laquelle il y a au cours de toute la campagne un nombre égal de cycles en « pas à pas » et en « autonome » soit vérifiée. Cette hypothèse correspond à une distribution uniforme des cycles d'injection.

On rappelle que le mode autonome peut être caractérisé par une application en ROM émulée ou sauvegardée en mémoire RAM. Il y a donc 4 cas possibles pour le calcul (Tableau 5-10).

Pour les expériences avec injection il faut rajouter le temps nécessaire à l'injection de la ou des fautes. M_{temp} représente le nombre de fautes injectées par expérience. Avec un chargement parallèle de la cible, l'injection se fait sans surcoût temporel. Pour le chargement en série nous utilisons le paramètre N_{load} qui est le nombre de cycles nécessaires pour charger une bascule lorsque les bascules ciblées sont connectées en chaîne de scan. Avec une interface la plus optimisée possible il doit être possible d'obtenir $N_{load} = 1$ mais une valeur plus probable est sans doute $N_{load} = 2$ soit une fréquence de chargement égale à la fréquence système divisée par deux.

Tableau 5-10 : Evaluation prédictive - Expériences

Fonctionnement	Chargement	t_{expe}
Pas à pas	Parallèle	$N_{expe} * [N_{cycle} * (N_{cmd} * t_{proc2HW} + N_{control} * T_{sys} + t_{soft})]$
	Série	$N_{expe} * [N_{cycles} * (N_{cmd} * t_{proc2HW} + N_{control} * T_{sys} + t_{soft}) + M_{temp} * N_{cibles} * N_{load} * T_{sys}]$
Autonome (DMA)	Parallèle	$N_{expe} * [N_{cmd} * t_{proc2HW} + N_{cycles} * (N_{control} * T_{sys}) + t_{soft}]$
	Série	$N_{expe} * [N_{cmd} * t_{proc2HW} + N_{cycles} * (N_{control} * T_{sys}) + t_{soft} + M_{temp} * N_{cibles} * N_{load} * T_{sys}]$
Autonome (ROM émulée)	Parallèle	$N_{expe} * [N_{cmd} * t_{proc2HW} + (N_{cycles} * R_{freq\_circuit} + N_{control}) * T_{sys} + t_{soft}]$
	Série	$N_{expe} * [N_{cmd} * t_{proc2HW} + (N_{cycles} * R_{freq\_circuit} + N_{control}) * T_{sys} + t_{soft} + M_{temp} * N_{cibles} * N_{load} * T_{sys}]$
Mixte	Parallèle	$N_{expe} * [(N_{cmd} * t_{proc2HW} + \frac{N_{cycles}}{2} * (N_{control} * T_{sys}) + t_{soft}) + \frac{N_{cycle}}{2} * (N_{cmd} * t_{proc2HW} + N_{control} * T_{sys} + t_{soft})]$
	Série	$N_{expe} * [(N_{cmd} * t_{proc2HW} + \frac{N_{cycles}}{2} * (N_{control} * T_{sys}) + t_{soft}) + (M_{temp} * N_{cibles} * N_{load} * T_{sys}) + \frac{N_{cycle}}{2} * (N_{cmd} * t_{proc2HW} + N_{control} * T_{sys} + t_{soft})]$

Contrairement à l'expérience de référence le temps de récupération des sorties n'apparait pas dans le Tableau 5-10 parce que son évaluation dépend du niveau d'analyse (matériel ou logiciel).

Le fonctionnement en mode pas à pas est le même que pour l'expérience de référence hormis pour la lecture des registres de sortie. En mode autonome le processeur écrit les registres de commande dont le registre contenant le nombre de cycles à réaliser et lance ainsi l'exécution. L'interface matérielle exécute le nombre de cycles défini puis génère une interruption pour signifier au processeur que l'exécution est terminée.

Les applications numériques pour le microcontrôleur peuvent être visualisées sur la Figure 5-5 et permettent de faire quelques remarques sur la durée des expériences. Si l'on s'intéresse à l'influence du mode de fonctionnement on remarque qu'en mode autonome uniquement la durée des expériences est plus faible mais surtout qu'en mode mixte l'accélération apportée par l'accès mémoire direct reste faible bien qu'en moyenne la moitié des expériences est effectuée en mode « interface autonome ». La durée des expériences n'est divisée que par un facteur 2 en mode mixte alors qu'elle est divisée par un facteur proche de 100 en fonctionnement autonome uniquement. La seconde remarque porte sur

l'accélération apportée par le chargement parallèle qui est négligeable. Pour le cas où l'ensemble des expériences est réalisé en mode « interface autonome », le gain n'est que de 0,5% et pour les autres cas il est inférieur à 0,01%. Le gain du chargement parallèle est en fait trop faible par rapport à la durée des expériences pour ce type d'étude de cas.

Les chiffres présentés sur la Figure 5-5 sont ceux d'une application pour un système donné ; ils ne sont donc pas généraux. Par exemple pour un circuit moins conséquent comme le multiplieur, le gain lié au chargement parallèle sur l'ensemble des expériences atteint 3%. Ici le système choisi est le microcontrôleur i8051 exécutant une application de près de 52000 cycles. Le nombre de bascules du circuit est 1343 et le nombre de cibles est égal à 20% de ce chiffre soit 268 bascules. Pour évaluer les performances d'un type de chargement par rapport à l'autre lors des expériences, deux caractéristiques du circuit interviennent principalement : le nombre de cibles et le nombre de cycles de l'application. Plus le nombre de cibles est important plus le temps de chargement sera long et plus le nombre de cycles est important moins la durée du processus d'injection a d'importance dans la durée totale de l'expérience.

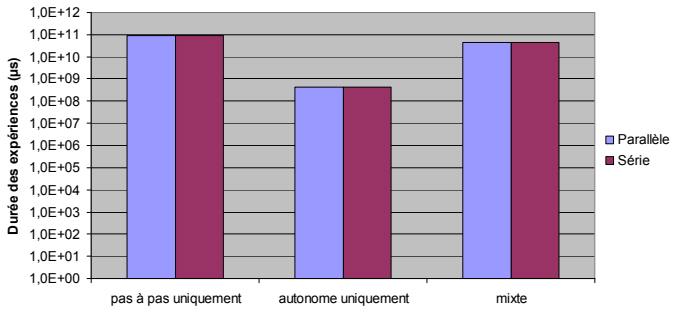

Figure 5-5 : Evaluation de la durée des expériences en fonction du mode de fonctionnement

Le graphe de la Figure 5-6 confirme l'impression rendue par le graphe de la Figure 5-5.

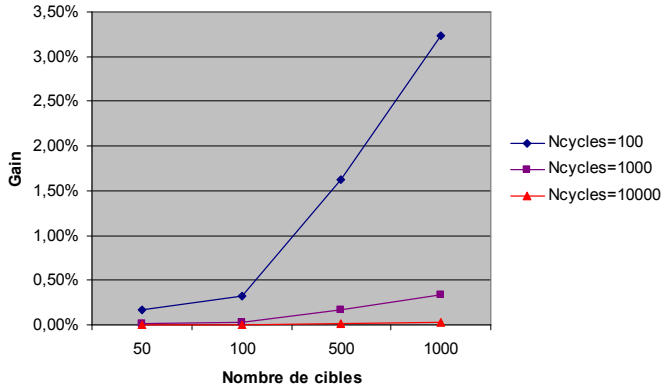

Figure 5-6 : Influence du nombre de cibles et de cycles sur le gain généré par le chargement parallèle

En effet on se rend compte, en fonctionnement mixte, des valeurs nécessaires pour que le chargement parallèle entraîne un gain significatif sur la durée des expériences. Il faut un nombre de

cibles supérieur à 500 et une application d'une centaine de cycles seulement pour que le gain soit supérieur à 1,5%. Nous pouvons en conclure que le chargement parallèle ne diminue significativement la durée des expériences que dans un nombre limité de cas.

5.2.4 Récupération des sorties du circuit et analyse

Les temps de récupération des sorties et les temps nécessaires pour l'analyse de ces sorties sont proposés dans le Tableau 5-11. Il n'est pas utile de mémoriser les sorties du circuit puisqu'elles sont directement traitées par le processeur embarqué. Le nombre de mots de 32 bits à lire dépend du nombre de sorties primaires et du nombre de sorties d'observation ((N_{PO}+N_{OO})/32). Et la récupération n'est nécessaire que dans le cas d'une analyse logicielle au niveau du processeur embarqué ou au niveau PC hôte.

La récupération des sorties pour chaque expérience de la campagne n'est effectuée qu'à partir du cycle d'injection. Le paramètre mis en jeu est le temps d'accès à l'interface à partir du processeur ($t_{hw2proc}$). Avec la simplification des cas possibles, décrite au début de la section 5.2.3, le paramètre N_{pas} ne peut donc prendre que trois valeurs possibles : 0, N_{cycles} ou $N_{cycles}/2$.

L'analyse en ligne par un bloc matériel se fait sans coût temporel. Lorsque l'analyse est effectuée par le processeur embarqué celui-ci compare les sorties récupérées avec les sorties de référence qui sont stockées en mémoire. Il y a donc pour chaque cycle et pour chaque mot un accès mémoire pour récupérer la valeur de référence ($t_{ram2proc}$) plus la comparaison des deux valeurs (t_{comp}).

Tableau 5-11 : Evaluation prédictive - Récupération des sorties et analyse

Niveau d'analyse	$t_{mem\_sorties}$
Analyse matérielle	0
Analyse logicielle	$N_{expe} * N_{pas} * \frac{N_{PO} + N_{OO}}{32} * (t_{hw2proc} + t_{ram2proc} + t_{comp})$

Il n'est pas nécessaire de mémoriser les résultats de l'analyse d'autant que la quantité de données peut être importante, notamment pour la génération des chemins de propagation d'erreurs. Ces données peuvent être directement transférées au PC hôte. Le calcul du temps de transfert entre le processeur embarqué et le PC hôte est présenté dans le paragraphe suivant.

5.2.5 Récupération des résultats de l'analyse

Les équations correspondant au temps de récupération des résultats d'analyse sont présentées dans le Tableau 5-12. La classification traditionnelle génère pour chaque classe de faute un mot qui contient le nombre de cas où la faute injectée appartient à la dite classe. Potentiellement, toujours si une faute est injectée par expérience, la valeur de ce mot est le nombre total d'expériences effectuées. Pour la classification raffinée le résultat correspond à un bit par nombre de classes et par expérience.

Ceci est valable que l'analyse soit faite au niveau matériel ou au niveau logiciel par le microprocesseur.

Tableau 5-12 : Evaluation prédictive – Remontées des résultats d'analyse

Analyse		$\mathbf{t_{upload\_résultats}}$
Bloc de classification	Traditionnelle	$N_{classes} * \left(\frac{\ln(N_{expe}) / \ln(2)}{32} \right) * (t_{hw2proc} + t_{proc2pc})$
	Raffinée	$N_{expe} * \left(\frac{N_{classes}}{32} \right) * (t_{hw2proc} + t_{proc2pc})$
Classification comparaison des sorties (µprocesseur)	Traditionnelle	$N_{classes} * \left(\frac{\ln(N_{expe}) / \ln(2)}{32} \right) * t_{proc2pc}$
	Raffinée	$N_{expe} * \left(\frac{N_{classes}}{32} \right) * t_{proc2pc}$
Chemins de propagation d'erreurs		$(N_{err\_c} + N_{err\_b} * 3) * t_{proc2pc}$

La durée de remontée des résultats au PC hôte pour l'analyse des chemins de propagation d'erreurs dépend du nombre d'erreurs qui sont détectées pendant la campagne. On caractérise ce nombre d'erreurs à l'aide des deux paramètres : N_{err_c} qui est le nombre de cycles avec erreur et N_{err_b} qui est le nombre total d'octets erronés, pour toute la campagne. Pour chaque octet erroné trois octets doivent être mémorisés : l'octet sans faute, l'octet avec faute et l'octet index. Travailler sur les octets permet d'avoir un bon compromis entre quantité de données à mémoriser et simplicité de l'algorithme. Si aucune erreur n'est générée N_{err_c} et N_{err_b} sont nulles. Le pire cas serait que chaque octet de sortie contienne une erreur à tous les cycles d'horloge (après le cycle d'injection) de toutes les expériences. Ceci est illustré par l'Équation 5-1 et l'Équation 5-2.

$$N_{err\_c\_max} = N_{expe} * N_{pas}$$

Équation 5-1 : Pire cas pour chemin de propagation d'erreurs
Nombre de cycles avec erreur

$$N_{err\_b\_max} = N_{expe} * \frac{(N_{PO} + N_{OO})}{8} * N_{pas}$$

Équation 5-2 : Pire cas pour chemin de propagation d'erreurs -
Nombre d'octets erronés

La Figure 5-7 illustre les temps de gestion des sorties du circuit analysé pour les deux cas d'illustration en fonction du type d'analyse. Les valeurs présentées sont les sommes des temps de récupération des sorties, des temps d'analyse et des temps de remontée des résultats d'analyse vers l'ordinateur hôte. Pour la génération des chemins de propagation d'erreurs on considère que 30% des cycles et des octets sont erronés.

Les valeurs obtenues confirment que l'analyse des chemins de propagation d'erreurs est très gourmande en temps. Ceci est principalement dû à la quantité de données à remonter de la carte vers le PC hôte car les temps de récupération et d'analyse (comparaison des sorties de référence et des sorties fautées) sont équivalents à ceux pour la classification logicielle. La connectique entre le PC et la plateforme est donc bien, pour ce type d'analyse, le nœud d'étranglement de l'environnement.

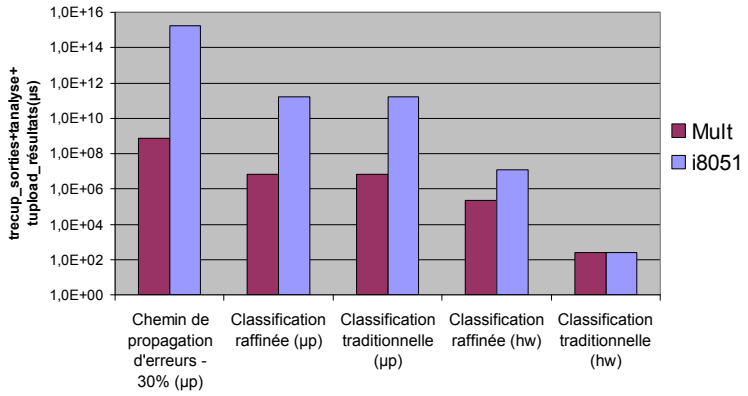

Figure 5-7 : Temps d'analyse
(récup. sorties+analyse+remontée résultats)

Pour la classification et pour un circuit relativement important, principalement en nombre de cycles de l'application, comme le microcontrôleur i8051 on paye la flexibilité de l'analyse logicielle par un temps de gestion des sorties important. Les temps de lecture des sorties et d'analyse sont du même ordre de grandeur (environ 5 10^{10} µs). L'analyse par un bloc matériel devient alors tout à fait intéressante même si elle est plus contraignante sur la définition des conditions d'analyse. Les temps de récupération des sorties deviennent alors infimes puisqu'il n'y a que très peu de transferts de données entre le périphérique d'injection et la RAM ou le processeur embarqué.

5.3 Bilan et poids des paramètres de l'environnement

Après avoir détaillé le calcul prédictif de la durée de chaque tâche indépendamment nous validons l'évaluation en comparant les temps de campagne calculés aux temps de campagne effectifs. La comparaison est effectuée pour le multiplieur et le microcontrôleur avec des répartitions de tâches correspondant à ces deux exemples.

Pour le multiplieur nous proposons une campagne exhaustive d'injections de *bit-flips* simples dans 40 bascules sur les 70 existantes. La charge de travail est une multiplication de deux valeurs (0x3FF * 0x3FF) pour une durée de 28 cycles. Cette campagne d'injection comprend donc 1120 expériences. La génération de la liste de fautes se fait au niveau du processeur embarqué suivant l'algorithme de Mersenne-Twister (cycles et cibles d'injection). La classification est logicielle et deux classes de fautes ont été utilisées en fonction du résultat de la multiplication : « Silencieuse » pour un résultat égal à 0xFF801, « Erreur » sinon. La répartition des tâches et la durée évaluée pour chaque tâche sont présentées dans le Tableau 5-13.

Tableau 5-13 : Evaluation prédictive pour l'analyse du multiplieur

Tâche	Niveau d'exécution/ Configuration	Durée évaluée (ms)	Durée effective (ms)
Génération des vecteurs d'entrées	PC hôte via RAM et processeur	67,7	
Génération de la liste de fautes	Processeur embarqué	59,6	
Expérience de référence	Classification par comparaison des sorties	0,56	
Expériences avec injection	Pas à Pas Uniquement -Parallèle	378,2	
Récupération des sorties & Analyse	Classification comparaison des sorties (µprocesseur) - Traditionnelle	313,6	
Récupération des résultats d'analyse	Classification comparaison des sorties (µprocesseur) - Traditionnelle	0,16	
Campagne complète		820	943
Expérience		0,732	0,842

La campagne d'injection permet de déterminer que 58% des fautes injectées entraînent une erreur dans la multiplication. La durée effective de la campagne est de 943 millisecondes ce qui donne 0,842 millisecondes par expérience. L'évaluation prédictive donne un temps de campagne légèrement inférieur de 13%. Ceci est dû à ce qui est négligé dans le calcul prédictif comme les appels et retours de fonction par exemple.

En ce qui concerne le microcontrôleur la version mise en œuvre est celle avec ROM émulée en logique programmable. Il n'y a donc aucun transfert pour la génération des vecteurs d'entrée. La liste de fautes est générée par le processeur embarqué selon l'algorithme de Mersenne-Twister. La classification (2 classes) est réalisée en ligne par un bloc matériel qui compare la séquence des sorties à une trace prédéfinie par simulation.

Les fautes injectées sont également des *bit-flips* simples et le microcontrôleur est partiellement instrumenté puisque 123 bascules de la partie contrôle ont été sélectionnées pour cette campagne type. L'application (sort.c) dure 71880 cycles d'horloge donc une campagne exhaustive serait constituée des plus de 8 millions d'expériences. Par soucis de rapidité seulement 1% des fautes sont injectées soit un total de 88417 fautes. Les valeurs calculées pour chaque tâche sont présentées dans le Tableau 5-14.

Tableau 5-14 : Evaluation prédictive pour l'analyse du microcontrôleur

Tâche	Niveau d'exécution/Configuration	Durée évaluée (ms)	Durée effective (ms)
Génération des vecteurs d'entrées	-	0	
Génération de la liste de fautes	Processeur embarqué	4937,5	
Expérience de référence	-	0	
Expériences avec injection	Autonome Uniquement -Parallèle	$128{,}2\ 10^3$	
Récupération des sorties & Analyse	Classification matérielle – séquence de sortie prédéfinie- Traditionnelle	0	
Récupération des résultats d'analyse	Classification Traditionnelle	0,32	
Campagne complète		$133\ 10^3$	$140\ 10^3$
Expérience		1,50	1,58

Près de 98% des fautes injectées (86609 fautes) entraînent une erreur dans le fonctionnement du microcontrôleur exécutant le programme sort.c. On observe dans le Tableau 5-14 que la durée de la campagne est évaluée à environ 133 secondes soit 1,5 millisecondes par expérience. Dans la pratique la campagne dure 140,34 secondes ce qui donne une expérience à 1,58 millisecondes. Pour ce cas d'étude l'erreur sur la durée de la campagne est de seulement 5%.

L'évaluation prédictive permet donc d'obtenir un ordre de grandeur raisonnable de la durée d'une campagne d'injection en fonction des paramètres de l'environnement d'analyse, des caractéristiques du circuit et des souhaits du concepteur en terme d'analyse.

Calculer a priori les durées nécessaires pour effectuer chaque étape d'une campagne met également en relief les paramètres qui ont le plus de poids dans cette durée. En effet suivant les caractéristiques du circuit analysé et suivant le type d'analyse, les différentes parties de l'environnement sont plus ou moins mises à contribution. Les Figure 5-8 et Figure 5-9 mettent en lumière l'importance des différents paramètres de l'environnement dans la durée d'une expérience. Pour chaque catégorie les paramètres sont réduits de 50% par rapport aux valeurs définies précédemment (Tableau 5-1 et Tableau 5-2) ce qui correspond à une amélioration des performances de l'environnement. Par exemple pour la catégorie « accès par l'interface » la durée d'un accès à un périphérique (typiquement un banc mémoire) par l'interface via le bus système est réduite de 50% par rapport à la durée de référence. Les six catégories sont : accès par le processeur ($t_{proc2hw}$, $t_{proc2ram}$), accès par l'interface (t_{hw2ram}), accès par le PC hôte (t_{pc2ram}), performances du processeur ($t_{algo\_\mu p}$, t_{soft}), fréquence d'horloge du système (tous les paramètres exceptés ceux liés à l'implémentation de l'interface) et implémentation de l'interface ($N_{control}$, $R_{fréq\_circuit}$).

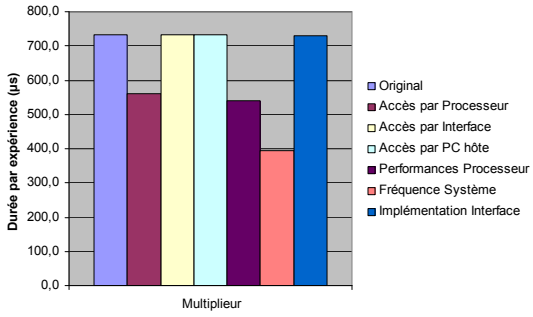

Figure 5-8 : Importance des paramètres pour une analyse du multiplieur

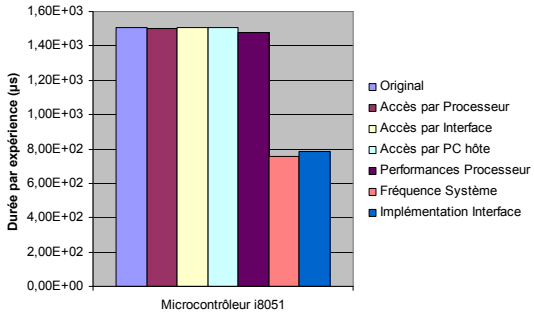

Figure 5-9 : Importance des paramètres pour une analyse du microcontrôleur i8051

L'accélération des accès mémoire par l'interface n'a aucune importance pour la simple raison qu'ils ne sont pas mis en œuvre pour les deux exemples considérés. Pour une analyse par classification, par le processeur ou par un bloc matériel, et bien que les vecteurs d'entrée soient générés par le PC hôte pour le multiplieur, la vitesse de transfert entre le PC hôte et la carte n'a pas non plus d'importance.

En revanche, pour une campagne comme celle du multiplieur, la réduction des temps de transfert entre le processeur embarqué et l'interface apporte une accélération de l'ordre de 25%. De la même manière les performances du processeur, qui est alors largement mis en œuvre, a une grande importance. Pour ce type de campagne on note également que l'optimisation de l'interface d'injection n'a pas spécialement d'importance.

Pour une campagne où plusieurs tâches (générations des vecteurs d'entrée, analyse) sont réalisées par des blocs matériels les performances de l'interface d'injection ont au contraire une importance prépondérante. Une amélioration de 50% de ces performances apporte une accélération voisine de 50%. La liste de fautes est générée au niveau du processeur embarqué mais les transferts processeur/interface correspondant sont trop peu nombreux pour entraîner une accélération conséquente (seulement 0,15%).

Enfin on remarque qu'il est possible de faire le lien entre les valeurs obtenues par prédiction et l'importance relative des paramètres de l'environnement sur la durée de la campagne. Pour le multiplieur les tâches qui prennent le plus de temps sont les expériences, la récupération des sorties et l'analyse. Au vu de la répartition des tâches on peut en déduire que les transferts processeur/interface et les performances du processeur ont une importance prépondérante dans la durée de la campagne. Ceci est confirmé par le graphe de la Figure 5-8. Pour le microcontrôleur on peut de la même manière anticiper l'importance de l'implémentation de l'interface d'injection puisqu'une grande partie des tâches y est exécutée et que les expériences représentent la quasi-totalité de la campagne d'un point de vue temporel.

5.4 Conclusion

Nous avons présenté dans ce chapitre l'évaluation prédictive des temps de génération des vecteurs d'entrées, de génération de la liste de fautes, de déroulement des expériences, de récupération des sorties, d'analyse et de récupération des résultats d'analyse par l'ordinateur hôte. Cette évaluation est basée sur les paramètres liés à l'environnement, sur les paramètres liés à l'implémentation de l'interface matérielle et sur les caractéristiques du circuit analysé ; il y a donc un grand nombre de paramètres.

Pour chaque tâche l'évaluation théorique est illustrée par les applications numériques correspondant à un environnement donné (carte ML310, PowerPC …) et à l'analyse de deux circuits (Multiplieur 16 bits et microcontrôleur i8051).

Ces applications permettent de mettre en lumière les différences de performances entre les 3 niveaux d'exécution des tâches et permettent d'étudier l'utilité et les performances des techniques que nous proposions dans le chapitre 4. Les résultats obtenus montrent que le codage permet de réduire le temps de génération de la liste de fautes et confirment que l'injection parallèle n'apporte d'accélération significative que pour un nombre de cas relativement restreint.

Nous avons enfin confronté l'évaluation prédictive au temps de campagne effectif pour les deux exemples de circuits. Les résultats obtenus sont très proches malgré les approximations faites. Il est donc désormais possible d'obtenir l'ordre de grandeur de la durée d'une campagne d'injection et de déterminer les paramètres les plus importants pour une analyse donnée d'un circuit donné. Cela offre au concepteur une ligne directrice pour le choix éventuel de l'environnement de prototypage à mettre en œuvre afin d'obtenir les meilleures performances possibles.

6 Automatisation et études de cas

Après avoir présenté le flot d'analyse, les possibilités offertes par l'environnement et les nouvelles techniques développées, ce dernier chapitre porte sur la mise en œuvre de l'ensemble dans le cadre de l'analyse de sûreté de circuits significatifs : une IP (*Intellectual Property*) d'encryptage AES et le microprocesseur Leon2 exécutant une application logicielle de cryptage AES.

6.1 Outil développé pour l'analyse

Nous avons vu qu'une des raisons de notre choix de l'instrumentation au niveau portes est une plus grande facilité d'automatisation par rapport à l'instrumentation au niveau RTL. L'intérêt de l'automatisation est d'éviter au concepteur d'effectuer à la main une tâche récurrente et qui peut prendre un temps important. Il est en effet inutile d'optimiser les performances de l'environnement sans se préoccuper de la configuration de celui-ci.

De plus, indépendamment de la plateforme sur laquelle est implémentée l'environnement d'analyse, il est nécessaire pour toute campagne d'injection de développer des modules logiciels et matériels pour la réalisation de la campagne. Le développement de ces modules dépend des choix de l'utilisateur (type d'analyse, répartition des tâches …) et doit prendre en compte les caractéristiques du circuit analysé. Il peut également prendre un certain temps alors qu'il est en partie automatisable.

J'ai donc développé un outil d'instrumentation et de génération automatique de modules appelé ANIFFI pour *Automatic Netlist Instrumentation For Fault Injection*. Celui-ci s'insère dans le flot d'instrumentation présenté sur la Figure 6-1.

L'outil d'instrumentation modifie des netlists au format EDIF (*Electronic Design Interchange Format*). Ce format a été développé dans les années 80 pour le transfert de données d'un environnement de travail à un autre. C'est un format neutre et non propriétaire contrairement aux formats utilisés par les entreprises d'EDA comme par exemple le format ngc de Xilinx. Si la description du circuit à analyser est une description EDIF générée après synthèse ciblant la technologie FPGA mise en œuvre, celle-ci peut être directement instrumentée. Mais dans une majorité de cas la description du circuit disponible est une description HDL (VHDL ou verilog) voire une description de plus haut niveau. Il faut donc d'abord synthétiser cette description et générer la netlist pour la technologie utilisée. En ce qui concerne les cas d'études nous avons utilisé les outils de synthèse (xst) et de traduction (ngc2edif) distribués par Xilinx.

La netlist originale, le fichier de primitives et le fichier de paramètres sont les trois fichiers d'entrée de l'outil d'instrumentation. Le fichier de primitives contient deux types de données : les noms des instances de bascules de la technologie et les déclarations des éléments nécessaires à l'instrumentation. Les premiers permettent d'identifier toutes les bascules qui sont présentes dans la description. Les secondes ont pour but d'être ajoutées à la description dans le cas où elles n'y figurent pas au départ. Pour la technologie Xilinx Virtex2 ce sont les déclarations de l'inverseur (INV), du multiplexeur (MUX) et du OU-exclusif (XOR).

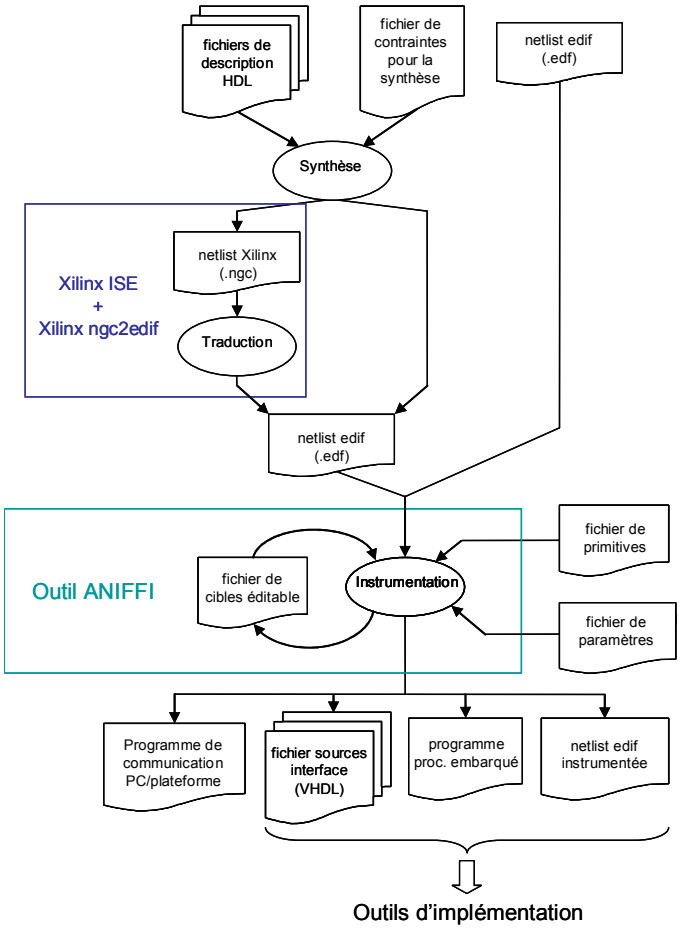

Figure 6-1 : Flot et outil d'instrumentation

La première tâche de l'outil est d'établir la hiérarchie de la description et d'identifier les bascules présentes dans la netlist. Un fichier contenant la liste des bascules est alors généré et présenté à l'utilisateur. Celui-ci peut éditer ce fichier et ainsi sélectionner les bascules dans lesquels il souhaite injecter des fautes comme cela est illustré sur la Figure 6-2. L'édition d'un fichier texte est une technique simple à mettre en œuvre mais peu conviviale et pourrait être remplacée par la sélection de cibles via une interface graphique.

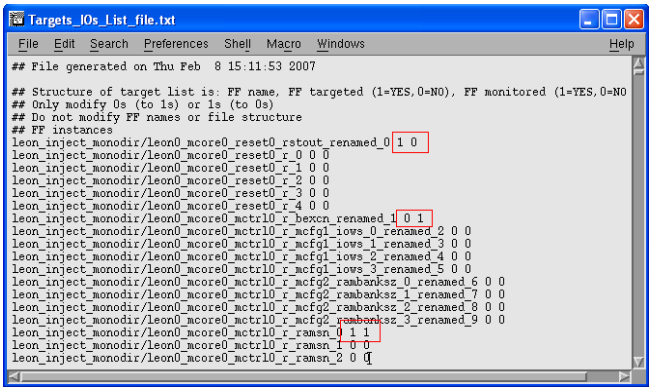

Figure 6-2 : Exemple de fichier de cibles

Un certain nombre de paramètres est ensuite nécessaire pour procéder à l'instrumentation puis pour générer les modules logiciels et matériels. Ces paramètres sont définis dans le Tableau 6-1. Ils peuvent être répartis en plusieurs groupes en fonction de leurs origines. Le premier regroupe les paramètres liés à l'environnement de développement. Le groupe « Circuit » inclut les paramètres liés aux caractéristiques du circuit analysé. Les trois groupes « Fautes », « Application » et « Analyse » sont la représentation des données d'entrée du flot d'analyse qui a été présenté avec la Figure 3-3. Le dernier groupe spécifie la répartition des tâches souhaitée par l'utilisateur.

Tableau 6-1 : Paramètres pour l'instrumentation

Nom	Valeurs possibles	Groupe
vendor	Xilinx	Environnement
implementation_tool	ISE	Environnement
instrumentation_style	LUT | XOR	Circuit
configuration	Pas_A_Pas | Autonome	Circuit & Analyse
campaign_type	exhaustive | pseudo_random | deterministic	Analyse
fault_type	SEU | MBU	Fautes
spatial_multiplicity	Tout nombre entier	Fautes
temporal_multiplicity	Tout nombre entier	Fautes
workload_cycles_nb	Tout nombre entier	Application
observation_cycle	No | Tout nombre entier	Analyse
experiments_nb	Tout nombre entier	Analyse
analysis_type	HW_class | SW_class_user | SW_class_comp_user | SW_class_comp_all | SW_EPPA	Analyse
fault_classes_nb	Tout nombre entier	Analyse
fault classes	(class_1, condition), (class_2, condition), (class_3, condition)	Analyse
pi_generation	PC | SW | HW | NO	Répartition des tâches
fii_generation	PC | SW | HW	Répartition des tâches
clk_active_edge	up | down	Circuit
reset_type	synchrones | asynchrones	Circuit
reset_active_level	0 | 1	Circuit
reset_clock_cycles_nb	Tout nombre entier	Circuit

Comme toutes les combinaisons de paramètres ne sont cependant pas possibles, une phase de vérification des paramètres est effectuée avant de commencer l'instrumentation. Par exemple une configuration « autonome » n'est pas compatible avec la génération des vecteurs d'entrée au niveau du processeur embarqué.

Au final l'outil génère la netlist instrumentée, les sources VHDL pour l'interface d'injection, les sources C pour le programme exécuté par le processeur embarqué et le programme de communication entre le PC hôte et la plateforme de prototypage. Le gain en temps est très important puisque l'instrumentation et la génération des fichiers ne prennent que quelques minutes pour des circuits conséquents du type processeurs.

6.2 IP AES

6.2.1 Présentation

L'occurrence d'erreurs lors du calcul est un réel défi pour les concepteurs d'IP de cryptage qui doivent concevoir des techniques qui effectuent une détection en ligne durant le fonctionnement du circuit. Une erreur non détectée peut induire un comportement ou un résultat inattendu, ce qui n'est pas souhaitable d'un point de vue sécurité.

L'algorithme AES (*Advanced Encryption Standard*) est un algorithme de chiffrement symétrique de données de 128 bits. Les 16 octets en entrée sont d'abord permutés selon une table définie au préalable et appelée S-Box pour *Substitution-Box* (opération non-linéaire). Ces octets sont ensuite placés dans une matrice 4x4 appelée *state* dont les lignes subissent une rotation vers la droite. L'incrément pour la rotation varie selon le numéro de la ligne. Une transformation linéaire est ensuite appliquée sur la matrice, elle consiste en la multiplication binaire de chaque élément de la matrice avec des polynômes issus d'une matrice auxiliaire. La transformation linéaire garantit une meilleure diffusion (propagation des bits dans la structure) sur plusieurs tours. Finalement, un XOR entre la matrice et une autre matrice permet d'obtenir une matrice intermédiaire. Ces différentes opérations définissent un « tour » et sont répétées plusieurs fois.

Deux techniques de détection basées sur la redondance d'information d'une part et sur la redondance temporelle d'autre part ont été appliquées à une IP de cryptage AES dont l'architecture originale est présentée dans [MANG-03]. Chaque octet est calculé indépendamment par deux blocs dédiés aux calculs linéaires (*Data-Cell*) et non linéaires (*S-Box*). Avec cette architecture 60 cycles sont nécessaires pour le cryptage d'une donnée de 128 bits et 4 cycles supplémentaires sont requis pour l'entrée et la sortie des données.

La protection par redondance d'information utilise le codage par parité de chaque octet. Elle nécessite un générateur de parité, un ensemble de règles de prédiction pour propager le bit de parité à travers les différentes opérations et un bloc de comparaison des bits de parité finaux aux bits de parité calculés [BERT-03]. La technique à base de redondance temporelle DDR (*Double Data Rate*) met en œuvre les deux fronts d'horloge afin de calculer deux octets en un seul cycle d'horloge [MAIS-07b].

6.2.2 Configuration de l'environnement et spécifications des campagnes d'injections

Le Tableau 6-5 reprend le tableau qui définit la répartition des tâches possible et montre les choix qui ont été faits pour l'analyse de l'IP de cryptage AES. Deux vecteurs d'entrée sont nécessaires ; la donnée à crypter et la clé de cryptage tous deux codées sur 128 bits. Ces deux données sont générées au niveau du PC hôte. Toutes les campagnes d'injection effectuées sont des campagnes exhaustives, la ou les cibles et le cycle d'injection sont incrémentés dans le programme exécuté par le PowerPC. La classification est réalisée au même niveau en comparant à la fin de chaque expérience la valeur cryptée à la valeur de référence qui est définie a priori dans le programme.

Tableau 6-2 : Répartition des tâches pour l'analyse de l'IP de cryptage AES

Tâche	Niveau d'exécution		
	PC hôte	Microprocesseur embarqué	Interface matérielle
Génération des entrées primaires	**Déterministe**	~~Exhaustive~~ / ~~Aléatoire~~	~~Exhaustive~~ / ~~Pseudo-aléatoire~~
Génération de la liste de fautes	~~Déterministe~~	**Exhaustive** / ~~Aléatoire~~	~~Exhaustive~~ / ~~Pseudo-aléatoire~~
Analyse des résultats	~~Chemins de propagation d'erreur~~	~~Chemins de propagation d'erreur~~ / **Classification**	~~Classification~~

Pour cette analyse le *bit-flip* a été choisi pour modéliser aussi bien les fautes naturelles que les fautes intentionnelles. L'algorithme AES est très régulier ce qui affecte la propagation d'erreurs dans le chemin de données. Il a été observé que la diffusion d'une faute dépend de sa position mais pas la façon dont évolue l'erreur dans la matrice. Ce qui signifie que l'on peut identifier un comportement erroné type indépendamment de la position initiale de la faute. On peut ainsi réduire la taille de la liste de fautes sans compromettre la validité des résultats.

Les campagnes ne comprennent donc que l'injection d'un sous ensemble de la liste de fautes et pour chaque campagne seule la partie concernée a été instrumentée. Quatre campagnes ont été réalisées correspondant à :

1. L'injection dans un octet et dans deux octets adjacents du bloc de calcul linéaire.
2. L'injection dans un seul composant S-Box.
3. L'injection dans l'unité de contrôle.

Chaque campagne d'injection a été partitionnée en trois sous-campagnes pour l'analyse des deux mécanismes de détection séparément et pour l'analyse de l'IP incluant les deux mécanismes en même temps. Lors de chaque sous-campagne des fautes ont été injectées dans toutes les cibles et à tous les cycles d'horloge exceptés ceux correspondant au chargement des données et au dernier cycle du calcul soit 59 cycles. Au dernier cycle de calcul les opérations sont linéaires donc non significatives du point de vue d'une attaque volontaire. Des fautes avec une multiplicité spatiale allant de 1 à 16 ont été injectées de façon exhaustive aux cycles d'horloge concernés. Pour l'injection dans deux octets cela correspond à 65535 expériences par cycle car toutes les valeurs possibles du « mot cible » ont été mises en œuvre.

La classification est réalisée par le processeur embarqué en fonction du résultat du cryptage de l'IP suivant quatre classes de fautes :

- silencieuse : cryptage correct et pas de détection de faute,
- erreur non détectée : cryptage incorrect mais pas de détection de faute,
- faux positif : cryptage correct mais détection de faute,
- erreur détectée : cryptage incorrect et détection de faute.

6.2.3 Résultats sur la robustesse de l'IP

Pour les blocs de calcul linéaire on peut remarquer dans le tableau de résultats (Tableau 6-3) qu'un grand nombre de fautes injectées n'a pas d'effet sur le cryptage (classes « Silencieuse » et « Faux positif »). Ceci peut s'expliquer par le fait que l'IP a une architecture « pipelinée » et donc qu'une faute dans un étage non utilisé de celui-ci reste sans effet. En ce qui concerne la détection par parité, il était prévu que son efficacité dépende de la multiplicité spatiale de la faute.

Tableau 6-3 : Résultats de l'analyse de l'IP AES - Injection dans un bloc de calcul linéaire

Classification	un octet			2 octets adjacents		
	Parité	DDR	DDR+Par.	Parité	DDR	DDR+Par.
Nombre de cibles	8	8	8	16	16	16
Nombre de fautes injectées	15045	15045	15045	$3{,}86\ 10^6$	$3{,}86\ 10^6$	$3{,}86\ 10^6$
Durée de la campagne	~1s	~1s	~1s	~3min	~3min	~3min
Silencieuse	16,88%	66,10%	58,44%	4,30%	66,10%	54,66%
Erreur non détectée	32,92%	0,00%	0,00%	20,70%	0,00%	0,00%
Faux Positif	17,02%	0,00%	7,66%	12,78%	0,00%	11,44%
Erreur détectée	33,18%	33,90%	33,90%	62,22%	33,90%	33,90%

Les injections de *bit-flips* multiples dans un seul octet confirment le faible niveau de détection de la parité. Au contraire l'architecture à base de DDR ne laisse aucune faute non détectée et n'engendre aucun « Faux positif » ; l'architecture DDR soit détecte la faute soit continue l'exécution sans être affectée. De plus on observe que la combinaison des deux mécanismes n'améliore pas le taux de détection.

Les résultats de la campagne d'injection dans un bloc de calcul non linéaire (*S-Box*) sont présentés dans le Tableau 6-4. Cette campagne inclut l'injection de fautes dans le registre de sortie du bloc de substitution et dans les registres internes du pipeline. A nouveau on peut voir que les performances du mécanisme de détection DDR sont presque parfaites alors que le niveau de détection de la parité reste peu élevé (~40%).

Tableau 6-4 : Résultats de l'analyse de l'IP AES - Injection dans un bloc de calcul non-linéaire (S-Box)

Classification	Registre Résultat			Registre interne		
	Parité	DDR	DDR+Par.	Parité	DDR	DDR+Par.
Nombre de cibles	8	16	16	24	24	24
Nombre de fautes injectées	15045	$3{,}86\ 10^6$	$3{,}86\ 10^6$	$989{,}8\ 10^6$	$989{,}8\ 10^6$	$989{,}8\ 10^6$
Durée de la campagne	~1s	~3min	~3min	~12h	~12h	~12h
Silencieuse	9,29%	33,90%	9,75%	11,56%	1,88%	1,79%
Erreur non détectée	41,36%	0,00%	0,00%	40,18%	0,06%	0,03%
Faux Positif	7,66%	33,90%	58,05%	7,58%	50,72%	50,81%
Erreur détectée	41,69%	32,20%	32,20%	40,69%	47,34%	47,37%

Tous ces résultats ainsi que ceux de la campagne d'injections dans la partie contrôle de l'IP AES sont détaillés dans [MAIS-07a].

6.2.4 Bilan sur l'utilisation de l'environnement développé

Les résultats ci-dessus confirment que l'environnement proposé permet une analyse relativement complète de cet IP de cryptage mais ce qui est le plus intéressant du point de vue de l'environnement d'analyse est :

- la rapidité de l'analyse,
- la possibilité d'injecter des fautes avec une multiplicité spatiale très importante.

En effet on voit qu'avec l'environnement proposé l'injection de près d'un milliard de fautes ne prend qu'une douzaine d'heures soit 43,6 microsecondes par expérience. On remarquera simplement que l'évaluation prédictive donne une durée d'expérience proche de 49 microsecondes. L'ordre de grandeur de l'évaluation est donc correct pour l'analyse de l'IP de cryptage.

Si l'on considère la simulation RTL de l'IP AES seule effectuée avec le simulateur Modelsim 6.0 sur une station SunBlade 2500 avec double processeur 1,6GHz et 2 GigaOctets de mémoire vive. Cette simulation RTL prend approximativement 2,4 secondes ce qui induit que nous avons une accélération voisine de 55000 pour ce cas d'études et cette configuration.

D'autre part l'environnement et les techniques proposées permettent d'injecter des fautes avec une multiplicité spatiale très importante, jusqu'à 24 pour l'exemple de l'AES. L'injection de fautes avec de telles multiplicités n'a peut être pas pour le moment de justification physique pour les fautes naturelles. Cependant on sait que les fautes transitoires affectant 2 ou 3 bits voire plus sont déjà de plus en plus fréquentes pour les technologies actuelles et que leur fréquence va en augmentant. De plus prendre en compte des multiplicités spatiales importantes devient indispensable si l'on considère les attaques intentionnelles.

La campagne d'injection a été réalisée par Paolo Maistri qui avait développé et intégré les mécanismes de détection dans l'IP AES. Ceci montre que la prise en main de l'environnement est rapide pour toute personne souhaitant effectuer l'analyse d'un circuit.

6.3 Leon2 avec application de cryptage

6.3.1 Présentation

Le Leon2 est un processeur basé sur le jeu d'instructions SPARC V8 et qui a été initialement conçu pour des applications spatiales : il dérive du processeur ERC32 de l'Agence Spatiale Européenne (ESA). L'IP est disponible sous la forme d'un projet VHDL synthétisable protégé par la licence GNU LGPL qui permet d'avoir pleine connaissance du code source et pleine liberté de modification (toujours sous licence LGPL) [GAIS-06]. A noter qu'il existe une version tolérante aux fautes appelé Leon2-FT [GAIS-02], que la société ATMEL a récemment produit un composant pour

applications spatiales basé sur le processeur Leon2 [ATME-06] et que ce processeur est utilisé dans un nombre croissant d'applications en dehors du domaine spatial.

Notre étude s'est portée sur l'analyse du processeur Leon2 exécutant une application de cryptage AES ce qui représente un système significatif du point de vue matériel et du point de vue logiciel. Ce cas d'étude a également été choisi parce que nous disposions d'une part d'une version durcie de Leon2 et d'autre part d'une version durcie du programme AES exécuté. L'étude comprend donc l'analyse de trois systèmes :

- Leon2 et programme AES non durci,
- Leon2 durci et programme AES original,
- Leon2 original et programme AES durci.

Les protections matérielles et logicielles mises en œuvre sont rapidement présentées avant les spécifications des campagnes d'injection.

❖ Protections matérielles

La version durcie du processeur Leon2 que nous analysons ici est celle qui inclut les protections au niveau de l'unité entière (Integer Unit - IU) présentées dans [PORTo-06]. La logique séquentielle est protégée contre les SEUs à l'aide de redondance d'information. En effet chaque registre entre les étages du pipeline est protégé par un code de parité suivant l'architecture de la Figure 6-3. Le code de parité est vérifié par deux entités indépendantes réalisées en logique inversée et deux signaux d'erreur complémentaires sont générés. Les signaux d'erreurs sont remontés jusqu'à l'interface du processeur ce qui permet d'observer le déclenchement du mécanisme de détection de l'un des registres protégés.

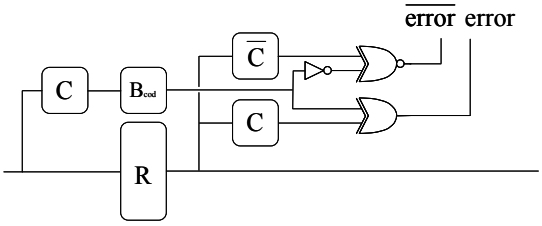

Figure 6-3 : Architecture de protection par prédiction de parité

L'approche ci-dessus était d'abord destinée à une analyse à base de simulation RTL donc une attention particulière a été portée sur les signaux indéfinis ou avec le label « don't care » qui peuvent rendre le bit de parité également indéfini. La solution choisie fut de filtrer les signaux suivant trois stratégies dans la prise en compte des signaux dans le calcul de la parité :

- « Statique minimale » : Seuls les signaux qui sont toujours définis sont pris en compte.
- « Dynamic » : A un cycle donné seuls les signaux définis à cet instant sont pris en compte.
- « Statique maximale » : Tous les signaux sont toujours pris en compte, indépendamment des 'U' ou 'X'.

Les signaux non définis ne sont plus un problème dans un environnement de prototypage puisque tous les signaux sont définis à chaque cycle d'horloge. Cependant les valeurs à un instant donné dépendent de la cible technologique choisie pour la synthèse et peuvent être différentes entre le prototype et le circuit final. Nous avons donc opté pour la stratégie de filtrage « statique minimale ». Ceci signifie que seul un sous-ensemble des signaux mémorisés dans un étage du pipeline est pris en compte pour le calcul de la parité de cet étage. Dans un premier temps le but n'est pas de comparer les stratégies de filtrage mais le mécanisme implémenté a un impact sur l'analyse des résultats qui peut être faite comme nous le verrons dans la partie 6.3.3.

❖ Protections logicielles

Les mécanismes de détection logicielle ont été développés et appliqués au programme de cryptage AES par le *Politecnico de Torino*. Ils exploitent les concepts de redondance d'information, temporelle, et d'opération afin de détecter l'occurrence de fautes lors de l'exécution du programme. La technique se base sur la modification du code source en insérant des instructions redondantes pour détecter les fautes affectant les données et les opérations manipulées par le programme ainsi que le séquencement correct des instructions. Plus de détails sur la protection des données et sur la protection du séquencement des instructions sont disponibles respectivement dans [CHEY-00] et [GOLO-03].

6.3.2 Spécification des campagnes d'injections

Le Tableau 6-5 reprend le tableau qui définit la répartition des tâches possible et montre les choix qui ont été faits pour l'analyse du système basé sur le processeur Leon2.

Tableau 6-5 : Répartition des tâches pour l'analyse du Leon2

Tâche	Niveau d'exécution		
	PC hôte	Microprocesseur embarqué	Interface matérielle
Génération des entrées primaires	**Déterministe**	~~Exhaustive~~ / ~~Aléatoire~~	~~Exhaustive~~ / ~~Pseudo-aléatoire~~
Génération de la liste de fautes	~~Déterministe~~	~~Exhaustive~~ / **Aléatoire**	~~Exhaustive~~ / ~~Pseudo-aléatoire~~
Analyse des résultats	~~Chemins de propagation d'erreur~~	~~Chemins de propagation d'erreur~~ / **Classification**	~~Classification~~

La génération des entrées primaires correspond pour un processeur à la génération des images RAM et ROM nécessaires pour le programme à exécuter. Celles-ci ont été générées au niveau du PC hôte avant d'être mémorisées dans la RAM de l'environnement via le processeur.

Pour réduire la durée des campagnes seulement une partie de la liste de fautes a été injectée. Pour chaque campagne le nombre de fautes injectées est approximativement égal à 1% du nombre de fautes correspondant à une campagne exhaustive (N_{cycles}*N_{cibles}). Les cycles et cibles d'injection ont été générés de façon aléatoire au niveau du processeur embarqué suivant l'algorithme de *Mersenne-Twister* (distribution uniforme).

Il a été précédemment montré que l'injection de 1% des fautes possibles permet d'obtenir des résultats représentatifs en un temps plus acceptable [AMMA-05]. Dans [NGUY-05] NGuyen et. al calculent le nombre minimum de bascules à cibler pour une précision et un indice de confiance donnés. Ce calcul se base sur le nombre total de bascules du circuit et sur un facteur de réévaluation du taux d'erreur estimé au niveau système aussi appelé facteur de déclassement (*derating factor*). Pour un processeur de type Intel Itanium contenant 150000 bascules ils estiment que le nombre minimum de bascules à cibler est 300. Pour une application d'un million de cycles d'horloge ils considèrent que l'injection dans seulement 100 cycles est également suffisante pour obtenir un facteur de déclassement avec une précision donnée. En conséquence 30000 expériences seraient suffisantes ce qui représenterait 0,0002 ‰ des fautes possibles (N_{cycles}*N_{cibles}). On notera qu'avec les mêmes conditions, pour les applications effectivement utilisées, M88KSIM et Vortex de SpecINT95, ce taux tombe respectivement à 3,52 10^{-10} et 1,8 10^{-11}. L'injection de seulement 1% des fautes possibles est donc tout à fait pertinente du point de vue de la validité des résultats obtenus.

Enfin nous avons choisi d'effectuer l'analyse des sorties du Leon2 au niveau logiciel à cause de la simplicité de mise en œuvre et de la flexibilité de cette approche. Une expérience sans injection est réalisée au début de chaque campagne pour récupérer la trace de référence qui sert à valider le fonctionnement correct du processeur au moment de l'injection. Pour chaque expérience l'interface d'injection contrôle le processeur étudié en mode « interface autonome » jusqu'au cycle d'injection. La faute injectée est classifiée en fonction de l'état de Leon2 à la fin de l'expérience.

La netlist générée après synthèse de la description RTL du processeur a été instrumentée afin de permettre l'injection de fautes. Toutes les bascules du pipeline de Leon2 sont des bascules avec entrée de validation donc l'instrumentation est à base de LUTs.

Au vu de l'environnement d'analyse et des mécanismes de protection le *bit-flip* simple a été choisi pour modéliser les SEUs dans les cellules mémoires.

Trois campagnes d'injections ont donc été menées ; la première sans aucun mécanisme de détection, la seconde et la troisième avec respectivement les protections matérielles et logicielles. L'analyse de la version non protégée du système nous indique la robustesse intrinsèque de celui-ci. Les campagnes d'injection dans les versions protégées nous donnent leurs éventuelles robustesses additionnelles et les couvertures de fautes des mécanismes implémentés.

Chaque campagne est divisée en cinq sous-campagnes, une pour chaque étage du pipeline, durant lesquelles des fautes sont injectées dans toutes les bascules de l'étage considéré. Les cinq étages du pipeline sont les suivants : Fetch (FE), Decode (DE), Execute (EX), Memory (ME) et Write (WR).

Pour la version avec protections matérielles la détection d'une erreur entraîne le positionnement des sorties d'erreurs du processeur. Celles-ci sont donc observées par le PowerPC à chaque cycle d'horloge à partir du cycle d'injection. La détection logicielle correspond quant à elle à l'écriture d'une valeur non nulle du compteur d'erreurs logiciel à une adresse spécifique en mémoire. En

conséquence le processeur embarqué vérifie les accès mémoires (adresse et donnée) après que la faute ait été injectée.

L'exécution du programme de cryptage AES se déroule en quatre phases : initialisation (ROM), positionnement de la clé (RAM), cryptage (RAM) et fin du programme avec « trap » à l'adresse 0x800 si le cryptage est correct. La validité du cryptage est obtenue par comparaison par le processeur Leon2 de la valeur cryptée avec la valeur attendue qui est prédéfinie et sauvegardée en RAM. Il faut noter que les fautes ne sont injectées que lors du positionnement de la clé ou lors du cryptage. Le programme de cryptage a été légèrement modifié pour détecter les cas où le programme se termine correctement mais avec une valeur cryptée différente de celle attendue. Ceci correspond à une erreur de donnée et dans ce cas le « trap » est effectué à l'adresse 0xff0. Si il y a une erreur durant l'exécution du programme le « trap » peut être à n'importe quelle adresse entre 0x0 et 0x1000 (excepté 0x800 et 0xff0) ou bien l'exécution peut ne pas se terminer par un « trap ».

L'exécution du programme AES original dure 17762 cycles d'horloges. Afin de détecter un éventuel retard dans l'exécution dû à l'injection d'une faute, s'il n'y a pas de trap à l'adresse 0x800 au cycle 17762 alors on laisse l'exécution se poursuivre jusqu'au cycle 18762. Sans mécanisme de détection les classes de faute sont les suivantes :

- silencieuse : trap à l'adresse 0x800 au cycle 17762,
- retard : trap à l'adresse 0x800 au cycle 18762,
- erreur : trap à l'adresse 0xff0 au cycle 18762,
- défaillance : trap à n'importe quelle adresse entre 0x0 et 0x1000 au cycle 18762,
- crash : pas de trap au cycle 18762.

La version protégée du programme AES dure plus longtemps que la version originale à cause des redondances qu'elle contient. Pour cette version le nombre de cycles nominal est 65608 et si nécessaire on laisse l'exécution se dérouler pendant 1000 cycles supplémentaires jusqu'au cycle 66608.

Lorsque des mécanismes de détection sont implémentés, au niveau logiciel ou au niveau matériel, chaque classe de faute est séparée en deux sous classes. La classe « Silencieuse » est décomposée en sous-classes « Silencieuse » et « Faux positif ». La sous-classe « Silencieuse » devient alors trap à l'adresse 0x800 sans détection de faute et « Faux positif » représente le cas avec trap à l'adresse 0x800 et détection de faute. Pour les autres classes, avec les mêmes conditions que ci-dessus, les sous-classes déterminent simplement si un mécanisme de détection a été activé ou non durant l'expérience. Par exemple la classe « Erreur » devient les sous-classes « Erreur détectée » (trap à l'adresse 0xff0 avec détection de faute) et « Erreur non détectée » (trap à l'adresse 0xff0 sans détection de faute).

6.3.3 Résultats sur les dispositifs évalués

Le Tableau 6-6 représente la classification obtenue pour l'injection de *bit-flips* simples dans les registres du pipeline sans aucun mécanisme de détection ni logiciel ni matériel.

Les taux de défaillance et de crash sont plus importants pour l'étage *Fetch* à cause des caractéristiques des cibles. En effet l'étage *Fetch* est composé de 30 bascules correspondant au « compteur de programme » et d'une bascule pour le signal de branchement. Il est indéniable qu'une faute injectée dans le compteur de programme a une très forte probabilité de conduire à un complet disfonctionnement du pipeline et donc du système. Le signal de branchement est également important mais peut être inactif au moment de l'injection. Peu de fautes n'ont pas d'impact sur le résultat de cryptage puisque seulement 13,4% des fautes sont silencieuses.

Les autres étages du pipeline ont un nombre plus important de bascules que l'étage *Fetch* et la plupart de ces cellules mémoires sont moins critiques que le compteur de programme. Ceci peut expliquer pourquoi la plupart des fautes sont classifiées comme « Silencieuse ». Cela signifie qu'il y a une grande probabilité qu'une faute dans l'un de ces étages n'ait pas d'influence sur le système. Moins de 10% des fautes injectées provoquent une défaillance ou un crash.

Tableau 6-6 : Résultats d'analyse du système « Leon2+AES » non protégé

Classification	Etage				
	FE	**DE**	**EX**	**ME**	**WR**
Nombre de FFs ciblées	31	70	150	144	135
Nombre de fautes injectées	5500	12000	27000	26000	24000
Durée de la campagne	~1/2h	~1h	~3h	~2h30	-2h30
Silencieuse	13,4%	74,4%	73,3%	90,3%	92,5%
Retard	14,0%	3,6%	4,7%	1,9%	1,0%
Erreur	11,2%	11,3%	16,1%	5,1%	3,4%
Défaillance	50,4%	7,2%	4,0%	1,9%	1,0%
Crash	11,0%	3,5%	1,9%	0,8%	2,1%

Pour le pipeline complet le pourcentage de fautes « Silencieuse » est proche de 76%. En d'autres termes 76% des fautes se produisant dans les registres du pipeline ne vont pas modifier le résultat de l'application. C'est un résultat important dans la perspective de l'évaluation du taux d'erreur au niveau système. Si l'on considère un taux de fautes transitoires (*SER*), obtenu de façon statique par exposition à un flux de particules par exemple, et l'application logicielle AES, le taux d'erreur au niveau système peut alors être estimé avec un facteur de déclassement non négligeable. En première approximation, si l'on considère que la probabilité de faute est la même pour toutes les bascules du pipeline alors le taux d'erreur fonctionnelle du système peut être réduit à 24% du *SER* statique.

A titre de comparaison le pourcentage de fautes « Silencieuse » pour un processeur Itanium exécutant une application tirée de SpecInt95 est proche de 80%-85% [NGUY-05]. Ce pourcentage pour le système basé sur Leon2 serait probablement plus important si les fautes avaient été injectées dans tout le processeur.

Les résultats obtenus avec la version protégée au niveau matériel sont disponibles dans le Tableau 6-7.

De la même façon que pour la version non protégée la classification correspondant à l'étage *Fetch* dépend largement du fait qu'il est principalement constitué du compteur de programme. Près de 97% des fautes qui y sont injectées le sont dans un registre protégé ce qui implique qu'elles sont massivement détectées.

En ce qui concerne la détection matérielle pour le pipeline complet, le poids des sous-classes « non détectées » (Silencieuse, Retard non détecté, Erreur non détectée, Défaillance non détectée et crash non détecté) est due à la stratégie de filtrage. Comme uniquement des *bit-flips* simples sont injectés ils devraient être détectés par tout mécanisme de détection basé sur la parité. Mais toutes les bascules d'un étage ne sont pas prises en compte dans le calcul de la parité de cet étage donc l'injection dans une bascule qui n'est pas prise en compte peut entraîner un cas « non détecté ». Le taux de cas « non détecté » dépend directement du pourcentage de bascules utilisées pour le calcul. Ainsi pour les étages *Fetch* et *Execute* respectivement 3,2% et 62,9% des signaux sont filtrés et les taux de « non détecté » sont respectivement de 2,9% et 63,9%.

Tableau 6-7 : Résultats d'analyse du système « Leon2+AES » avec protections matérielles

Classification	Etage				
	FE	**DE**	**EX**	**ME**	**WR**
Nb de FFs ciblées	31	75	159	145	140
FFs considérées par le calcul de la parité	30	42	57	55	48
Nombre de fautes injectées	11000	13500	29500	29500	25000
Durée de la campagne	~1h	~1h	~3h	~3h	-2h30
Silencieuse	2,1%	25,6%	43,6%	57,5%	58,8%
Faux positif	11,1%	50,6%	32,0%	33,1%	33,9%
Retard non détecté	0,8%	2,8%	3,7%	1,7%	0,8%
Retard détecté	13,0%	0,3%	0,5%	0,1%	0,0%
Retard non détecté	0,0%	9,9%	12,8%	4,7%	3,4%
Erreur détectée	11,5%	0,9%	2,1%	0,3%	0,1%
Défaillance non détectée	0,0%	6,0%	2,7%	0,9%	0,3%
Défaillance détectée	46,2%	0,7%	1,0%	0,9%	0,6%
Crash non détecté	0,0%	3,1%	1,0%	0,3%	0,7%
Crash détecté	15,3%	0,1%	0,6%	0,5%	1,4%

Le nombre important de « Faux positif » est également à noter. La stratégie de filtrage mise en œuvre consiste à filtrer les signaux qui ne sont pas définis à chaque cycle. On peut supposer que les signaux filtrés ont en moyenne un impact moins important sur le comportement du processeur pour l'application donnée. Néanmoins une grande partie des fautes injectées dans les bascules non filtrées, et détectées, n'entraîne pas de modification du résultat de cryptage. Ceci montre qu'une telle protection conduit à des activations inutiles des éventuelles procédures de recouvrement.

On peut remarquer qu'avec une stratégie de filtrage « Statique Maximal » toutes les fautes devraient en théorie être détectées ce qui entraînerait des taux de « Faux positif » encore plus

importants. L'approche « Dynamique » nécessite une phase d'analyse longue et complexe qui ne pourrait toutefois pas conduire à un taux de « Faux positif » moins important et qui n'est peut-être pas justifiée connaissant le taux de fautes silencieuses du système non durci.

Considérons maintenant les résultats de l'analyse du système avec logiciel durci qui sont présentés dans le Tableau 6-8.

La première chose que l'on peut noter est que la classification est cohérente avec celle obtenue pour le système original et pour le système avec détection matérielle.

Pour l'étage *Fetch* la seule différence concerne le taux de détection qui est plus faible qu'avec le calcul de parité ceci toujours à cause de l'architecture de l'étage. Le pourcentage de fautes qui ne sont pas détectées et qui entraînent un cryptage incorrect est proche de 63% ce qui est non négligeable. Il faut cependant rappeler que les techniques mises en œuvre ne visent pas la détection de fautes survenant dans les registres internes du pipeline. Ce taux de non détection n'est donc pas étonnant.

Tableau 6-8 : Résultats d'analyse du système « Leon2+AES » avec protections logicielles

Classification	Etage				
	FE	**DE**	**EX**	**ME**	**WR**
Nb de FFs ciblées	31	70	150	144	135
Nombre de fautes injectées	20000	46000	100000	95000	90000
Durée campagne	~3h	~15h	~37h	~37h	-32h
Silencieuse	18,8%	77,4%	82,2%	93,9%	94,6%
Faux positif	0,2%	0,0%	0,1%	0,0%	0,0%
Retard non détecté	16,7%	8,1%	10,3%	2,9%	1,6%
Retard détecté	0,2%	0,2%	0,2%	0,1%	0,0%
Erreur non détectée	5,9%	2,4%	2,7%	0,8%	0,5%
Erreur détectée	1,4%	0,5%	0,5%	0,2%	0,2%
Défaillance non détectée	39,8%	7,6%	2,6%	1,5%	1,0%
Défaillance détectée	0,0%	0,0%	0,0%	0,0%	0,0%
Crash non détecté	17,0%	3,8%	1,4%	0,6%	2,1%
Crash détecté	0,0%	0,0%	0,0%	0,0%	0,0%

Pour les quatre autres étages du pipeline deux aspects intéressants peuvent être soulignés : le nombre de « Faux positif » et le nombre de fautes « Silencieuse ».

Pour les quatre derniers étages très peu de fautes (moins de 0,2%) sont classifiées « Faux Positif » ce qui signifie que le mécanisme de détection n'est pas activé quand ce n'est pas nécessaire. Comparée à la version non protégée du programme AES il y a un plus grand nombre de fautes qui n'empêchent pas le système d'effectuer un cryptage correct. La version durcie du programme inclut le code de cryptage lui-même (1/4 du programme) mais également une part importante de code de détection (3/4 du programme). Il est donc très probable que l'injection de faute ait lieu durant l'exécution de la partie détection ce qui n'a pas d'impact sur le cryptage et au pire retarde la fin du programme.

L'analyse précédente peut être étendue à tout le pipeline et aux trois versions du système. La Figure 6-4 montre la comparaison des « taux de résultat correct » qui incluent tous les cas où le cryptage est correct et au pire retardé.

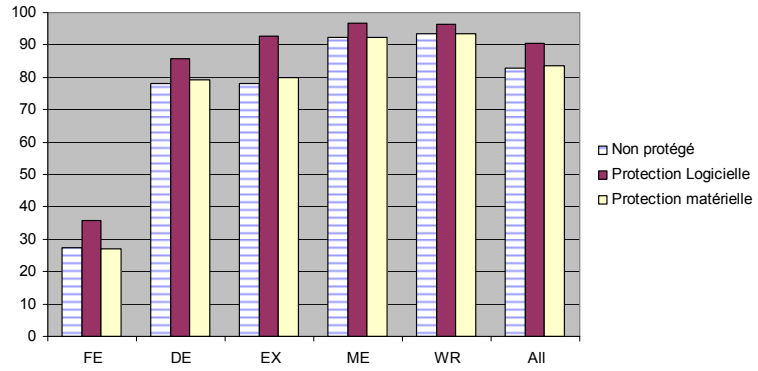

Figure 6-4 : Taux de résultat correct pour les systèmes « Leon2+AES » en fonction de l'étage du pipeline où survient l'erreur

Le « taux de résultat correct » correspond aux classes « Silencieuse », « Retard » et à toutes les sous-classes correspondantes. Cette comparaison met en avant le fait que le système dans sa version protégée au niveau logiciel est probablement plus robuste que les autres versions mais cela doit être corrélé avec le nombre de cycles plus important pour cette version.

6.3.4 Temps de campagnes

Pour compléter la comparaison des approches avec et sans DMA il est nécessaire de prendre en compte un système qui nécessite une mémoire externe et qui exécute une application significative comme c'est le cas pour le processeur Leon2 exécutant l'application AES. Le Tableau 6-9 permet de comparer le temps d'exécution de l'application (17762 cycles d'horloge) par le processeur en fonction de l'approche. Pour l'approche « Pas à Pas » la mémoire de Leon2 est implantée dans la mémoire du PowerPC alors qu'en mode « autonome » avec DMA l'interface d'injection se charge des accès mémoire. Les valeurs présentées sont pour l'exécution complète de l'application (sans injection de faute) et on voit que l'utilisation du DMA apporte une accélération conséquente.

L'accélération apportée en pratique par le mode « autonome avec DMA » est cohérente avec celle évaluée pour le mode « autonome avec ROM émulée ». En effet la Figure 5-5 montre que l'on peut obtenir pour ce dernier une accélération de l'ordre de 200.

Tableau 6-9 : Accélération apportée par le DMA

	Pas à Pas uniquement	**Autonome avec DMA**	**Accélération**
Temps d'exécution	1,316 s	0,041 s	32,1

Le Tableau 6-10 donne l'évaluation prédictive des temps d'exécution pour la campagne d'injection dans l'étage *Decode* du pipeline de Leon2 sans détection matérielle ou logicielle.

Tableau 6-10 : Evaluation prédictive pour l'analyse du processeur Leon2 avec application logiciel AES

Tâche	Niveau d'exécution/Configuration	Durée (ms)
Génération des vecteurs d'entrées	PC hôte via RAM (programme logiciel)	$3{,}59\ 10^{4}$
Génération de la liste de fautes	Processeur embarqué	$1{,}24\ 10^{3}$
Expérience de référence	Classification par comparaison logicielle	$6{,}02\ 10^{2}$
Expériences avec injection	Mode mixte avec chargement parallèle de la cible	$1{,}36\ 10^{6}$
Récupération des sorties & Analyse	Classification classique & logicielle	$2{,}81\ 10^{6}$
Récupération des résultats d'analyse	Classification classique & logicielle	$8{,}00\ 10^{-1}$
Campagne complète	-	$4{,}22\ 10^{6}$
Expérience		351,3

En pratique la campagne dure 4024 secondes (environ 1 heure) ce qui donne une durée par expérience de 335 millisecondes. Contrairement aux exemples du chapitre précédent l'évaluation prédictive donne une durée d'expérience évaluée supérieure à la durée d'expérience effective obtenue lors de la campagne. Ceci s'explique par les optimisations logicielles qui prennent plus d'importance compte tenu de la taille du projet. Par exemple le nombre de sorties à récupérer est ici plus important. Or deux accès consécutifs (sur 32 bits au niveau logiciel) aux parties hautes et basses (MSB et LSB) d'un registre de l'interface (64 bits) peuvent être optimisés en un seul accès sur 64 bits puisque dans le cas de l'analyse du Leon2 le bus a une taille de 64 bits. Les valeurs restent cependant très proches. L'évaluation prédictive permet donc d'obtenir l'ordre de grandeur de la durée d'une campagne pour un système complexe

En ce qui concerne la durée des campagnes on peut à nouveau observer que l'environnement que nous présentons accélère considérablement l'analyse de sûreté. Le Tableau 6-11 donne l'accélération obtenue par rapport à la simulation RTL pour trois modes de fonctionnement. La durée pour le mode « autonome » uniquement est extrapolée à partir de la durée d'une expérience en mode « pas à pas » et de l'accélération apportée par le DMA (Tableau 6-9). Ce mode de fonctionnement est possible si l'on analyse l'état du Leon2 uniquement à la fin de chaque expérience.

Le mode mixte est celui qui a été mis en œuvre pour les campagnes ci-dessus et dans ce cas l'émulation avec l'environnement que nous proposons est 180 fois plus rapide que la simulation RTL. On note également qu'en mode autonome uniquement il est possible de réduire la durée de la campagne de 3 ordres de grandeurs.

Tableau 6-11 : Accélération par rapport à la simulation RTL

	Simulation RTL	Emulation mode « pas à pas »	Emulation mode « mixte »	Emulation mode « autonome » uniquement
Durée d'une expérience (s)	64	1,791	0,351	0,055
Accélération		~35	~180	~1100

La simulation RTL indiquée dans le Tableau 6-11 est effectuée avec le simulateur Modelsim 6.0 sur une station SunBlade 2500 avec double processeur 1,6GHz et 2 GigaOctets de mémoire vive.

L'analyse du processeur Leon2 par simulation RTL (ModelSim) a été effectuée et présentée dans [TOUL-07]cesseur 1,6GHz et 2 GigaOctets de mémoire vive.

L'analyse du processeur Leon2 par simulation RTL (ModelSim) a été effectuée et présentée dans [TOUL-07]. Les conditions d'analyse sont globalement les mêmes et les résultats d'analyse publiés sont proches des notres, ce qui est tout à fait logique. Nous pouvons juste remarquer que le nombre de fautes injectées pour l'analyse avec simulation RTL est significativement plus faible que le nombre de fautes que nous avons injectée 3000 fautes ont été injectées par étage alors que nous avons injecté jusqu'à 100000 fautes dans l'étage *Execute* pour le système avec protection logicielle. Les durées des expériences ne sont pas spécifiquement reportées dans l'article.

6.3.5 Bilan sur l'utilisation de l'environnement développé

L'environnement que nous proposons permet d'effectuer l'analyse de sûreté d'un système complexe comprenant du matériel et du logiciel. Grâce à un contrôle optimisé des injections, l'accélération par rapport à la simulation RTL est proche de 180. Si l'on considère que la configuration de l'environnement jusqu'au téléchargement du *bitstream* prend 1 heure alors l'analyse du système non durci avec injections dans le pipeline entier nécessite moins de 16 heures.

Au-delà de l'accélération apportée les résultats donnent un aperçu du niveau de robustesse intrinsèque du système sans aucun durcissement. En considérant une tâche spécifique à exécuter (i.e. le programme de cryptage AES) et un retard maximum inférieur à 6%, une faute se produisant dans un registre du pipeline du processeur a une probabilité moyenne de 76% de ne pas conduire à un résultat erroné. Cette caractéristique du système permet deux choses :

- Estimer le taux d'erreur au niveau système.
- Poser les bases de la réflexion sur l'utilité d'un quelconque mécanisme de protection.

Si la robustesse intrinsèque du système est estimée insuffisante alors des mécanismes de protection peuvent être ajoutés à la description de haut niveau ou bien dans la partie logicielle. L'environnement offre alors la possibilité de rapidement déterminer les avantages et les inconvénients de ces mécanismes de protection.

6.4 Conclusion

La configuration de l'environnement est une étape importante dans un flot d'analyse à base de prototypage. Elle l'est d'autant plus quand on rajoute un niveau d'exécution des tâches de l'analyse comme c'est le cas pour notre approche. L'instrumentation du circuit à analyser est également une étape qui peut être gourmande en temps et en partie annuler les bénéfices apportés par l'optimisation

de l'environnement. Nous avons donc automatisé l'instrumentation et la génération des sources matériels et logiciels devant être inclus dans l'environnement d'analyse.

Cette automatisation a été appliquée pour l'analyse d'un IP de cryptage AES et pour l'analyse d'un système basé sur le processeur Leon2 exécutant une application logicielle AES. Ces études de cas montrent les performances de l'environnement pour l'analyse de systèmes significatifs.

Pour l'analyse de l'IP AES un grand nombre de fautes a été injecté en un temps réduit ce qui a permis la validation de différents mécanismes de détection. Un des intérêts de l'environnement pour l'analyse de circuits liés à la sécurité est la possibilité d'injecter des fautes avec une grande multiplicité spatiale. Ceci permet d'étudier la réponse d'un circuit à des attaques intentionnelles qui peuvent être caractérisées par une multiplicité spatiale importante.

Le système basé sur le processeur Leon2 a permis la mise en œuvre de l'environnement pour l'analyse d'un système significatif tant au niveau matériel qu'au niveau logiciel. L'analyse de trois versions du système (non protégé, protégé au niveau matériel, protégé au niveau logiciel) donne la robustesse de chacune d'elle pour des fautes se produisant dans l'unité entière du processeur. On en déduit en particulier la robustesse intrinsèque du système non protégé et la valeur du facteur de déclassement qui est de 24%.

Pour les deux cas d'études la durée des campagnes est significativement inférieure à celle qui serait nécessaire pour le même type d'analyse à partir des simulations RTL.

En conclusion ce que nous pouvons retenir de l'analyse des deux circuits est que :

- l'environnement permet l'injection de fautes avec une grande multiplicité ce qui est très intéressant du point de vue sécurité,
- l'environnement offre la possibilité de valider ou de nuancer la pertinence d'un FIT technologique en fonction de l'application,
- le temps nécessaire pour effectuer l'analyse est relativement court et grandement réduit par rapport à la simulation RTL.

Conclusion et perspectives

La conception des circuits intégrés sûrs exige de plus en plus de techniques et d'outils pour une évalutation de leur niveau de robustesse. A cause de la réduction des dimensions les circuits sont désormais sensibles aux attaques naturelles au niveau de la mer. De plus il faut considérer les attaques intentionnelles sur les circuits liés à la sécurité. L'analyse des circuits utilisés dans des applications critiques est donc indispensable. Nous avons opté pour une analyse tôt dans le flot de conception afin d'offrir le choix de la réponse à apporter et éventuellement de réduire le temps de reprise de la description originale (insertion de mécanismes de protection au niveau RTL).

L'analyse des phénomènes physiques mis en jeu nous a d'abord permis de définir les modèles de fautes les plus représentatifs. Ceux-ci sont basés sur les phénomènes extérieurs liés à des attaques intentionnelles ou non (impact de particules, laser…) et sur les phénomènes liés à la perte de l'intégrité du signal. 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 et ce qu'il était possible d'améliorer.

A partir du travail ci-dessus et des objectifs que nous souhaitions atteindre nous avons ensuite défini un environnement à base de prototypage et certaines techniques afin de remédier aux faiblesses des approches existantes.

Nous avons choisi le prototypage parce qu'il permet une analyse à partir d'une description à haut niveau et parce qu'il permet d'accélérer les campagnes d'injection par rapport à la simulation RTL. Les approches à base de simulation RTL induisent en effet des temps d'expérience très importants. L'exécution de certaines tâches au niveau matériel s'accompagne d'une accélération mais aussi d'une diminution de la flexibilité. C'est pour cette raison que nous avons proposé d'utiliser un processeur embarqué comme niveau intermédiaire entre l'ordinateur hôte et la logique programmable. Nous conservons ainsi un certain niveau de flexibilité en limitant les transferts entre le PC et la plateforme de prototypage. Les trois niveaux qui composent l'environnement présentent chacun des caractéristiques différentes qui permettent d'obtenir le meilleur compromis entre la complexité de la tâche à effectuer et les performances obtenues lors de son exécution. L'environnement offre en outre une grande flexibilité à plusieurs égards : circuits ciblés, implantation de l'environnement, niveau et type d'instrumentation et type d'analyse.

L'utilisation d'une version instrumentée du circuit à analyser permet de s'affranchir de l'implémentation d'une version de référence et donc de conserver un maximum de logique programmable disponible. Néanmoins l'instrumentation a un coût matériel qu'il faut minimiser. Les techniques d'instrumentation que nous avons proposées visent à limiter le plus possible ce coût matériel et s'adaptent aux types de bascules ciblées. En complément nous proposons une nouvelle approche pour le contrôle des injections qui est caractérisée par un chargement parallèle de données d'injection et par le codage de ces données. Les avantages des techniques que nous proposons ont été

discutés et validés à l'aide de cas concrets. Lorsque cela était possible nous avons comparé notre approche aux approches existantes comme par exemple pour le coût matériel lié à l'instrumentation.

La flexibilité de la méthodologie au niveau du type de circuit, du type d'analyse ou bien de la plateforme d'implantation entraine une grande variabilité de la durée des campagnes d'injection qui est un paramètre important de l'analyse de sûreté d'un point de vue utilisateur. Afin de le renseigner sur cette durée en fonction de ses choix, nous avons prêté une grande attention au calcul prédictif du temps d'exécution de chaque tâche. Cette prédiction permet à l'utilisateur d'évaluer la durée nécessaire pour une campagne d'injection donnée en fonction des paramètres du circuit analysé, des caractéristiques de la plateforme d'implantation et des options retenues (type d'analyse, répartition des tâches etc). Ceci peut en outre guider l'utilisateur dans ses choix notamment en ce qui concerne la plateforme de prototypage. Pour une analyse donnée il peut évaluer les caractéristiques à optimiser afin qu'elle soit la plus rapide possible.

Nous avons appliqué la méthodologie pour l'analyse de sûreté de deux cas d'étude : une IP de cryptage AES et un système basé sur le processeur Leon2 exécutant une application logicielle de cryptage AES. Les résultats obtenus ont permis de comparer les mécanismes de protection intégrés à l'IP de cryptage. L'analyse du système basé sur le processeur Leon2 a quant à elle permis d'évaluer la robustesse de ce système et de comparer deux approches pour la détection d'erreur, l'une matérielle, l'autre logicielle.

Une caractéristique importante de l'environnement est qu'il permet d'obtenir la robustesse intrinsèque d'un système matériel/logiciel tôt dans le flôt de conception et en un temps relativement court. L'analyse croisée entre le SER statique et le niveau de sûreté de fonctionnement dynamique augmente alors la pertinence de l'évaluation du niveau de robustesse obtenu.

Ce travail de thèse offre plusieurs perspectives, d'un point de vue développement d'une part, en terme de recherche d'autre part.

L'automatisation complète du flot d'analyse, de la description à haut niveau d'abstraction jusqu'aux résultats de l'analyse, est l'un des premiers axes de développement afin de permettre une prise en main par l'utilisateur encore plus simple.

A moyen terme il est envisageable d'augmenter le niveau de complexité du système analysé. Les plateformes de prototypage actuelles offrent une quantité de ressources importante (logique programmable, mémoire embarqué) qui pourrait être plus utilisée. Nous pouvons par exemple penser à l'analyse de mécanismes de recouvrement implantés au niveau d'un système d'exploitation exécuté sur le processeur Leon2.

Un des avantages de l'approche est sa flexibilité or pour le moment les expériences n'ont été menées que sur une seule plateforme. Nous pourrions donc implanter l'environnement d'analyse sur un émulateur matériel industriel plus complexe afin de valider le portage de l'environnement et d'observer les performances qu'il est possible d'obtenir. Ce type d'émulateur peut également être mis

en œuvre pour augmenter la complexité du système analysé ; SoC avec co-processeur, SoC multiprocesseurs, etc.

Pour l'analyse des périphériques de processeur un des objectifs serait de mettre en œuvre deux processeurs : le premier pour le fonctionnement du périphérique et le second pour l'injection de fautes. Il faudrait alors veiller à la bonne synchronisation de l'ensemble. Certaines plateformes incluent deux cœurs de processeur et il serait intéressant de les mettre en œuvre simultanément.

En terme de recherche nous avons vu que l'injection de 1% des fautes possibles pour les systèmes complexes n'est peut-être pas nécessaire et que l'injection d'un nombre beaucoup plus faible de fautes reste pertinent. Il serait donc utile de mettre en oeuvre une technique de réduction de la liste de fautes afin de réduire encore la durée des expériences

Il serait également intéressant de comparer les résultats d'une analyse à plus haut niveau aux résultats obtenus avec l'environnement proposé. Cela permettrait de rapidement valider des méthodes d'analyse plus probabilistes.

Enfin l'axe de recherche le plus intéressant est peut-être l'utilisation de l'environnement pour l'analyse de systèmes complexes par composition. L'analyse par composition correspond à l'analyse de systèmes composés de plusieurs éléments en procédant à l'analyse de chacun de ces éléments indépendamment. En effectuant l'analyse des éléments d'un système puis l'analyse du système complet, notre approche pourrait permettre de mettre en avant les relations entre la robustesse intrinsèque de chaque élément et le niveau de robustesse du système complet.

Bibliographie

[ABBA-04] M. Abbas, M. Ikeda, K. Asada, "Noise Effects on Performance of Low Power Design Schemes in Deep Submicron Regime", 19th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'04), Cannes, France, pp. 87-95, Octobre 2004.

[ALEX-04] Alexandrescu, L. Anghel, M. Nicolaidis, "Simulating Single Event Transients in VDSM ICs for Ground Level Radiation", Journal of Electronic Testing: Theory and Applications (JETTA), pp. 413-421, Août 2004.

[AMMA-05] A. Ammari, K. Hadjiat, R. Leveugle, "Combined fault classification and error propagation analysis to refine RT-level dependability evaluation", J. Electron. Test. Theory and Applications, vol. 21, pp. 365-376, 2005.

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

[ARLA-90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.C. Laprie, E. Martins, D. Powell, "Fault Injection For Dependability Validation – A Methodology and Some Applications", IEEE Trans. on Software Engineering, vol. 16, pp. 166-182, Février 1990.

[ATME-06] AT697E Datasheet. www.atmel.com, dernière mise à jour: Février 2006.

[BAI-01] X. Bai, S. Dey, "High Level Crosstalk Defect Simulation for System On-Chip Interconnects", Proc. 19th VLSI Test Symposium (VTS'01), Los Angeles, USA, Avril 2001.

[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.

[BENS-98] A. Benso, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "EXFI: A Low-cost Fault Injection System for Embedded Microprocessor-based Boards", ACM Trans. on Design Automation of Electronic Systems (TODAES), vol. 3, issue 4, pp.626-634, Octobre 1998.

[BERT-03] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard". IEEE Transactions on Computers, vol. 52, No. 4, pp. 493-505, ISSN 0018-9340, Avril 2003.

[BERR-02] L. Berrojo, I. Gonzales, F. Corno, M. Sonza-Reorda, G. Squillero, L. Entrena, C. Lopez, "New Techniques for Speeding Up Fault Injection Campaigns", Design, Automation and Test in Europe Conference (DATE'02), Paris, France, pp. 847-852, Mars 2002.

[BOLC-01] C. Bolchini, L. Pomante, F. Salice, D. Sciuto, “Reliability Properties Assessment at System Level: a Co-Design Framework”, Proc. 7th Int. On-Line Testing Workshop (IOLTW’01), Taormina, Italie, pp. 165-171, Juillet 2001.

[BONE-98] D. Boneh, R.A. DeMillo, R.J. Lipton, “On the Importance of Checking Cryptographic Protocols for Faults”, Lecture Notes in Computer Science, Advances Cryptology, Proc. of EUROCRYPT’97, pp. 37-51, 1997.

[BOUE-98] J. Boué, P. Pétillon, 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.

[CARR-98] J. Carreira, H. Madeira, J.G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", IEEE Trans. on Software Engineering, vol. 24, pp. 125-136, Février 1998.

[CHA-96] H. Cha, E.M. Rudnick, J.H. Patel, R.K. Iyer, G.S. Choi, “A Gate-Level Simulation Environment for Alpha-Particle-Induced Transient Faults”, IEEE Trans. on Computers, vol. 45, issue 11, pp. 1248-1256, Novembre 1996.

[CHAN-97] Y. Chang, S.K. Gupta, M.A. Breuer, "Analysis Of Ground Bounce In Deep Sub-Micron Circuits", Proc. of the 15th IEEE VLSI Test Symposium (VTS'97), Mai 1997.

[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.

[CHEN-02] L. Chen, X. Bai, S. Dey, "Testing for Interconnect Crosstalk Defects Using On-Chip Embedded Process Cores", Journal of Electronic Testing: Theory and Applications (JETTA), vol 18, pp. 529-538, 2002.

[CHEY-00] P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M. Sonza Reorda, M. Violante, “Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors”, IEEE Trans.on Nuclear Science, Vol. 47, n° 6, pp. 2231-2236, Décembre 2000.

[CIVE-01a] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting FPGA for accelerating fault injection experiments", Proc. 7th Int. On Line Testing Workshop (IOLTW’01), Taormina, Italie, pp. 9-13, Juillet 2001.

[CIVE-01b] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "FPGA-based Fault Injection for Microprocessor Systems", 10th Asian Test Symp. (ATS’01), Kyoto, Japon, p. 3004, Novembre 2001.

[CIVE-03] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, “New Techniques for Efficiently Assessing Reliability of SoCs”, Microelectronics Journal, vol 34, issue 1, pp. 53-61, Janvier 2003.

[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.

[DELO-96] T.A. Delong, B.W. Johnson, J.A. Profeta, "A Fault Injection Technique for VHDL Behavioral-Level Models", IEEE Design and Test of Computers, pp.24-33, hiver 1996.

[DUZE-00] S. Duzellier, D. Falguère, 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, Décembre 2000.

[EJLA-04] A. Ejlali, S. Ghassem Miremadi, "FPGA-based Fault Injection into Switch-level Models", Elsevier Jounal of Microprocessors and Microsystems, vol. 28, issue 5-6, pp.317-327, Août 2004.

[FOGL-04] A.D Fogle, D. Darling, R.C. Blish, G.Daszko, "Flash Memory under Cosmic & Alpha Irradiation", IEEE Trans. on Device and Materials Reliability, vol. 4, n° 3, pp. 371-376, Septembre 2004.

[GAIS-02] J. Gaisler, "A Portable and Fault Tolerant Microprocessor Based on the SPARC V8 Architecture", Proc. of the 2002 Int. Conf. on Dependable Systems and Network (DSN 2002), pp. 409-415, 2002.

[GAIS-06] www.gaisler.com

[GARC-06] M. Garcia-Valderas, M. Portela-García, C. López-Ongil, L. Entrena, "An Extension of Transient Fault Emulation Techniques to Circuits with Embedded Memories", 9th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'06), Prague, République Tchèque, pp. 218-219, Avril 2006.

[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.

[GASI-02] G. Gasiot, V. Ferlet-Cavrois, J. Baggio, P. Roche, P. Flatresse, A. Guyot, P. Morel, O.Bersillon, J.du Port de Pontcharra, "SEU sensitivity of Bulk and SoI Technologies to 14-MeV Neutrons", IEEE Trans. on Nuclear Science, vol. 49, n°6, Décembre 2002.

[GOLO-03] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Soft-error Detection Using Control Flow Assertions", IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 581-588, 2003.

[GRAC-01] J. Gracia, J. C. Baraza, D. Gil, P. J. Gil, "Comparison and Application of different VHDL-Based Fault Injection Techniques", IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'01), San Francisco, USA, Octobre 2001.

[GUI-02] P. Gui "VLSI Design and Lab Fall 2004: Timing Issues", Adapted from the book Digital Integrated Circuits, J. Rabaey, 2002.

[GUNN-89] U. Gunneflo, J. Karlsson, J. Torin, "Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation", 19th Symp. Fault-Tolerant Computing (FTC-19), pp. 340-347, 1989.

[GUTI-04] D. Gonzáles Gutiérrez, "Single Event Upsets Simulation Tool Functional Description", ESA document, Juillet 2004.

[HADD-05] N. Haddad, C. Hatfield, R. McPeack, S. Shaffer, M. Shoga, "SEU Sensitivity of an Advanced 0.13µm SoI Microprocessor", 8th European Conference on Radiation and its Effects on Components and Systems (RADECS'05), Cap D'Agde, France, Septembre 2005

[HADJ-05] K. Hadjiat, “Evaluation Predictive de la Sûreté de Fonctionnement d’un Circuit Intégré Numérique”, Thèse de Doctorat, Laboratoire TIMA, INPG Grenoble, Juin 2005.

[HARE-01] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walsta, C. Dai, “Impact of CMOS Process Scaling and SoI on the Soft Error Rates of Logic Processes”, Symposium on VLSI Technology Digest of Technical Papers, pp. 73-74, Kyoto, Japon, 2001.

[HAZU-00] P. Hazucha, C. Svensson, "Impact of CMOS Technology Scaling on the] Atmospheric Soft Error Rate", IEEE Trans. on Nuclear Science, vol. 47, issue 6, Décembre 2000.

[HEID-05] W. Heidergott, “SEU Tolerant Device, Circuit and Processor Design”, Proceedings of the 42th Annual Design Automation Conference (DAC’05), Anaheim, USA, pp.5-10, Juin 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.

[IROM-05] F. Irom, F. Farmanesh, “Single-Event Upset and Scaling Trends in Commercial SoI PowerPC Microprocessors”, 8th European Conference on Radiation and its Effects on Components and Systems (RADECS’05), Cap D’Agde, France, Septembre 2005.

[ITC-99] http://www.cad.polito.it/tools/itc99.html.

[JEDE-01] JEDEC Standard, "Measurement and Reporting of Alpha particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices", JESD89, Août 2001.

[JEDE-05] JEDEC Standard, "Test Method for Beam Accelerated Soft Error Rate", JESD89-3, Septembre 2005.

[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.

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

[KANA-92] G.A. Kanawati, N.A. Kanawati, J.A. Agraham, "FERRARI: a tool for the validation of system dependability properties", Proc. 22nd Symp. on Fault-Tolerant Computing (FTCS-22), pp. 336-344, 1992.

[KARL-91] J. Karlsson, U. Gunneflo, P. Lidén, J. Torin "Two Fault Injection Techniques to Test of Fault Handling Mechanisms", Proc. of IEEE Int. Test Conference (ITC’91), pp. 140-149, 1991.

[KARL-95] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, "Comparison And Integration Of Three Diverse Physical Fault Injection Techniques", Predictably Dependable Computing Systems, pp. 309-327, 1995.

[KARN-04] T. Karnik, P. Hazucha, J. Patel "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes", IEEE Trans. on Dependable and Secure Computing, vol 1, n°2, Juin 2004.

[KOGA-96] R. Koga, "Single Event Effect Ground Test Issues", IEEE Trans. on Nuclear Science, vol. 43, issue 2, pp. 661-670, Avril 1996.

[LAJO-00] M. Lajolo, M. Rebaudengo, M. Sonza-Reorda, M. Violante, L. Lavagno, "Evaluating System Dependability in a Co-Design Framework", Proc. of IEEE Design, Automation and Test in Europe (DATE'00), Paris, France, pp. 586-590, Mars 2000.

[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-03a] R. Leveugle, L. Antoni, "Dependability Analysis: A New Application for Run-Time Reconfiguration", Int. Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, pp. Avril 2003.

[LEVE-03b] R. Leveugle, K. Hadjiat, "Multi-Level Fault Injections in VHDL Descriptions: Alternative Approaches and Experiments", Journal of Electronic Testing: Theory and Applications (JETTA), vol. 19, pp. 559-575, Octobre 2003.

[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.

[LOPE-05] C. López-Ongil, M. García-Valderas, M. Portela-García, L. Entrena-Arrontes, "Autonomous Transient Fault Emulation on FPGAs for Accelerating Fault Grading", 11th IEEE Int. On-Line Testing Symposium (IOLTS'05), Saint-Raphael, France, pp. 43-45, Juillet 2005.

[MADE-94] H. Madeira, M. Rela, F. Moreira, J.G. Silva, "RIFLE: A General Purpose Pin-Level Fault Injector", Proc. First European Dependable Computing Conference, pp.199-216, 1994.

[MAIS-07a] P.Maistri, P. Vanhauwaert, R.Leveugle, "Evaluation of Register-Level Protection Techniques for the Advanced Encryption Standard by Multi-Level Fault Injections", Proc. 22nd Int. Symp.on Defect and Tolerance in VLSI Systems (DFT'07), Rome, Italie, pp. 499-507, Septembre 2007.

[MAIS-07b] P.Maistri, P. Vanhauwaert, R.Leveugle, "A Novel Double-Data-Rate AES Architecture Resistant against Fault Injection", 4th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'07), Vienne, Autriche, pp. 54-61, Septembre 2007.

[MANG-03] S. Mangard, M. Aigner, S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," IEEE Transactions on Computers, vol. 52, no. 4, pp. 483-491, Avril 2003.

[MART-99] R.J. Martinez, P.J. Gil, G. Martin, C. Pérez, J.J. Serrano, "Experimental Validation of high-Speed Fault Tolerant Systems Using Physical Fault Injection", Dependable Computing for Critical Applications, San Jose, Californie, p. 249, Janvier 1999.

[MCNU-99] P.J. McNulty, L.Z. Scheick, D.R. Roth, M.G. Davis, M.R.S. Tortora, “First Failure Predictions for EPROMs of the Type Flown on the MPTB Satellite”, IEEE Trans. on Nuclear Science, vol. 47, n°6, Décembre 2000.

[MICH-94] T. Michel, R. Leveugle, G. Saucier, R. Doucet, P. Chapier, "Taking Advantage of ASICs to Improve Dependability with Very Low Overheads", in European Design and Test Conference, pp. 14-18, 1994.

[NGUY-05] H. T. Nguyen, Y. Yagil, N. Seifert, M. Reitsma, “Chip-Level Soft Error Estimation Method”, IEEE Trans. On Device and Materials Reliability, vol. 5, n° 3, pp. 365-381, Septembre 05.

[NORM-96] E. Normand, “Single Event Upset at Ground Level”, IEEE Transactions on Nuclear Science, vol. 43, n°6, Décembre 1996.

[PORTe-04] M. Portela-Garcia, C. lópez-Ongil, M. García-Valderas, L. Entrena-Arrontes, "Analysis of Transient Fault Emulation Techniques in Platform FPGAs", XIX Conf. on Design of Circuits and Integrated System (DCIS’04), Bordeaux, France, Novembre 2004.

[PORTo-06] M. Portolan, R. Leveugle, “A Highly Flexible Hardened RTL-Processor Core Based on Leon2”, IEEE Trans. on Nuclear Science, vol. 53, n°4, pp. 2069-2075, Août 2006.

[PORTo-07] M. Portolan, “Conception d’un Système Embarqué Sur et Sécurisé”, Thèse de Doctorat, Laboratoire TIMA, INPG Grenoble, Janvier 2007.

[ROTH-05] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, “A Smart Card Test Environment Using Multi-Level Fault Injection in SystemC”, 6th Latin-American Test Workshop (LATW’05), Salvador, Brésil, pp. 103-108, Mars 2005.

[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.

[SEGA-88] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, T. Lin, "FIAT – Fault Injection Based Automated Test Environment", Proc. 18th Symp. on Fault Tolerant Computing (FTCS-18), Tokyo, Japan, pp. 102-107, June 1988.

[SEIF-02] N. Seifert, X. Zhu, L. W. Massengill, "Impact of Scaling on Soft-Error Rates in Commercial Microprocessors", IEEE Trans. on Nuclear Science, vol. 49, n° 6, Décembre 2002.

[SIEH-97] V. Sieh, O. Tschäche, F. Balbach, “Comparing Different Fault Models Using VERIFY”, Proc. 6th Conf. on Dependable Computing for Critical Applications, Grainau, Germany, pp. 59-76, Mars 1997.

[SHIV-02] P. Shivakumar, M. Kistler, "Modeling the Effect Of Technology Trends on the Soft Error Rate of Combinational Logic", Proc. of 2002 Int. Conference on Dependable Systems And Networks, pp. 389-398, 2002.

[SUBH-05] M. Subhasish, T. Karnik, N. Seifert, M. Zhang, “Logic Soft Errors in sub-65nm Technologies Design and CAD Challenges”, Proceedings of the 42nd Annual Design Automation Conference (DAC’05), Anaheim, USA, pp. 2-4, Juin 2005.

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

[TOUL-07] E. Touloupis, J.A. Flint, V.A. Chouliaras, D.D. Ward, “Study of the Effects of SEU-Induced Faults on Pipeline-Protected Microprocessor”, IEEE Trans. On Computers, vol. 56, n°12, Décembre 2007.

[VARG-00] F. Vargas, A. Armory, R. Velazco, "Fault Tolerance in VHDL Description: Transient Fault Injection & Early Reliability Estimation", 9th Asian Test Symp., Taipei, Taiwan, p. 417, Décembre 2000.

[VARG-05] F. Vargas, D.L. Cavalcante, E. Gatti, D. Prestes, D. Lupi, “On the Proposition of an EMI-Based Fault Injection Approach”, 11th IEEE Int. On-Line Testing Symposium (IOLTS’05), Saint-Raphael, France, pp. 207-208, July 2005.

[VELA-00] R. Velazco, S. Rezgui, R. Ecoffet, "Predicting Error Rate for Microprocessor-Based Digital Architecture through C.E.U (Code Emulating Upsets) Injection", IEEE Trans. Nuclear Science, vol. 47, n°6, pp. 2405-2411, Décembre 2000.

[ZARA-03] H.R. Zarandi, S.G. Miremadi, A. Ejlali, “Fault Injection in Verilog Models for Dependability Evaluation of Digital Systems”, Proc. 2nd Int. Symp. on Parallel and Distributed Computing (ISPDC’03), Ljubljana, Slovénie, pp. 281-287, Octobre 2003.

Publications obtenues pendant la thèse

Conférences Internationales

[C.I.1] L. Anghel, R. Leveugle, P. Vanhauwaert, "Evaluation of SET and SEU effects at multiple abstraction levels", 11th IEEE Int. On-Line Testing Symposium (IOLTS'05), Saint-Raphael, France, pp. 309-312, Juillet 2005.

[C.I.2] P. Vanhauwaert, R. Leveugle, P. Roche, "A Flexible SoPC-based Fault Injection Environment", IEEE workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'06), Prague, République Tchèque, pp. 190-195, Avril 2006.

[C.I.3] P. Vanhauwaert, R. Leveugle, P. Roche, "Dependability Analysis Performance Evaluation of Environment Configurations", IEEE conf. Design and Test of Integrated Systems in Nanoscale Technology (DTIS'06), Tunis, Tunisie, pp. 335-340, Septembre 2006.

[C.I.4] P. Vanhauwaert, R. Leveugle, P. Roche, "Reduced Instrumentation and Optimized Fault Injection Control for Dependability Analysis", IFIP Int. Conference on Very Large Scale Integration (VLSI-SoC'06), Nice, France, pp. 391-396, Octobre 2006.

[C.I.5] P. Maistri, P. Vanhauwaert, R. Leveugle, "A Novel Double-Data Rate AES Architecture Resistant Against Fault Injection", IEEE workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'07), Vienne, Autriche, pp. 54-61, Septembre 2007.

[C.I.6] P. Maistri, P. Vanhauwaert, R. Leveugle, "Evaluation of Register-Transfer Level Protection Techniques for the AES Standard by Multi-Level Fault Injections", IEEE Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'07), Rome, Italie, pp. 499-507, Septembre 2007.

[C.I.7] (Actuellement Soumis) P. Vanhauwaert, M. Portolan, R. Leveugle, P. Roche, "Usefulness and Effectiveness of HW and SW Protection Mechanisms in a Processor-Based System", 15th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS'08), Malta, Septembre 2008.

Conférences Nationales

[C.N.1] P. Vanhauwaert, R. Leveugle, "Environnement d'Analyse de Sûreté sur SoPC", Actes des Journées Nationales du Réseau Doctoral de Microélectronique, Rennes, France, Mai 2006.

Glossaire

Terme	Description	Première occurence
AES	Advanced Encryption Standard : standard de chiffrement avancé suivant un algorithme de chiffrement symétrique	Page 107
Bascule	Elément mémoire actif sur front d'horloge	Page 17
Bit-flip	Inversion de la valeur mémorisée dans une ou plusieurs cellules mémoires	Page 22
CCE	Charge Collection Efficiency : efficacité de la collection de charge qui est le rapport entre la charge collectée sur la charge générée	Page 6
CE	Chip-Enable : Entrée de sélection/validation	Page 73
CTR	Compile-Time Reconfiguration : reconfiguration statique	Page 35
DDR	Double Data Rate : transfert des données à la fois sur fronts montant et descendant de l'horloge	Page 110
Decode	Decodage : indique la phase de décodage d'une instruction à exécuter	Page 116
DMA	Direct Memory Access : caractéristique permettant d'accéder à la mémoire système indépendemment du processeur central	Page 57
Execute	Execution : indique la phase de d'éxecution d'une instruction	Page 116
Fetch	Extraction : indique la phase de lecture à partir de la mémoire d'une instruction à exécuter	Page 116
FF	Flip-Flop : bascule	Page 19
FIT	Failure In Time : nombre de défaillances, ou d'erreurs suivant les cas, en 10^9 heures	Page 7
FPGA	Field Programmable Gate Array : réseau de portes programmables in-situ	Page 2
Instrumentation	Modification de la description originale d'un circuit pour permettre l'injection de fautes	Page 2
IP	Intellectual Property : Bloc fonctionnel complexe pouvant être réutilisé dans plusieurs conceptions	Page 110
LET	Linear Energy Transfer : énergie moyenne perdue (ΔEt) par unité de longueur de trace (Δx) normalisée à la densité du matériau	Page 5
LFSR	Linear Feedback Shift Register : registre à décalage avec rétroaction linéaire	Page 54
LUT	Look-Up-Table : table de configuration	Page 35
MBU	Multiple Bit Upset : basculement simultané de plusieurs points mémoire	Page 7
MCU	Multiple Cell Upset : basculement simultané de plusieurs points mémoire n'appartiennant pas au même mot mémoire	Page 7
Memory	Mémorisation : indique la phase d'accès mémoire lors de l'éxécution d'une instruction	Page 116
Netlist	Description de circuit à l'aide des éléments qui le constituent et des interconnexions entre ces éléments	Page 36
Particule α	Particule hautement ionisée et peu pénétrante, constituée de 2 protons et de 2 neutrons	Page 5
Pipeline	Architecture qui découpe l'exécution d'une tâche en plusieurs étages qui sont opérés en parallèle, chacun sur une instruction différente	Page 27

RAM	Random Access Memory : mémoire volatile accessible en lecture et en écriture	Page 19
RTL	Register Transfer Level : niveau de description du matériel qui permet de mettre en évidence les échanges d'informations entre composants	Page 2
RTR	Run-Time Reconfiguration : reconfiguration dynamique	Page 35
SEE	Single Event Effects : Evénements singuliers	Page 7
SER	Soft Error Rate : Taux d'erreurs transitoires en un temps donné	Page 7
SET	Single Event Transient : événement singulier - pic de courant sur un nœud dans une partie de logique combinatoire	Page 7
SEU	Single Event Upset : événement singulier - basculement d'un point mémoire	Page 7
Soft Error	Erreur transitoire caractérisée par une modification de donnée réversible ou un état erroné temporaire	Page 7
Sûreté de fonctionnement	Propriété qui permet de placer une confiance justifiée dans le service qu'il délivre (Dependability)	Page 2
SWIFI	Software Implemented Fault Injection : approche d'injection de faute dans le logiciel	Page 27
SoPC	System on Programmable Chip : Système sur circuit programmable	Page 53
Write	Write : indique la phase de retour vers les registres	Page 116
XOR	Porte logique réalisant un Ou-exclusif	Page 72

RESUME

L'évolution des technologies microélectroniques augmente la sensibilité des circuits intégrés face aux perturbations (impact de particules, perte de l'intégrité du signal…). Le comportement erroné d'un circuit peut être inacceptable et une analyse de sûreté à haut niveau d'abstraction permet de choisir les protections les plus adaptées et de limiter le surcoût induit par une éventuelle reprise de la description.

Cette thèse porte sur le développement d'une méthodologie et d'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. Nous proposons 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.

MOTS CLES

VLSI, fautes transitoires, SoPC, analyse de sûreté, injection de fautes

TITLE

FAULT-INJECTION BASED DEPENDABILITY ANALYSIS IN A FPGA-BASED ENVIRONMENT

ABSTRACT

Technology downscaling increases the sensitivity of integrated circuits faced to perturbations (particles strikes, lose of signal integrity…). The erroneous behaviour of a circuit can be unacceptable and a dependability analysis at a high abstraction level enables to select the most efficient protections and to limit timing overhead induced by a possible rework.

This PhD aims at developing a methodology and an environment which improves the dependability analysis of digital integrated circuits. The proposed approach uses a hardware prototype of an instrumented version of the design to be analyzed. The environment includes three levels of execution including an embedded software level that enables to speed-up the experiments while keeping an important flexibility: the user can obtain the best trade-off between the complexity of the analysis and the duration of the experiments. We also propose new techniques for the instrumentation and for the injection control in order to improve the performances of the environment.

A predictive evaluation of the performances informs the designer on the most influent parameters and on the analysis duration for a given design and a given implementation of the environment. Finally the methodology is applied on the analysis of two significant systems including a hardware/software system built around a SparcV8 processor.

KEYWORDS

VLSI, transient faults, SoPC, dependability analysis, faults injection

ISBN : 978-2-84813-115-3

ISBNE : 978-2-84813-115-3