Environnement d'observation générique pour systèmes embarqués exposés aux radiations

Sebastien Thomet

Thèse de doctorat

En vue de l'obtention du titre de

DOCTEUR

délivré par

l'Université de Cergy-Paris

École Doctorale n°405 : Economie, Management, Mathématiques, Physique et Sciences Informatiques (EM2PSI) - Sciences et Technologies de l'Information et de la Communication (STIC)

Environnement d'observation générique pour systèmes embarqués exposés aux radiations

---

présentée par et soutenue publiquement par

Sébastien THOMET

le 24 mai 2022

devant le jury composé de :

Pr. Frédéric WROBEL	Professeur, IES Montpellier	*Président et examinateur*
Dr. Karine COULIÉ	Maîtresse de conférences, Université d'Aix-Marseille	*Rapporteuse*
Pr. Otmane AIT MOHAMMED	Professeur, Concordia University, Canada	*Rapporteur*
Dr. Besma ZEDDINI	Professeure associée, SATIE Cergy	*Examinatrice*
Pr. Amor NAFKHA	Professeur, CentraleSupélec Rennes	*Examinateur*
Dr. Fakhreddine GHAFFARI	Maître de conférence, ETIS Cergy	*Co-directeur de thèse*
Dr. Serge DE PAOLI	Docteur, STMicroelectronics Crolles	*Encadrant industriel*
Pr. Olivier ROMAIN	Professeur, ETIS Cergy	*Directeur de thèse*

Thèse préparée dans le cadre d'une convention CIFRE entre la société STMicroelectronics et le laboratoire ETIS (UMR CNRS 8051)

À mes parents, Marie-Hélène et Patrice, dont le dévouement a fait de moi ce que je suis aujourd'hui. Qu'ils trouvent dans ce manuscrit le témoignage de ma profonde reconnaissance.

Abstract

Enhanced observation Framework for embedded systems exposed to radiations

The growing complexity of semiconductor devices combined with the use of advanced technology tend to increase the sensitivity to radiation effects. Radiations are presents in space outside the magnetosphere and even on the earth at sea-level. They are overwhelmingly responsible of the decrease in lifespan and the failure of electronic systems. Radiations also became a concern with the upsurge of safety-critical applications. Today embedded electronics is the heart of security systems for automotive, of manned space mission, etc. In such domains, a simple failure from a silent data corruption to a system crash may cost people's life. Therefore, industries must ensure the reliability of semiconductors for the concerned applications. Indeed, technologies and system used for safety-critical application are qualified under radiation. They are exposed to particle beams following accelerated soft-error rate methods that emulate the expected fluence of billions of hours of operation.

In this study, through the contributions in two test-chips, we focus on space and automotive application domains where systems are respectively qualified under heavy-ions and protons. The vast majority of embedded systems are processor-based systems-on-a-chip (SoC). In such design, flip-flops are carrying the most sensitive information, in particular because they form every register. In a system, the soft-error rate (SER) coming from the flip-flops is therefore predominant with respect to other logical elements. That's why flip-flops are qualified under beam for a specific environment. It is then possible to estimate the SER of a system based on these cells in such an environment. To perform the qualifications, we have designed two test-chips especially for beam testing purposes. The implemented testing structure used to expose and monitor a large number of flip-flops is a shift register. The existing embedded self-testing bloc used to operate the shift register in an autonomous way, generates a pattern which propagates along the shift register, and monitors the output. However, as the register in composed of thousands of flip-flops and a large clock tree made of buffers, it is also exposed to clock tree events that may generate clusters of bit-flips. The count of bit-flips in the existing embedded testbed no longer reflects the number of events that will be used to estimate the error rate. In such context, we propose an embedded module able to classify single events due to SEU in flip-flop and SET in the clock tree. Implemented in the self-testing bloc using hardening techniques, in particular with TMR cells, it provides a direct overview on the expect results in terms of event number.

To validate the SER estimation and ASIC design rules on a real platform, we have designed a processor-based SoC platform which includes memory protected by an error correcting code (ECC). We have also developed a software application dedicated for tests, which ensures of the integrity of the memory, monitors a benchmark application, and

provides as diagnostic information as possible in case of an error. The platform also includes a debug architecture accessible from external JTAG connection. Thereby, the JTAG gives an access to the system, which is used to load the memory, as well as to recover data in an autonomous way that doesn't require the software to run. This reliable access is an asset to ensure the recovering of information related to a crash when the processor can no longer respond.

To enable further investigation of the failure reason, we propose a technique to capture the fault propagation in a backtrace buffer. Having an insight of a detailed execution backtrace is an asset to understand how a single-event upset (SEU) led to a system failure. The technique is based on an online trace buffering, and an error detection based on hardware assertions combined with triggering mechanisms. It also provides a hardened register bank dedicated to the user for the exportation of data from the application. Once a error is detected, the recording of the trace is stopped to keep in the backtrace the most relevant information, and the application saves the context and the corresponding error code into the user register bank. It has been integrated as an IP within the two SoCs platforms implemented in both test-chips designed.

The approach has been evaluated through a comprehensive fault injection campaign based on the RTL model. Injecting SEU-like faults during the execution of the applications provides a fault dictionary composed of errors cases including the fault injection point, information captured by the IPs and the status of the application. It enables to evaluate the coverage rate of the error detection bloc and the propagation capture rate of the execution trace buffering. To exploit in-depth the fault injection dictionary, a machine-learning classification model has been trained on it to recognize the fault signature in the buffered trace and infer the register affected by the SEU. The overall model accuracy allows to estimate the observation rate provided by the IP.

Both test test-chips have been irradiated under a proton and a heavy-ion beam for the shift registers and under a focused X-rays beam for the SoC platform. The results show that the event classifier is effective in up to 40 % of the cases, identifying multiple events in the shift register. And the IP provides up to 86 % of processor observation rate thanks to the machine-learning classification model analysis of the captured backtrace after a detected error. Moreover, the IP was able to identify the affected registers during focused X-ray testing of the SoC platforms.

Remerciements

Cette thèse CIFRE a pour moi été une expérience passionnante, pleine de découvertes et d'opportunités. Rien de tout ceci n'aurait pu se réaliser sans les personnes formidables qui m'ont entouré au sein de l'équipe HiRel et du laboratoire ETIS. Ces trois années extraordinairement enrichissantes, autant sur le plan personnel que professionnel, ne sont pas près d'être oubliées.

Je souhaite remercier chaleureusement mon encadrant industriel Serge DE PAOLI pour son soutien exceptionnel tout au long de nos travaux, et en particulier pour sa volonté de partager ses connaissances et son expérience. J'aimerais également te remercier pour m'avoir incité à prendre du recul sur notre travail, ce qui est difficile à accomplir dans un domaine aussi spécifique. Je me souviendrai notamment des revues des premiers diaporamas de présentation dans lesquels tu commençais toujours par ajouter une planche intitulée "Contexte". Je te remercie également pour ton aide à conduire nos expérimentations, pour lesquelles ton expérience a été très instructive. J'en profite également pour remercier Isabelle CASAGRANDA pour son accueil au laboratoire de ST et Alix VOLTE pour son aide et ses précieux conseils lors de nos tests à l'ESRF.

Je souhaite remercier chaleureusement Fakhreddine GHAFFARI, qui malgré la distance qui nous sépare, a toujours été très présent et très impliqué. J'ai beaucoup apprécié ta bienveillance, ton optimisme et ton soutien durant cette thèse. Nos réunions quotidiennes étaient très enrichissantes et permettaient de construire une approche à la fois industrielle et académique. Je remercie également Olivier ROMAIN pour les discussions très intéressantes que nous avons pu avoir lors des réunions et des CSTs.

Je suis très reconnaissant envers les personnes qui ont accepté de faire partie de mon jury de thèse : Frédéric WROBEL, Karine COULIÉ, Otmane AIT MOHAMMED, Besma ZEDDINI et Amor NAFKHA.

Je souhaite remercier Jean-Marc DAVEAU qui a également été très impliqué dans ces travaux. Mr. l'architecte a bâti les SoCs sur lesquels nous avons pu apporter nos contributions et m'a initié aux campagnes d'injection de fautes qui ont permis d'obtenir la majorité des résultats de ces travaux. Tu as aussi été le compagnon fidèle de nos tours de vélo, malgré le froid, la nuit et parfois même la tempête. J'ai également eu le plaisir de partager ma motivation avec Ali JOUNI et Fabio SUREAU pour aller s'entrainer d'arrache-pied au parc des Berges.

J'aimerais remercier Fady ABOUZEID pour tous les enseignements, les précieux conseils et l'optimisme dont tu as fait preuve. Tu as toujours été présent à toute heure du jour et de la nuit lors des *tapeout* pour apporter les corrections de dernière minute sur

les circuits. J'ai aussi beaucoup apprécié ta méthode d'auto-formation à l'environnement de simulation, qui a été extrêmement bénéfique pour tout le reste des travaux.

Capucine LECAT-MATHIEU DE BOISSAC, Guénolé LALLEMENT et Victor MALHERBE je vous remercie d'avoir été de si bons modèles à suivre pour cette thèse. Capucine, tu avais toujours le mot pour rire dans nos discussions sur Teams, tu as aussi été d'un grand soutien et je t'en remercie. Guénolé, je me souviens encore de tes cours accélérés de design alors que nous rentrions à vélo ensemble de ST, dont je te suis encore reconnaissant. Victor, tu incarnes l'encyclopédie de l'équipe, je te remercie pour la manière avec laquelle tu détaillais les réponses à mes questions concernant les radiations.

Valérie BERTIN, Thomas THERY, Gilles GASIOT et Dimitri SOUSSAN je vous remercie de l'aide que vous m'avez fourni, de toutes vos explications et vos partages de connaissances. Vos domaines d'expertise connexes ont été un réel atout pour les contributions que nous avons apportées. Je remercie également Anna ASQUINI et Calogero TIMINERI pour le partage de votre expérience dans la conception des circuits.

Vincent LORQUET, Olivia RIEWER et Philippe ROCHE, je souhaite sincèrement vous remercier de votre bienveillance, et de vos encouragements pour tout ce que nous avons accompli dans le cadre de ces travaux. Je vous remercie également de votre patience et de vos piqûres de rappels, face à ma tendance à toujours attendre la dernière relance.

Je terminerai cette série de remerciements par les personnes sans lesquelles rien de tout cela n'aurait été possible. Celles qui m'ont soutenu tout le long de mes études et continuent de le faire. Je remercie donc très sincèrement tous les membres de ma famille, en particulier mes parents, pour leur amour et la force qu'ils m'ont transmise. Je tiens aussi à remercier tout particulièrement ma compagne Lena de tous ses encouragements, et de m'avoir supporté et épaulé durant cette période.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

ALU Arithmetic-Logic Unit – Unité arithmétique et logique.

ASER Accelerated Soft-Error Rate – Taux d'erreur accéléré.

ASIC Application-Specific Integrated Circuit – Circuit intégré pour une application spécifique.

ATE Automated Test Environment – Environement de test automatisé.

AU Arbitrary Unit – Unité arbitraire.

BIST Built-in self test – Mécanisme d'auto-diagnostic.

CMOS Complementary MOS – MOS à grille complémentaire.

COTS Commercial Off-The-Shelf – Produit informatique standard.

CPU Central Processing Unit – Processeur d'un ordinateur.

CRC Cyclic Redundancy Check – Contrôle de redondance cyclique.

CSR Control and Status Register – Registres de contrôle et d'état.

DD Displacement Damage – Dommage par déplacement d'atomes.

DFF D flip-flop – Bascule D.

DOE Document Of Experiment – Plan d'expériences.

DSP Digital Signal Processor – Processeur optimisé pour le traitement du signal.

ECC Error Correcting Code – Code correcteur d'erreur.

ECI Error Capturing Instruction – Instruction de capture d'erreur.

EMFI Electromagnetic Fault Injection – Injection de fautes par impulsion électromagnétique.

ETM Embedded Trace Macrocell – Bloc de trace embarqué.

EXU Execution Unit – Unité d'exécution.

FD-SOI Fully-Depleted Silicon On Insulator – Silicium sur isolant totalement déserté.

FIFO First-In First-Out – Premier entré premier sorti.

FinFET Fin Field-Effect Transistor – Transistor à effet de champ à ailettes.

FIT Failure In Time – Taux de défaillance.

FMC FPGA Mezzanine Card – Connecteur FPGA pour carte fille de Xilinx.

FPGA Field-Programmable Gate Array – Circuit logique programmable.

GDS Graphic Design System – System de conception graphique.

GEO Geostationary Earth Orbite – Orbite géostationnaire.

GSL GNU Scientific Library – Bibliothèque scientifique GNU.

HDU Hart Debug Unit – Unité de débogage.

HTM AHB Trace Macrocell – Bloc de trace du bus AHB.

HTS High Throughput Satellites – Satellites à hautes bandes passantes.

IC Integrated Circuit – Circuit intégré.

IDU Instruction Decode Unit – Unité de décodage d'instructions.

IFU Instruction Fetch Unit – Unité de récupération des instructions.

IO Input/Output – Entrée/Sortie.

IoT Internet of Things – L'internet des objets.

IP Intelectual Property – Propriété intelectuelle.

IR Instruction Register – Registre d'instruction.

ITM Instruction Trace Macrocell – Bloc de trace par instructions.

LASER Light Amplification by the Stimulated Emission of Radiation – Amplification de la lumière par émission stimulée de radiation.

LEO Low Earth Orbit – Orbite de basse altitude.

LET Linear Energy Transfer – Transfert linéique d'énergie.

LSU Load Store Unit – Unité de chargement et de stockage de données en mémoire.

MBU Multiple Bit Upset – Erreur logicielle multiple.

MCU Multiple Cell Upset – Inversion binaire multiple.

MEO Medium Earth Orbit – Orbite de moyenne altitude.

MOS Metal-Oxide-Semiconductor – Structure métal-oxide-semiconducteur.

MOSFET MOS Field-Effect Transistor – Transistor à effet de champ à grille métal-oxide.

MPRF Multi-Port Register File – Fichier de registres multi-ports.

NGSO Non-GeoStationary Orbite – Orbite non géostationnaire.

PC Program Counter – Pointeur d'instruction.

PLL Phase-Locked Loop – Boucle à verrouillage de phase.

PTM Program Trace Macrocell – Bloc de trace d'exécution.

RA Return Address – Adresse de retour.

RTL Register-Transfer Level – Niveau de transfert de données entre registres.

SBC Single-Board computer – Ordinateur à carte unique.

SBU Single Bit Upset – Erreur logicielle simple.

SCU Single Cell Upset – Inversion binaire simple.

SECDED Single Error Corrected Double Error Detected – Erreur simpe corrigée erreur double détectée.

SEE Single-Event Effects – Effets d'une particule isolée.

SEGR Single-Event Gate Rupture – Rupture de grille isolée.

SEHE Single-Event Hardware Error – Erreur matérielle.

SEL Single-Event Latchup – Déclenchement d'un parasite isolé.

SER Soft-Error Rate – Taux d'erreur logiciel.

SET Single-Event Transient – Evénement transitoire.

SEU Single-Event Upset – Erreur logicielle.

SoC System on a Chip – Système sur Puce.

SP Stack Pointer – Pointeur de pile.

STM System Trace Macrocell – Bloc de trace système.

TCM Tightly-Coupled Memory – Mémoire proche du processeur.

TMR Triple Modular Redundancy – Redondance modulaire triple.

TPIU Trace Port Interface Unit – Bloc d'interfaçage vers le port de trace.

UART Universal Asynchronous Receiver Transmitter – Emetteur-récepteur asynchrone universel.

UDRB User Data Register Bank – Banque de registres pour les données utilisateur.

USB Universal Serial Bus – Bus série universel.

UTBB Ultra-Thin Buried Oxide – Couche ultra mince d'oxyde enterrée.

WFI Wait For Interrupt – Instruction d'attente d'interruption.

Introduction

Depuis l'apparition de la microélectronique, les technologies utilisées pour la fabrication des circuits n'ont cessé d'évoluer. La finesse de gravure des transistors est un paramètre prépondérant qui définit un nœud technologique. Au fil du temps, les industries ont cherché à réduire la taille des transistors gravés sur circuit, afin d'augmenter la densité d'intégration, de diminuer les tensions d'alimentation et d'augmenter les fréquences de fonctionnement. Néanmoins, l'évolution de cette caractéristique physique entraîne une sensibilité accrue des technologies face aux perturbations liées à l'environnement dans lequel les systèmes évoluent. Compte tenu du rôle de la microélectronique dans notre société, ses faiblesses font l'objet des défis majeurs d'aujourd'hui.

Les systèmes autonomes sont depuis quelques années au cœur de nombreux dispositifs dont la fiabilité est critique pour la sécurité des personnes, la réussite de projets de grande envergure, ou la réputation d'entreprises de renommée internationale. D'après un rapport de tendance datant de 2020 sur la vérification fonctionnelle [32], jusqu'à 42% des *Application-Specific Integrated Circuit* (ASIC) ou *Integrated Circuit* (IC) recensés parmi 1492 participants du monde entier concernent des application critiques pour la sécurité. Les systèmes embarqués, en particulier les systèmes sur puces basés sur microprocesseurs, sont notamment utilisés dans des applications automobiles et spatiales, deux domaines dans lesquels nous nous plaçons dans le cadre de ces travaux.

Dans les applications automobiles, les équipements de sécurité actifs (ABS, ESP, contrôle de trajectoire, etc.), passifs (airbags, etc.) ou encore le contrôle de l'ensemble des paramètres moteur sont entièrement régis par des systèmes à processeurs. Les véhicules autonomes sont d'autant plus concernés puisque leur comportement dépend directement des décisions prises par le système. La conduite nécessite de lourds traitements de données provenant des capteurs du véhicule afin d'évoluer dans la circulation, ce qui implique des besoins en performance et en mémoire à coûts limités que seuls les nœuds technologiques de dernière génération peuvent adresser. Les décisions prises par le système dans ce cadre peuvent donc être critiques pour la sécurité des personnes. Des fautes allant d'une simple erreur de calcul jusqu'à l'arrêt du système peuvent avoir de lourdes conséquences. À la suite d'un accident assez suspect d'un véhicule de grande marque, une enquête de la NASA avait révélé les failles de fiabilité de l'unité de contrôle du moteur [1]. Le conducteur avait en effet pu appeler les secours en expliquant que la commande d'accélération était bloquée au maximum et que les freins ne répondaient plus. D'autres véhicule de cette même grande marque étaient impliqués dans une série d'accidents ayant pour seule et même cause la perte de contrôle de l'accélération et l'inefficacité des freins. Les failles de fiabilité identifiées étaient nombreuses, et concernaient tant la partie matérielle que logicielle. Une prise de conscience collective des potentielles erreurs logicielles pouvant apparaître dans les systèmes a mené à la spécification de normes concernant les méthodes de validation, la protection mémoire

et la vérification en ligne des systèmes.

Les applications spatiales ont les mêmes enjeux quant à la fiabilité des systèmes, d'autant plus si les vols spatiaux emportent un équipage humain. Dans le contexte de ces applications, il est parfois impossible d'opérer toute forme de maintenance une fois le système déployé. Il est donc crucial de s'assurer de la fiabilité et de la durée de vie des systèmes avant même d'envisager leur lancement en orbite. On distingue trois orbites dans lesquelles évoluent les engins, d'après la Figure 1. La distance par rapport à la Terre affecte les performances des communications, les services sont donc répartis selon des compromis entre les propriétés des différentes orbites :

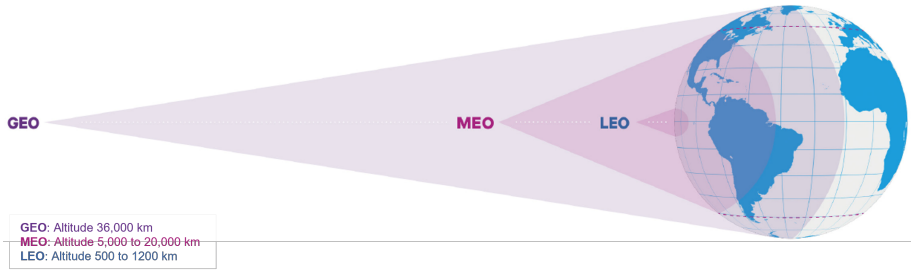


FIGURE 1 – Représentation des orbites et de la couverture des transmissions adaptée de [35]

— **Orbite géostationnaire (GEO - *Geostationary Orbite*)** : Ces satellites suivent la rotation de la Terre à environ 36,000 km d'altitude et gardent la même couverture de transmission de données au sol. Ils fournissent généralement des services tels que la diffusion de chaînes TV, de données météo, et quelques transmissions de données bas débit. Ces services sont progressivement supplantés par des équipements haut débit spécialement conçus pour l'échange de données.
— **Orbite de moyenne altitude (MEO - *Medium Earth Orbite*)** : Les satellites MEO évoluent entre 5,000 et 20,000 km d'altitude. Historiquement, ils ont été utilisés pour des applications de navigation. Plus récemment, une constellation a été déployée pour fournir une connectivité haut débit équivalente à celle fournie par la fibre en particulier dans des zones non couvertes par les installations, comme les zones maritimes ou à accès difficile.
— **Orbite de basse altitude (LEO - *Low Earth Orbite*)** : Cette orbite entre 500 et 1,200 km est densément peuplée par des milliers de satellites, répondant principalement aux besoins de la science, de l'imagerie et des télécommunications bas débit. La nouvelle génération de satellites a l'intention de desservir les marchés de la communication tels que l'internet haut débit.

Les domaines d'application spatiaux exposent inéluctablement les systèmes embarqués aux radiations puisqu'ils évoluent dans un environnement qui n'est pas protégé par la magnétosphère terrestre. La Figure 2 est une vue d'artiste des radiations présentent dans l'espace provenant des vents solaires et venant interagir avec la magnétosphère terrestre. Les particules impliquées sont des protons, des électrons, des particules alpha et des ions lourds. Le blindage des applications peut en partie protéger les applications des effets de ces radiations. Certaines particules, majoritairement des protons et des électrons restent piégées dans la magnétosphère et forment une ceinture de radiations appelée la ceinture de Van Allen. Cette ceinture est composée d'une partie intérieure (*inner belt*) entre 1,000 km et 6,000 km et d'une partie extérieure (*outer belt*) entre 13,000 km et 60,000 km. La ceinture intérieure est la seule source de radiation qui va affecter les applications LEO et MEO.

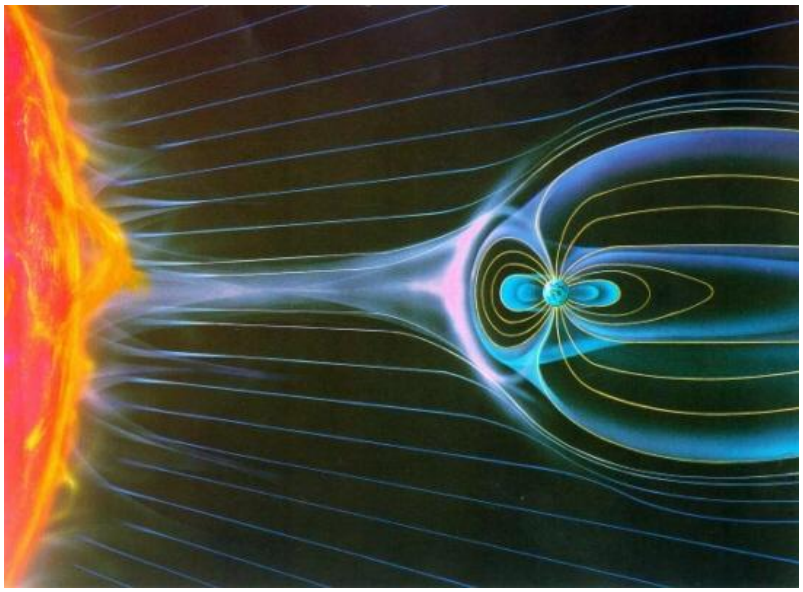


FIGURE 2 – Vue d'artiste du rayonnement cosmique en direction de la Terre [5]

Enfin, la dernière source de radiation dans l'espace provient des rayonnements cosmiques composés de protons, de particules alpha et d'ions lourds. Dans ce cas, le blindage est très difficile, les applications GEO doivent donc être en mesure de tolérer les effets de ces radiations, en particulier en provenance des ions lourds.

Dans l'environnement terrestre, des radiations sont aussi naturellement présentent et constituent un facteur majeur de diminution de la fiabilité des semiconducteurs. Ces radiations sont présentes sous deux formes : les neutrons provenant de rayons cosmiques (illustrés Figure 2) qui passent à travers l'atmosphère, et les particules alpha issues de la radiactivités naturelle provenant de la décroissance d'atomes lourds. Les impacts de particules sur les circuits ont des conséquences directes sur leur fonctionnement instantané et leur durée de vie.

Les effets des radiations sur les circuits peuvent se traduire par un vieillissement dû à l'action mécanique des particules (DD - *Displacement Damage*), une dégradation du fonctionnement suite aux effets ionisants (TID - *Total Ionizing Dose*), et l'apparition localisée et aléatoire d'événements isolés (SEE - *Single-Event Effect*) dans les circuits. Les événements isolés peuvent être de nature irréversible menant à la perte définitive de toute ou partie de la fonctionnalité d'un système suite à la destruction matérielle d'un élément du circuit. Les SEEs sont majoritairement des événements réversibles se traduisant par des aléas logiques appelés erreurs logicielles, un redémarrage de l'application suffit alors à retrouver un état stable.

Ainsi toutes les applications peuvent être sujettes à des erreurs dues aux radiations. Le taux d'erreur (SER - *Soft-Error Rate*) acceptable dépend de la criticité de l'application et de la sensibilité du système. Les facteurs prépondérants de la sensibilité sont : le nœud technologique utilisée, la tension d'alimentation, la fréquence de fonctionnement et la température.

Enjeux de l'étude

Dans cette étude, nous abordons les deux domaines d'application cités précédemment à savoir l'automobile et le spatial, à travers la conception de modules matériels dédiés à l'observation des erreurs qui surviennent dans les circuits suite à l'effet des radiations. Les contributions ont été implémentées dans deux circuits de tests et expérimentées sous radiations. Comme nous l'avons évoqué, la fiabilité des systèmes embarqués dans de telles applications est critique, les circuits nécessitent une implémentation robuste et une phase de qualification en milieux radioactifs. La phase de qualification est réalisée sous la forme de test accéléré sous radiation selon le profil de mission visé.

Les tests sous radiations sont utilisés pour évaluer la réponse des circuits aux événements isolés, et de leur faire subir un vieillissement accéléré afin d'évaluer le taux d'erreur sur une très longue durée de vie. Ces particules peuvent causer des dommages suite à des déplacements d'atomes de la surface active (DD), à l'insertion de particules chargées dans le silicium ou à l'ionisation du circuit (TID). Les particules chargées ont un effet ionisant sur les matériaux. Le transfert linéique d'énergie (LET - *Linear Energy Transfer*) est la quantité qui décrit l'énergie transférée par une particule ionisante traversant la matière. Le LET varie en fonction de l'épaisseur du matériau, ou de l'énergie des particules, le pique de Bragg correspond au maximum global de la courbe. Ces phénomènes se traduisent par une modification du comportement des transistors menant à un dysfonctionnement partiel ou total du circuit. Dans les circuits logiques, le TID se traduit par une modification du temps de réponse des portes et bascules.

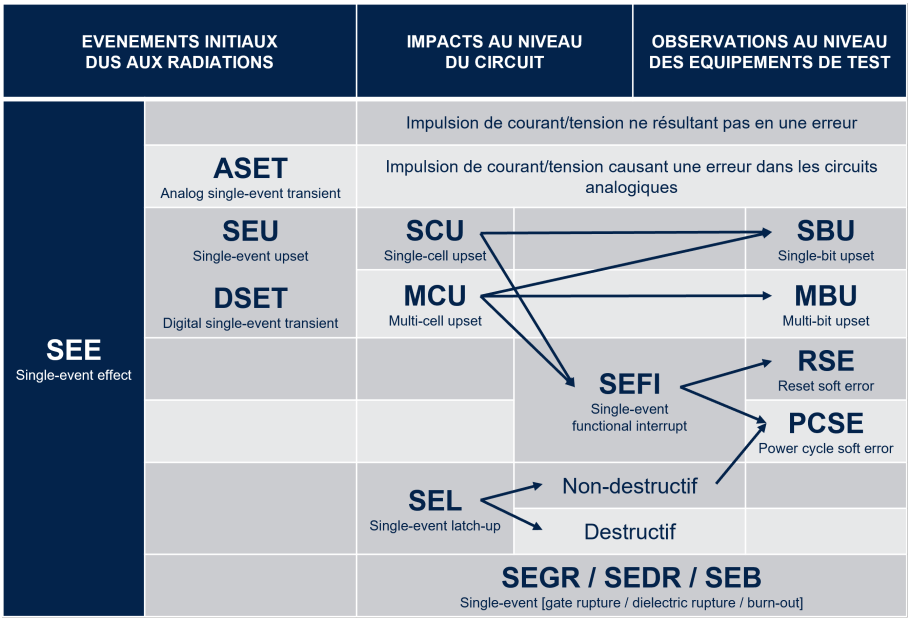


FIGURE 3 – Taxonomie des erreurs suite à l'effet d'une particule isolée dans les composants à semiconducteurs adaptée de [49]

Ce qui nous intéresse dans les circuits sont les impacts de particules sur le silicium pouvant générer des évènements isolés (SEEs) qui se manifestent sous différentes formes selon la zone du composant électronique affectée. Ces événements sont causés par les effets ionisants des particules qui pénètrent dans le silicium du circuit, déposent des charges électriques qui interagissent avec les transistors et génèrent des perturbations. La taxonomie des erreurs issues des SEEs est illustrée sur la figure 3 adaptée du standard JESD89B [49].

Les SEEs peuvent se traduire par différents phénomènes physiques énumérés dans les point suivants :

- **Générer des impulsions de courant ou de tension** dans le circuit (SET - *Single-Event Transient*). Ces impulsions peuvent se traduire par l'inversion binaire d'un signal, le masquage ou l'ajout d'un front montant ou descendant. Les signaux particulièrement sensibles à ce phénomène sont les signaux d'horloge ou de reset des circuits logiques puisqu'ils adressent un grand nombre de cellules. Ils sont propagés au travers des circuits grâce à des buffers afin d'équilibrer les temps de propagation, ces buffers constituent alors les parties sensibles.
- **Conduire à une ou plusieurs inversions binaires** dans des éléments de stockage tels que des cellules mémoires (SRAM, DRAM), ou des cellules logiques séquentielles (bascules D ou JK). Cette inversion binaire peut être due à la propagation d'un signal corrompu ou au fait qu'un transistor de la cellule ait été affecté par l'impact d'une particule. Si à la suite d'un impact, un changement d'état s'est produit, il s'agit d'une erreur logicielle (SEU - *Single-Event Upset*). Si plusieurs éléments de stockage ont été affectés en même temps, on parle de MCU (*Multiple-Cell Upset*), et plus précisément de MBU (*Multiple-Bit Upset*) si plusieurs bits du même mot mémoire ont été inversés.
- **Déclencher des parasites isolés** (SEL - *Single-Event Latchup*) qui engendrent une consommation de courant au sein du composant, dû à l'armement d'un thyristor parasite entre les zone dopées ou déplétées, et le substrat du silicium. Ce phénomène est réversible sous réserve de couper la tension d'alimentation. La surconsommation de courant peut monter de quelques mA à plusieurs ampères et provoquer un échauffement par effet Joule allant jusqu'à endommager le composant. La surveillance du courant délivré par les alimentations lors des tests permet de détecter ce phénomène.
- **Provoquer un changement irréversible** associé à des dommages permanents dans le circuit (SEHE - *Single-Event Hard Error*), comme une rupture de grille (SEGR - *Single-Event Gate Rupture*) entrainant le claquage d'un transistor à effet de champ (MOSFET).

L'ensemble de ces risques potentiels implique la nécessité de qualifier le comportement des composants durant des tests sous radiations pour s'assurer de leur fiabilité de fonctionnement dans des environnements hostiles.

Besoins émergents

Pour qualifier des composants pour un domaine d'application ciblé, des expérimentations sont réalisées sous faisceau de particules ou en haute altitude durant lesquelles le circuit testé est surveillé depuis un banc de test automatisé (ATE) pour détecter l'apparition d'erreur. Les normes JESD89B [49] de JEDEC et 25100 [30] de ESCC spécifient des protocoles de test et établissent une correspondance entre l'environnement d'évolution du système et les types de particules à utiliser. L'objectif principal est de s'assurer que les expérimentations ont été menées selon un protocole qui garantit la pertinence des résultats. Les différents industriels du marché des semi-conducteurs qui tentent de répondre aux besoins de fiabilité des circuits sont amenés à réaliser ce genre d'expérimentations. C'est donc par souci d'équité et de validité des résultats que ces standards ont été créés.

La technique de test accélérée (ASER - *Accelerated Soft Error Rate Technique*) sous faisceau consiste à imposer un flux de particules important sur un composant de test pour obtenir une fluence équivalente à plusieurs années en situation, en seulement quelques minutes ou quelques heures. Cette technique, émule le fonctionnement d'une application en situation au profil de mission, elle est notamment utilisée pour qualifier des systèmes dans les domaines d'applications spatiales et terrestres. Les points suivants résument la correspondance avec l'environnement ciblé et les principaux établissements dans lesquels les tests peuvent être réalisés :

— **Alpha** : Ce type de particule est généré par le boitier du composant lui-même à cause d'impuretés radio-isotopiques. Il s'agit d'un flux d'ions He^{2+}. Les qualifications sont effectuées grâce à une source aplha dont l'activité est connue. Les mesures données en FIT sont normalisés par rapport à certains type de boîtiers *Ultra Low Alpha* (ULA) dont le rayonnement de référence est de 0.001 $\alpha.cm^{-2}.s^{-1}$. On trouve assez facilement des sources alpha dans les laboratoires, en particulier chez STMicroelectronics. Les principales difficultés sont l'accréditation pour en posséder une et les contraintes de stockage ;
— **Neutrons** : Il représentent donc la menace la plus importante de génération de SEEs et de détérioration des circuits pour les applications terrestres. Les mesures données en FIT sont normalisés par rapport à une référence terrestre, généralement le flux mesuré dans la ville de New York, de 13 $n.cm^{-2}.h^{-1}$. Les qualifications sont réalisées dans des accélérateurs à particules tels que ISIS ChipIR [43] en Angleterre, TRIUMF [95] au Canada, ou encore LANSCE [56] aux États-Unis. Le spectre énergétique du faisceau de neutrons est assez large et correspond au spectre neutronique atmosphérique ;
— **Protons** : Les protons de hautes énergies sont utilisés pour émuler un environnement d'orbite basse. Dans certaines conditions d'énergie, ils ont des effets similaires à ceux des neutrons. Lors de tests dans des accélérateurs à particules, le faisceau fournit des protons monoénergétique, dont l'énergie peut être modulée dans une certaine gamme de valeurs selon les capacités des établissements. Les mesures effectuées sont des sections efficaces pour chaque niveau d'énergie utilisé. Les qualifications sont réalisées dans des accélérateurs à particules tels que PSI [74] en Suisse ;
— **Ions lourds** : Les ions lourds sont quant à eux utilisés pour émuler les conditions en orbite géostationnaire. Les accélérateurs à particules peuvent fournir des faisceaux d'ions ou de cocktails d'ions. Chacun des types d'ions disponibles possède une énergie et un LET différents. Les mesures effectuées sont des sections efficaces pour chaque LET utilisé. Les principaux sites d'irradiation en Europe sont CYCLONE [25] en Belgique et RADEF en Finlande [83].

Les blocs développés pour ces tests sont en général des répétitions de cellules afin de maximaliser la surface exposée au faisceau, et ainsi réduire le temps de test. Plus généralement, lors de l'exposition des composants sous radiations, les objectifs sont les suivants :

— **Établir la liste des SEEs** auxquels la technologie ou le circuit peut être sujet. Cette étape permet notamment de vérifier que l'exposition n'induise pas d'événements à caractère destructif.
— **Émuler le vieillissement** induit par la dose de radiations et l'action mécanique des particules. En connaissant le flux moyen reçu par une application dans son environnement, il est possible de lui imposer la même fluence lors des tests que

celle correspondante à sa durée de vie en mission.
— **Mesurer le taux d'erreur logicielles**, à savoir des erreurs de type SET, SEU ou MCU, pour en déduire la section efficace (ou *cross-section*) ou le FIT (Failure in Time). La section efficace est une grandeur expérimentale normalisée permettant de comparer des résultats de test. C'est le nombre d'erreurs, divisé par la fluence, et le nombre de cellules. Son unité est le cm2. Le FIT est généralement utilisé pour les applications terrestres, et dérivé de la section efficace par un facteur multiplicatif. Cette grandeur donne le taux de pannes par milliard d'heures pour un flux de référence de neutrons ou de particules alpha.

Dans le cadre de nos travaux, les qualifications s'effectuent à différents niveaux d'abstraction matérielle. Selon le niveau d'abstraction, le banc de test permettant d'opérer le circuit est plus ou moins complexe. Nous verrons dans l'état de l'art quels sont les avantages et limites de chacun des cas suivants :

— **Qualification d'une propriété intellectuelle :** Il s'agit dans ce cas de qualifier un bloc fonctionnel (IP - *Intellectual Property*). Pour ce faire, une méthode de BIST (*Built-In Selft Test*) est mise en œuvre afin d'automatiser le test et la détection d'erreur d'un élément matériel, comme le stipule la norme [49]. Des structures spécifiques peuvent être implémentées en tant que capteur pour réaliser des mesures physiques telles que le taux d'erreur (SER), le taux d'ionisation (TID), la tension seuil, la température ou la largeur des SETs.
— **Qualification d'un système basé sur microprocesseur :** Ce niveau d'abstraction est le plus élevé et implique le développement d'un logiciel pour le mettre en œuvre le système. Le BIST est donc partiellement ou entièrement logiciel. Le banc de test est plus complexe car il doit opérer le SoC afin de le programmer et d'échanger des informations, le tout en utilisant des protocoles spécifiques (JTAG, etc.).

Problématiques

Ce travail de thèse se concentre sur deux problématiques dans le cadre des qualifications sous radiations suivant la méthode de test accéléré sous faisceau de particules. Les contributions constituent des moyens d'observation non intrusifs des fautes induites par l'effet des radiations. Les méthodes de mesure misent en œuvre ne nécessitent pas d'action de la part de l'élément à observer. Elles se basent en revanche sur des modules matériels intégrés dans le composant de test pour accéder aux informations utiles à la mesure. Ces modules ont été intégrés au sein des architectures présentes sur les circuits de test pour assurer l'observation des éléments d'intérêt et devront être durcis par conception pour garantir la fiabilité de la mesure et l'accès aux informations. En effet lors de l'exposition, les moyens de mesure seront soumis aux radiations au même titre que structure de test à qualifier.

Les techniques proposées adressent une IP dédiée à la qualification de cellules logiques séquentielles et un système basé sur microprocesseur. Dans le cadre de ces travaux, deux composants dédiés aux tests comportant les contributions évoquées ont été développés. Ces circuits de test ont été fabriqués dans l'usine de STMicroelectronics à Crolles en technologie 28 nm en silicium totalement déserté sur isolant composé d'une couche enterrée

ultra mince d'oxyde (UTBB - *Ultra-Thin Buried Oxide*) (FD-SOI - *Fully-Depleted Silicon On Insulator*). Ces deux circuits de tests dénommés SERVAL et SQUAL ont pour vocation de qualifier des applications respectivement automobiles et spatiales. Développés dans un contexte industriel, ils contiennent tous deux des structures de tests dédiées à la qualification de cellules logiques séquentielles et des systèmes basés sur microprocesseurs.

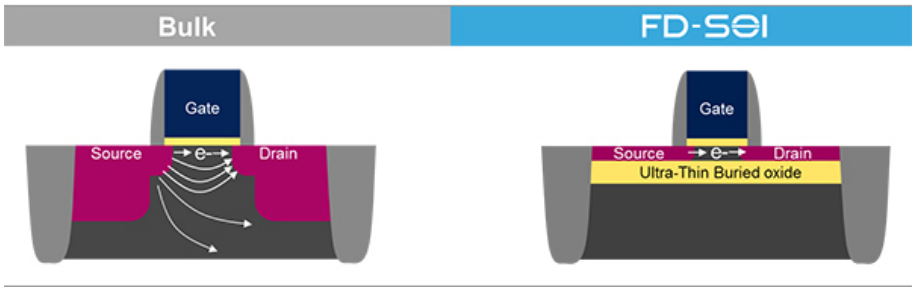


FIGURE 4 – Comparaison des technologies *Bulk* et FD-SOI [58]

Sur la Figure 4, on peut voir une représentation de la structure de la technologie UTBB FD-SOI comparée à celle d'une technologie Bulk standard. Une fine couche d'isolant sépare la source et drain du substrat, le canal est donc plus étroit, ce qui diminue la capacitance entre la source et le drain, et tend à diminuer le courant de fuite, la dépendance en température et augmenter le rendement [40]. Il est donc possible d'appliquer une tension de polarisation entre le corps du transistor et son substrat, ce qui modifie son comportement [10]. Cette technologie a un meilleur comportement face aux SEEs et aux SELs que la technologie Bulk grâce à la couche d'isolation SOI, qui constitue une barrière à la diffusion des charges dans le canal. Ces propriétés font de la technologie UTBB FD-SOI un atout pour les applications automobiles et spatiales en raison de sa robustesse, ainsi qu'un très bon compromis entre efficacité, performances et coûts. En raison de ses performances et de son efficacité énergétique, cette technologie est également utilisée pour des applications très basse consommation [66]. La robustesse intrinsèque de cette technologie a cependant été une difficulté pour le test de nos contributions car, comme nous l'avons constaté lors des expérimentations sous radiations des circuits de test, les propriétés de la technologie ne sont pas favorables à l'apparition d'erreurs logicielles que nous étudions. Le fait d'appliquer une tension de polarisation augmente la section efficace à bas LET [10], ce qui est exploité lors des tests sous particules alpha.

L'IP de qualification de cellules logiques séquentielles se base sur une méthode existante CREST (*Circuit for Radiation Effects Selft-Test*) [63] pour opérer un registres à décalage et détecter les erreurs logicielles qui surviennent dans les bascules. La problématique de cette approche émane de la structure du registre composée essentiellement de bascules mais aussi de l'arbre d'horloge qui induit d'autres types d'erreurs qui viennent perturber la mesure. Les tests au profil de mission ne permettent pas d'accéder aux signaux depuis l'extérieur, et les limitations du banc de test n'offre pas suffisamment de bande passante pour garantir l'exactitude de la mesure. De fait, distinguer les deux types d'événements aurait le double avantage d'améliorer la précision du taux d'erreur des cellules logiques séquentielles et d'évaluer le taux d'erreur des buffers d'horloge. Pour cela, l'utilisation d'un motif de test avec changement d'état est requise, et la conception du registre et de son arbre d'horloge peut suivre les règles de design conventionnelle. Une classification en ligne au profil depuis un module intégré à l'IP permettrait d'adresser limitation et de donner en temps réel un compte exact du nombre d'événements par classe.

Concernant le système basé sur microprocesseur, l'objectif est de pouvoir qualifier un

système complexe tel qu'il serait conçu pour une application et ainsi, valider les règles de conception de circuits durcis. Initialement, le taux d'erreurs de ce type d'architecture est une estimation basée sur les taux respectifs des éléments qui le composent. Par ailleurs, les qualifications d'un tel système sont actuellement très pauvres en termes d'observabilité. Les données de diagnostic accessibles ne permettent pas d'avoir d'informations quant à l'origine potentielle de l'erreur, elles se limitent aux registres d'état du processeur. Il n'est donc pas possible d'envisager d'optimiser une technique de durcissement partielle suite à des tests sous faisceau de particules. Notre contribution consiste à intégrer une IP d'observation du processeur visant à collecter des informations en ligne et à les restituer sous forme de données de diagnostic lors d'une erreur. Elle est accompagnée d'une méthode d'analyse des résultats qui fournit le registre de provenance de la faute ayant conduit à l'erreur. L'IP d'observation de l'activité du processeur a pour vocation d'être implémentée de manière durcie pour garantir l'intégrité des données lors des tests, non-intrusive envers l'application et agnostique vis-à-vis du processeur.

Organisation du manuscrit

Le manuscrit est organisé selon les cinq chapitres décris dans les points suivants :

Chapitre 1. État de l'art
Ce chapitre dresse un état de l'art des techniques présentes dans la littérature, et liste les solutions existantes en rapport avec les travaux. L'objectif est de définir le contexte scientifique et industriel dans lequel notre contribution s'inscrit.

Chapitre 2. Qualification de cellules logiques séquentielles et classification en ligne d'événements isolés
Dans ce chapitre, nous introduisons le contexte des qualifications sous faisceau de particules dans les établissements évoqués, ainsi que toute la partie matérielle qui a permis d'opérer les circuits de test dans ce cadre. Nous présentons la structure de test utilisée pour mesurer le taux d'erreur de cellules logiques séquentielles et les limites du banc de test embarqué existant. Puis nous proposons une technique de classification en ligne des événements afin d'identifier leur provenance dans le circuit.

Chapitre 3. Plateformes de test basées sur microprocesseurs et simulation avec injection de fautes
Ce chapitre se concentre sur les plateformes SoC et l'environnement logiciel misent en place pour le développement et la validation des techniques proposées pour l'observation des systèmes basés sur microprocesseurs. Nous présentons également la méthode de simulation utilisée pour évaluer ces techniques.

Chapitre 4. Observation en ligne et capture des erreurs sur SoC
Deux version de la technique d'observation des microprocesseurs basée sur une IP intégrée dans un système sur puce sont proposées dans ce chapitre. Elles ont été implémentées dans deux circuits de test différents. Une méthode basée sur un modèle de classification à apprentissage automatique est utilisée pour exploiter les données récoltées, elle apporte un éclairage sur les bénéfices de la démarche et les optimisations possibles de son implémentation.

Chapitre 5. Résultats de simulations et d'expérimentations
Les résultats des campagnes de simulation constituent une première approche pour évaluer l'efficacité de la technique et fournissent les éléments nécessaires à l'optimisation des moyens mis en œuvre pour l'observation. Les résultats des campagnes de tests sous radiations mettent en lumière l'apport et les limites des différentes contributions. L'ensemble de ces résultats s'articule respectivement autour des deux circuits de test développés.

Au terme de ces cinq chapitres, la conclusion récapitule les principaux aspects de cette thèse et met en exergue les bénéfices et les limites de la démarche. Dans la continuité des travaux réalisés, les perspectives résument les points pouvant être améliorés suite aux résultats et proposent de nouvelles pistes de recherche.

Chapitre 1

État de l'art

Ce chapitre est consacré à l'état de l'art des domaines concernés par les deux axes de recherche abordés dans ce manuscrit. Dans la Section 1.1, au travers de plusieurs articles sur la qualification de cellules logiques séquentielles sous radiations, nous allons mettre en exergue les limites des études précédentes face aux nouveaux besoins de qualification au profil de mission d'une technologie durcie, le 28 nm UTBB FD-SOI. Les articles étudiés se basent sur la même structure de test, implémentée dans des composants dédiés au expérimentations sous faisceau, pour surveiller le taux d'erreur d'un grand nombre de cellules.

Alors que l'essor des systèmes basés sur microprocesseurs bat son plein depuis plusieurs décennies, de nombreux outils de mise au point ont été développés dans le but d'aider à la validation du fonctionnement des circuits réalisés et au développement des applications logicielles. Un aperçu des techniques industrielles les plus utilisées aujourd'hui dans les systèmes embarqués est présenté dans la Sections 1.2. Nous étudions en particulier les techniques de débogage non intrusives qui sont maintenant devenues des standards de l'industrie. Elles fournissent un support à l'observabilité dans les systèmes à microprocesseur dont certains mécanismes peuvent être repris pour l'observation du comportement sous radiations. Nous nous concentrons sur les mêmes contraintes que celles évoquées dans la classification des cellules logiques séquentielles, à savoir de connectivité limitée et de haute fréquence de fonctionnement.

Du fait de leur finesse de conception, les nœuds technologiques avancés utilisés pour fabriquer les circuits basés sur microprocesseurs dans des applications critiques souffrent d'une sensibilité accrue aux hostilités de leur environnement. Pour y faire face, des techniques de durcissement matériel ou logiciel s'adressent au contrôle en ligne du fonctionnement dans le but de détecter les anomalies et de garantir un fonctionnement cohérent. Une analyse d'un ensemble de techniques couvrant les approches logicielles et matérielles est présentée dans la Section 1.3.

Les problématiques d'observation du comportement de l'application et de détection des erreurs dans les microprocesseurs adressées dans la littérature nous permettent de structurer notre approche pour parvenir à capturer la propagation d'erreurs. Cette méthode est utilisée dans le but de la qualification de système à microprocesseurs sous radiations. Les procédures de test sous radiations ont fait l'objet de normes proposés par JEDEC et ESCC dont les principaux points sont décrits dans la Sections 1.4.

1.1 Qualification de cellules logiques séquentielles sous radiations

Les cellules logiques séquentielles (DFF - *D flip-flop*) sont la base de tous les circuits logiques. Ce sont notamment les constituants des registres d'un processeur ou de périphériques au sein d'un SoC. De plus, les informations contenues dans les bascules peuvent être critiques et provoquer un dysfonctionnement majeur en cas de corruption. Ces cellules sont utilisées dans des gammes de tension et de température assez large, il est donc important d'évaluer l'impact de ces paramètres sur le taux d'erreurs qui peut se produire pour un profil d'utilisation donné. Pour qu'une étude soit pertinente, il est nécessaire de récolter un nombre conséquent d'événements (plus d'une centaine en général) dans le but de diminuer l'incertitude sur les grandeurs mesurées. Dans un contexte de test accéléré, cela se traduit par exposer une surface importante à un faisceau de particules, et donc utiliser un nombre conséquent de cellules et vérifier leurs comportements. C'est pour parvenir à cette fin que des architectures dédiées sont conçues pour la qualification de cellules logiques séquentielles. Dans ce cadre, un registre à décalage d'une taille souvent très importante (jusqu'à plusieurs milliers de bascules), illustré sur la Figure 1.1, est privilégié [16, 34, 50, 61, 63, 101]. En effet, cette structure permet d'alimenter une chaîne de cellules logiques séquentielles avec des données connues, qui se propagent le long de cette chaîne, et de les vérifier en sortie. Elle ne permet cependant pas d'avoir accès à la cellule dans laquelle une erreur est apparue. La chaine impose également un délai de détection de l'erreur en fonction de la position de la cellule. Toutefois, un registre à décalage permet de voir toutes les erreurs, peu importe le nombre de cellules qui le composent, pour un minimum de logique de contrôle. Les erreurs sont simplement détectées par comparaison de la donnée sortante avec la donnée attendue. Cette structure est utilisée dans tous les articles présentés par la suite.

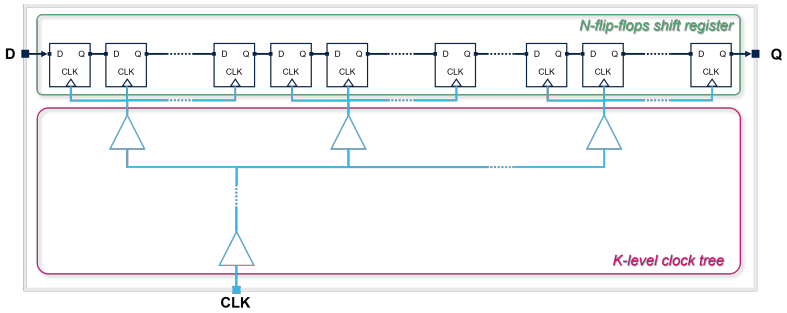


FIGURE 1.1 – Strucutre d'un registre à décalage composé de bascules D

Cette méthode de test a initialement été définie dans l'article [63] et permet de capturer des signatures complexes d'erreurs dans des circuits fonctionnant jusqu'à 5 GHz. Elle se présente sous la forme d'une méthode BIST, illustrée Figure 1.2, à savoir une IP matérielle embarquée qui permet de tester des fonctionnalités implémentées sur le DUT de manière autonome. L'architecture de test présentée dans cet article est composée d'un registre à décalage d'une profondeur de 127 bits, alimenté par un générateur de données. Afin de détecter les erreurs en sortie du registre, les données en entrée sont comparées avec celles en sortie. Grâce au fait que le flux binaire provient d'un motif répété de manière cyclique, la comparaison peut se faire directement entre l'entrée et la sortie du registre. Le block de détection peut de stopper l'horloge afin de permettre le diagnostic du registre. Une pile FIFO mémorise de manière continue les valeurs en sortie du registre ainsi que le signalement d'une erreur sur une profondeur d'un octet. La petite taille du registre à

tester rend possible une telle implémentation. Il faut également que la FIFO soit durcie pour ne pas générer de fausses erreurs car elle est également exposée aux radiations. Cette technique a été réutilisée dans d'autres travaux afin de qualifier différents types de DFFs ou de buffer d'horloge.

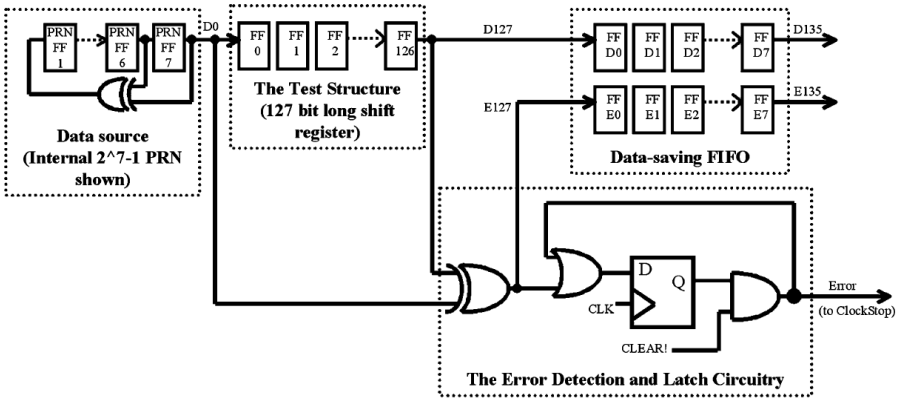


FIGURE 1.2 – Architecture CREST (*Circuit for Radiation Effects Self Test*) de détection d'erreurs autonome [63]

Nous verrons au travers d'autres articles traitant de la même problématique que cette technique présente des limites dans certaines conditions. Ces limites sont exacerbées selon la structure du registre à décalage, la fréquence ou le SER mesuré lors des qualifications.

1.1.1 Comparaison des méthodes de qualification

L'article [50] présente les résultats des tests d'une chaine de DFFs exposée à des particules Alpha. L'expérience est effectuée en posant directement la source radioactive sur un circuit ouvert. Pour ce type d'expérience, le boîtier est ouvert du côté de la face active du circuit, afin de pouvoir approcher la source au plus près. L'étude montre l'influence de la tension d'alimentation et de la fréquence sur le SER induit par les radiations. Cette étude s'adresse à la technologie 14/16nm FinFET bulk. A cet égard, un circuit spécifique aux tests a été fabriqué. Il implémente 4 registres à décalage d'une profondeur de 8 kb. Chaque registre utilise différents types de cellules dans le but de pouvoir comparer leur robustesse. La technique [63] est mise en œuvre afin de pouvoir opérer le DUT à des fréquence allant jusqu'à 1.3 GHz. Le motif utilisé dans le registre est uniquement composé de '0' logique. Ce motif permet de masquer les SETs dans l'arbre d'horloge qui provoquent des translations de données parasites dans le registre, au niveau des DFFs concernées. A l'inverse, ce motif ne permet pas d'étudier la sensibilité de toutes les transitions des cellules logiques séquentielles ni même la vulnérabilité de l'arbre d'horloge. Il ressort de cette publication que le SER augmente avec la diminution de la tension d'alimentation ainsi que l'augmentation de la fréquence, comme le montre la Figure 1.3. Les auteurs fournissent un modèle qui permet, pour un SER et une fréquence donnée, d'optimiser la tension d'alimentation dans le but de réduire la consommation.

L'étude [16] consiste à qualifier l'apparition de SETs dans les circuits logiques. Elle met en avant l'effet de la tension et de la fréquence sur la section efficace en fonction du LET, lors de tests sous ions lourds. Un circuit spécifique a été fabriqué avec la technologie CMOS bulk 65 nm. Ce circuit permet d'évaluer des structures de tests qui mettent en œuvre différents types de buffers et de DFFs basées sur différentes techniques de durcissement.

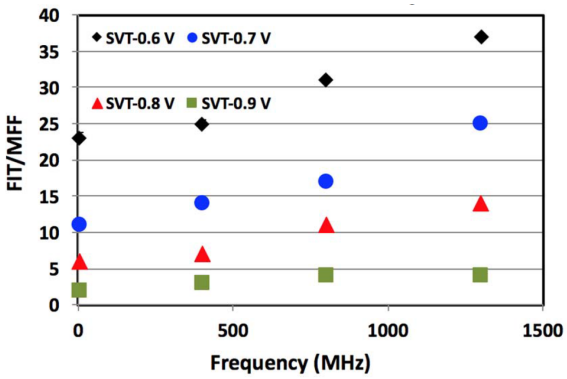


FIGURE 1.3 – FIT en fonction de la Fréquence pour une irradiation Alpha de DFFs à différentes tensions d'alimentation [50]

La particularité de certaines structures, basée sur des registres à décalage de 500 DFFs de profondeur, est que dans certaines chaines, chaque DFF est séparée de la suivante par un à quatre buffers. Cela permet d'étudier l'apparition de SET dans les buffers. Le DUT est opéré à 50 Mhz en raison du banc de test externe implémenté sur FPGA, utilisé pour injecter les signaux en entrée des registres, et surveiller les sorties. Les différentes structures de test et les résultats des irradiations sous ions lourds sont illustrés Figure 1.4 et 1.5. Le caractère externe des éléments permettant de superviser les registres impose une contrainte sur la fréquence de fonctionnement qui ne dépasse pas les 50 MHz. Les auteurs concluent sur une dépendance forte de la section efficace à la tension d'alimentation à bas LET, tandis que cette dépendance diminue avec l'augmentation du LET. Le fait d'ajouter des inverseurs entre les DFFs n'influe pas le taux d'erreur à cette fréquence.

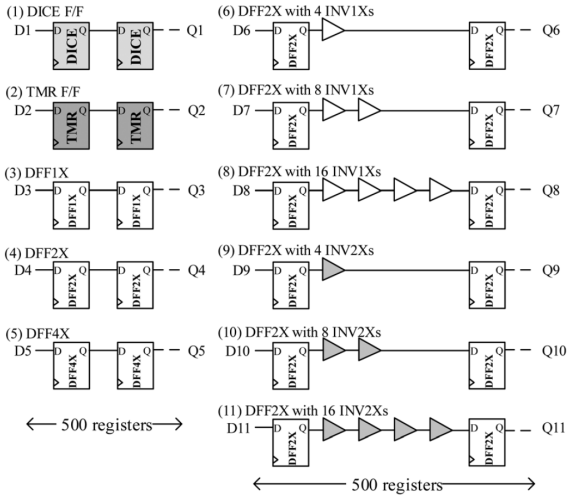


FIGURE 1.4 – Structures de test basées sur un registre à décalage sans et avec inverseurs [16]

Dans l'article [34] sont présentés des résultats de tests sous particules alpha sur des registres à décalage. La particularité de cette étude est que plusieurs motifs sont utilisés à des fréquences différentes, et qu'elle présente des résultats en test statique. Le véhicule de test est comme précédemment, un circuit spécifique en 32 nm dans lequel sont implémentés plusieurs registres à décalage. Plusieurs types de DFF sont évalués, mais chaque registre est composé d'un seul type de cellule. On retrouve des cellules standards avec différentes propriétés, et des cellules durcies par différentes techniques. Les registres composés de

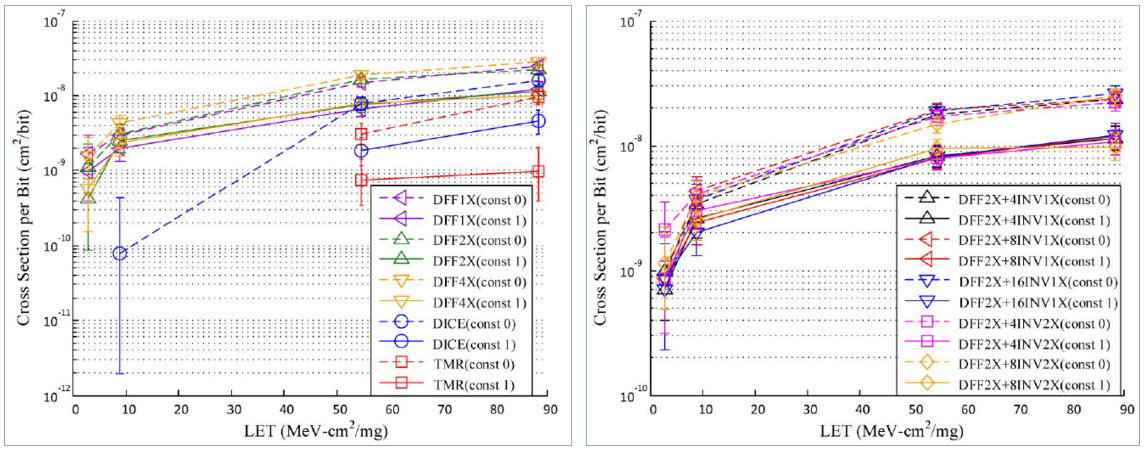


FIGURE 1.5 – Section efficace par bit en fonction du LET sous ions lourds des différentes structures de test[16]

cellules durcies sont d'une profondeur de 80k DFF, et les autres de 40k DFF. Une telle profondeur se justifie par la nécessité d'obtenir un nombre important d'événements en un temps limité, ce qui procure des résultats de SER avec de faibles taux d'incertitude. Par conséquent, cela engendre des contraintes de construction de l'arbre d'horloge, qui a dans ce cas été durci afin de minimiser l'apparition d'événements venant de cette partie du circuit. Cependant, les tests ont été effectués à basse fréquence (20 MHz) pour que le banc de test FGPA puisse échantillonner les compteurs d'erreur à la même fréquence. Une analyse des données est nécessaire à postériori afin de garantir l'absence d'événement venant de l'arbre d'horloge qui pourrait générer artificiellement une grande quantité d'erreur. En effet, si un front d'horloge vient à manquer ou s'ajouter au sein de l'arbre d'horloge, toutes les cellules concernées sont désynchronisées, ce qui génère autant d'erreurs en sortie que de cellules dans la branche de l'arbre ayant été touchée. Malgré la difficulté à opérer des tests avec un motif qui rend sensible le système aux événements survenant dans l'arbre d'horloge, cette étude met en lumière l'importance du motif utilisé dans les registres à décalage. En outre, les auteurs proposent un motif de test sur 4 bits, utilisé dans le cas de tests dynamiques, avec le taux d'erreur le plus élevé, voir Figure 1.6.

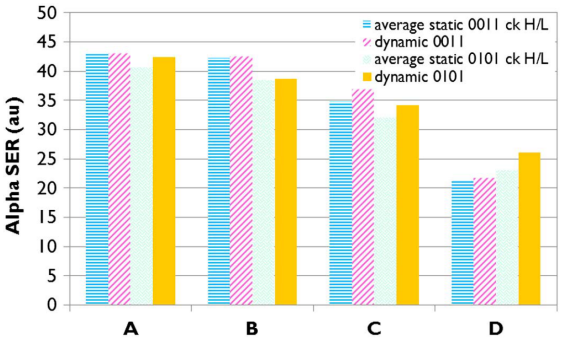


FIGURE 1.6 – SER en fonction des motifs de test pour différents types de DFFs sous particules Alpha [34]

L'approche présentée dans l'article [101] se propose d'évaluer la dépendance en fréquence du SER en ne considérant que les événements de type SET dans les DFFs. En particulier, cela s'adresse à la technologie FinFET en 16 nm avec laquelle le circuit de test a été réalisé. Il implémente quatre registres à décalage d'une profondeur de 8 kb, gérés par une méthode BIST [63]. Les DFFs de trois registres à décalage sont durcies, elles

contiennent 4 éléments de stockage, avec une configuration maître-esclave, détaillées sur la Figure 1.7. En outre, chaque élément de stockage est espacé d'une distance suffisante à faire diminuer la probabilité qu'une particule puisse affecter deux éléments, ce qui diminue le SER de ces DFFs. Parmi les 3 registres à décalage avec DFF durcies, les trois implémentent des DFFs avec des distances inter-éléments différentes. Cette structure permet de diminuer le nombre de transistors sensibles dans les DFFs. Le DUT a été testé sous un flux d'ions lourds à basse fréquence (2,5 MHz - considéré comme tests statiques) et à des fréquences allant jusqu'à 1,3 GHz. Les résultats montrent donc que la section efficace augmente avec la fréquence, illustré sur la Figure 1.8, ce qui est cohérent avec les résultats de [50]. Ils montrent également que la duplication des éléments de stockage est suffisante pour mitiger les effets des SEEs. Le fait d'augmenter l'espacement des éléments ayant une faible efficacité montre que les registres fabriqués en technologie FinFET sont peu sujets au MCUs.

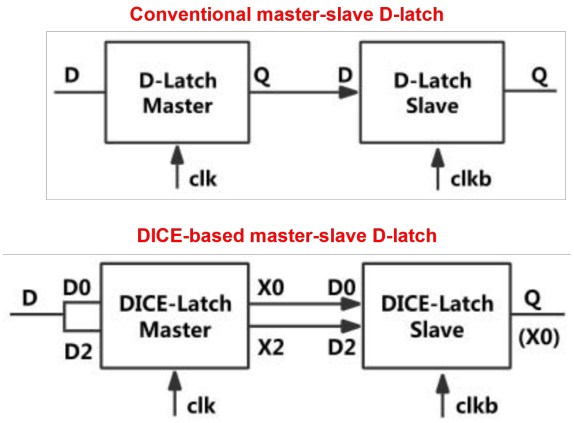


FIGURE 1.7 – Structure d'une DFF maître-esclave conventionnelle et d'une DICE [04]

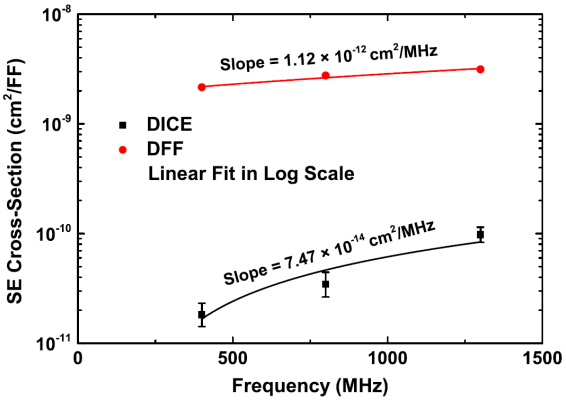


FIGURE 1.8 – Section efficace par DFF en fonction de la fréquence de registres basés sur DFFs de type maître-esclave conventionnelle et DICE sous ions lourds [101]

L'approche présentée dans [61] propose d'évaluer le taux d'erreurs dans les buffers qui composent l'arbre d'horloge. Dans le cadre de ce travail, un test chip a été réalisé et implémente notamment deux registres à décalage. Le premier registre est composé de 13,5 kb avec des DFFs standards, et le second de 28 kb avec des DFFs durcies. La particularité de la structure de test est qu'elle implémente une méthode BIST [63] avec comptage des erreurs. Les test chip a donc pu être opéré à haute fréquence, entre 200 et 600 MHz. Le motif utilisé lors des tests est basé sur quatre bits '0101'. Les registres à décalage sont uniquement composés de DFF en série, cadencées par un arbre d'horloge qui se propage de

manière synchrone dans toute la chaine de DFFs. Les buffers d'horloge des deux registres commandent respectivement au minimum entre 56 et 96 DFFs. Ce qui induit des groupes de minimum 56 ou 96 erreurs lorsqu'un buffer d'horloge est touché par une particule. Or la fréquence d'échantillonnage du compteur d'erreur embarqué est de 1,3 Hz. Ce qui signifie que lors des tests, le SER doit rester bien inférieur à cette valeur pour que la distinction d'événement d'horloge ou de DFF soit pertinente.

1.1.2 Adaptation au contexte industriel

Au travers de ces travaux, les limites de l'architecture CREST[63] apparaissent. Tout d'abord les registres à décalage mis en œuvre pour la qualification des DFFs sous radiations nécessitent dans certains cas une profondeur importante afin de capturer un nombre suffisant d'événements. Les auteurs évoquent des chaînes de 8 kb [50, 101], 13,5 kb, 28 kb [61], jusqu'à 40 kb ou encore 80 kb [34]. De telles structures composées de logiques synchrones nécessitent un arbre d'horloge conséquent, composé de buffers d'horloge afin de répartir les courants de commande des portes et les délais de propagation. Ces buffers constituent eux-mêmes une surface non négligeable de silicium, exposée lors des tests sous radiations, et qui peut être sujette aux SETs. Le signal d'horloge, en tant que signal de référence pour la logique synchrone, est très sensible à ce type d'événement qui peut se matérialiser par un front d'horloge masqué ou injecté en plus. Ce qui a pour conséquence de désynchroniser les DFFs dans la branche de l'arbre d'horloge qui a été touchée. Un moyen de masquer ce type d'erreur est de n'utiliser que des motifs statiques (composés uniquement de '0' ou de '1') [16, 50, 101]. Afin de prévenir ce type d'événement, les auteurs utilisent également des techniques de durcissement [34], ou de duplication [101] de l'arbre d'horloge. Se prémunir de ce type d'événement ne permet pas de qualifier les buffers d'horloge, et les techniques de durcissement de l'arbre ne sont pas toujours applicables à l'échelle d'un circuit. Dans [61], les auteurs ont conçu l'arbre d'horloge manuellement. Le nombre minimal de DFFs contrôlées par buffer d'horloge est connu et peut être maintenu à une valeur assez élevée, ce qui facilite le discernement des événements dans les DFFs ou l'arbre d'horloge. Mais cette technique n'est pas représentative d'un circuit logique conçu de manière conventionnelle.

Il est à noter que le SER est dépendant de la fréquence d'utilisation qui induit une plus grande probabilité de capture des SETs à haute fréquence [16, 50]. Or, la configuration du banc de test peut induire de fortes contraintes sur la fréquence maximale admissible par le DUT [16, 34], il est donc nécessaire d'embarquer une méthode de type BIST [63] dans le DUT. Lors de tests sous particules Alpha et sous ions lourds, le boîtier du composant doit être ouvert pour permettre à la source de générer des erreurs dans le circuit. En outre, baisser la tension d'alimentation rend le circuit plus sensible et peut permettre d'augmenter le SER [50].

Par conséquent, à la suite de l'utilisation de technologie durcie par construction (28 nm FD-SOI) et de leur utilisation à haute fréquence, il est crucial de distinguer de manière fiable, la source des erreurs dans les registres à décalage. Les erreurs provenant de DFFs ou de l'arbre d'horloge possèdent des signatures différentes qui pourraient être exploitées depuis le harnais de test embarqué au niveau du détecteur d'erreur. Nous reviendrons sur cet aspect dans le Chapitre 2, en proposant une technique de classification statistique en ligne ne nécessitant que peu de ressources et indépendante de la structure de l'arbre d'horloge.

1.2 Observabilité des systèmes à microprocesseurs

Le test radiatif de processeurs est un moyen de valider une plateforme dans son ensemble en incluant les règles de conception, et en testant le comportement d'une application. Au niveau d'un système à processeurs, on retrouve des cellules logiques séquentielles pour les registres mais aussi de la logique combinatoire, ainsi qu'un arbre d'horloge. Lors des tests radiatifs, des événements très hétérogènes surviennent, et nous nous intéressons aux moyens de les détecter. Nous passons en revue dans ce paragraphe les solutions les plus courantes.

Le développement matériel et logiciel génère inévitablement des bogues. Qu'il s'agisse de certifier un système par rapport à une norme (ISO26262, etc.) ou de vérifier l'intégralité des ressources une fois le circuit fabriqué, les outils de débogage et de trace sont devenus cruciaux face à la complexité croissante des systèmes. Nous distinguons ici les techniques de débogage intrusif qui contrôlent l'exécution du programme par l'intermédiaire de points d'arrêts, et les techniques d'exportation de trace d'exécution peu ou pas intrusives. Dans le cadre d'une approche d'observation non intrusive du flot d'exécution, nous privilégions dans notre étude les techniques basées sur les traces d'exécution.

L'environnement de débogage consiste en une partie matérielle embarquée dans le système, d'une partie logicielle sur un poste hôte, et d'un moyen de transport des informations entre les deux. Pour ce qui est des traces, la partie embarquée est généralement constituée d'un moyen de collecte, de stockage et d'exportation. Les traces matérielles permettent l'enregistrement de données, d'instructions ou de signaux. Elles sont généralement non intrusives vis à vis de l'exécution du programme. Les traces logicielles permettent l'émission de messages via de points de passage dans le programme. Ces traces peuvent être explicites (cas de la fonction *printf*) ou implicites. Les traces implicites sont nativement implémentées dans les systèmes d'exploitation, comme les *hooks* de FreeRTOS ou certains mécanismes Linux (*ftrace*, *strace*, *perf*, *SystemTrap*, *kprobes*, *OProfile*, etc.). Le moyen de transport des traces est adapté aux interfaces et protocoles d'encodages utilisés. Nous passons ci-après en revue un ensemble de solutions et de standards de débogage et de traces pour les systèmes embarqués.

1.2.1 ARM - Architecture CoreSight

L'architecture de débogage et de trace du système CoreSight [23] de ARM est certainement la plus connue du fait de l'utilisation très répandue de processeurs ARM dans les applications embarquées à faible consommation. On retrouve ce type de cœur dans nombreux appareils du commerce de masse tels que les smartphones, les boîtiers multimédia, l'IoT, etc. Dernièrement, ils commencent à apparaître dans les ordinateurs portables notamment les notebooks dotés de SoCs multicœurs pour des raisons d'autonomie. Ces cœurs sont aussi présents dans les SoCs sur les cartes de développements, on peut citer les *Single-Board Computers* (SBC) telles que la lignée des RaspberryPi, ou encore les plateformes de développement STM32 MCU et MPU. En particulier, certains circuits de Xilinx ou Intel FPGA sont dotées de SoCs dans lesquels sont implémentés des cœurs ARM ainsi que d'une partie de logique reconfigurable. Ce type circuit, tel que le Zynq-7000, est très utilisé dans la littérature [75-78, 80] du fait premièrement de l'implémentation des cœurs couplés à l'architecture de débogage CoreSight. Deuxièmement, la partie FPGA offre la

possibilité de développer des IPs intégrables dans l'environnement du SoC.

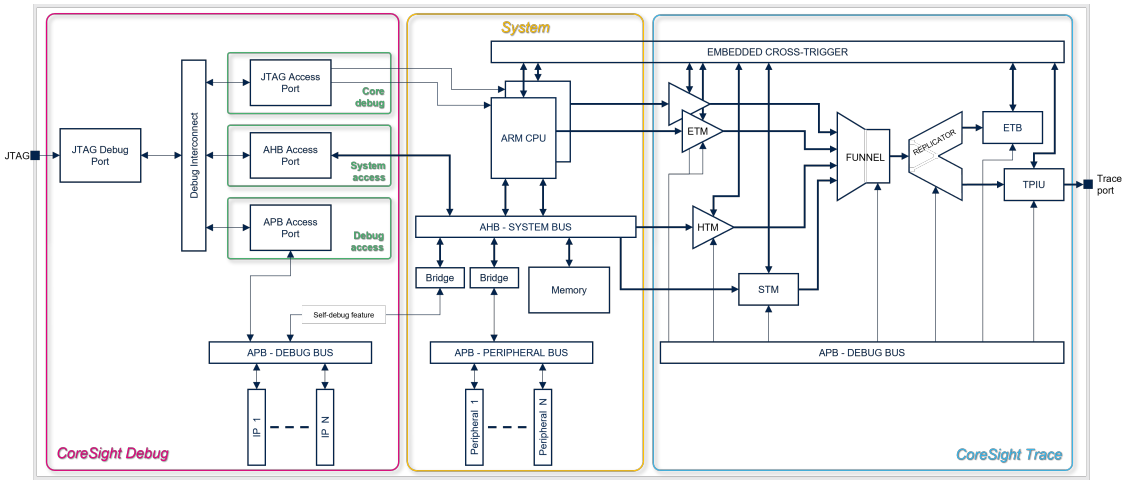


FIGURE 1.9 – Architecture avancée du CoreSight avec débogage et trace

L'architecture du système CoreSight permet de réaliser deux fonctions majeures. La première est le débogage du ou des cœurs processeurs et l'accès au système, très utile pour accéder aux ressources dans le cadre d'une mise au point du logiciel, pour exécuter pas-à-pas un programme, ou lorsque le système est dans un état indéterminé. La seconde, est l'extraction en ligne de la trace d'exécution du processeur, et d'informations provenant du logiciel. Ces deux fonctions sont passées en revue ci-dessous :

1. **CoreSight Debug**

   L'infrastructure de débogage de l'écosystème CoreSight est opérable depuis le protocole de débogage JTAG, ou encore le protocole ARM Serial Wire ne nécessitant que 2 broches. Cette dernière assure plusieurs fonctionnalités permettant d'accéder à différentes parties du SoC. Voici le schéma d'architecture, Figure 1.9, illustrant les différentes parties du CoreSight dont la partie de débogage. Leurs fonctions sont détaillées ci-dessous.

   (a) *JTAG debug port*

   Point d'entrée unique de l'infrastructure de débogage, cette IP supporte les protocoles JTAG et Serial Wire. Elle peut être adressée par un debugger standard, incluant OpenOCD [71]. Il s'agit d'un accès faible bande passante mais ayant une couverture d'accès très importante. La surcouche protocolaire induite par la configuration de l'architecture diminue d'autant plus la bande passante. Les fonctions décrites dans les points suivants impliquent des protocoles propriétaires spécifiques.

   (b) *JTAG access port*

   Cette IP offre une alternative au chaînage des JTAG lorsque le système comporte plusieurs blocs pilotables par cette interface. Elle permet de générer des trames JTAG pour adresser les fonctionnalités de débogage des IPs du SoC. Cela permet notamment d'accéder aux points d'arrêt, ou encore aux fichiers de registres des processeurs. Malgré la surcouche protocolaire induite à l'accès JTAG en entrée, l'intérêt de cette IP est de pouvoir éviter les chaînes JTAG qui diminuent drastiquement la bande passante.

   (c) *AHB access port*

L'accès au bus système principal est assuré par cette IP. Cet accès est considéré comme intrusif au regard du système, car les requêtes sont en concurrences avec celles des processeurs et doivent être arbitrées. Cette IP permet d'accéder à toutes les ressources du système mappées dans l'espace d'adressage du bus AHB. Il de fait possible d'accéder à la mémoire du SoC et de charger un programme. Cette fonctionnalité est intéressante dans le cas de SoC conçus pour les tests radiatifs car elle permet de s'affranchir de l'implémentation de mémoire non volatile pour le programme et de différents modes de *boot*.

(d) *APB access port*

Dans la majorité des SoCs est implémenté un bus APB indépendant dédié à des fonctionnalités de débogages. Cette IP permet d'y accéder directement, de manière non-intrusive. Ce bus permet d'adresser différentes fonctionnalités attrait à la surveillance du système de manière transparente et indépendante vis-à-vis du fonctionnement du système. Les IPs périphériques ont la possibilité d'être reliées à ce bus afin de rendre accessible le débogage de fonctions qui leurs sont propres. Le bus system AHB possède un accès en tant que maître sur le bus de débogage, ce qui permet de configurer les éléments de ce bus depuis le logiciel embarqué (mode *Self-Hosted Debug*).

2. **CoreSight Trace**

L'infrastructure de trace de l'écosystème CoreSight est configurée depuis le bus spécifique de débogage, dont l'accès est décrit précédemment. Cette infrastructure est modulaire, et permet d'extraire en ligne des informations provenant du flot d'exécution de manière automatique ou déclenchée par logiciel. Cette structure entièrement configurable permet de s'adapter aisément aux différentes architectures de SoC. Le schéma d'architecture, illustré Figure 1.9, montre une configuration complète du CoreSight sur un SoC composé de deux cœurs et d'un bus principal. Les fonctions des principaux blocs sont détaillées dans les points suivants.

(a) *Embedded Trace Macrocell* (ETM)

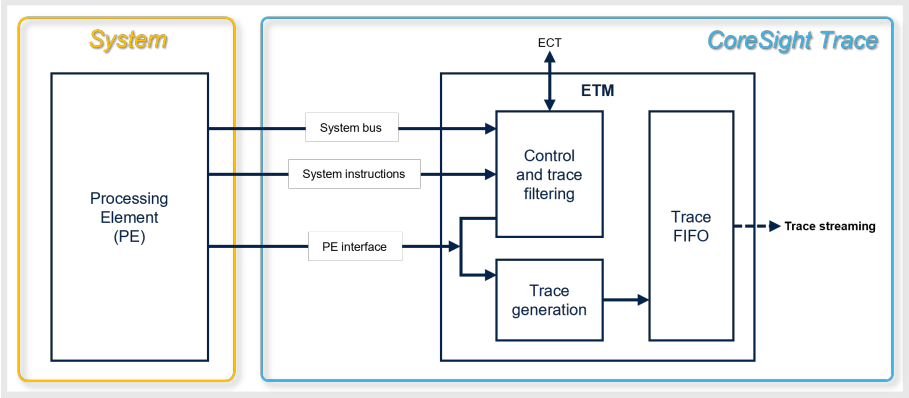


FIGURE 1.10 – Exemple de configuration du bloc ETM (Embedded Trace Macrocell)

Voici un exemple de SoC avec un bloc ETM connecté au processeur, Figure 1.10. Le bloc ETM extrait, formate et génère la trace d'exécution en ligne du processeur afin d'observer en détails son activité. Ce bloc est directement connecté au cœur à observer. Il récupère des informations sur les accès bus du cœur, les instructions exécutées, ainsi que des signaux provenant du pipeline permettant de synchroniser PC, instructions et mouvements de données. La

partie génération de trace compresse les données et génère des événements de trace datés. La partie filtrage de trace permet de configurer les données à exporter afin de ne récupérer que branchements, certaines variables, les retours d'exceptions, etc. La déclinaison *Program Trace Macrocell* (PTM) est parfois utilisée pour les systèmes *low end*.

(b) *AHB Trace Macrocell* (HTM)

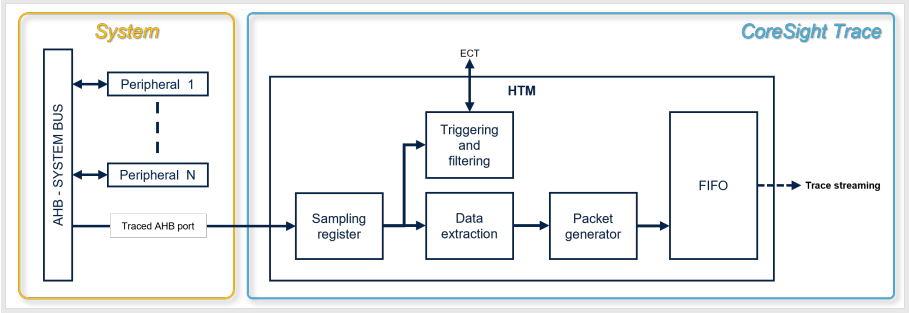


FIGURE 1.11 – Architecture du bloc HTM (AMBA AHB Trace Macrocell)

Le bloc HTM permet d'exporter la trace du bus système, et d'observer les requêtes des IPs du processeur en incluant adresses et données. Il observe le bus système depuis un des ports, comme l'illustre la Figure 1.11. Un module de filtrage est intégré dans ce bloc afin de discerner des transactions en fonction de leurs adresses, des données ou encore d'un nombre de répétition.

(c) *System Trace Macrocell (STM)*

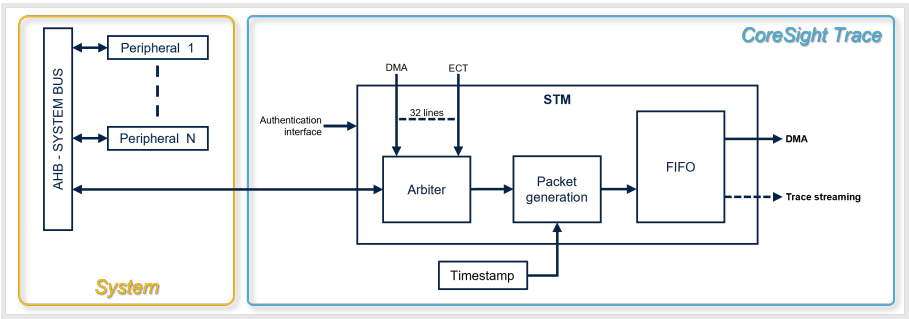


FIGURE 1.12 – Architecture du bloc STM (System Trace Macrocell)

Le bloc STM, dont l'architecture est illustrée Figure 1.12 fournit un banque de registres dans laquelle un processeur peut écrire. Les registres sont associés à des canaux, et une écriture dans cet espace mémoire génère un événement daté dans un format particulier. Ceci permet d'exporter des donner depuis le logiciel de la même manière qu'un programme le ferait via la sortie standard (avec la fonction *printf* par exemple). L'avantage de cette configuration est qu'elle offre une datation système permettant de synchroniser les événements émis par plusieurs processeurs, et une grande bande passante allant de pair avec une faible latence pour exporter des événements. La déclinaison *Instrumentation Trace Macrocell* (ITM) est parfois utilisée pour les systèmes *low end*.

(d) *Gestion du flux de trace*

Les flux de trace générés par les blocs précédents sont très hétérogènes, ils se présentent tous sous forme de paquets composés d'un en-tête permettant de les identifier. Selon la configuration des blocs, notamment pour les blocs ETM

et HTM, le flux généré peut être continu et nécessiter une bande passante élevée. Tandis que le bloc STM a plutôt tendance à générer des événements sporadiques, selon l'application logicielle. Chacun de ces flux doit pourtant converger vers un seul point de sortie sous forme d'un seul flux multiplexé. Sur la droite de la Figure 1.9, on peut voir un cas d'usage typique de gestion de la trace dans un SoC avec une architecture de trace avancée.

Le bloc *Funnel* est chargé de multiplexer les flux entrants, composés de paquets, en un seul flux. Il associe un identifiant au paquet suivant sa provenance. Le bloc *Replicator* duplique le flux de trace, ceci afin de pouvoir simultanément l'exporter vers l'extérieur et le stocker dans un buffer. Pour exporter les données vers le port de trace destinées à un analyseur de trace externe, le bloc *Trace Port Interface Unit*(TPIU) formate les paquets sous forme de trames incluant l'identifiant de l'expéditeur. De la même manière, le buffer de trace, *Embedded Trace Buffer* (ETB), génère ces mêmes trames et les stocke dans une mémoire interne. Cette mémoire peut être configurée pour ne garder que les premiers ou les derniers événements en cas de surcharge. La taille typique de ce buffer est de 4 à 16 kB.

(e) *Embedded Cross-Trigger (ECT)*

Le bloc ECT permet de mutualiser les événements détectés par les différents blocs qui composent la structure du CoreSight, et de les associer à une action. Chaque bloc précédemment cité, est capable de détecter des éléments dans le flot d'exécution, comme le STM, le HTM ou encore les ETMs des différents processeurs. L'ECT est composé du module *Cross-Trigger Interface* (CTI) qui permet de collecter ces différents événements, et du module *Cross-Trigger Matrix* (CTM) qui redirige les actions vers certains blocs. Ces actions visent à aider la phase de débogage, elles peuvent par exemple prévenir un accès mémoire corrompu, ou détecter une exception inattendue en stoppant l'exécution ou en arrêtant l'exportation d'événements. Ceci permet de cibler des événements en particulier et de capturer les informations pertinentes en lien avec leur cause.

L'aperçu des fonctionnalités offertes par l'architecture du CoreSight de ARM permet de constater l'avancée des techniques de débogage des systèmes embarqués. Cette architecture est une référence dans le domaine et les principes ont été repris par certains des développeurs d'IPs matérielles libres cités par la suite. L'aspect modulaire de cette structure permet d'optimiser les composants en fonction de la complexité du SoC, des besoins en termes d'observabilité de l'unité d'exécution et de l'espace disponible sur le composant. L'inconvénient majeur du CoreSight est qu'il s'agit d'un ensemble d'IPs propriétaires ARM. Dans le contexte d'une utilisation à processeur agnostique, il n'est pas possible d'accéder au code source des IPs afin de modifier leur implémentation pour les adapter ou utiliser des techniques de durcissement. Cette infrastructure est également conséquente en termes de ressources, où rare sont les microcontrôleurs de faible taille et faible consommation qui embarquent l'ensemble des fonctionnalités citées. Toutefois, l'architecture supporte certains protocoles standards, tels que le JTAG ou des protocoles définis par MIPI (STP - *System Trace Protocol* et TWP - *Trace Wrapper Protocol*), ce qui a pu permettre d'utiliser certains blocs pour gérer l'accès de débogage.

1.2.2 Standard Nexus 5001

La norme Nexus 5001 définit une interface universelle de débogage pour processeurs embarqués [38]. Créée en 1999, le but initial était de proposer des spécifications générales pour les interfaces de débogage. Face à la complexité croissante des systèmes liée à l'envolée du nombre de transistors dans les circuits, allant de pair avec celle des logiciels, la visibilité devient de plus en plus critique. Ce standard s'adresse aux développeurs de matériel dans le cadre de la vérification du circuit ainsi qu'aux développeurs de logiciels. Pour cela, elle définit des exigences en termes de contrôle, d'analyse de trace et d'outils de développements. Cette fonctionnalité est similaire à la partie débogage du CoreSight.

— **Contrôle de l'exécution**
Cette fonction de débogage s'adresse directement au cœur. Elle permet de configurer des *watchpoint* ou *breakpoint*, de stopper le cœur et d'accéder à tous ses registres. Le diagnostic potentiel est donc très détaillé. Cette méthode implique une exécution étape par étape, et les accès aux ressources du processeur ou la configuration des *watchpoint* ou *breakpoint* ne peut se faire qu'une fois l'exécution stoppée.
— **Analyse de la trace**
La spécification d'analyse de la trace contribue à améliorer l'observabilité de l'utilisateur directement sur le flot d'exécution du processeur. Cette méthode se doit d'être la moins intrusive possible afin de récolter des informations pertinentes lors d'une exécution normale. La trace d'exécution comprend l'accès aux instructions exécutées mais aussi aux données circulant dans le système. Cette surveillance en ligne du système permet de corréler les instructions avec les flux de données et d'évaluer les performances. Elle permet de surveiller en ligne le fonctionnement du système, et est utilisée dans le cadre de la validation post-fabrication de circuits ou dans le cadre du développement logiciel.
— **Exigences en outils de développement**
Cette partie de la norme concerne l'interface physique avec le système et les protocoles de communication pour extraire les données. Cela consiste en une interface physique sur laquelle peuvent être reliés plusieurs outils de développement externes. Une couche protocolaire permet de gérer les différents clients au sein du système embarqué, à savoir plusieurs processeurs, de manière indépendante. Cette interface procure une visibilité et une contrôlabilité sur l'ensemble du système depuis la même interface.
— **Niveau de conformité et performances**

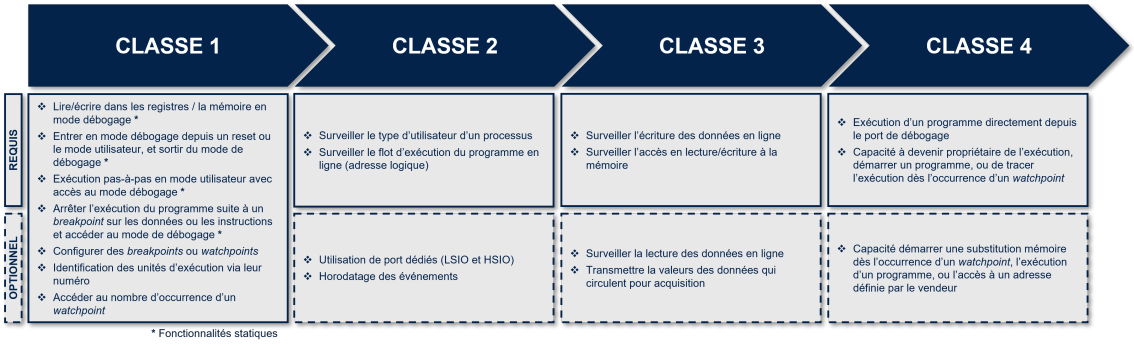


FIGURE 1.13 – Classification des conformités aux outils de développement statiques et dynamiques [38]

La Figure 1.13 présente les différents niveaux de classification de fonctionnalités liées au débogage. Elles sont réparties en classes selon le besoin de visibilité en fonction de la complexité du SoC et du type de logiciel. Par classe on retrouve des éléments dits statiques et dynamiques. Les éléments statiques sont uniquement opérables lorsque l'exécution est arrêtée, alors que les éléments dynamiques peuvent être adressés sans affecter l'exécution de l'application.

1.2.3 SiFive Insight

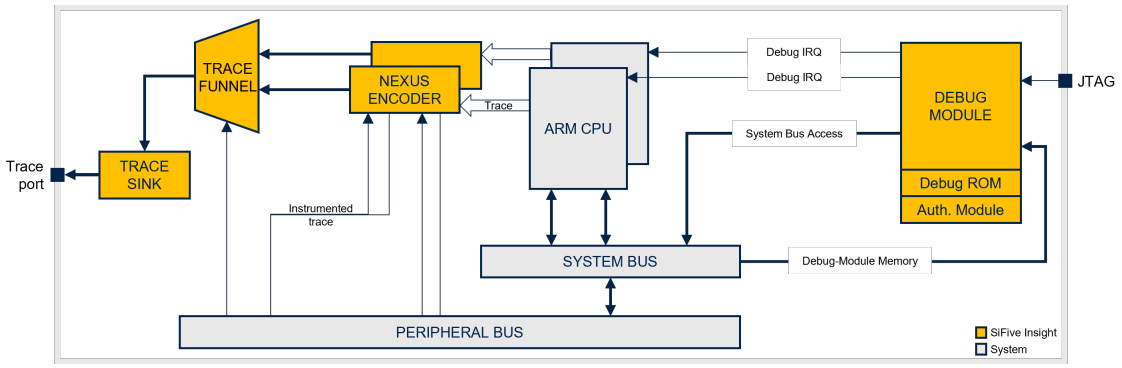


FIGURE 1.14 – Architecture de SiFive Insight - IP de débogage et de trace [3]

L'entreprise SiFive, qui développe des IPs majoritairement autour de CPU au jeu d'instruction RISC-V, propose une IP de débogage et de trace basée sur un encoder de trace répondant aux spécifications de Nexus5001 [38]. L'implémentation de cet encodeur est disponible sur ce dépôt [92]. L'architecture, illustrée Figure 1.14, est semblable à celle du CoreSight de ARM dans laquelle on retrouve une partie de débogage intrusive, ainsi qu'un mécanisme de trace CPU en ligne. On peut noter que leur concepteur de SoC en ligne [20] propose également deux techniques de gestion de la trace. Il est possible soit de la stocker dans une pile FIFO accessible depuis l'environnement de débogage, soit de l'exporter en ligne par un port dédié. La taille de la FIFO est configurable depuis l'interface de conception [20] de 256 B à 64 kB.

1.2.4 Spécifications RISC-V pour la trace et le débogage

Les spécifications sur l'architecture trace de processeurs RISC-V, dans ce document [86], établissent la structure de l'encodeur de trace, illustrée de manière synthétique Figure 1.15. Ces spécifications font la lumière sur les points suivants :

— L'interface de configuration de l'IP est décrite sous forme d'une table de registres, incluant la description de chaque champ.
— L'interface instructions et données entre le processeur et l'encodeur est détaillée. Il s'agit ici de lister tous les signaux nécessaires à la génération d'une trace d'exécution correcte.
— La détection des branchements est explicitée en détails. Elle confère à l'encodeur de trace une plus grande efficacité en termes de données à exporter afin de pouvoir reconstituer un flot d'exécution cohérent depuis le port de trace. L'approche se base sur le fichier binaire du programme pour optimiser la trace en fonction : des sauts, des appels ou retours de fonction, des instructions de branchement, des

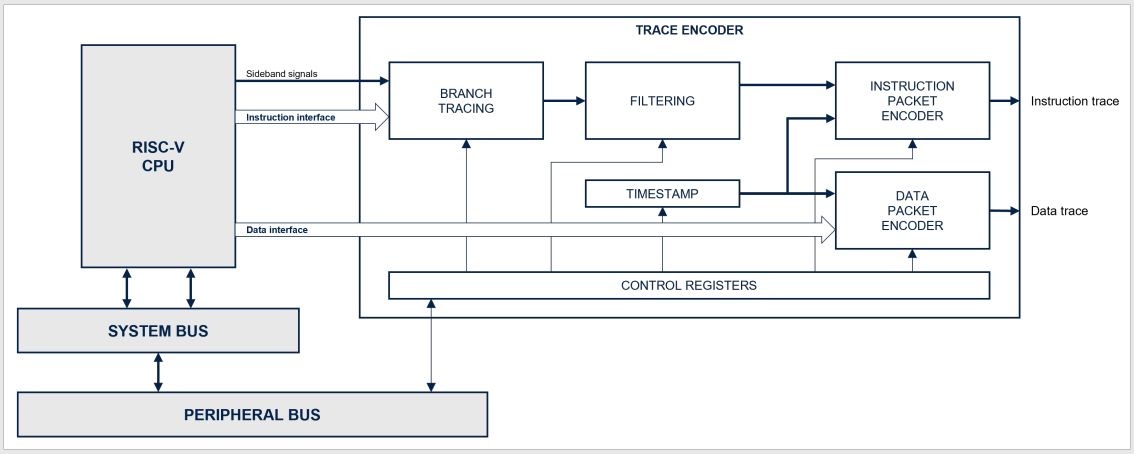


FIGURE 1.15 – Débogage et Trace pour processeurs RISC-V selon les spécifications [86]

interruptions et des exceptions.

— Le filtrage des événements proposé se base sur le contexte d'exécution déduit de la détection de branchements. Il s'agit ici de cibler un ensemble d'adresses d'instructions, de niveaux de privilèges, d'interruptions, d'exceptions, d'index temporels, etc. pour déclencher ou arrêter la génération de la trace.

— L'encodeur d'événements pour la trace d'instructions ou de données génère des trames formatées et datées dont les données à transmettre sont compressées. Cette étape est importante car elle tend à réduire le volume des données générées sans perdre d'information, et à les compacter en paquets cohérents. En effet la mémoire embarquée étant très limitée, tout comme la bande passante du port de sortie, la richesse des informations exportées dépend des optimisations offertes par l'encodeur de trace dans son ensemble.

Ces spécification [86] sont très récentes et encore à l'état d'ébauche. Le document cité est mis à jour régulièrement au moment où cet état de l'art a été rédigé. Il n'y a pour l'instant pas encore d'implémentation libre respectant ces spécifications.

1.2.5 MIPI Alliance - Application de débogage et de trace

L'alliance MIPI [27] est une organisation collaborative qui développe des solutions pour l'industrie dans le domaine de l'électronique embarquée. L'objectif de cette alliance est de concevoir et de promouvoir des interfaces matérielles et logicielles standards qui facilitent l'intégration de nouvelles IPs dans les systèmes embarqués. Les défis à relever dans ce contexte consistent à offrir une bande passante élevée, tout en maintenant une faible consommation et un faible rayonnement électromagnétique, pour le support de la trace et du débogage d'applications.

Dans cet objectif, MIPI développe des protocoles qui permettent de gérer des flux de données hétérogènes de différentes applications. Les principaux protocoles sont listés ci-dessous de manière inclusive du haut vers le bas :

— ***SneakPeek Protocol* (SPP)** : Protocole de communication d'informations de débogage entre un système et un terminal mobile comme le protocole JTAG ou

I3C.
— ***System Software* (Sys-T)** : Format de donnée standard pour transmettre une trace logicielle ou des informations de débogage.
— **System Trace Protocol (STP)** : Protocole générique de base permettant la coexistance des protocoles de trace spécifiques aux applications.
— **Trace Wrapper Protocol (TWP)** : Protocole utilisé afin de combiner les traces de différentes sources en un seul et même flux.

La sortie standard est probablement l'interface la plus ancienne et est encore très utilisée en particulier par les techniques de traces logicielles, et par exemple pour le *boot* de Linux. Elle est généralement exportée via UART, mais les débits sont très limités. Afin d'exploiter les flux de trace récoltés, des interfaces physiques ou réseaux sont également proposées pour exporter les données du SoC vers une machine hôte. Les principales interfaces proposées par MIPI sont récapitulées dans la liste ci-dessous :

— ***SensWire* (I3C)** : Interface physique minimaliste, ne nécessitant qu'une ligne d'horloge et de donnée (push-pull, open-drain), pour le débogage. Ce protocol est similaire au *SerialWire* de ARM.
— ***Gigabit Debug for IP Sockets* (GbD IPS)** : Adaptateur pour exploiter les fonctionnalités de débogage (protocole SPP) et exporter une trace via le réseau par sockets IP.
— ***Gigabit Debug for USB* (GbD USB)** : Adaptateur pour exploiter les fonctionnalités de débogage (protocole SPP) et exporter une trace via USB (supporte notamment l'USB 3.1).
— ***High-Speed Trace Interface* (HTI)** : Permet de tirer parti des connections série haute bande passante du SoC pour exporter des données de trace, comme les accès PCI Express, DisplayPort, HDMI ou encore USB.
— ***Narrow Interface for Debug and Test* (NIDnT)** : Permet d'exploiter les ports fonctionnels pour le débogage et les tests, comme les accès HDMI, DisplayPort, microSD, USB, etc.
— ***Parallel Trace Interface* (PTI)** : Port parallèle dédié pour exporter une trace sur plusieurs signaux de données et d'horloge, il est configurable selon les connexions disponibles.

1.2.6 Outils de développement pour microprocesseurs

Les outils de développement offrent des solution de débogage et fournissent en général le moyen de transport des traces ainsi que leur exploitation. Comme nous avons pu le voir au travers des différentes techniques de trace, les données de diagnostic peuvent être transmises vers l'extérieur par des ports dédiés quand les ressources le permettent. En configuration minimale, on trouve le port JTAG donnant accès à toutes les fonctionnalités mais avec une bande passante très limitée. Pour les flux de trace à haute bande passante (plusieurs centaines de Mb/s), des ports dédiés sont utilisés, mais ils nécessitent l'emploi de sondes spécifiques pour récupérer les données sur une machine hôte. LAUTERBACH [64] notamment développe des solutions dans la cadre du débogage et de la trace s'adressant à des protocoles standards libres ou propriétaires (MIPI, ARM, etc.). ARM propose la sonde DSTREAM accompagnée d'un environnement logiciel pour exploiter les ports de trace à haute bande passante et accéder au JTAG, ou encore la sonde ULINK de KEIL plus adaptée au MCUs. Ces solutions se présentent sous la forme de sondes connectées à

une machines hôte et reliée au système à étudier via l'interface appropriée.

1.3 Détection et analyse des défaillances

La qualification de systèmes à microprocesseurs sous radiations est requise dans plusieurs contextes. Dans des applications critiques, basées sur des ASICs, l'objectif est d'immuniser le système face aux événements induits par les radiations. Cette immunité provient premièrement du processus de fabrication de la technologie qui permet de faire drastiquement baisser la sensibilité des transistors aux rayonnements [89]. Ensuite, par le design des cellules logiques, il est possible de masquer l'apparition de SEEs dans les circuits. Plusieurs techniques existent à ce propos permettant de dupliquer les éléments de stockage d'information au sein d'une cellule, de tripliquer des cellules logiques séquentielles puis d'ajouter un voteur majoritaire, ainsi que de concevoir des cellules capables de se rafraîchir elle-même, afin de supprimer les erreurs masquées. Les articles [54, 55] proposent un aperçu des techniques les plus courantes de durcissement des cellules logiques séquentielles. Certaines techniques de durcissement logiciel visent également à utiliser des COTS pour des applications critiques.

Les circuits peuvent être implémentés avec plusieurs types de cellules durcies. En fonction de l'architecture, certaines zones peuvent être plus ou moins critiques. La persistance des données peut aussi varier d'une zone de registre à l'autre, or une méthode telle que la triplication de cellule avec voteur nécessite un rafraîchissement afin de rester efficace. En outre, les contraintes de synchronisation lors du placement et routage durant la conception du circuit peuvent contraindre, selon le choix du type de cellule, à diminuer la fréquence de fonctionnement [94]. Dans l'article [57], les auteurs proposent une implémentation durcie du microprocesseur HERMES (Highly Efficient Radiation-hardened Microprocessor for Enabling Spacecraft) en adaptant le choix de la technique de durcissement à la zone de l'architecture. Sur la Figure 1.16, on peut voir le découpage de la zone critique (TMR) avec l'interface du bus et les registres d'état et de contrôle. Cette zone est implémentée avec des cellules tripliquées capables de corriger un duplicata erroné, comme [54, 55]. L'autre zone (DMR) correspondant au pipeline, est quant à elle implémentée avec des cellules dupliquées capables de lever une exception si une donnée est corrompue. Le processeur entame alors une procédure de restauration.

Lors de qualifications sous radiations de tels systèmes, il est nécessaire d'avoir un minimum d'information à interpréter afin de connaître la cause de la défaillance. Les SEEs peuvent avoir des effets complexes sur les microprocesseurs. L'article [82] établit une taxonomie des erreurs qui surviennent dans les microprocesseurs suites à des fautes dues aux radiations. La Figure 1.17 restitue cette taxonomie et fait apparaître des éléments notables de l'architecture du processeur que nous allons étudier, à savoir les registres internes du pipeline et la pile FIFO pour l'interface avec le bus système.

Les corruptions de données (SDC) ne sont pas les cas d'erreur les plus courant, comme intuitivement on pourrait l'envisager [94] à cause du flot massif de données et d'opérations dans un processeur. En revanche les cas d'exception programme, de crash ou encore de *Timeout* représentent une partie importante des erreurs, et peuvent être causés par une faute dans n'importe quelle partie de l'architecture.

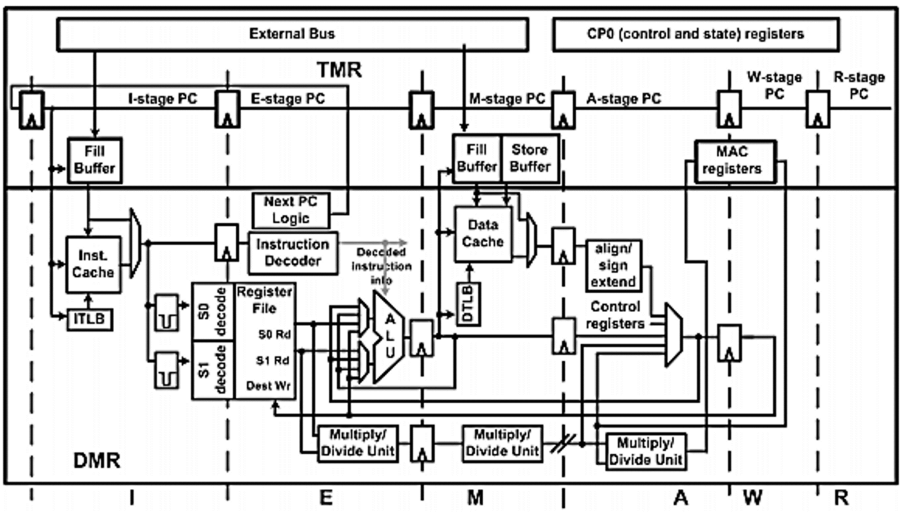


FIGURE 1.16 – Architecture du processeur HERMES et type d'implémentation (TMR/DMR) par zone du pipeline [57]

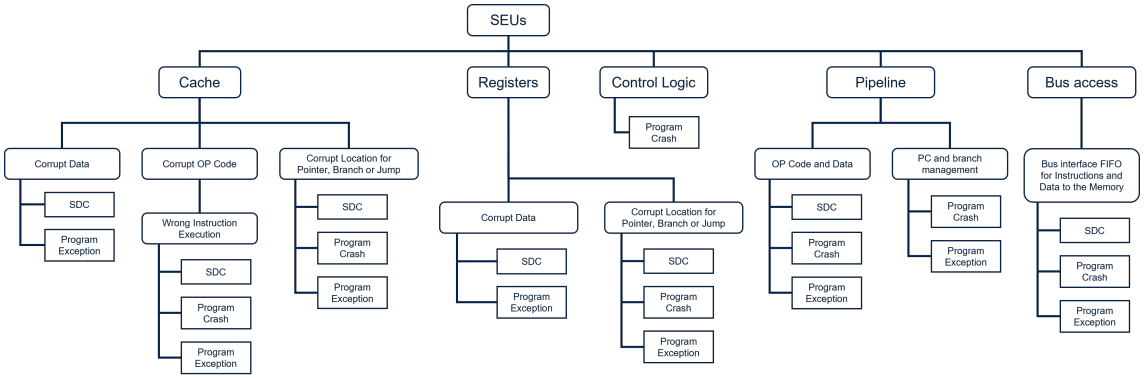
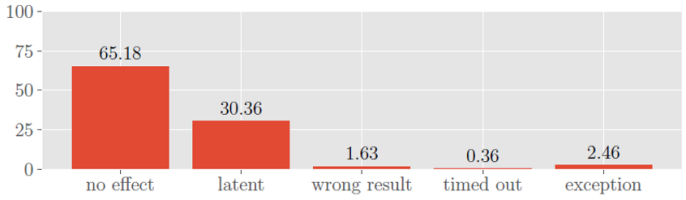


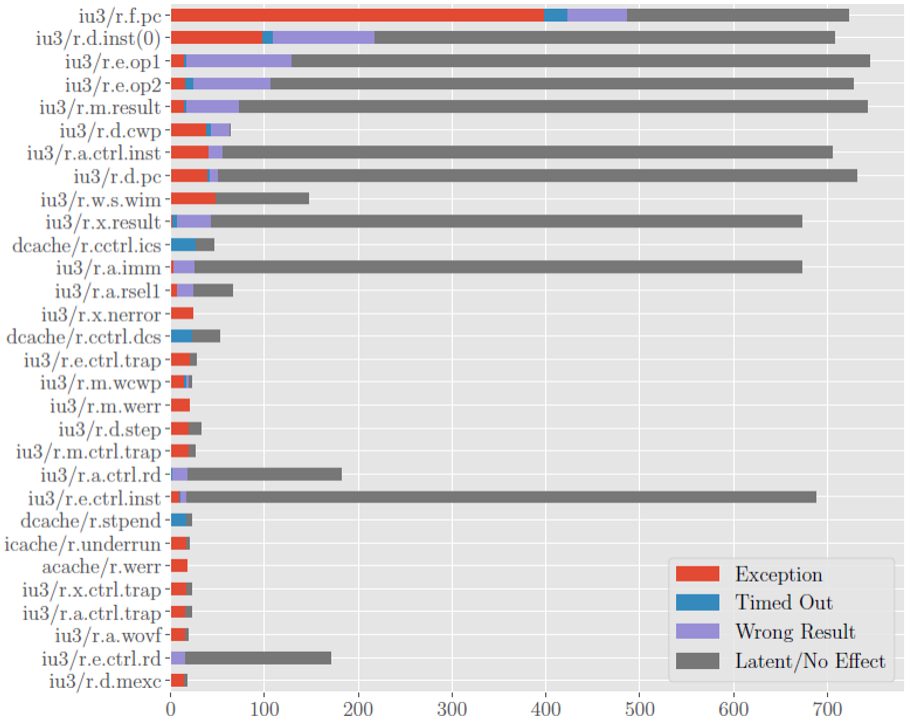
FIGURE 1.17 – Taxonomie des erreurs dans les microprocesseurs à la suite de SEUs adaptée de [82]

Certains auteurs proposent d'évaluer la criticité des zones du processeurs par une campagne d'injection de faute [11, 62, 94, 96]. Une simulation avec injection de faute consiste à reproduire l'effet d'un SEU sur le circuit, à savoir effectuer une inversion binaire aléatoirement dans une cellule logique séquentielle du processeur à un instant donné. Une simulation sans injection de faute sert de référence pour analyser les résultats. Une campagne d'injection de fautes consiste à couvrir toutes les bascules du processeur à plusieurs répétitions de sorte à répartir les injections aléatoirement durant l'exécution d'un programme par le processeur. Le programme en question comporte généralement un algorithme assez complexe, mettant à l'épreuve la mémoire et les ressources du processeur, comme le CoreMark [21]. Le but est de mettre à contribution toutes les zones du processeur. Le taux de couverture est une métrique utilisée dans ce cadre, pour évaluer un programme de test. Le programme de test se termine sur la comparaison du résultat du calcul avec une valeur de référence afin de constater de la qualité de son exécution par le processeur ayant subi une injection de faute. De fait, il est possible de dresser la cartographie des zones sensibles en fonction du taux d'erreur par faute injectée. Cependant, l'issue d'une simulation peut s'avérer être beaucoup plus complexe, la taxonomie [82] des erreurs en fonction des zones affectées sera étudiée par la suite. Ci-dessous se trouve la liste des modes de défaillance identifiés par les auteurs :

— ***No effet*** – Le résultat de l'algorithme est correct, aucune divergence n'est détectée dans le fichier de registres du processeur par rapport à la référence.
— ***Latent*** – Le résultat de l'algorithme est correct, mais le fichier de registres du processeur contient des erreurs.
— ***Wrong result*** – Le résultat de l'algorithme ne correspond pas à la référence. Dans la littérature, on retrouve aussi la dénomination *SDC* pour corruption silencieuse de données.
— ***Timed out*** – Le programme ne s'est pas terminé pour une raison inconnue.
— ***Exception*** – Une exception a été levée par le processeur pour une raison inconnue.



(a) Taux de répartition des modes de défaillance



(b) Nombre de cas d'erreur par mode de défaillance selon les registres du processeur

FIGURE 1.18 – Résultat de la campagne d'injection de fautes sur un processeur LEON3 [94]

Les résultats de la campagne d'injection de fautes menée par les auteurs sur un processeur LEON3 sont présentés sur cette Figure 1.18, on peut voir le taux de distribution des cas d'erreur, Figure 1.18a, et le taux de distribution par registre des cas d'erreur, Figure 1.18b. A la vue de ces résultats les auteurs proposent une version durcie du processeur par triplication partielle. Les registres sélectionnés pour être implémentés avec des cellules logiques séquentielles TMRs sont les 30 les plus sensibles figurant sur la Figure 1.18. Le bénéfice mis en valeur est qu'une erreur survient toutes les 133 injections contre 22 injections pour le processeur standard. Néanmoins, la fréquence de fonctionnement est passée de 59 MHz à 52 Mhz, soit une diminution de 12%.

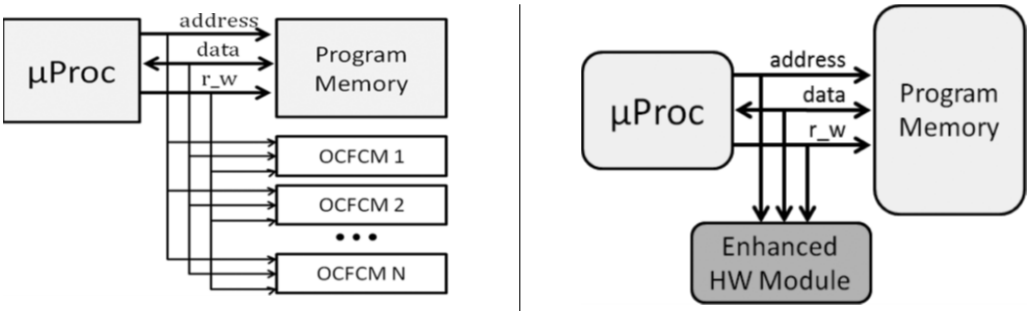
Comme nous avons pu le voir, les systèmes basés sur microprocesseurs utilisés dans des applications critiques possèdent des mécanismes de détection ou de tolérance aux fautes afin d'assurer leur fiabilité de fonctionnement. La technologie utilisée pour fabriquer le circuit, son implémentation et sa conception sont des facteurs qui peuvent être travaillés en faveur de la robustesse aux fautes induites par les radiations. Les ASICs pour les applications spatiales mettent souvent en œuvre plusieurs des trois niveaux de durcissement cités [57]. Ces techniques de durcissement matérielles font que les ASICs souffrent de piètres performances comparées aux COTS. Les COTS fabriqués avec des nœuds technologiques plus avancés, fonctionnent à des fréquences plus élevées, des tensions d'alimentation plus basses et possèdent des architectures plus complexes. Ceci au détriment de leur fiabilité dans des environnements hostiles. Par conséquent, les ASICs ont de plusieurs génération de retards [37, 79].

En plus du retard générationnel des ASICs, ils sont faits pour répondre à un besoin spécifique, et pas nécessairement fabriqués en grande série, donc très coûteux. La tendance actuelle est de plus en plus tournée vers l'utilisation de COTS pour les parties les moins critique de mission spatiales ou des missions à faibles coûts [98]. Ce type de mission s'oriente vers des satellites légers, compacts, avec une durée de vie réduites, où l'utilisation de COTS est très appropriée. On retrouve par exemple le format CubeSat [24] qui correspond à un satellite cubique de 10 cm de côté, qui donne aux étudiants l'opportunité de pouvoir déployer leur projet dans l'espace.

En raison de la plus grande sensibilité des COTS aux événements transitoires liés aux radiations, de nombreuses techniques sont apparues en liens avec la détection et la corrections des fautes [7, 8, 39, 59, 72, 73, 75-78, 80]. Ces approches proposent des méthodes hybrides où le logiciel et l'architecture matérielle sont mis à contribution dans le but de durcir les COTS [51]. Certains auteurs proposent des architectures de SoC basées sur des processeurs logiciels avec des IPs dédiées pour le contrôle du flot d'exécution [7, 8, 39, 72, 73, 80]. De nos jours, les FPGAs et notamment les cartes de développement basées sur FPGA sont devenues très abordables et offrent des performances parfois presque équivalentes aux SoCs *low-end*. Il est possible d'y implémenter des SoC multiprocesseurs et de conserver des fréquences de fonctionnement allant jusqu'à 500 MHz pour les hauts de gamme. En outre certaines carte de développement proposent des SoC possédants plusieurs cœurs et une partie FPGA ayant accès système processeur, en particulier le Zynq7000 est un cible privilégiée de certains auteurs [75-78]. Les travaux [7, 59] ont quant à eux implémenté le SoC développé sur FPGA et fourni des résultats de tests sous radiations. Ces approches sont très efficaces car elles permettent d'étudier le SoC par simulation et par la suite de l'expérimenter sur carte de développement.

Les techniques [7, 8] proposent des modules hardwares de surveillance du processeur depuis l'accès au bus principal. Les SoC mis en œuvre ont été testés via des simulations avec injection de fautes. Elles utilisent également des méthodes de surveillance du flot d'exécution par logiciel pour couvrir de manière exhaustive toutes les erreurs.

Les modules OCFCM, illustrés Figure 1.19a, (on-line control flow checker module) de [7] sont dédiés à une application. Ils sont capables de surveiller le comportement de l'exécution, à savoir l'accès aux zones mémoires et données correspondantes, les valeurs du PC et du pointeur de pile, le nombre de coups d'horloge par instruction, ainsi que la cohérence du flot d'exécution. Le module de surveillance HW de [8], illustrés Figure 1.19b, effectue des vérifications d'accès mémoires uniquement, et est opéré depuis le logiciel au travers de l'ajout d'instructions statiques dans le code pour calculer des signatures de code. Les deux



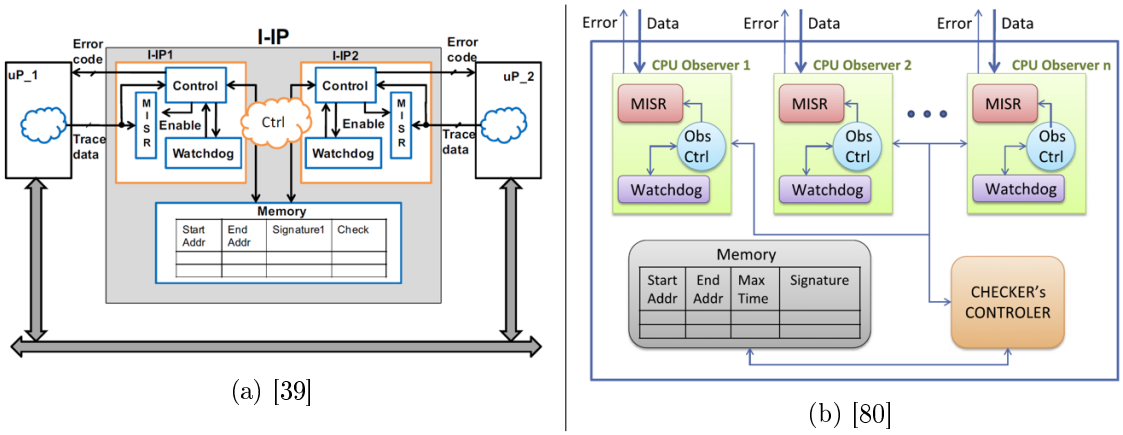
(a) OCFCM [7]

(b) HW [8]

FIGURE 1.19 – Modules de surveillance du processeur depuis l'accès au bus système présentés dans [7, 8]

approches [7, 8] affirment détecter 100% des erreurs. Cependant elles impliquent l'utilisation d'un logiciel adapté et ont des impacts importants sur l'utilisation de la mémoire et les performances du système (entre +50% et +126% de temps d'exécution et entre +159% et +192% de mémoire pour [7] ; +55% de temps d'exécution et +11% de mémoire pour [8]). L'approche [8] s'est efforcée de diminuer l'impact sur les ressources. Néanmoins, ces deux techniques [7, 8] dépendent d'un processeur et de son jeu d'instructions.

Dans les travaux [39, 80], les auteurs ont implémenté un SoC basé sur deux processeurs logiciels dans un FPGA, les deux cœurs exécutent de manière alternée les mêmes tâches. Les signatures de trace sont calculées à la volée par un module dédié, Figure 1.20, depuis le port de trace et comparées entre elles pour détecter les erreurs. Cette technique se rapproche de la technique de vérification dynamique DIVA [97], mais ne nécessitant aucune modification des processeurs. Dans ces travaux les cœurs utilisés sont des processeurs logiciels LEON3, le port de trace est directement connecté au module d'observation. Dans l'article [80], un portage a été fait sur des cœurs ARM qui nécessitent d'utiliser l'infrastructure du CoreSight pour exploiter la trace, ce qui génère une latence significative, en plus du fait que seulement les instructions de branchement génèrent des événements, comme vu précédemment.



(a) [39]

(b) [80]

FIGURE 1.20 – Module de surveillance du processeur depuis le port de trace et calcul en ligne de la signature du code

Les deux approches précédentes offrent un taux de détection d'environ 99%. Elles n'impliquent qu'une faible dégradation des performances, mais nécessitent l'emploi de deux cœurs, ce qui équivaut à un coût matériel d'environ +100%.

D'autres techniques proposent des approches hybrides pour contrôler les données et le flot d'exécution du processeur. Premièrement, le logiciel est durci selon les points suivants :

— Les données sont répliquées [17] et peuvent être récupérées en cas d'erreur [85] ;
— Les procédures calculatoires sont dupliquées et réexécutées en cas d'erreur [70] ;
— Le flot d'exécution contient des instructions de vérification des branches [7, 69, 84].

Les méthodes de durcissement du logiciel permettent d'atteindre d'excellents taux de couverture des erreurs. Il est à noter que ces techniques ne protègent pas contre les erreurs du flot d'exécution, elles permettent seulement de les détecter. Les inconvénients majeurs de ces techniques sont la complexification du flot d'exécution, une baisse importante des performances et un impact mémoire pouvant aller jusqu'à augmenter d'un facteur 3 [7]. En outre, le durcissement logiciel ne permet pas de se prémunir de fautes provenant de toutes les parties de l'architecture. Certains registres du processeur, non-accessibles depuis le logiciel, ne peuvent bénéficier d'aucune protection. Il peut en résulter les effets opposés, où comme le montre cette étude [26], une sensibilité de l'architecture a été amplifiée par l'usage d'un logiciel durci par triplication des procédures et variables. Les auteurs ont testé ce logiciel sur une plateforme reconfigurable dans laquelle a été implémenté un processeur logiciel. La méthode de durcissement logiciel a été efficace dans le cadre d'injection de fautes en mémoires. En revanche, les fautes injectées dans l'architecture du processeur ont révélé une plus grande sensibilité du système avec le logiciel durci par réplication. Une autre approche [18] consiste à évaluer par injection de fautes les registres les plus sensibles et à mitiger cette sensibilité par réplication logicielle. Ceci impose néanmoins une campagne d'injection de faute mémoire ce qui est un processus assez lourd d'autant plus si le logiciel est conséquent. Il en résulte que le taux de détection d'erreur mémoire est de 93% pour une augmentation du temps d'exécution de 77% et de l'utilisation mémoire de 73%.

C'est pourquoi, le flot d'exécution est quant à lui surveillé depuis le port de trace du processeur qui donne accès au PC et aux instructions exécutées sans affecter les performances. Les modules de détection développés [59, 72, 73] se basent sur une prédiction de la valeur suivante du PC selon l'instruction courante. A la suite de l'exécution d'une instruction de branchement, le PC aura soit la valeur du saut, soit la valeur de l'adresse suivante. Seules des méthodes basées sur la signature du code permettent de surveiller le flot de manière plus fine [28].

Les modules de contrôle du flot présentés dans [59, 72, 73] utilisent une interface personnalisée avec des processeurs logiciels de type PicoBlaze-3 et LEON3. Les techniques [59, 72] propose également de comparer les données issues de trace avec celles des entrées/sorties du processeur. En effet, les processeurs complexes possèdent parfois une profondeur de pipeline conséquente dans lequel peuvent se propager des erreurs. Les modules de contrôle du flot permettent donc de comparer ces deux points au prix d'un buffer de requêtes sur le bus dont la taille dépend de celle du pipeline. Les Figures 1.21, et 1.22 présentent les architectures précédemment décrites. Cette Figure 1.21 met en lumière le bloc de prédiction des valeurs du PC de [59, 72, 73], avec la pile nécessaire pour stocker les accès/retours de fonctions. Et celle-ci, 1.22, détaille les points d'observation et les éléments supplémentaires pour réaliser la surveillance du pipeline depuis le port de trace et l'accès bus dans [59, 72].

La détection et la correction d'erreurs logicielles selon les méthodes précédemment citées entrainent une complexification significative du flot d'exécution pouvant avoir des effets inattendus sur la sensibilité d'un SoC. Dans le cadre d'une évaluation de logiciel durci

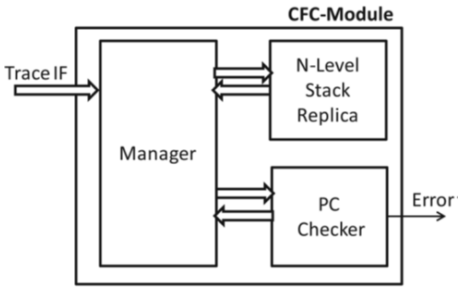
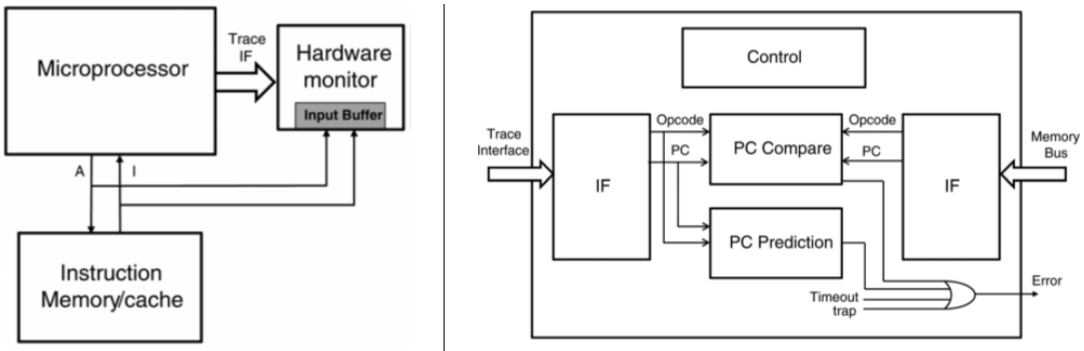


FIGURE 1.21 – Module de surveillance du processeur depuis le port de trace [73]



(a) Points d'observation du système

(b) Architecture du module matériel

FIGURE 1.22 – Module de surveillance du processeur et de l'accès au bus système [59, 72]

par triplication des procédures [26], via un SoC logiciel implémenté dans une plateforme FPGA testée sous ions lourds, les auteurs mettent en lumière une augmentation significative du SER par rapport à un logiciel standard.

Les approches [75-78] se basent sur l'architecture de trace non intrusive présente dans les microprocesseurs, à savoir le CoreSight de ARM dans un SoC Zynq7000 possédant deux cœurs ARM Cortex-A9 ainsi qu'une zone de logique programmable. Dans ces travaux, les auteurs vont se concentrer sur une technique visant à expérimenter ce COTS dans diverses conditions de test, que ce soit sous protons, neutrons, ou LASER. Le but est de savoir comment et pourquoi les défaillances apparaissent. Le diagnostic des erreurs est une tâche difficile à cause des nombreuses parties pouvant collecter des fautes dans les processeurs comme vu dans la Figure 1.17. Par conséquent, très peu d'information sont disponibles concernant les vulnérabilités des processeurs ou pour évaluer leur section efficace. Cet axe de recherche a été adressé depuis de nombreuses années dans la littérature [90]. Des techniques commencent à apparaître pour l'automatisation de test de circuit logiques [12]. Cette approche [67, 68] basé sur le calcul d'une fonction de hachage sur toutes les entrées/sorties d'un microprocesseur, fait apparaître clairement les limites en termes de bande passante et de ressources de ce type d'observation considérant les processeurs actuels. Des techniques [2, 29, 52] sont également à l'œuvre afin de faciliter le travail de validation des composants une fois fabriqués. La complexité des systèmes basés sur processeurs est telle que la tâche de validation du fonctionnement de l'ensemble de circuit après sa fabrication représente une partie importante du temps de développement [32]. Ces techniques offrent une visibilité très détaillée sur l'architecture d'un SoC. Elles sont néanmoins très spécialisées pour cette tâche et s'apparentent plus à des techniques de débogage des cœurs par accès JTAG, qu'à une surveillance en ligne du flot d'exécution. Au

vue des outils de trace du CoreSight détaillé dans la Section 1.2, le travail des auteurs de [75-78] pourra directement être lié à des blocs connus de l'architecture de trace. La Figure 1.23 illustre l'architecture implémentée dans le SoC et la manière dont elle est utilisée dans le but de faire du diagnostic en ligne, lors de tests sous radiations.

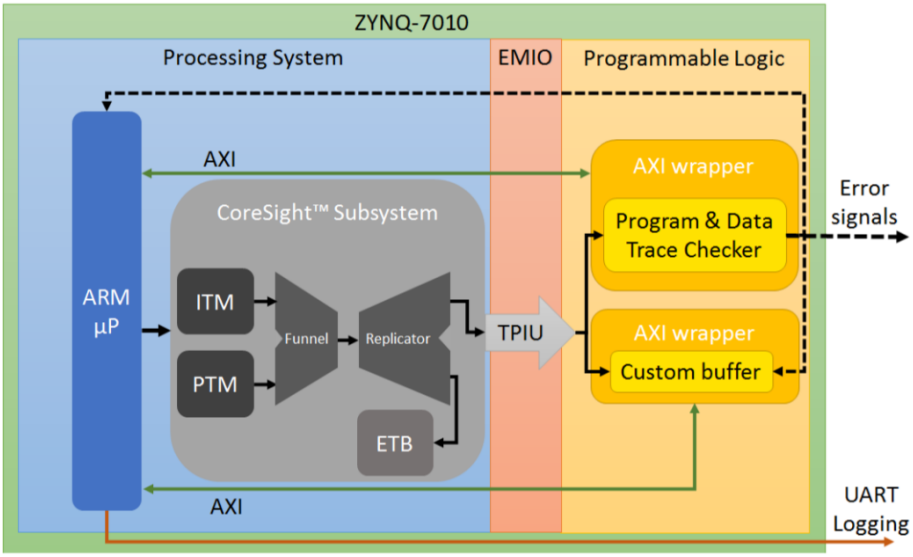


FIGURE 1.23 – Module de surveillance du processeur depuis le port de trace de l'architecture du CoreSight pour le SoC d'un Zynq7000 [73]

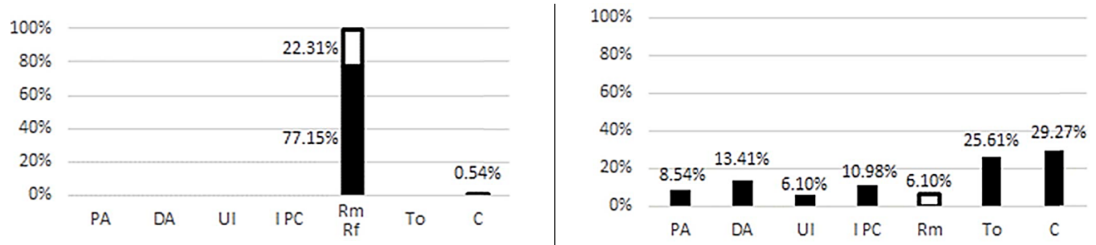
Cette architecture est composée des blocs PTM et ITM (équivalents respectifs de l'ETM et STM pour des SoCs de faibles ressources). Les flux de trace sont ensuite dirigés vers un *Funnel*, puis vers un *Replicator* pour obtenir deux flux d'événements étiquetés par bloc. L'un est stocké dans un buffer embarqué (ETB), et l'autre est exporté par le TPIU vers l'IP PDTC dans la logique programmable du SoC. Le logiciel peut transférer des résultats de calculs vers le PDTC afin d'en vérifier les valeurs. L'ITM possède 32 registres mappés en mémoire, l'algorithme possède donc 32 canaux d'exportation de résultats en vue de les vérifier. De cette manière, il est possible de contrôler en ligne les résultats de calcul de l'algorithme avec le minimum d'intrusion. En ce qui concerne la trace, l'ETM génère un événement contenant l'instruction et son adresse, à chaque branchement dans le flot d'exécution, L'ETM fournit donc régulièrement la valeur du PC. Le PDTC contrôle l'évolution du PC afin qu'il reste dans une zone définie, les bornes sont extraites à la compilation. Une banque de registres a également été implémentée dans la logique programmable de manière à pouvoir stocker des éléments du contexte d'exécution au moment de l'erreur, comme des données pertinentes de la mémoire et le pointeur de pile. À la suite d'une erreur, les données sont exportées via le port série, à savoir les informations capturées par le PDTC, la zone mémoire dédiée, et le contenu de l'ETB. Cette approche a permis aux auteurs d'enrichir les données de diagnostic dans le cas d'erreurs ayant pour cause des fautes induites par les radiations. On retrouve les signatures d'erreurs suivantes :

— Erreur de données : plusieurs déclinaisons existent en fonction du nombre de bits erronés ;
— Exception matérielle : le CPU possède des mécanismes de détection des erreurs et est capable de fournir des informations de diagnostic :
  Prefetch Abort (PA) : Accès mémoire en zone interdite pour une instruction ;
  Data Abort (DA) : Accès mémoire en zone interdite pour une donnée ;
  Undefined Instruction (UI) : Instruction illicite ;

Invalide PC (IPC) : PC hors zone mémoire.

— Délais dépassé : le programme n'a plus donné signe de vie durant un temps trop long, l'application est bloquée sans raison apparente.

— Problème de communication : les données de diagnostic n'ont pas pu être récupérée.



(a) Cache de données de niveau L1

(b) Cache d'instructions de niveau L1

FIGURE 1.24 – Histogramme des erreurs de l'injection de fautes par LASER dans les caches [78]

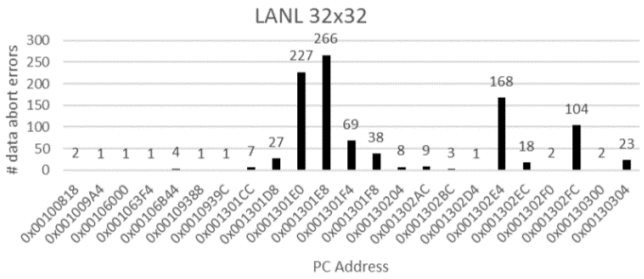
Il apparaît que le point faible de cette approche lors des tests sous radiations réside dans la manière d'exporter les données vers l'extérieur. La Figure 1.24 montre que nombreux sont les cas où il a été impossible de récupérer les informations de diagnostic lorsque les caches ont été touchées par l'injection de faute par LASER (0,5% des cas pour les caches de données, ce qui représente le 2^{me} maximum) (30% des cas pour les caches d'instruction, soit le maximum de cas). De plus, l'article [75] fait mention que les cas où la trace récupérée depuis l'ETB est exploitable sont faibles lors de tests sous flux de protons, tout comme [77]. Ces deux articles [75, 77] remettent directement en cause la fiabilité des éléments du CoreSight et notamment du buffer dans le cadre de tests sous radiations.

Une autre technique [91] propose d'enregistrer les informations disponibles lors d'un crash sous forme d'événements dans un buffer afin de faciliter le diagnostic. Ces travaux s'adressent à des produits finis dans le but d'améliorer l'efficacité du support pour le système d'exploitation. Le principe se base sur la génération d'événements contenant des informations sur l'exception causant le crash sur l'état courant de l'unité d'exécution. Ces événements sont stockés dans des buffers embarqués selon leurs priorités. L'auteur a notamment précisé que lors d'un crash sévère, de multiples rapports pouvaient être générés. Dans ce cas, les buffers d'événements atteignent leurs limites et les nouveaux rapports écrasent les plus anciens. Pourtant, seuls les premiers rapports liés au crash possèdent réellement des informations pertinentes. Un agent de tri a donc été développé dans le but de filtrer les événements selon leurs priorités et de les répartir dans les trois buffers, en cas de congestion de la mémoire. Cette approche permet d'avoir accès à l'instruction ayant provoqué une exception mettant fin à l'exécution du programme. On retrouve cette démarche dans les travaux [77, 78] où les auteurs, tout en injectant des erreurs dans le système, collectent les adresses des instructions erronées étant la cause de l'exception. Il s'avère que certaines instructions collectent beaucoup plus d'erreur que les autres comme on peut le voir sur la Figure 1.25. Il s'agit notamment d'instructions de branchement conduisant à un saut, ou d'accès mémoire. Certains exemples cités illustrent la propagation d'une corruption de donnée, tel un compteur avec un inversion binaire, ayant conduit à un accès hors zone mémoire. Il apparaît donc que des instructions sont plus sensibles si leurs actions dépendent d'un certain nombre d'opération effectuées auparavant. L'étude [18] met

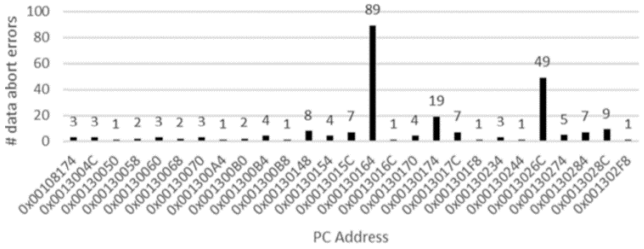
également en avant les sensibilités hétérogènes des instructions au travers d'une campagne d'injection de fautes.

0x001301C4	ldr	r8, [r3, #4]
0x001301C8	add	r3, r3, #2048
0x001301CC	ldr	r6, [r3, #-2040]
0x001301D0	add	r2, r2, #16
0x001301D4	**ldr**	**r4, [r3, #-2036]**
0x001301D8	cmp	r3, r9
0x001301DC	**ldr**	**r5, [r2, #-8]**
0x001301E0	ldr	lr, [r2, #-4]
0x001301E4	**ldr**	**r7, [r2, #-12]**
0x001301E8	mla	r0, r6, r5, r0
0x001301EC	mla	ip, r8, r7, ip
0x001301F0	mla	r1, r4, lr, r1
0x001301F4	bne	1301c4

(a) Code désassemblé



(b) Tests sous neutrons

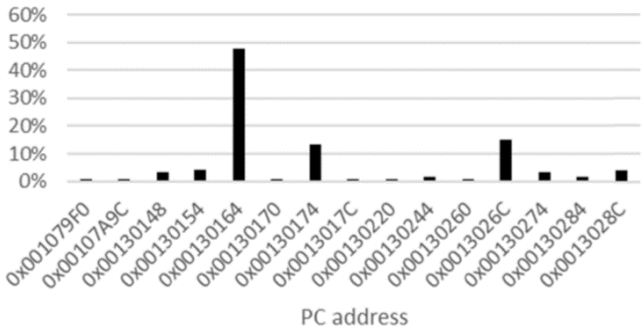


(c) Tests sous protons

FIGURE 1.25 – Histogrammes du nombre de cas de *fault exception* en fonction de l'adresse de l'instruction fautive lors de tests sous radiations accompagné du code désassemblé [78]

Address	Content	Instruction	
0x00130230	ea000003	b	130244
0x00130234	e2822010	add	r2, r2, #16
0x00130238	e28cc010	add	ip, ip, #16
0x0013023C	e1520004	cmp	r2, r4
0x00130240	**0a000013**	**beq**	**130294**
0x00130244	e992000a	ldmib	r2, {r1, r3}
0x00130248	e592000c	ldr	r0, [r2, #12]
0x0013024C	e5871000	str	r1, [r7]
0x00130250	e1530000	cmp	r3,r0
0x00130254	01510003	cmpeq	r1, r3
0x00130258	e5863000	str	r3, [r6]
0x0013025C	13a0e001	movne	lr, #1
0x00130260	03a03001	moveq	r3, #1
0x00130264	13a03000	movne	r3, #0
0x00130268	e5850000	str	r0, [r5]
0x0013026C	11a0800e	movne	r8, lr
0x00130270	1afffef	bne	130234
0x00130274	e59c0004	ldr	r0, [ip, #4]

(a) Code désassemblé



(b) Injection de fautes par LASER dans le cache

FIGURE 1.26 – Histogramme du nombre de cas de *fault exception* en fonction de l'adresse de l'instruction fautive lors de tests sous impulsions LASER accompagné du code désassemblé [78]

Cette approche d'analyse de la sensibilité d'instructions du code a également été adressée par campagne d'injection de fautes basée sur simulation dans [18], dans le cadre d'un durcissement logiciel sélectif. Cependant, les résultats illustrés sur la Figure 1.25 ne se basent que sur 3 valeurs de PC précédant l'exception qui termine l'exécution. L'observabilité de l'erreur est par conséquent très réduite.

Afin d'améliorer la profondeur de stockage d'événements de trace, la compression est une technique ayant été abordée dans le but d'optimiser l'utilisation de la mémoire embarquée. Cette approche [53] a été expérimentée sur un module de trace embarqué pour signaux logiques dans des FPGAs, Integrated Logic Analyser (ILA), développé par Xilinx.

Ce module permet de tracer des signaux internes, d'enregistrer les données dans un buffer embarqué et de les récupérer depuis une machine durant la phase de développement. Les auteurs affirment avoir augmenté jusqu'à 90% le nombre d'événements stockés grâce à l'utilisation d'un algorithme de compression par dictionnaire, Lempel-Ziv-Welch (LZW). En revanche, le dictionnaire a lui seul une taille de 1 kB, ce qui est déjà conséquent au regard de la taille des buffers de trace vu précédemment. Outre le dictionnaire, cette méthode de compression est d'autant plus efficace sur un volume de donnée important. Ce qui implique de surcroît l'emploi de buffer de trace d'autant plus important.

1.4 Procédures de qualification sous faisceau de particules

L'organisme Join Electron Device Engineering Council (JEDEC) œuvre dans un but de normalisation des semi-conducteurs. Cet organisme a rédigé la norme JESD89B [49] qui s'adresse aux SEEs dans les composants électroniques dû aux radiations dans le domaine terrestre. Les perturbations correspondantes à l'environnement d'évolution du système, ou aux impuretés du boîtier du composant, sont émulées par différents types de particule. Cette norme spécifie tous les paramètres et éléments qui gravitent autour de la qualification, afin de rendre comparable les résultats d'expérience entre les différents acteurs des tests. Les domaines d'application ciblés sont le terrestre, et les orbites basses et géostationnaires.

Deux méthodes d'exposition du composant à son environnement ont été déterminées dans la cadre de qualifications de cellules mémoires ou de cellules logiques séquentielles. Les tests en temps réels, ou non-accélérés, peuvent être conduits en haute altitude pour des application spatiales [6]. Ces tests peuvent durer des mois, voir même des années, et ne sont pas compatibles avec les besoins industriels. Pour cette raison, nous nous intéressons aux tests accélérés (ASER), durant lesquels le composant est soumis à un flux de particule contrôlé, supérieur à celui de son environnement. L'intérêt est d'atteindre des fluences de plusieurs années d'exposition dans l'espace en quelques minutes, éventuellement quelques heures, sous faisceau dans des accélérateurs à particules. Lors des irradiations deux modes de test sont identifiés, le principe est illustré sur la Figure 1.27. Le mode statique concerne les éléments logiques avec rétention d'information, comme les mémoires volatiles (SRAM, DRAM, etc.) et les bascule (D, JK, etc.). Dans ce mode, la structure à tester est initialisée avant l'exposition, et son contenu est examiné après. L'autre mode, le mode dynamique implique de vérifier en permanence l'apparition des erreurs et de les corriger le cas échéant.

Afin de qualifier des circuits sous radiations, les standards (JEDEC, ESCC) recommandent d'élaborer un plan de tests complet (DOE - *Document of Experiment*). Ce plan comprend :

— Le type de SEE d'intérêt (SEU, MBU, SET, SEL, etc.) ;
— Les conditions de tests (tension, température, fréquence) ;
— La source d'irradiation (Alpha, Neutrons, Protons, Ions lourds) avec le flux et la fluence à atteindre ;
— Le nombre minimum d'événements à collecter pour établir des métriques, et arrêter un test si ce nombre est atteint avant la fluence spécifiée.

Pour obtenir des métriques fiables lors des expérimentations, il est nécessaire de connaître les paramètres suivants :

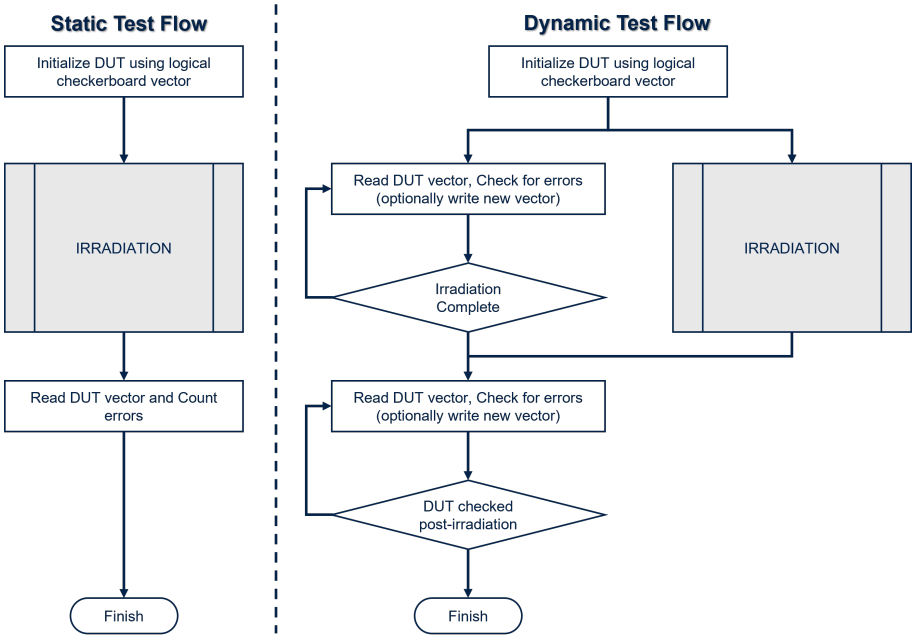


FIGURE 1.27 – Procédures de test statique et dynamique de cellules logiques séquentielles sous radiations adaptée de [49]

— La fluence effective sur le DUT lors d'une exposition ;
— Le nombre de DUTs exposés durant les tests ;
— Le nombre et le type d'événements collectés ;
— Il est intéressant d'obtenir une estimation au travers de la littérature du nombre d'événements que l'on est supposé obtenir dans les conditions de test selon les trois points précédents. Ceci permet de s'assurer du bon déroulement des tests.

Comme évoqué dans les Sections 1.1 et 1.3, les installations présentent dans les accélérateurs à particules notamment, sont très contraignante vis-à-vis du matériel utilisé pour opérer le DUT. Il faut s'assurer d'être en mesure d'adresser les points suivants :

— Contrôler le DUT à distance, pouvoir l'initialiser et exécuter des fonctions basiques de vérification de son fonctionnement ;
— Choisir entre le mode de test statique ou dynamique et supporter ce mode ;
— Opérer un reset matériel du circuit à distance ;
— Être en mesure de détecter les événements d'intérêt et de journaliser leur apparition en incluant une datation.

Les conditions de tests sont à contrôler et à suivre de manière précise afin que l'expérience soit répétable et que les mesures faites puissent être pertinentes dans le contexte de l'application. Elles regroupent l'ensemble des points suivants :

— Comme évoqué dans la Section 1.1, dans le cadre de test de tableaux mémoire ou de cellules logiques, le choix du motifs a son importance et aura un impact sur la mesure ;
— Il en est de même pour le choix de la fréquence de fonctionnement du circuit, de sa tension d'alimentation et de sa température ;
— Pour la mesure du taux d'erreur, le type d'élément logique à tester est à prendre en compte. Il faut être en mesure d'opérer chaque IP selon une procédure qui lui

est propre (tableau mémoire, cellule logique séquentielle, FPGA, microprocesseur, etc.). En particulier, les FPGAs sont sujet à des fautes dans leur mémoire de configuration, ce qui complexifie la taxonomie des erreurs.

Pour procéder aux expérimentations, il faut sélectionner les échantillons à tester, et les monter sur la carte DUT afin de les opérer. Cette carte DUT devra être fixée et positionnée de manière précise en face du faisceau, la position angulaire pouvant avoir un impact conséquent. Les tests avec une source Alpha imposent d'ouvrir le boîtier du composant de sorte que la source soit en face de la couche active du silicium. Les tests sous ions lourds nécessitent également d'amincir le boitier et le substrat du circuit afin de viser un certain LET. Une procédure de test (DOE) est ensuite détaillée par type de particule.

Une série de documents rédigés par JEDEC viennent en complément de la norme JESD89B définissant les exigences standards de mesures des taux d'erreur, pour spécifier en détails les méthodes de tests pour les différents types de qualifications. Cette série de documents proposent des méthodes pour déterminer le taux d'erreur de bascules et de cellules mémoires dans des conditions de test ambiantes (JESD89-1B [46]), en conditions de test accélérées par une source Alpha (JESD89-2B [47]) et sous faisceau de particules (JESD89-3B [48]). D'autres documents spécifient les techniques de test sous faisceau d'ions lourds (JESD57A [45]) et de protons (JESD234 [44]). Ces techniques de tests accélérées s'adressent quant à elle aux applications spatiales.

1.5 Conclusion

Au terme de cet état de l'art, nous avons couvert les thématiques de recherche en lien avec les contributions, les solutions et standards clés pour la mise en œuvre de la plateforme, et les normes de tests sous radiations qui définissent le cadre des expérimentations radiatives.

La qualification de cellules logiques séquentielles est un besoin industriel existant. Cependant, les articles passés en revue proposent tous des techniques différentes, il n'y a donc pas de standard excepté pour l'utilisation d'un registre à décalage comme structure de test. Selon les cellules à qualifier et le profil de mission, il n'est pas forcément nécessaire d'exposer un grand nombre de cellules. Dans notre contexte, la technologie utilisée pour les vecteurs de test est durcie par conception. Néanmoins, un nombre suffisant d'événements est requis pour obtenir une mesure représentative du taux d'erreur. L'utilisation d'un grand nombre de cellules est préférable pour diminuer les temps de faisceau tout en collectant suffisamment d'événements, ce qui limite les coûts des expérimentations dans les accélérateurs à particules. Nous avons également vu qu'utiliser un motif possédant des changements d'états est plus représentatif du fonctionnement attendu des cellules dans un système, et donc du taux d'erreur. Les événements provenant de l'arbre d'horloge sont donc détectables et génèrent de nombreuses inversions binaires. L'échantillonnage de la valeur du compteur permet, à posteriori lors du traitement des données récoltées, de comptabiliser les événements selon les types : SETs dans l'arbre d'horloge ou SEUs dans les DFFs. Cependant réside une incertitude dans la mesure due à la faible fréquence d'échantillonnage qui peut masquer des événements rapprochés. Comme nous avons pu le voir, certains auteurs ont dû restreindre la fréquence de fonctionnement du DUT durant les tests pour s'assurer d'être en mesure de différencier les types d'événements. Notre première contribution s'ins-

crit donc dans ce cadre en adressant les limitations de fréquence du DUT, et d'observabilité des événements qui surviennent, dans le but d'améliorer les conditions des tests et la fiabilité des résultats. Ce travail a un intérêt industriel direct puisqu'il adresse une difficulté rencontrée lors d'expérimentations très onéreuses, dont la fiabilité des composants produits par la suite, et la satisfaction client dépendent.

L'observabilité des microprocesseurs dans le cadre des tests radiatifs est une thématique très peu abordée dans la littérature. Seulement quatre publications récentes du même auteur [75-78] ont présenté des résultats de tests sous faisceau et sous LASER d'un SoC. Cependant, le point de vue de l'approche s'adresse plutôt à l'étude des sensibilités de l'application. L'utilisation du CoreSight s'est révélée être pertinente dans ce cas, mais pas suffisamment fiable pour exploiter la trace récoltée. Les méthodes de détection et d'analyse des erreurs se sont révélées être très enrichissantes et ont permis d'appréhender les modes de défaillances d'un processeur. Notamment la technique de simulation avec injection de fautes permet d'évaluer la cartographie de la sensibilité d'un processeur et de son application, elle a servi de base aux investigations durant ces travaux. La détection d'erreur est un sujet connexe à la contribution, mais dimensionnant pour notre approche visant à capturer la propagation d'une faute. Cette thématique est adressée par de nombreux auteurs puisqu'elle permet de s'assurer de la cohérence du fonctionnement d'une application et de l'intégrité des résultats. Ces objectifs sont atteints dans l'optique de répondre au standard ISO26262 pour l'automobile et les véhicules autonomes, et de détecter des attaques malveillantes pour la sécurité des systèmes. Les techniques de détection d'erreur matérielles sont très intéressantes pour l'aspect non intrusif, mais pour atteindre des taux de détection d'erreurs de plus de 90%, l'impact mémoire peut être conséquent, ce qui n'est pas en adéquation avec les contraintes des tests radiatifs. En effet, la fiabilité des moyens de mesure mis en œuvre est absolument nécessaire pour la collecte et l'intégrité des résultats. Les techniques logicielles sont plus flexibles et peuvent être utilisées sur une plateforme sans support matériel dédié. Elles restent néanmoins très intrusives envers l'application, ce qui résulte en une baisse significative des performances et nécessite beaucoup de développement. Les modules matériels de surveillance du processeur, se basant sur l'évolution des valeurs du PC et des adresses des accès aux données, représentent le meilleur compromis entre taux de détection et impact sur les ressources. Les mécanismes de détection d'erreurs du processeur sont également un moyen efficace et sans impact sur les ressources pour accéder aux premières informations de diagnostic. La capture de la propagation des fautes dans un processeur n'est apparemment pas une thématique couverte à ce jour. Seules quelques approches proposent de surveiller la propagation de valeurs de PC dans le pipeline jusqu'aux requêtes d'accès aux instructions en mémoire. Une approche propose également de sauvegarder le maximum de messages d'erreur dans le but de restituer le maximum d'information en cas de défaillance. Ces deux techniques ont permis de consolider l'approche basée sur la capture de la propagation de fautes dans le processeur, dans le cadre de tests radiatifs.

Chapitre 2

Qualification de cellules logiques séquentielles et classification en ligne d'événements isolés

Dans ce chapitre, nous décrivons l'ensemble de la démarche de qualififation de cellules logiques séquentielles sous radiations, ainsi que la contribution visant à classifier les événement selon leur provenance dans la structure de test. La techniques mise en œuvre a été implémentée dans deux circuits dédiés aux tests sous faisceau de particules, fabriqués durant la deuxième et la troisième année de thèse. Ces deux circuits, SERVAL et SQUAL, sont conçus respectivement pour les domaines d'application automobile et spatial, en technologie 28 nm UTBB FD-SOI.

En premier lieu, nous passons en revue les contraintes techniques liées aux conditions de test sous radiations, notamment dans les accélérateurs à particules, afin de justifier l'ensemble des moyens déployés pour opérer au profil de mission les composants sous faisceau. Les procédures de test et les méthodes utilisés sur les circuits adressés sont décrites, elles concernent également les contributions des chapitres suivants.

Ensuite, nous présentons la technique proposée adressant les limites des techniques à l'état de l'art pour la qualification de cellules logiques séquentielles dans le cadre industriel. La contribution repose sur la méthode CREST (*Circuit for Radiation Effects Selft-Test*) [63] et rend possible les qualifications au profil de mission de manière autonome selon les exigeances industrielles de cette technologie durcie.

2.1 Banc et protocoles de test de systèmes logiques sous radiations

La classification d'événements intervient lors de tests sous radiations utilisés pour mesurer de manière expérimentale le taux d'erreur d'éléments logiques lors de test accélérés. Ces tests sont opérés en exposant directement le DUT sous faisceau de particules, ce qui constitue le principe de la technique ASER. Comme abordé au travers de l'état de l'art, Section 1.1, cette technique impose de nombreuses contraintes. Ces contraintes sont directement répercutées sur le circuit et le banc de test. Les choix techniques sont justifiés dans les sous-sections suivantes. Le banc de test décrit dans cette section a également été utilisé pour opérer les plateformes SoCs présentées dans le Chapitre 3.

2.1.1 Infrastructures d'irradiation et contraintes des installations

Les infrastructures d'irradiations qui fournissent les faisceau d'intérêt pour la qualification de circuits sont principalement des accélérateurs à particules [15, 25, 43, 56, 74, 83, 95]. Ces infrastructures imposent de nombreuses contraintes sur le dispositif permettant de mettre en œuvre le DUT. La chambre d'irradiation est isolée de la salle de contrôle par d'épais murs de béton. L'opérateur des tests est donc contraint d'être éloigné d'une à plusieurs dizaines de mètres du DUT. Dans la chambre d'irradiation, tout le matériel présent est exposé. Les équipements de laboratoire de mesure et d'acquisition de données, tels que les analyseurs numériques, sont donc sujets à des erreurs ou des pannes. Étant donné leur coût et le manque de fiabilité dans ces conditions, il est préférable de ne pas les utiliser. En outre, dans le cas de tests sous ions lourds le DUT peut parfois être placé dans une cuve sous vide, ce qui limite le nombre de connexions avec l'extérieur. Le cumul des contraintes sur le banc de tests résulte en une diminution drastique de la bande passante pour échanger des informations entre l'utilisateur et le DUT.

Pour résumer, le stockage d'information dans le testeur lorsqu'il est situé dans la chambre d'irradiation n'est pas à privilégier à cause du manque de fiabilité dû aux radiations. Lors des tests actifs, il faut également un contrôle en ligne des taux d'erreurs afin de redémarrer le test sans délai si besoin. Les solutions sur étagère d'échange de données sans fils sont à proscrire. Le signal serait très atténué par les murs de béton, les plaques de plomb protégeant le testeur, ou encore une éventuelle cuve sous vide. Il ne serait pas raisonnable de mettre en péril une campagne de tests à cause d'instabilités de communication. Les bancs et le plan de test doivent être parfaitement prêts avant la campagne. L'organisation de telles campagnes s'effectue à distance, des informations sur les contraintes mécaniques du banc de test sont échangées avec les ingénieurs en charges des infrastructures. Une fois sur le site d'irradiation, la campagne commence avec un temps limité. Étant donné que les coûts de faisceau peuvent s'élever à 1 K$ de l'heure, l'objectif est de rentabiliser de manière fiable les expérimentations. Plus précisément, le temps de faisceau est optimisé en minimisant les temps morts qui peuvent être liés à une éventuelle prise de décision qui nécessite du temps d'analyse ou de dépouillement des données. L'observabilité en ligne sur le déroulement d'une expérience est donc un atout de taille dans ce contexte. Le point de départ de la réflexion sur la manière de concevoir une plateforme de test commence donc par choisir une manière d'opérer le DUT et de communiquer avec l'opérateur en respectant ces contraintes.

2.1.2 Plateforme de test basée sur FPGA contrôlée à distance

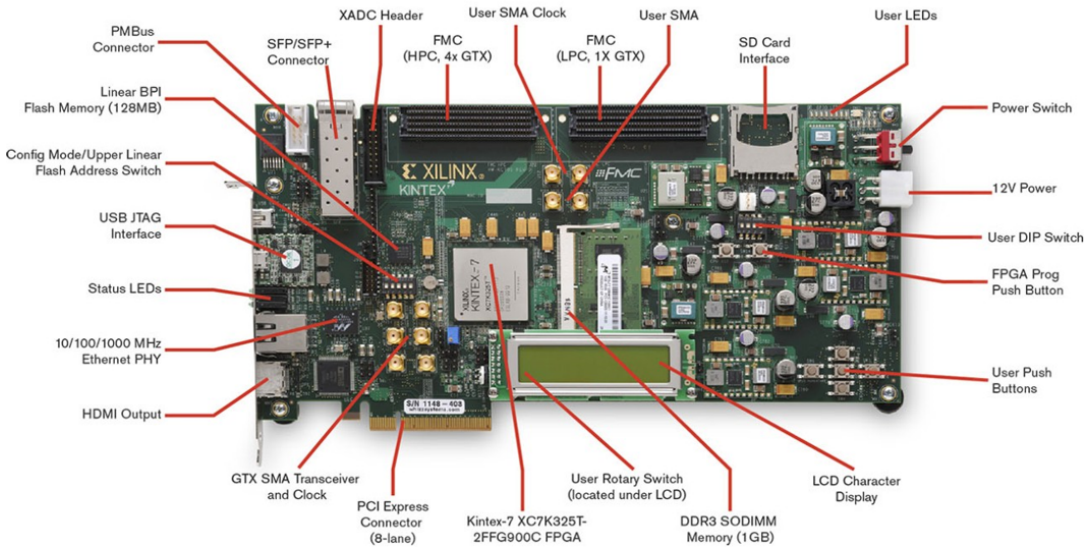


FIGURE 2.1 – Carte de développement Xilinx KC705

La solution utilisée pour notre plateforme de test est basée sur une carte de développement FPGA de Xilinx, le modèle Kintex-7 KC705[100], illustré Figure2.1. Le FPGA de la famille des Kintex-7 est gravé en 28nm. Cette carte a été choisie en particulier pour les connecteurs FMC qui offrent une grande capacité de connexions avec le circuit à tester et permettent d'utiliser une carte fille. Pour opérer le DUT à tester sous faisceau une carte fille dédiée est conçue pour faire l'interface entre le testeur et le composant. Cet élément est fait au besoin en fonction des contraintes et du plans de tests. Certaines cartes filles possèdent un socket pour interchanger les composants, sur d'autres le composant est soudé à la carte, permettant ainsi d'utiliser des tapis chauffants pour les tests en température. C'est un des éléments à modifier en fonction du boitier du DUT. Voici une illustration, Figure2.2, de l'ensemble carte FPGA et carte fille pour les deux DUTs SERVAL et SQUAL, comportant les contributions de ces travaux, ces contributions sont détaillées par la suite. Dans cette configuration utilisée pour les tests en laboratoire, la carte fille est directement connectée sur la carte FPGA, il est possible d'insérer un prolongateur FMC entre des deux cartes lors des tests radiatifs.

Lors des tests dans des infrastructures d'irradiation, les contraintes mécaniques liées aux installations des différents accélérateurs à particules, en particulier le positionnement face au faisceau, font qu'il est préférable de séparer la carte FPGA et la carte fille. Les deux cartes sont éloignées d'au moins un mètre, ce qui permet de fixer la carte fille sur un support en face du faisceau et d'éloigner la carte FPGA. Dans les accélérateurs à protons, comme [74], sont fournis des blocs de plombs afin de protéger le plus possible les instruments de mesure. Enfin, selon les fluences à atteindre, des cartes dédiées comportant plusieurs DUTs peuvent être fabriquées pour maximiser la surface exposée au faisceau.

Dans le FPGA est implémenté un banc de test permettant d'opérer les DUTs. L'architecture du banc de test est représentée sur la Figure 2.3, on distingue la carte mère basée sur FPGA, le banc de test embarqué implémenté dans le FPGA, ainsi que la carte fille avec le DUT. Ce banc est dédié à chaque DUT selon les IPs à tester. Le moyen de transmission retenu pour communiquer avec celui-ci est une liaison série RS232. Par cette

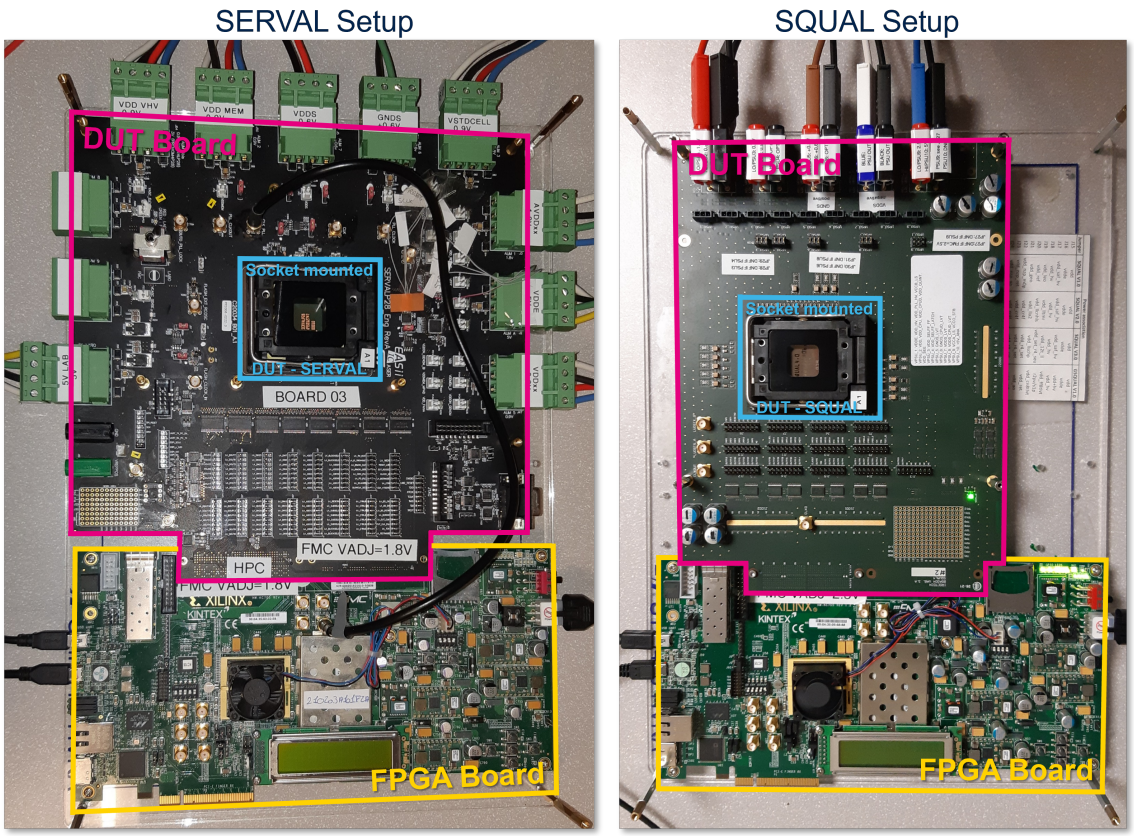


FIGURE 2.2 – Plateformes de test basés sur FPGA et cartes DUT

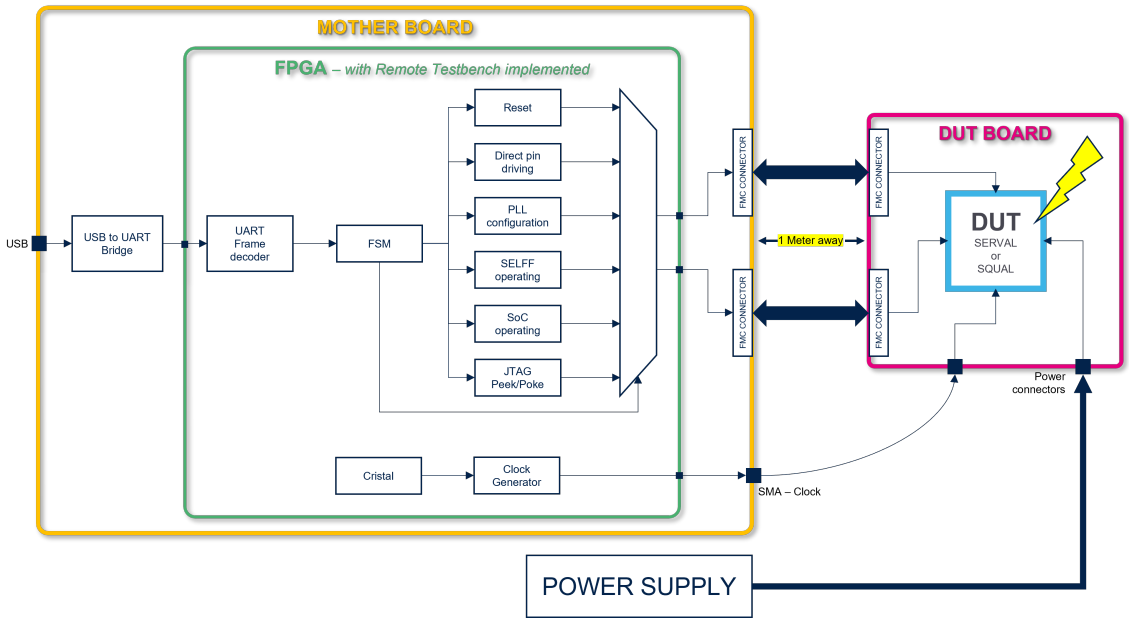


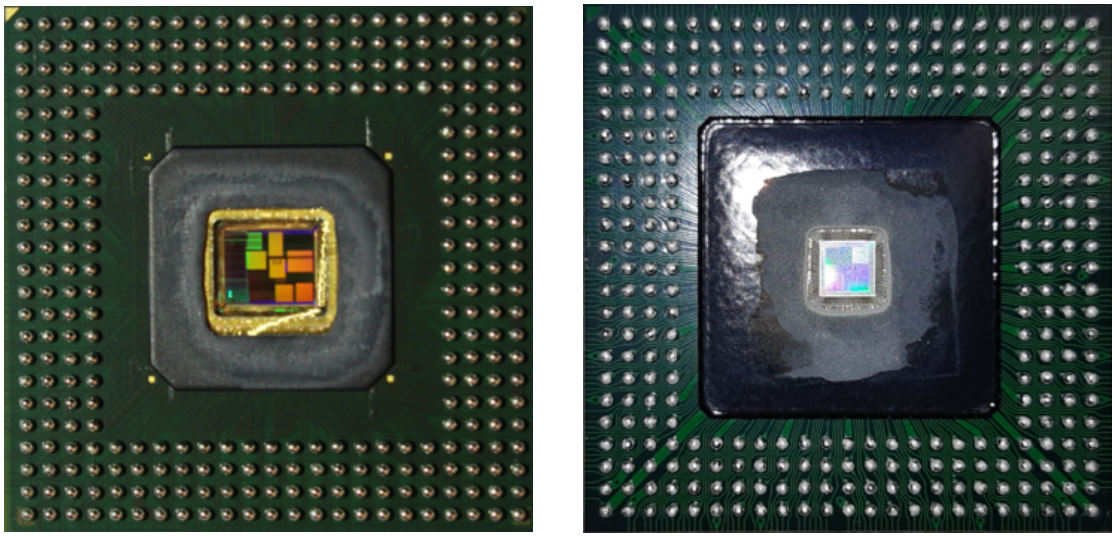
FIGURE 2.3 – Configuration du banc de test du DUT basé sur FPGA

liaison, l'opérateur, depuis sa machine hôte, peut transmettre des commandes de haut niveau au banc de test. Le FPGA interprète ces commandes et les traduit en séquence de signaux à appliquer et à lire aux bornes du circuit sous test. Ces commandes peuvent donc déclencher le changement d'état de broches sur le DUT jusqu'à générer des trames suivant un protocole (JTAG par exemple). Elles sont spécifiques aux IPs à tester et sont détaillées

par la suite selon les IPs d'intérêts.

À chaque bloc du circuit à tester correspond un bloc de contrôle dans le FPGA. La commande reçue par le FPGA contient l'identifiant du block à programmer ou à interroger, que le bloc de contrôle traduit en séquence de signaux. Dans la mesure où le nombre de fonctions du circuit est largement plus important que le nombre de broche, le bloc à tester est également relié aux entrées sorties du circuit par des multiplexeurs instanciés dans le circuit de test.

2.1.3 Composants à tester



(a) SERVAL (b) SQUAL

FIGURE 2.4 – Composants de test - vue arrière avec boîtier BGA304 ouvert pour tests sous particules Alpha

Notre contribution est intégrée dans deux circuits de qualification de technologie, ils comportent les techniques présentées dans ce chapitre, ainsi que le Chapitre 4. Les circuits SERVAL et SQUAL ont été fabriqués par STMicroelectronics dans l'usine de Crolles avec la technologie 28 nm UTBB FD-SOI. Les circuits SERVAL et SQUAL sont des circuits de tests d'applications qui s'adressent respectivement aux secteurs automobile et spatial d'orbite géostationnaire (GEO). Les deux circuits sont illustrés sur la Figure 2.4. Ils sont montés dans des boitiers BGA304 qui permettent une ouverture côté couche active et substrat afin d'exposer le silicium sans nuire à la fonctionnalité du circuit. Cette vue face arrière (côté broches) dévoile la *die* de deux pièces dont le boîtier a été ouvert.

2.1.4 Procédures de test

Les procédures de test appliquées pour les qualifications sous faisceau de particules suivent les différentes méthodes de test [30, 44-48] correspondantes aux spécifications de la norme JESD89B de JEDEC pour le terrestre et de la norme 25100 de ESCC pour le spatial. Étant donné les domaines d'application des circuits de test, les qualifications sous radiations se feront donc respectivement sous faisceau de neutrons et d'ions lourds pour

SERVAL et SQUAL.

De telles expérimentations nécessitent beaucoup de préparation afin de garantir un bon déroulement et d'obtenir des résultats. Il s'agit pour cela de tester toutes les capacités du circuit pour s'assurer que le banc de test est en mesure de les opérer et que les pièces à tester fonctionnent. Cependant, dans le cadre de la mesure du taux d'erreurs logicielles, les fonctionnalités de détection ou de comptage des erreurs ne peuvent être mise à l'œuvre que grâce à l'injection d'erreur dans le circuit. Le plus simple serait d'avoir une méthode BIST embarquée, mais cela ne permet pas toujours de s'assurer du fonctionnement de l'ensemble de chaîne de mesure. Les autres moyens de validation du circuit utilisés consistent à injecter des erreurs en son sein, de sorte à faire apparaître les erreurs attendues lors des campagnes d'irradiation.

Durant les phases de qualification sous faisceau de particules, il est important de s'assurer de la fiabilité de l'ensemble de l'installation et que les taux d'erreurs mesurés soient cohérents avec les perturbations induites par le flux de particules, comme le suggère les recommandations de JESD89B [49]. En pratique, une grande quantité de matériel doit être déplacée pour tester les circuits dans les infrastructures d'irradiation. Les campagnes de tests se font à toute heure du jour ou de la nuit, il faut donc redoubler de vigilance quant au bon fonctionnement de l'installation. Pour cela, avant chaque début de test sous faisceau, le circuit est laissé en fonctionnement quelques minutes sans perturbation. Puis il est exposé aux radiations durant l'expérience, en prenant soin de noter le début de l'exposition. Ensuite, l'exposition prend fin et le circuit est de nouveau laissé en fonctionnement. Ce découpage en trois phases permet de vérifier que le circuit démarre d'un état stable et retrouve un état stable une fois hors du faisceau, ce qui confirme que le taux d'erreur mesuré correspond aux perturbations induites par les radiations.

Dans la mesure où le faisceau n'est pas constant, à part dans le cas des sources radioactives, une surveillance du flux est réalisée durant le test. Cette surveillance peut être obtenue auprès de la facilité, ou en utilisant un compteur de particules.

Les sous-sections ci-dessous, listent les moyens d'injection de fautes et de qualification mis en œuvre dans le cadre de nos travaux, dont les résultats sont présentés dans le Chapitre 5.

Injection de fautes par impulsion électromagnétique (EMFI) :

L'outils utilisé durant ces travaux pour l'EMFI est le ChipSHOUTER [19]. Cet appareil permet de générer une impulsion électromagnétique, contrôlée en largeur et en amplitude, de manière focalisée, et proche d'un circuit. La sonde où est générée l'impulsion se situe à l'extrémité du boîtier. Elle est composée d'un noyau de ferrite entouré de quelques tours de fil de cuivre. L'appareil génère une impulsion de courant dans la sonde par décharge capacitive. Il est possible de paramétrer la tension de charge de la capacité, la largeur de l'impulsion ainsi qu'un train d'impulsion paramétrable. Lors de son utilisation, l'appareil est placé sur une potence et la sonde est positionnée contre le circuit de manière orthogonale. L'appareil est contrôlé via une liaison série par des commandes envoyées depuis un terminal. Bien que le comportement physique des perturbations générées par les impulsions n'ait rien en commun avec l'effet des particules sur le circuit, il est possible de créer des fautes qui s'apparentent à des SEUs ou des SETs. Les impulsions électroma-

gnétiques proches de ce dernier induisent des pics de courant dans les lignes métalliques, ce qui provoque les fautes. Ce devrait donc principalement être des SETs qui surviennent dans les signaux d'horloge ou de reset du circuit. En effet, ces signaux se propagent dans l'intégralité du circuit, et possèdent en majorité les lignes les plus longues. Néanmoins, cet appareil a permis de générer des fautes de type SEU dans les circuits en optimisant les paramètres tels que le positionnement, la tension de charge de la capacité et le type de sonde. Les principaux intérêts de son utilisation sont la praticité et la flexibilité de manipulation lors d'expérience en laboratoire. Grâce à cet outil d'injection de fautes, il a été possible de couvrir tous les modes de défaillance attendus, et de valider le fonctionnement de l'ensemble du banc de test.

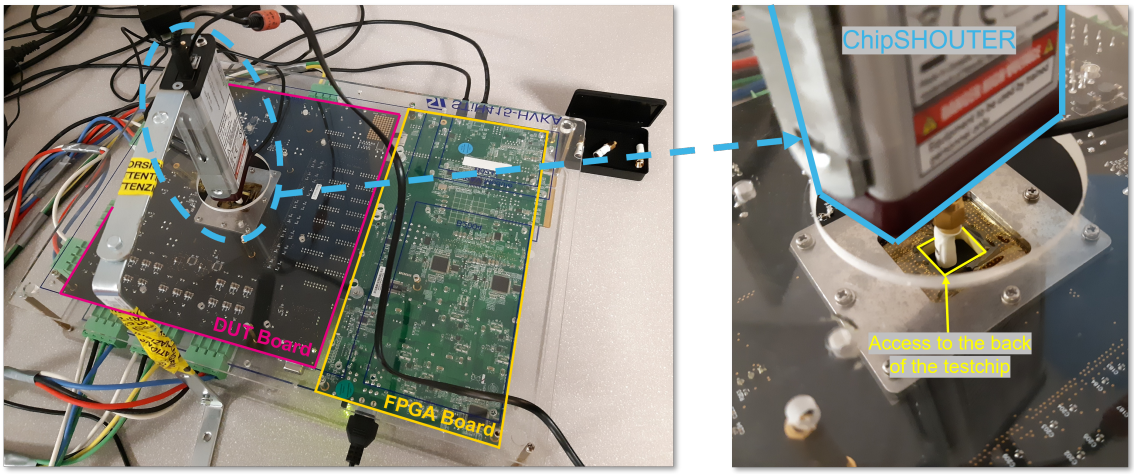


FIGURE 2.5 – Expérience en laboratoire avec le ChipSHOUTER - sous impulsions électromagnétiques

La Figure 2.5 illustre la mise en place du banc de test. Le ChipSHOUTER est maintenu par une potence au-dessus du banc de test. La sonde est placée directement contre le composant pour avoir la configuration ou le champ magnétique est le plus concentré au niveau du circuit. Ce positionnement permet aussi de s'affranchir du paramètre de la profondeur qui aurait été difficilement maîtrisable dans ces conditions. Nous avons choisi d'attaquer le composant coté couche active pour que les impulsions rencontrent directement les couches d'interconnexions métaux.

Particule Alpha :

Les particules Alpha sont potentiellement générées par des impuretés radio-isotopiques du boîtier même du composant, les expérimentations sous alpha peuvent donc faire l'objet d'une qualification. Dans le cadre de nos travaux, ces expérimentations sont seulement utilisées à des fins de test via injection de fautes dans le circuit. Les tests sous faisceau de particules Alpha sont assez simples et pratiques à réaliser en laboratoire. Ils demandent toutefois un balisage préalable de la zone où sont effectués les tests, qui par ailleurs doivent être réalisés à proximité du lieu de stockage sécurisé de la source. La source utilisée est une source Alpha [4], la partie active est composée de l'isotope américium 241. Son activité est de 3.7 MBq. Elle émet un flux d'ions He^{2+} de $2.44e^{6}\alpha/cm^{2}/s$. Ce type de particules nécessitent de la prudence de manipulation de la source, mais leur portée se limite à quelques centimètres dans l'air, et une feuille de papier suffit à les arrêter. Cette source peut donc être manipulée directement par nos soins dans le laboratoire où elle est à disposition.

A cause de la faible capacité de pénétration des particules dans la matière, le boîtier du composant à irradier doit être retiré afin que la source soit le plus proche possible de la surface active. Pour se faire, la face avant du boîtier est retirée par action mécanique ou chimique, il ne reste donc plus qu'une très fine couche de résine et l'enchevêtrement des lignes de métaux pouvant atténuer la particules Alpha. Comme l'illustre la Figure 2.4, une fois le boîtier ouvert, on peut distinguer les différents blocs sur les circuits. Etant donnée la technologie de fabrication des composant de test considérés et le durcissement intrinsèque qu'elle procure au circuit, il est difficile de provoquer des événements à tension nominale. Pour garantir une sensibilité maximale et obtenir les événements d'intérêt dans le cadre de nos tests, la tension d'alimentation a été diminuée au maximum, comme le stipulent les travaux [50]. Ce faisant, il est possible de générer des fautes dans les circuits et de procéder à des campagnes de test. L'objectif de cette démarche est double, elle permet à la fois de finaliser la validation de l'ensemble du banc de test, et d'obtenir des statistiques sur l'apparition des erreurs ou une section efficace, au même titre qu'une phase de qualification. Les résultats des sessions de test sont présentés dans la Section 5.3.2, la configuration de l'installation est illustrée sur la Figure 2.6. On voit en bas à droite de l'illustration, une photo montrant l'avant du composant avec le boîtier ouvert et l'autre avec la source radioactive directement posée dessus.

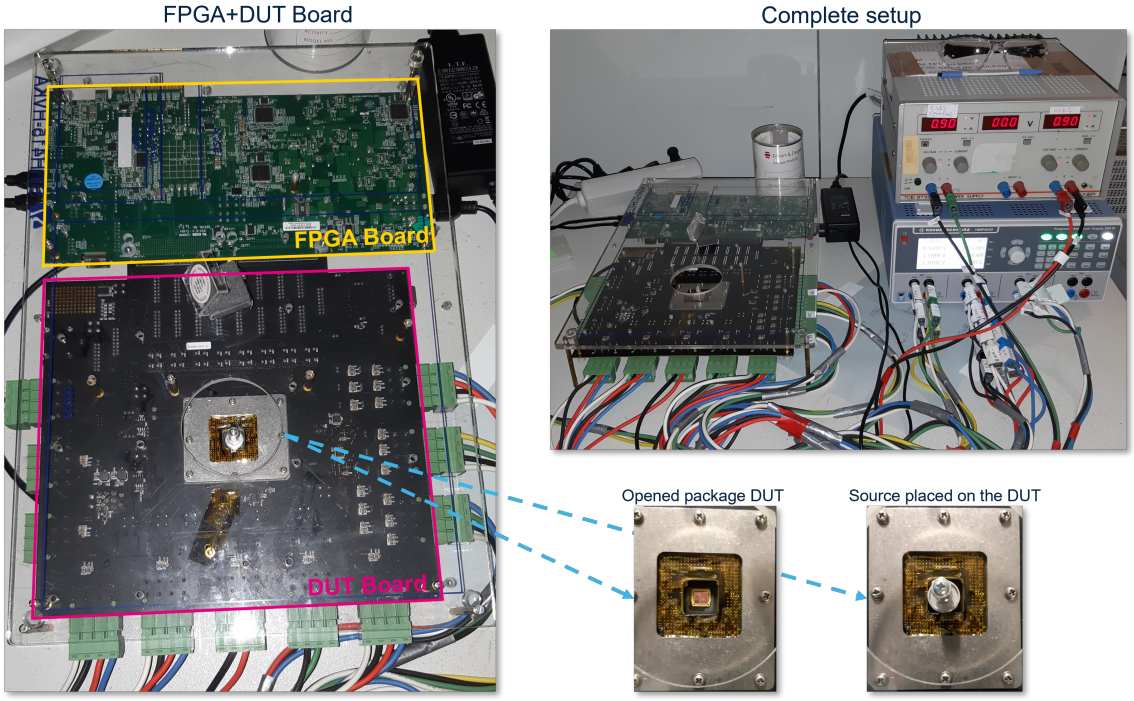


FIGURE 2.6 – Expérience en laboratoire avec une source radioactive d'américium 241 - Circuit SERVAL sous faisceau de particules alpha

Rayons X :

Dans l'industrie du semiconducteur, les machines à rayon X sont plutôt utilisées à des faibles niveaux d'énergie pour contrôler l'assemblage de produits, notamment au niveau des soudures. Les rayons X utilisés dans ces tests sont de toute autre nature, ils sont utilisés pour évaluer leur impact sur le circuit. Les rayons X ont des effets importants sur le silicium mais non favorables à l'apparition de SEUs dans nos circuits de test étant donnée leur technologie de fabrication. Les rayons X ont un effet ionisant très important. Il en résulte

donc que, lors des expérimentations sous rayon X, il est difficile de collecter des cas d'erreurs logicielles avant que la dose en arrive à empêcher le composant de fonctionner. Ce type de rayonnement est très pénétrant, il n'est donc pas nécessaire d'amincir le boîtier, la résine de ce dernier n'atténue que très peu le rayonnement. Il reste néanmoins préférable d'irradier le composant par l'avant pour éviter aux rayons d'avoir à traverser le substrat de silicium avant d'atteindre la couche active. Plusieurs techniques d'exposition aux rayons X existent. Soit le composant est entièrement exposé sous un large faisceau, soit le rayonnement est délivré de manière pulsée à travers un faisceau de quelques μm de diamètre. La première méthode est couramment utilisée pour tester la résistance des circuits à la dose, tandis que la deuxième permet de cibler précisément des éléments logiques et de réaliser une cartographie de sensibilité par scan du design. Ceci constitue un atout considérable pour l'évaluation des différentes techniques proposées dans ces travaux. Des tests sous rayons X ont été effectués à l'ESRF à Grenoble [31] sur la ligne ID09, les résultats sont détaillés dans la Section 5.3.3. Voici une illustration d'expérimentations menées à L'ESRF [31], Figure 2.7, sur laquelle on distingue la carte FPGA, la carte fille en face du faisceau et les alimentations de laboratoire. Les résultats, ainsi que les propriétés et la configuration du faisceau lors des tests sont détaillés dans la Section 5.3.3.

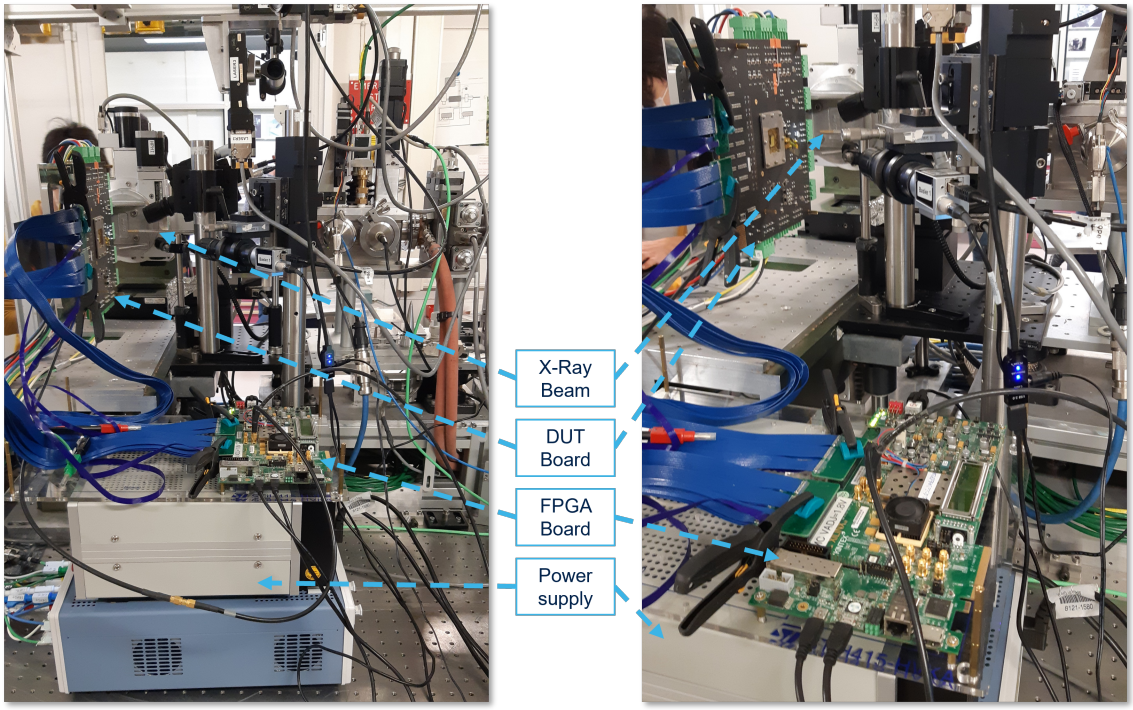


FIGURE 2.7 – Expérience menée à l'ESRF [31] - Circuit SERVAL exposé à un faisceau pulsé de rayons X

Faisceau de protons :

Les faisceaux de protons sont communément utilisés pour la qualification des circuits destinés aux orbites LEO. Dans notre cas, nous avons utilisé les protons en tant qu'équivalence aux neutrons, suivant les spécifications JESD89B [49]. L'accès aux établissements fournissant des flux de neutrons a été fortement contraint par la crise sanitaire, et les tests n'auraient pas pu être menés en temps voulu. Ces tests ont donc permis de qualifier le circuit SERVAL pour des applications terrestres. Les campagnes ont eu lieux dans le la-

boratoire Paul Scherrer Institut (PSI) [74], en Suisse. Il n'est pas nécessaire d'amincir le boîtier pour ces expérimentations, les protons sont très pénétrants et traversent le boîtier de part en part. A l'inverse, ils sont très dur à arrêter, rebondissent sur toutes les surfaces, et menacent d'autant plus le reste du banc de test, à savoir le testeur basé sur carte FPGA et l'alimentation. Afin de protéger les alimentations, lors de notre expérience, elles ont été disposées au-dessus de la chambre d'irradiation et connectées au circuit à travers une chicane dans le blindage. La carte FPGA est déportée de la carte fille et protégée du mieux possible avec des blocs de plomb placés autour, ce qui améliore grandement la fiabilité du banc de test. Les résultats de cette campagne de test sont présentés dans la Sections 5.1.1. Sur la Figure 2.8, on peut voir différentes photos des installations et de la configuration du banc de test. Dans la salle d'irradiation, la carte fille est fixée sur un support mobile afin d'aligner le faisceau en face du DUT. La carte FPGA est éloignée au maximum du faisceau et entourée de bloc de plomb. Une dalle de 2 mètres d'épaisseur de béton confine la salle d'irradiation. Les alimentations et la salle de contrôle sont donc situées deux étages plus haut. Les câbles sont acheminés dans la salle à travers une chicane dans le mur de béton, et arrivent par le plafond. De cette contrainte naît l'ensemble des exigences en termes de fiabilité et de simplicité du banc de test. Comme ces tests devaient être normatif et des fluences élevées devaient être atteintes, une carte dédiée à ce test comportant 4 circuits irradiés simultanément a été développée, elle est présentée sur la Figure 2.8.

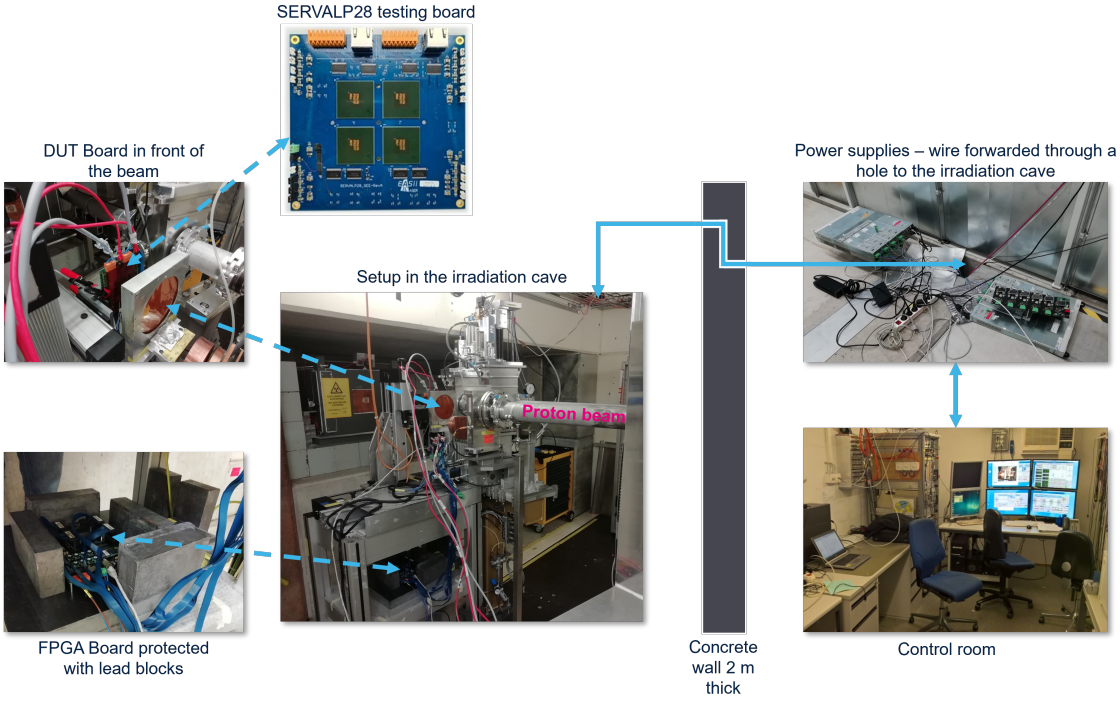


FIGURE 2.8 – Expérience menée à PSI [74] - Circuit SERVAL exposé à un faisceau de protons

Faisceau d'Ions lourds :

Les ions lourds sont utilisés pour qualifier des circuits destinés à des orbites géostationnaires. Les faisceaux sont communément de diamètre assez faible (de l'ordre de 3 cm). Pour qualifier le circuit SQUAL, circuit de test d'applications spatiales, des pièces ont été exposées sous faisceau d'ions lourds à l'air libre dans le laboratoire RADiation Effects

Facility (RADEF) [83]. L'intérêt de cet établissement est que le faisceau peut être calibré pour irradier les puces dans l'air, et non dans une cuve sous vide. En effet, les tests sous vide imposent des contraintes sur la carte, elle ne doit pas comporter de condensateurs à électrolyte liquide qui risquent d'éclater. La cuve impose aussi des contraintes de temps de mise sous vide entre chaque changement de pièce, et de connectivité vers l'extérieur qui se limite aux connecteurs présents sur la flasque étanche. La pénétration des ions lourds est limitée, la configuration des tests est donc faite afin d'optimiser le transfert linéique d'énergie (LET) dans le circuit, plus précisément au niveau de la surface active. Pour ce faire, le circuit est irradié depuis l'arrière, les ions traversent donc tout le substrat et vienne se loger dans la surface active, ce qui constitue le cas le plus défavorable. Le paramètre sur lequel nous pouvons jouer pour effectuer l'optimisation est l'épaisseur du substrat du circuit, afin de positionner le curseur de l'énergie déposée sur la valeur maximum appelée le pic de Bragg. Sur la Figure 2.9, on peut voir en bas à droite un graphique représentant le LET en fonction de l'épaisseur du composant et cocktail d'ions lourds utilisé [42]. On peut également remarquer que les DUT ont été soudé sur les cartes filles, ces dernières sont donc aussi nombreuses que les pièces à tester. Cette solution était plus rentable et imposait moins de délais de livraison plutôt que d'utiliser un socket. On aperçoit sur les cartes filles une sonde de température coté avant, et une plaque chauffante à l'arrière, utilisées pour les tests en température. Les résultats de cette campagne sont présentés dans la Section 5.1.2.

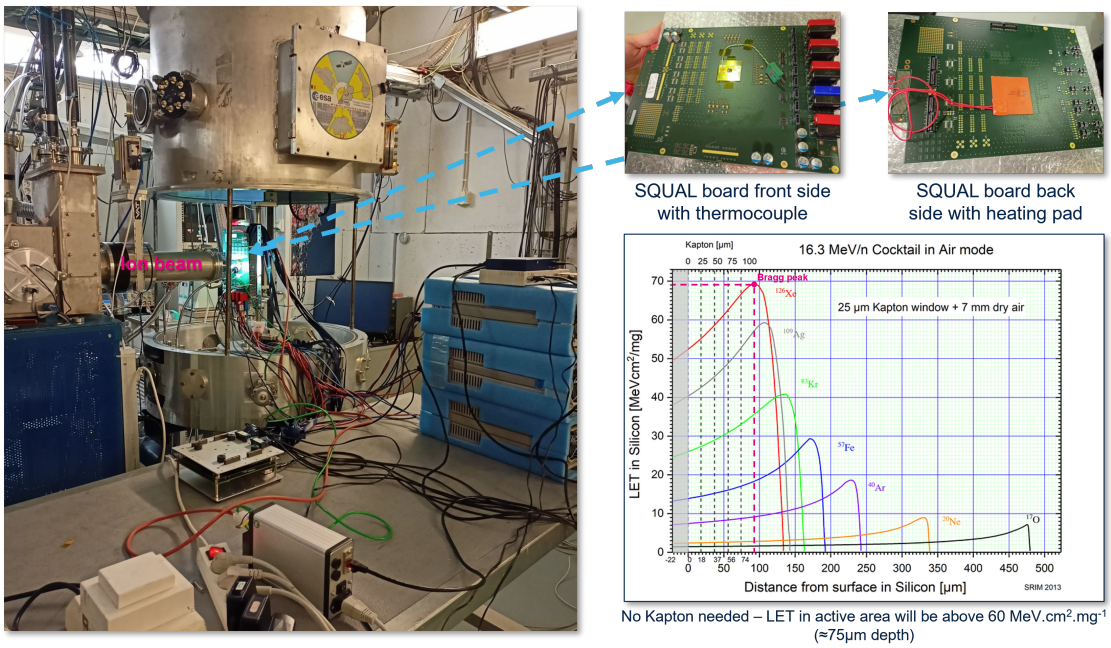


FIGURE 2.9 – Expérience menée à RADEF [83] - Circuit SQUAL exposé à un faisceau d'ions lourds

2.1.5 Calcul de la section efficace

Le taux d'erreur logicielle de cellules logiques sous faisceau de particules est mesuré durant la phase de qualification. Le principe de la technique du taux d'erreur logiciel accéléré (ASER) est d'exposer un circuit sous un flux de particule beaucoup plus important que celui imposé par son environnement, et par conséquent de lui imposer un taux d'erreur plus élevé. Cette méthode de test permet d'émuler plusieurs dizaines d'années de fonc-

tionnement dans l'espace en quelques heures dans des infrastructures d'irradiation, comme des accélérateurs à particules. Comme décrit dans le standard JESD89B [49] de JEDEC, à l'environnement dans lequel le système est conçu pour évoluer correspond un certain type de particule avec une certaine énergie. La fluence, imposée de manière accélérée, est équivalente à une durée de fonctionnement en situation.

La section efficace $\sigma[cm^2]$ s'exprime en fonction de la fluence de particules $\Phi[cm^{-2}]$ et du nombre d'événements N_{evt} durant le temps d'exposition $T_{beam}[s]$, suivant la relation 2.1. La fluence est l'intégrale du flux de particule $\phi(t)[cm^2.s^{-1}]$ durant l'exposition, 2.2.

$$\sigma = \frac{N_{evt}}{\Phi} \tag{2.1}$$

$$\Phi = \int^{T_{beam}} \phi(t).dt \tag{2.2}$$

La section efficace est une caractéristique expérimentale d'un design. Elle est propre au type de particule et dépend du transfert linéique d'énergie (LET). En pratique, la section efficace est mesurée sur un circuit à son profil de mission (fréquence de fonctionnement et courant d'alimentation). Par extension, connaissant le flux moyen de particule dans lequel le circuit est prévu pour évoluer, le SER peut être calculer. Les circuits concernés par ce type de tests sont principalement des blocs spécifiques, tels qu'une mémoire volatile (SRAM), une PLL, des IPs dédiées basées sur une méthode BIST telle que CREST, ou encore des processeurs.

De nombreux articles dans la littérature [16, 34, 50, 61, 63, 101] proposent de qualifier des cellules logiques séquentielles sous radiations. Une fois la section efficace d'un type de cellule déterminée, il est possible d'estimer le taux d'erreur logiciel d'un circuit constitué de ce type de cellule. C'est cette approche que nous adressons dans ce chapitre. D'autres grandeurs servent à qualifier la fiabilité d'un système, notamment le MTBF (Temps moyen entre deux défaillances) 2.3 et le FIT (Taux de défaillance - nombre d'erreurs par milliard d'heures de fonctionnement) 2.4.

$$MTBF = \frac{1}{SER} \tag{2.3}$$

$$FIT = \frac{10e^9}{MTBF} \tag{2.4}$$

2.2 Indentification de SEEs dans un registre à décalage

Dans cette section, la technique de qualification de cellules logiques séquentielles utilisée dans ces travaux est détaillée, elle se base sur la méthode CREST [63]. Le but est de récolter un nombre important de fautes lors des expérimentations afin d'obtenir des métriques fiables de taux d'erreur et de section efficace.

2.2.1 Architecture dédiée à la qualification

La qualification de cellules logiques séquentielles se base sur un registre à décalage, une structure dédiée très utilisée dans la littérature. La profondeur choisie est très conséquente et n'a d'autre utilité que de mettre en œuvre un grand nombre de cellules. Ces travaux sont hérités de ceux des articles [34, 61] et sont effectués dans un contexte industriel. L'intérêt de cette approche est qu'un nombre conséquent de cellules peut être surveillé, avec un minimum de logique de contrôle. Un motif connu est propagé le long de la chaîne de registre et vérifié en sortie. Ainsi, les inversions binaires traduisant l'apparition de SEUs ou de MCUs, sont constatées en sortie de la chaîne. Cette approche ne permet pas de connaître l'instant d'apparition d'un événement. Les fautes sont constatées en sorti avec un retard borné, dont le pire cas est le nombre de cellules dans la chaîne divisé par la fréquence. La section suivante présente en détails l'architecture conçue dans ce cadre.

2.2.2 Harnais de test embarqué

Dans le Chapitre 1, nous avons pu constater les limites de fréquence de fonctionnement imposées par les entrées/sorties du composants. La technique de BIST utilisée dans ces travaux est donc basée sur la méthode CREST [63]. Chaque registre à décalage fait donc partie d'IP autonome, possédant un générateur de données pour alimenter la chaîne et un bloc de vérification des données en sortie, cette architecture est illustrée Figure 2.10.

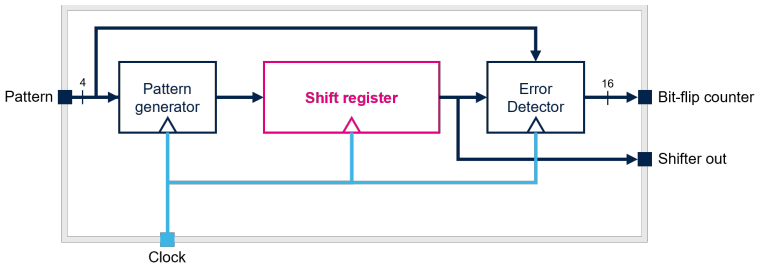


FIGURE 2.10 – Architecture du registre à décalage avec mécanisme d'autotest

Cette architecture est composée des éléments suivants :

Générateur de motifs :

Le générateur de motifs se basent sur un motif de quatre bits configurés en entrée du bloc. Ce motif est répété de manière cyclique dans le registre à décalage. Quatre bits suffisent à faire les motifs d'intérêt, Tableau 2.3, tels que proposés dans l'article [34].

Registre à décalage :

Le registre à décalage, Figure1.1, est constitué de cellules logiques séquentielles possédant les mêmes caractéristiques. Toutes les cellules sont pilotées de manières synchrones, ce qui impose beaucoup de contraintes sur l'arbre d'horloge comme décris dans la Section 2.2.5.

Détecteur d'erreurs :

Le bloc de détection d'erreur détecte les inversions binaires par comparaison de la référence avec les bits en sortie du registre à décalage. Un *watchdog* est instancié dans ce bloc. Il s'agit de compteur de front d'horloge qui permet, lors de la mise en route de l'IP, de désactiver la détection d'erreur le temps de la propagation du motif le long de la chaîne avant de procéder à la comparaison. Lorsqu'une inversion est détectée un compteur est incrémenté.

2.2.3 Implémentation dans les circuits de tests

Plusieurs IPs sont implémentées par circuits de tests. Les registres à décalage peuvent être opérés simultanément et implémentent différents types de cellules logiques séquentielles. Dans SERVAL, 6 types de cellules différents sont implémentés, le Tableau 2.1 récapitule les instances et le type de cellule. Au total, 24 IPs sont implémentées, ce qui représente 1,2 millions de cellules. Dans SQUAL, 4 types de cellules différents sont implémentés, comme le stipule le Tableau 2.2. En tout, 14 IPs sont implémentées, ce qui représente 180 mille cellules.

IP	**Propriétés des cellules**	**Surface du registre**	**Taille du registre**	**Nb. d'instances de l'IP**
SELFF-9T	Cellules de référence	1.23 mm^2	50 kb	4
SELFF-9T-ALP	Cellules DICE	1.14 mm^2	50 kb	4
SELFF-9T-CT		1.17 mm^2	50 kb	4
SELFF-9T-L	Cellules TMR	1.23 mm^2	50 kb	4
SELFF-9T-SC		1.26 mm^2	50 kb	4
SELFF-9T-SS		1.27 mm^2	50 kb	4
SELFF-9T-CG	Clock gates	0.946 mm^2	50 kb	4

TABLE 2.1 – Liste des IPs de qualification de cellules logiques séquentielles implémentées dans SERVAL

IP	**Propriétés des cellules**	**Surface du registre**	**Taille du registre**	**Nb. d'instances de l'IP**
SELFF-8T	Cellules de référence	0.305 mm^2	20 kb	4
SELFF-8T-GEOP2	Cellules DICE	0.633 mm^2	20 kb	4
SELFF-DPHD	Bascules RS	0.753 mm^2	5 kb	2
SELFF-QUINT		0.678 mm^2	5 kb	2

TABLE 2.2 – Liste des IPs de qualification de cellules logiques séquentielles implémentées dans SQUAL

Les Figures 2.11 et 2.12 illustrent l'implémentation physique des circuits de test SERVAL et SQUAL. Cette vue est obtenue depuis l'exploitation du fichier décrivant l'implémentation physique du circuit en mettant en surbrillance les couches actives en vert et les contacts avec les couches de métaux supérieures en jaunes. Cette représentation met en évidence les différents blocs afin de pouvoir les identifier. Les IPs d'intérêts sont encadrées en rouge et labellisées. Le format de fichier standard dans l'industrie du semiconducteur pour la description physique des circuits est le *Graphic Data System* (GDS), directement utilisé lors du contrôle de la géométrie des masques pour la lithographie.

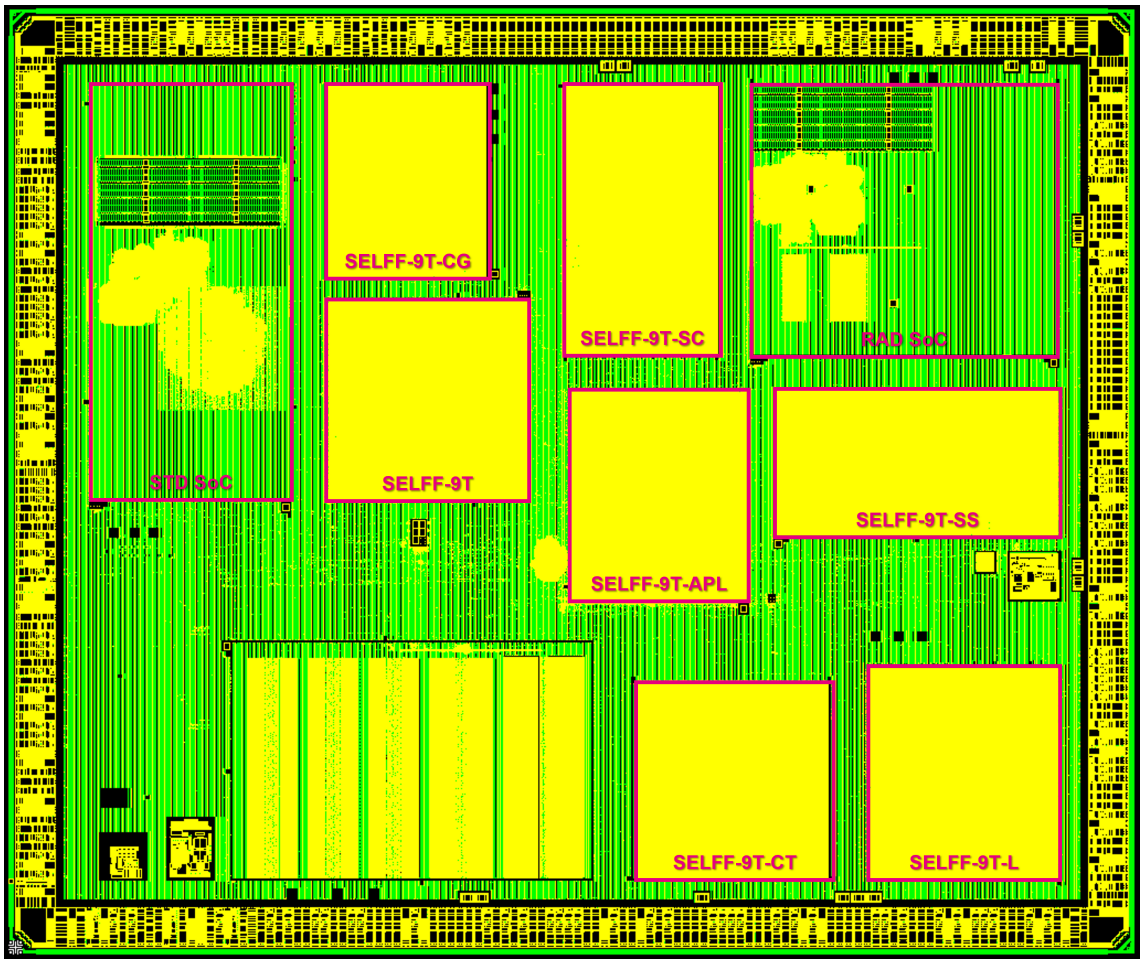


FIGURE 2.11 – Vue GDS du circuit SERVAL - couches actives en vert et contacts métaux en jaune

2.2.4 Impact des différents types de pattern

Le Tableau 2.3 récapitule les différents types de pattern qui pourront être utilisés lors des qualifications

Motif	Appellation
'0000'	ALL0 or Solid0
'1111'	ALL1 or Solid1
'0101'	Checkerboard or Patt1
'0011'	Patt2

TABLE 2.3 – Motifs utilisés lors des tests du registre à décalage [34]

Les motifs ALL0 et ALL1 sont les motifs les plus courant comme vu dans l'état de l'art. Ces motifs ne nécessitent pas de générateur de motifs et rendent le registre à décalage insensible aux SETs dans l'arbre d'horloge. Les motifs '0101' et '0011' possèdent des changements d'état, tout événement dans l'arbre d'horloge pourrait donc désynchroniser les données dans le registre et générer un nombre conséquent d'erreurs en sortie, qui ne représente plus le nombre de SEEs dans le registre. Comme décrit dans l'article [34], le motifs '0011' permet de couvrir tous les états possibles sur chaque cellule logique séquen-

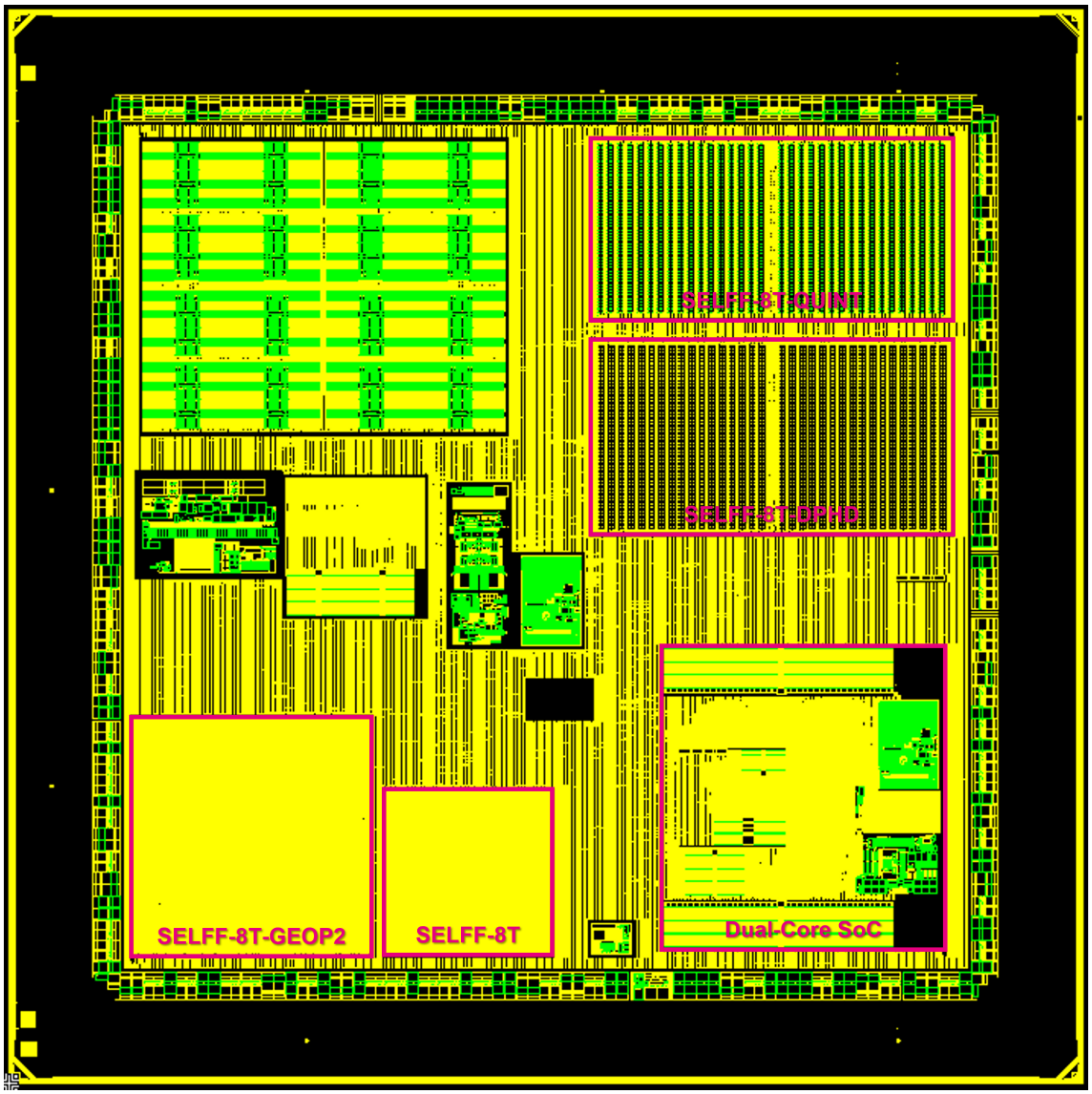


FIGURE 2.12 – Vue GDS du circuit SQUAL - couches actives en vert et contacts métaux en jaune

tielle, listés dans le Tableau 2.4, et d'obtenir le SER le plus élevé. Ce motif rend également sensible le registre au événements transitoires dans l'arbre d'horloge, ce qui permet donc de tirer parti des qualifications des cellules logiques séquentielles pour aussi évaluer la section efficace des buffers d'horloge.

Flux de données	Bit stocké	Bit suivant
'001100110011'	'0'	'1'
'001100110011'	'0'	'0'
'001100110011'	'1'	'0'
'001100110011'	'1'	'1'

TABLE 2.4 – Changement d'état des cellules logiques séquentielles pour le motifs '0011'

2.2.5 Structure du registre à décalage et arbre d'horloge

La Figure 2.13 illustre la structure du registre à décalage exploité dans ces travaux et met en lumière l'impact des SEEs selon leurs provenances dans le design. Le deuxième élément à considérer dans cette structure est l'arbre d'horloge, car les chaînes implémentées dans les composants de tests contiennent 20 k cellules pour SQUAL à 50 k cellules pour SERVAL. Nous avons fait le choix de concevoir l'arbre d'horloge de la même manière qu'il l'aurait été dans un produit. Ce dernier est généré automatiquement par les outils de placement routage selon la fréquence spécifiée, et en fonction des caractéristiques des cellules. Cet arbre est composé uniquement de buffers d'horloge insérés par l'outil afin de répartir les temps de propagation et de synchroniser les cellules. Il en résulte donc des structures d'arbres très hétérogènes, pouvant aller jusqu'à plus de 10 niveaux. Les buffers d'horloge, bien que leur nombre ne soit pas significatif face à celui des cellules logiques séquentielles dans la chaîne, représente une source d'erreur pouvant affecter les mesures, telle que décrite dans l'état de l'art.

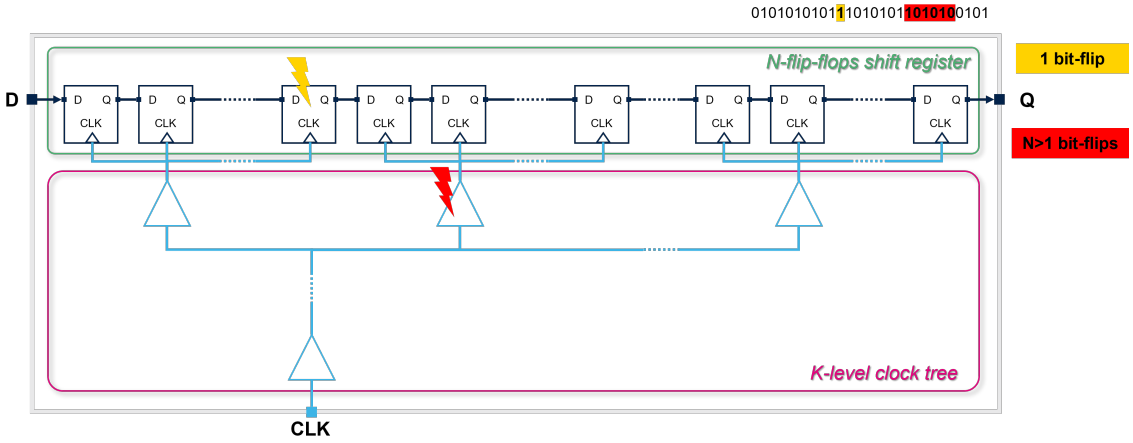


FIGURE 2.13 – Structure du registre à décalage avec les impacts des SEUs (en jaune) et des SETs (en rouge) sur le compte des inversions binaires en sortie

2.2.6 Limites de la mesure

Comme nous avons pu le constater dans la littérature, les SETs pouvant survenir dans l'arbre d'horloge contraignent le déroulement des qualifications. L'évolution du compteur d'erreur peut être très chaotique, par conséquent l'opérateur des tests n'a plus accès en direct au nombre d'événements récoltés. En outre, le besoin de devoir exposer une grande quantité de cellules logiques sous faisceau (de 20 k à 50 k cellules), cumulé à l'utilisation d'un arbre d'horloge conçu de manière classique, rendent les conditions très favorables à l'apparition de SETs. Les profils de missions imposent des tests à haute fréquence, il est donc impossible d'exporter directement sur les entrées/sorties du composant les données brutes en sortie du registre, ou encore d'échantillonner à une fréquence suffisante la valeur du compteur d'inversion binaire (*bit-flip*). Il faut pouvoir trouver un moyen d'effectuer une classification par type d'événement en ligne embarquée dans l'IP.

La méthode classique utilisée dans le cadre industriel consiste à échantillonner la valeur brute du nombre d'inversions binaires, et de ne tracer que les incréments. La première limitation concerne la différence entre les fréquences de fonctionnement du banc de test

FPGA et du DUT. Le profil de mission du DUT peut imposer des fréquences allant jusqu'au gigahertz, tandis que celle du FPGA est maintenu à 10 MHz pour garantir un fonctionnement fiable des entrées/sorties. La granularité de l'observation du compteur d'inversion binaire est donc dépendante de la fréquence de fonctionnement du DUT. La deuxième limitation imposée par le banc de test est due à la bande passante entre celui-ci et la machine hôte depuis laquelle l'opérateur récupère les événements. En cas d'événements rapprochés dans le temps, la congestion du canal de transmission limite la fréquence d'exportation des événements, ce qui limite d'autant plus la fréquence d'échantillonnage du compteur d'erreur.

Une classification embarquée et en ligne des événements basée sur l'analyse de l'évolution du compteur d'inversions binaires peut s'effectuer au profil de mission. Dans ce cas, elle donne accès directement au nombre d'événements détectés, ce qui permet un suivi en temps-réel plus fin des expérimentations par l'opérateur, et simplifie l'analyse des données récoltées durant la campagne.

2.3 Classification statistique des SEUs et SETs

C'est dans le cadre de cette IP dédiée à la qualification de cellules logiques séquentielles et à celles des buffers de leur arbre d'horloge que nous allons proposer une nouvelle technique de classification en ligne des événements. Cette technique s'appuie sur les conditions de tests et s'adaptent aux caractéristiques de la technologie et du design du registre à décalage.

2.3.1 Hypothèses

Afin d'identifier les caractéristiques discriminantes des événements d'intérêts, deux hypothèses sont faites grâce aux conditions de test ainsi qu'à la technologie.

Types d'événements attendus

La technologie 28 nm UTBB FDSOI utilisée pour fabriquer les deux circuits de test, à savoir SERVAL et SQUAL, est durcie par conception. Sa constitution interne la rend moins sensible au SEEs. L'isolation entre le *body* des transistors MOS et le substrat diminue la collecte et la diffusion des charges dues à l'impact d'une particule. En plus d'être moins sensible aux SBUs, cette technologie est immunisée contre les MCUs. Les propriétés de cette technologie font qu'une particule ne peut affecter qu'un seul transistor MOS lors d'un impact. On peut donc faire l'hypothèse qu'un SEE lors des tests se traduit soit par un SEU dans une cellule logique séquentielle, soit par un SET dans un buffer d'horloge. Deux classes se dégagent alors dans lesquelles doivent être comptés les événements.

Temps minimum en deux événements

Le paramètres régissant les conditions de test sous faisceau sont choisies pour imposer un certain taux d'erreur au circuit. Si ce taux d'erreur est trop élevé, il devient difficile d'identifier et de caractériser les événements. A l'inverse, si celui-ci est trop faible, il n'est pas possible de cumuler suffisamment de fluence en un temps limité. Ce faisant la période minimum entre deux événements est très faible devant la fréquence de fonctionnement. Le SER est de l'ordre du Hertz pour les cellules de référence, alors que la fréquence de fonctionnement est de l'ordre de la centaine de Hertz. La deuxième hypothèse est que lorsqu'un événement survient dans l'IP, le registre à décalage est entièrement rafraîchi avant l'événement suivant, en effet la valeur du SER est négligeable devant la fréquence de rafraîchissement 2.5.

$$F_{flush} = \frac{F}{N_{flipflops}} = \frac{100MHz}{50k} \approx 1kHz << SER \tag{2.5}$$

2.3.2 Méthode de classification

La méthode de classification proposée se base sur les deux hypothèses évoquées précédemment. Lorsqu'un événement survient dans le registre à décalage, il ne peut s'agir que d'un SEU dans une cellule logique séquentielle, ou d'un SET dans l'arbre d'horloge. De plus, deux événements sont séparés au minimum d'un rafraîchissement complet du registre. Lors de la conception de l'arbre d'horloge, les cellules logiques séquentielles sont regroupées au minimum par deux par buffer d'horloge. Ce qui signifie qu'un SET dans l'arbre qui génèrerait un masquage de front d'horloge ou un front supplémentaire, provoquerait au minimum deux inversions binaires détectées en sortie.

La technique de classification développée se base donc sur la statistique d'apparition des événements. Elle a été implémentée dans l'IP sous forme d'un module qui surveille l'évolution du compteur d'inversion binaire. Lorsque le compteur est incrémenté, un décompteur est lancé à la valeur initiale du nombre de cellules, $N_{flipflops}$, dans le registre et qui décompte sur chaque coup d'horloge. Si une autre erreur est détectée dans les $N_{flipflops}$ coups d'horloge suivants, l'événement est considéré comme SET, sinon comme SEU. Le schéma de la Figure 2.14 détaille l'architecture basée sur le principe évoqué.

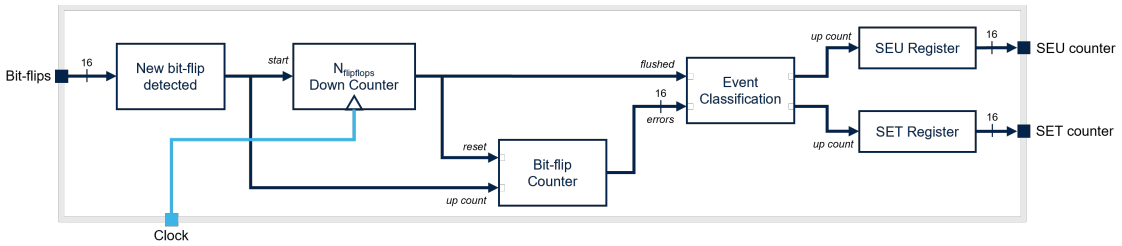


FIGURE 2.14 – Architecture du module de classification en ligne d'événements SEUs et SETs

2.3.3 Architecture finale et implémentation

La Figure 2.15 illustre l'architecture finale de l'IP de qualification de cellules logiques séquentielles. On retrouve le module de classification des événements en fin de chaîne. Ce module effectue donc la classification en ligne, au profil de mission. Cette opération est faite dans le but d'améliorer l'estimation de la section efficace à la fois des cellules logiques séquentielles et des buffers d'horloge. Ce module permet d'avoir accès directement aux nombres et au type d'événement survenus durant les tests ce qui est très pratique lors d'une campagne. L'ancienne méthode nécessitait d'analyser l'entièreté des fichiers de journalisation afin d'avoir le compte des événements, ce qui ne pouvait se faire qu'une fois les expérimentations terminées. L'accessibilité en direct de l'état d'avancement lors d'une campagne est aussi un atout permettant de surveiller le bon déroulement des opérations. L'opérateur peut éventuellement décider de continuer un test lorsque la fluence prévue est atteinte mais pas le nombre d'erreurs, ou de l'écourter si le nombre d'erreurs ciblé est atteint.

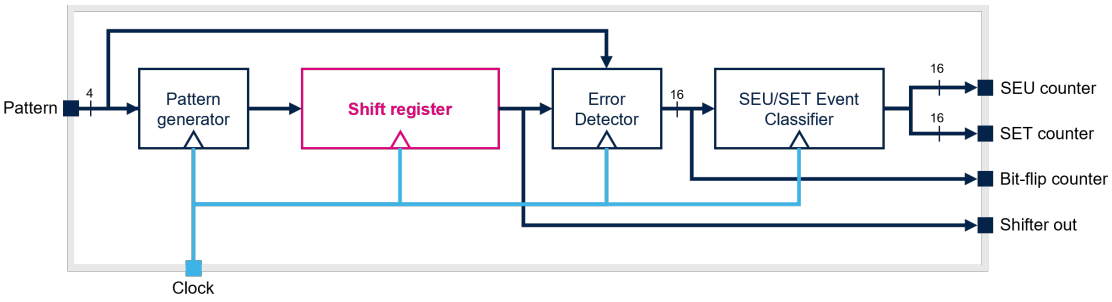


FIGURE 2.15 – Nouvelle architecture du registre à décalage avec mécanisme d'autotest et classification en ligne

Dans cette IP, tous les éléments faisant partie du harnais de test embarqué, à savoir le générateur de données, le détecteur d'erreurs et le classifieur d'événements, sont exposés aux radiations au même titre que les structures testées. Comme ils font partie de la chaîne de mesure, ils sont implémentés avec des cellules logiques séquentielles durcies. Ce type de cellules est composé de 3 cellules indépendantes et d'un voteur (TMR). Une écriture déclenche l'écriture dans les 3 cellules, et une lecture se fait depuis voteur majoritaire. Un bit erroné dans une cellule est alors masqué par les deux autres. Cette implémentation permet de garantir la fiabilité du harnais de test lors de l'exposition.

Les circuits de test SERVAL et SQUAL ont été irradiés respectivement sous neutrons et sous ions lourds lors de tests qualificatifs des technologies. Les résultats sont présentés dans la Section 5.1. Lors des tests sous faisceau, l'opérateur contrôle le banc de test depuis une machine hôte. Pour opérer cette IP, la procédure de démarrage du circuit et de test en continu est constituée des points suivants.

— **Horloge** : Configuration et démarrage de la PLL ;
— **Entrées/Sorties** : Configuration des multiplexeurs pour mapper les signaux de l'IP à adresser vers les IOs du composant.
— **Activation de l'IP** : Configuration des états des broches, sélection du motif de test, reset de l'IP, et lancement de la procédure BIST.
— **Surveillance de l'exécution** : Les compteurs d'erreur sont lus de manière périodique. L'échantillonnage des valeurs permet de suivre en temps réel l'évolution du nombre d'événements et de sauvegarder la progression au fur et à mesure de

l'expérience. Ce faisant, lors du post-traitement des données, la cohérence de la classification peut être vérifiée et comparée à l'ancienne méthode.

2.4 Conclusion

Pour conclure ce chapitre, nous avons proposé une méthode de classification en ligne des événements provenant d'une structure de test dédiée à la qualification de cellules logiques séquentielles dans le contexte industriel. Cette technique permet de différentier les événements selon l'élément logique affecté dans le registre à décalage depuis l'analyse de la signature des erreurs constatées en sortie. En suivant la méthode de test accéléré, ce type de qualification doit être réalisé dans des accélérateurs à particules selon le domaine d'application. Les limitations du banc de tests embarqué existant ne permettaient pas d'atteindre de hautes fréquences de fonctionnement pouvant être imposées par le profil de mission. En outre, le banc de test imposait un dépouillement des données récoltées lors de la mesure et ne permettait pas d'avoir accès au nombre d'événements.

Grâce à la classification en ligne proposée, le nombre d'événements est directement accessible et tient compte de l'élément logique affecté, soit les DFFs du registre à décalage, soit l'arbre d'horloge. Ceci représente un atout lors des qualifications sous faisceau de particules, car l'objectif est d'atteindre un compte d'erreur ou une certaine fluence pour établir le SER, et en déduire le FIT ou la section efficace des cellules. Cette technique a été implémentées dans deux circuits de test. Ces circuits ont été irradiés, les résultats sont présentés dans le Chapitre 5 et mettent en lumière les bénéfices de l'approche et les limites de la méthode.

Dès lors que le SER de cellules logiques séquentielles est évalué, il est possible d'estimer celui de systèmes conçus avec ces cellules. Afin de vérifier cette estimation et de valider les règles de conception des circuits, nous avons développé une plateforme SoC basée sur microprocesseurs dans l'objectif de réaliser des tests sous faisceau, tout en analysant les erreurs qui surviennent dans le processeur. Cette démarche fait l'objet du chapitre suivant.

Chapitre 3

Plateformes de test basées sur microprocesseurs et simulation avec injection de fautes

Ce chapitre détaille l'architecture et l'environnement logiciel des plateformes SoC, développées dans le but de qualifier le comportement du processeur RISC-V lors de tests radiatifs. Nous allons également nous concentrer sur l'architecture du processeur utilisé, en particulier sur les mécanismes de détection d'erreurs intégrés, ainsi que celle de l'infrastructure de débogage, le seul accès à la plateforme depuis l'extérieur.

La procédure de test spécifique aux systèmes processeurs est décrite, depuis l'opérabilité du circuit par le banc de test FPGA, jusqu'à la supervision de l'application depuis la machine hôte. L'application logicielle joue un rôle très important dans la collecte des fautes à travers l'architecture du processeur et fournit les premiers éléments permettant d'appréhender l'erreur. Elle a fait l'objet de nombreux développements pour améliorer la capture et l'identification des erreurs dans le flot d'exécution et assurer l'intégrité de la mémoire. L'application proposée est expliquée en détails ainsi que l'adaptation utilisée en simulation.

Afin d'étudier la propagation des erreurs, d'estimer les taux de couverture des modes de défaillance et de capture des erreurs, nous avons réalisé des campagnes de simulations avec injection de fautes. Le choix du modèle de simulation, et de la méthode d'injection de faute sont justifiés en s'appuyant sur les propriétés de l'implémentation physique. L'ensemble des outils de gestion des simulations, de l'extraction et de l'analyse des données est également présenté. Les données obtenues sont exploitées dans le Chapitre 4 pour le développement des techniques proposées pour aider à la compréhension des erreurs dans le circuit.

3.1 Présentation des plateformes de test

Deux plateformes de test ont été développées dans le cadre de ces travaux. Les deux plateformes sont des systèmes sur puces basées sur des processeurs RISC-V SCR1 développés par Syntacore et disponible en *open-source*. L'architecture des cœurs ainsi que celle des deux SoCs sont détaillées dans les prochaines sections.

3.1.1 Processeur 32-bits Syntacore SCR1

Le processeur SCR1 [88] est un microcontrôleur 32 bits basé sur le jeu d'instruction RISC-V [93]. Le pipeline comporte 4 étages, le mode machine est le seul mode d'exécution. Ce cœur développé par Syntacore est destiné à des applications embarquées de faible consommation. Les sources sont disponibles au format RTL en *open-source*. La configuration implémentée ne comporte pas de mémoire TCM, elle est composée de 2.827 cellules mémoires. Les modules qui composent ce processeur sont illustrés sur la Figure 3.1, leurs fonctions sont explicitées dans les points suivants.

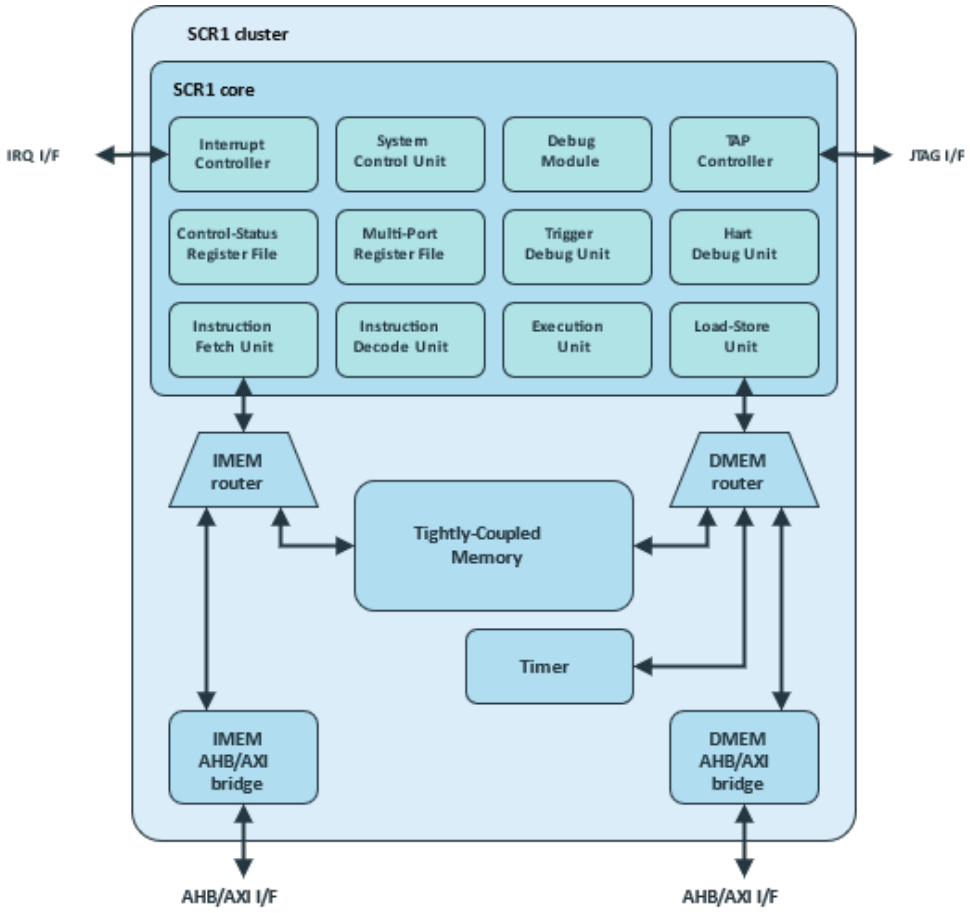


FIGURE 3.1 – Architecure du cœur RISC-V SCR1 de Syntacore [88]

Test-Access Port Controller (TAPC), Debug Module (DM), Trigger Debug Unit (TDU), Hart Debug Unit (HDU)

Cet ensemble de module supporte les fonctionnalités de débogage du cœur. Le module TAPC est le point d'entrée JTAG du cœur. Il est en charge de la chaîne JTAG interne, et dialogue avec deux IPs chaînées, le SCU et le DM. Le DM interprète les commandes JTAG et interagit avec HDU et le TDU. Le TDU détecte les *breakpoints* et événements définis dans le code pour déclencher des actions. Le HDU permet d'interagir avec le pipeline, dérouler l'exécution pas-à-pas, et consulter les registres du cœur.

System Control Unit (SCU)

Cette IP gère les commandes de reset des différentes parties du cœur. Elle peut être opérée depuis les entrées de reset externes et depuis la chaîne JTAG interne.

Integrated Programmable Interrupt Controller (IPIC)

Cette IP est un contrôleur d'interruption programmable. Il a été configuré pour supporter une ligne d'interruption externe. Cette ligne est connectée à une IP du SoC qui elle réunit d'autres lignes d'interruption.

Multi-Port Register File (MPRF)

Ce bloc gère le fichier de registres du processeur. Il possède 31 registres de 32 bits, ce qui représente 124 B. Il contient notamment le pointeur de pile et pointeur d'adresse de retour. La fonction de chaque registre est décrite dans le document RISC-V - *Instruction Set Manual* [93].

Instruction Fetch Unit (IFU)

Chargé de la requête de lecture en mémoire de l'instruction suivante, ce bloc transmet le couple instruction et adresse à l'IDU. A l'exception d'une instruction de branchement, l'instruction suivante est automatiquement lue et transmise. Les requêtes de lecture sont transmises au bloc IMEM. Ce bloc lève les exceptions *instruction access fault*, ou *instruction address misaligned* en cas d'adresse inaccessible en lecture ou non alignée sur 32 bits.

Instruction Decode Unit (IDU)

Les instructions sont décodées par ce bloc. En cas d'instruction illégale ou de *software breakpoint* une exception est générée à ce niveau. L'opérande et les éventuels arguments sont transmis au bloc EXU.

Execution Unit (EXU)

Ce module regroupe plusieurs blocs en charge de l'exécution des différents opérandes. Il contient deux unités arithmétiques et logiques (ALU). La première supporte tous les opérandes, elle est utilisée pour les instructions de calcul, le résultat est donné un à plusieurs cycles après la requête. La seconde ne supporte que les additions, elle est utilisée en particulier pour calculer les sauts de PC. En cas d'accès à la mémoire, TCM ou mémoire vive, les requêtes sont transmises au bloc LSU.

Load-Store Unit (LSU)

Ce bloc reçoit des requêtes de lecture/écriture du bloc EXU. Il lève les exceptions *Load/Store address misaligned* ou *Load/Store access fault* en cas d'adresse inaccessible en lecture ou non alignée sur 32 bits pour la lecture ou l'écriture de données en mémoire.

Instruction/Data Memory access router (IMEM/DMEM)

Ces blocs redirigent les accès lecture d'instructions (IMEM) ou lecture/écriture de données (DMEM) vers la TCM ou vers le bus système par l'intermédiaire d'un routeur. Ils se basent uniquement sur les adresses des requêtes.

Tightly-Coupled Memory (TCM)

Cette mémoire est optionnelle, elle est utilisée pour augmenter les performances du processeurs et fonctionne à la même vitesse. Elle est divisée en une partie pour les données et une pour les instructions. Il s'agit d'une mémoire mappée en interne, elle n'est accessible que depuis le processeur. Les principaux avantages sont que les accès ne peuvent pas être congestionnés comme ils pourraient l'être sur le bus selon la charge système, et se font à la vitesse du processeur. Cependant, toutes les données à placer dans cette mémoire nécessitent d'être copiées par le processeur. La TCM n'a pas été implémentée dans tous les SoCs, elle n'est pas non plus supportée par l'application, pour des raisons de temps de développement et de fiabilité. Elle n'est donc pas utilisée dans ces travaux.

Control-Status Register (CSR) file

Ce bloc réunit tous les registres d'état du processeur. Il contient notamment des registres d'identification du cœur (numéro de vendeur, identifiant, etc.) et des registres dédiés aux interruptions et exceptions, listés Tableau 3.1. Lorsqu'une interruption ou une exception survient, le processeur en mode machine, effectue un saut sur le vecteur d'exception *Machine Trap* qui renvoie l'exécution au *handler* d'interruption. La fonction à cette adresse commence par consulter le registre MCAUSE afin de connaître la raison de cet événement. En cas de *fault exception*, le registre MEPC contient l'adresse de l'instruction fautive. Le registre MTVAL contient des informations spécifiques à l'exception, comme une adresse ayant causée une faute d'accès ou encore une instruction illicite.

Status Register	Description
MSTATUS	Interrupt enabling state and privilege mode
MCAUSE	Exception code (related to Table 3.2)
MEPC	Virtual address of exception that encountered the error
MTVAL	Exception-specific information on fault exception

TABLE 3.1 – Registres d'état du SCR1 en lien avec les interruptions et exceptions

Liste des interruptions et exceptions

Il est important d'identifier la liste de toutes les interruptions et exceptions possibles. Lors des campagnes d'injection de fautes ou des expérimentations sous faisceau, des événements inattendus surviennent, et connaître leurs conséquences permet de les identifier. Les exceptions sont mises à profit par l'application dans le but de comprendre la cause d'une défaillance. Chacune d'entre elle appelle une fonction d'erreur, exceptés les interruptions légales. Voici la liste exhaustive des interruptions et exceptions du cœur SCR1, Tableau 3.2

0 : Exception 1 : Interrupt	Cause
0	Instruction address misaligned
0	Instruction access fault
0	Illegal instruction
0	Breakpoint
0	Load address misaligned
0	Load access fault
0	Store/AMO address misaligned
0	Store/AMO access fault
0	Ecall from M-mode
1	Machine software interrupt
1	Machine timer interrupt
1	Machine external interrupt

TABLE 3.2 – Code d'exception du processeur SCR1

3.1.2 Architecture des systèmes sur puce

Dans cette section sont présentées les architectures des deux plateformes implémentées sur SERVAL et SQUAL.

Plateforme SERVAL

La Figure 3.2 présente l'architecture des deux SoCs du vecteur de test SERVAL. Les deux instances de ce SoC, appelées STD (standard) et RAD (radiations hardened) diffèrent par l'implémentation du processeur dans le SoC RAD. Le cœur du RAD est partiellement implémenté avec des cellules TMR. L'identification des zones à durcir a été faite sur la base de campagne d'injection de fautes, le processus est détaillé par la suite.

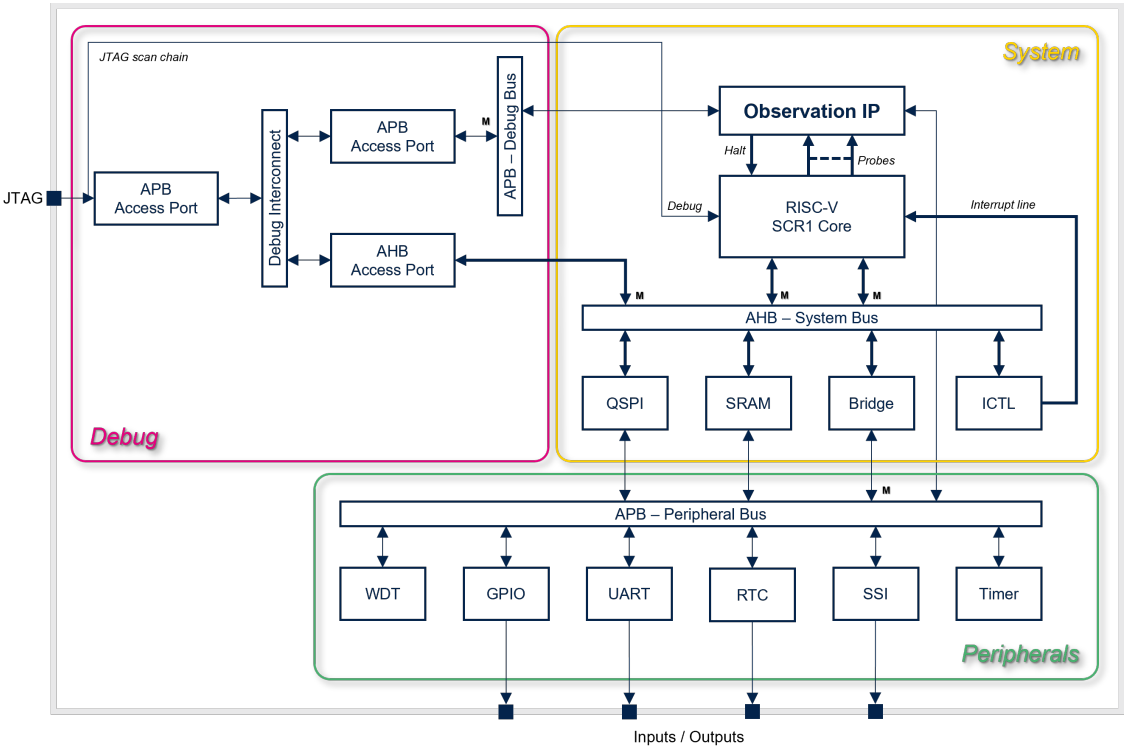


FIGURE 3.2 – Architecture du système sur puce de SERVAL

L'architecture du SoC de SERVAL, illustrée Figure 3.2, peut être divisée en trois parties : une partie système incluant le processeur le bus système et ses périphériques ; une partie périphérique basée sur le bus périphérique et toutes les IPs qui gravitent autour ; et une partie support de débogage depuis l'accès JTAG en passant par le bus de débogage et toutes les IPs associées.

Système

— **Processeur** : Le cœur SCR1 utilisé ne possède pas de TCM. Son architecture est de type Harvard, deux accès en tant que maître le relient donc au bus système pour les instructions et les données.

— **Mémoire** : Le programme peut être stocké dans la mémoire SRAM ou dans une mémoire flash externe accessible depuis le Quad Serial Peripheral Interface (QSPI). Lors des tests, seulement la mémoire SRAM est utilisée car la carte DUT n'a pas été dotée de mémoire flash. Cette mémoire est composée de 32,768 mots de 32 bits (soit 128 kB) protégés par 7 bits d'ECC. Le code correcteur d'erreur utilise l'encodage de Hamming étendu, cette configuration permet de corriger une erreur simple et de détecter une erreur double (SECDED - *Single Error Corrected Double Error Detected*). Au-delà de 2 erreurs par case mémoire, le résultat de l'ECC n'est plus garanti. Lorsqu'une valeur est écrite dans la SRAM, le syndrome est calculé et automatiquement écrit.

— **Contrôleur d'interruption** : Cette IP gère les interruptions externes de toutes les IPs du SoC. Si une interruption est levée, elle est transmise au processeur via l'unique ligne d'interruption externe. C'est donc au *handler* d'interruption de consulter le contrôleur pour connaître la raison de cette interruption.

— **Bus système et périphériques** : Le bus système est un bus haute performance

AHB dont les périphériques maîtres sont les deux accès du processeur et l'accès de débogage. Les requêtes sont arbitrées selon la méthode Round-Robin. Il est possible, depuis le bus système, d'accéder au bus périphérique via l'IP *Bridge*, qui est l'unique maître sur ce bus. La particularité de la configuration de ce bus est que les IPs ne peuvent pas le bloquer en cas de défaillance pour cause de requête erratique.

— **IP d'observation** : Permet de sauvegarder un historique de l'activité du processeur. Développée dans le cadre de nos travaux, la conception de cette IP est détaillée dans la Section 4.1. Elle est mappée sur le bus de débogage et sur le bus système, ce qui la rend accessible depuis l'accès JTAG et depuis le processeur. Cette IP contient notamment une banque de 16 registres mappés en mémoire et implémentés avec des cellules TMR, elle est dédiée à l'utilisateur afin d'exporter des données. Cette banque de registres, l'UDBR (*User Debug Register Bank*) reste donc accessible depuis l'extérieur même en cas de blocage système car le seul élément dont elle dépend est l'horloge. Elle a également la possibilité de transmettre un signal au pipeline du processeur pour geler l'exécution.

Si une erreur simple est détectée lors d'une lecture en mémoire, la donnée lue est automatiquement corrigée mais l'inversion binaire subsiste. Une intervention de l'application est nécessaire pour corriger cette inversion. La gestion matérielle de l'ECC déclenche alors exception et le *handler* associé écrase la valeur dans la case mémoire. Cette action du logiciel assure de pouvoir à nouveau corriger cette case mémoire dans le cas où une autre erreur simple surviendrait. Si deux erreurs sont détectées dans une case mémoire, la donnée n'est pas récupérable, et la gestion matérielle de l'ECC déclenche une autre exception qui appelle un *handler* configuré pour signaler l'erreur et arrêter l'exécution.

Périphériques

— **Watchdog (WDT)** : Permet de programmer le reset du SoC de manière périodique via un timer.
— **General Purpose Input/Output (GPIO)** : Permet de contrôler l'état des broches du circuit. Les GPIOs peuvent être utilisés comme port parallèle pour exporter des données au banc de test FPGA.
— **Universal Asynchronous Receiver Transmitter (UART)** : Lien série asynchrone
— **Real-Time Clock (RTC)** : Permet de générer des interruptions sur de longues bases de temps via une horloge basse fréquence.
— **Synchronous Serial Interface (SSI)** : Lien série synchrone.
— **Timer** : Permet de générer des interruptions sur des bases de temps courtes.

Support de débogage

Le support de débogage correspond à l'environnement de débogage, présenté dans la Section 1.2.1. Accessible depuis un port JTAG, il implémente deux fonctionnalités, l'accès au bus de débogage et l'accès au bus système. Le débogage du cœur est chaîné avec l'accès JTAG en entrée.

L'intérêt majeur du bus de débogage est son indépendance vis-à-vis du système pro-

cesseur, sauf pour ce qui concerne l'horloge. Cette indépendance lui confère la garantie de rester accessible peu importe l'état de l'application. En outre, l'accès débogage est prioritaire sur le bus système, ce qui permet donc de demander l'arrêt du processeur via l'IP d'observation, avant de poursuivre l'investigation du SoC. Ce moyen est en particulier utilisé pour charger le programme en mémoire avant de démarrer le processeur.

Plateforme SQUAL

La Figure 3.3 présente l'architecture du SoC implémenté dans le vecteur de test SQUAL. Ce SoC possède deux cœurs SCR1 dont un est implémenté suivant la même méthode de durcissement partiel (RAD), et l'autre n'utilise que des cellules standards (STD). On retrouve donc comme dans SERVAL deux processeurs ayant deux niveaux de durcissement. Lors des tests sous radiations, les résultats de l'observation des processeurs peuvent être comparés, si les différences mises en lumières correspondent à l'implémentation, il est possible de justifier de la cohérence de l'approche. Le logiciel de test n'exploite donc qu'un seul cœur à la fois, et place l'autre en attente d'interruption.

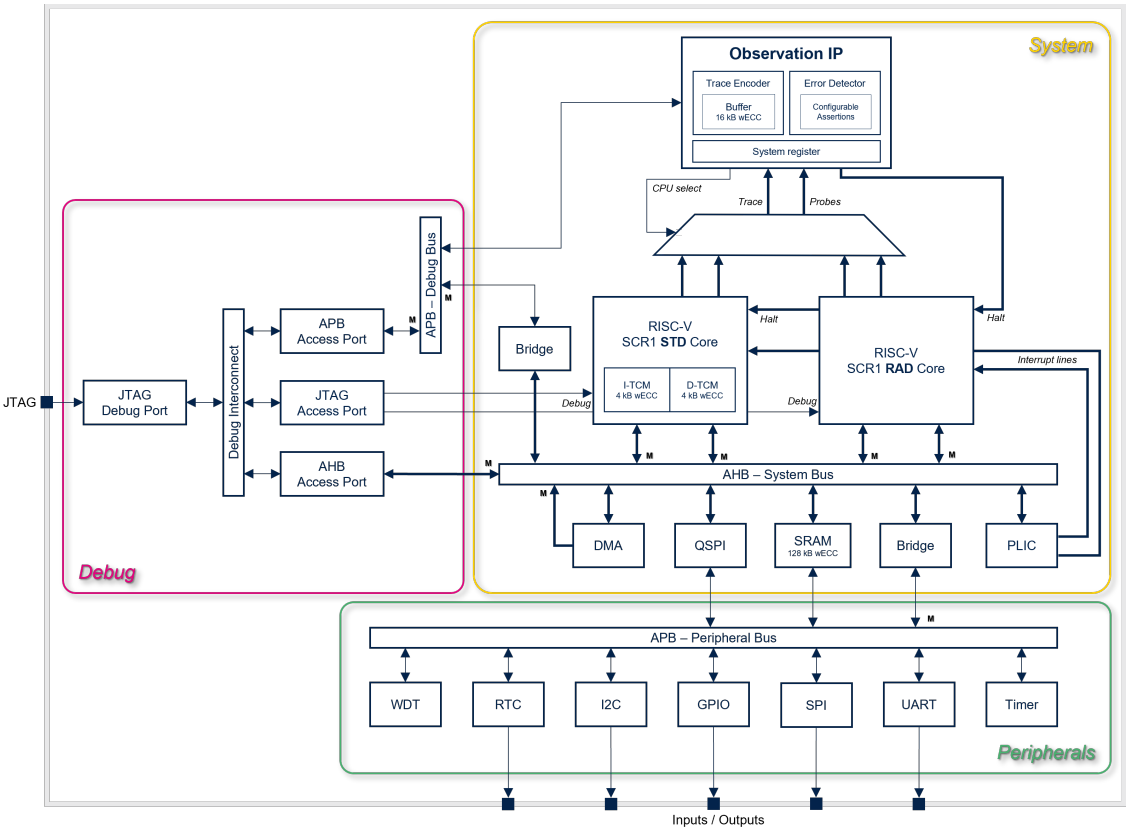


FIGURE 3.3 – Architecture du système sur puce de SQUAL

L'architecture du SoC de SQUAL hérite de celle de SERVAL, elle possède des améliorations notamment au niveau de la structure de débogage et de l'IP d'observation. Ce SoC est dorénavant basé sur deux processeurs prévus pour être testés de manière distincte dans le même esprit que le vecteur de test SERVAL, le but étant de comparer les deux implémentations. Sur la Figure 3.3, on peut voir les différentes améliorations et changements, qui sont détaillés dans les points suivants, selon les trois branches du SoC : système, périphériques et support de débogage.

Système

Les deux processeurs sont indépendants, il sont différentiés par des identifiants matériels accessibles en tant que registres. Le processeur STD possède une mémoire TCM de 8 kB protégée par ECC, divisée en deux zones de 4 kB, une pour les instructions et une pour les données. L'ECC est de type SECDED. Cette mémoire n'est pas utilisée dans le cadre de nos travaux. Le programme étant stocké dans la SRAM, le gain de performance n'est pas dimensionnant. En outre l'utilisation de cette mémoire impliquerait l'utilisation d'un logiciel plus complexe et nécessiterait des procédures de vérification de l'intégrité des données. Une IP de DMA (*Direct Memory Access*) a été ajoutée permettant d'automatiser les copies mémoires, ceci permettrait d'optimiser le temps de copies du programme de la mémoire flash à la mémoire SRAM. Un *Bridge* a été ajouté du bus système vers le bus de débogage afin de pouvoir accéder à l'IP d'observation depuis le processeur. Cette plateforme est dotée d'une nouvelle IP d'observation, dont la conception est détaillée dans la Section 4.2. Cette IP hérite de la précédente, elle conserve les mêmes principes de fonctionnement, l'UDRB, et la possibilité de stopper le processeur. Elle n'est dorénavant mappée que sur le bus de débogage.

Périphériques

L'environnement des périphériques a été conservé, seules certaines IPs ont été remplacées. Le SSI a été retiré, mais un SPI et I2C ont été ajoutés.

— **Serial Peripherial Interface (SPI)** : Permet de communiquer directement avec des circuits selon un schéma de maître-esclave, et d'adresses de registres mappées en mémoire.

— **Inter-Integrated Circuit (I2C)** : Permet de communiquer avec des périphériques d'un système grâce à seulement deux lignes.

Support de débogage

Le support de débogage hérité de SERVAL a été amélioré. La chaîne JTAG comprenant l'architecture de débogage CoreSight et le débogage du cœur a été supprimée. De fait, la bande passante sur les accès bus depuis le JTAG est donc significativement supérieure. Les fonctionnalités de débogage des cœurs sont dorénavant assurées par l'IP *JTAG access port*, décrite en Section 1.2.1, permettant de générer des trames JTAG en interne, depuis des commandes envoyées par le port JTAG externe. L'accès depuis le processeur au bus de débogage n'impose plus de mapper les IPs à la fois sur ce bus et sur le bus périphérique, si ces dernières doivent être adressées depuis le processeur et le JTAG.

3.1.3 Banc de test basé sur FPGA

La Figure 2.3 représente la carte FPGA connectée à la carte fille supportant le DUT. L'architecture du banc de test implémentée dans le FPGA est aussi détaillée. La configuration représentée sur cette illustration est celle utilisée lors des tests sous radiations avec la carte fille déportée d'environ un mètre de la carte FPGA. Le banc de test supporte

le *Peek/Poke* JTAG et génère des trames depuis l'envoie de commandes de haut-niveau par la machine hôte via le lien série. Le *Peek/Poke* est une fonctionnalité qui fournit un accès atomique en lecture/écriture aux blocs mappés sur les différents bus, comme le bus système ou le bus de débogage. Cette fonctionnalité suffit pour charger un programme en mémoire, accéder à l'ensemble du SoC et récupérer toutes les informations de diagnostic. L'ensemble des IPs de l'environnement de débogage du CoreSight implémentées dans ce SoC possèdent leur propre signal de reset. Il est donc possible en cas de défaillance de restaurer un état suffisamment stable du système de débogage sans pour autant écraser les données de diagnostic.

Pour démarrer le SoC, la procédure suivante est appliquée :
— **Horloge** : Configuration et démarrage de la PLL
— **Entrées/Sorties** : Configuration des IOs du circuit pour y connecter le SoC à tester.
— **Activation du SoC** : Configuration des états des broches et reset du SoC.
— **Chargement en mémoire** : Mise en pause de l'exécution via *Poke* JTAG dans le registre de contrôle de l'IP, puis chargement du programme. La mémoire SRAM étant dans un état inconnu après avoir coupé l'alimentation du circuit, il est préférable de geler l'exécution avant le chargement pour éviter tout comportement erratique du processeur qui pourrait conduire à une congestion du bus ou à la corruption du programme en cours de chargement.
— **Effacement de l'UDRB** : Remise à zéro de toutes les cases mémoire de l'UDRB pour lever toute ambiguïté quant à l'apparition de nouvelles données
— **Démarrage du programme** : Reset du SoC, qui se faisant dégèle le pipeline et lance l'exécution.
— **Surveillance de l'exécution** : Des registres d'état dans l'UDRB sont lus de manière périodique depuis le JTAG de sorte à s'assurer du fonctionnement correct de l'application. Ces registres ont des valeurs qui évoluent de manière cohérente, on utilise pour cela un compteur du nombre d'itération d'un algorithme de test afin d'être sûr de la pertinence des valeurs lues. Ces registres lus en continu donnent également accès au nombre d'erreurs mémoire corrigées par l'application, un indicateur tangible de l'hostilité de l'environnement.

3.1.4 Procédure de test sous faisceau

Le banc de test durant les expérimentations sous faisceau des SoCs est le même que celui présenté dans la Section 2.1.2. Il embarque dorénavant le support des fonctionnalités nécessaires à adresser le SoC, en particulier le support JTAG, comme illustré sur la Figure 2.3.

La procédure au lancement des tests est la même que précédemment. On commence par s'assurer du fonctionnement stable de l'application durant quelques minutes, puis on démarre l'irradiation tout en notant l'heure d'ouverture du faisceau. Durant l'expérience, l'application est surveillée via la lecture en continu des registres d'état depuis l'accès JTAG dans l'UDRB. En cas de défaillance détectée, les informations de diagnostic sont récupérées, puis la procédure de démarrage du SoC, est de nouveau appliquée. De cette manière, on répète l'expérience en s'assurant qu'aucune erreur résiduelle de subsiste. Les dispositifs logiciels d'identification des erreurs sont détaillés dans la Section 3.2.

Les actions de surveillance du banc de test contrôlé depuis la machine hôte via le lien série ont été automatisées afin de gagner en réactivité et en rigueur, les tests pouvant durer plusieurs heures. Un script en langage Python [22] supervise l'expérience. Il répète la procédure de démarrage du SoC, puis en se basant sur la lecture d'un compteur d'itérations de l'algorithme de test (décrit dans la Section 3.1.3), le script est en mesure de détecter un code d'erreur ou un blocage de l'application. Suite à un événement, un fichier de journalisation daté est créé incluant toutes les données de diagnostic, avant de relancer l'expérience. Le fait de gagner en réactivité sur la phase de récupération des données de diagnostic après qu'un événement se soit produit est crucial, et permet de s'assurer de leur intégrité. En effet, durant les expérimentations sous faisceau, il est impossible de couper le flux de particules entre chaque événement, cela permet donc de limiter au maximum leurs temps d'exposition. Cela contribue également à réduire le temps mort durant lequel le circuit exposé n'est pas observable.

3.2 Environnement logiciel

L'environnement logiciel a été développé dans le but d'étudier le comportement du processeur sous radiations. Il est composé d'une procédure d'initialisation, d'un algorithme de *benchmark*, d'une fonction de scrutation mémoire et d'un ensemble de fonctions associées à des cas d'erreurs spécifiques. Toutes les exceptions sont prises en charge et correspondent soit à un code d'erreur pour les *fault exception*, soit à la détection d'une erreur par ECC pour assurer la correction. Le code est conçu pour vérifier en ligne l'intégrité des résultats de calcul, du contenu de la mémoire et de la configuration de l'IP d'observation. Dans ces travaux, l'IP à tester est le processeur uniquement, il faut donc garantir qu'aucune erreur ne puisse venir de la mémoire. En outre, il faut s'assurer que toutes les fonctionnalités de détection d'erreur et de sauvegarde de la trace d'exécution restent opérationnelles. Une version du code a été adaptée pour les simulations en campagne d'injection de fautes.

3.2.1 Application et couverture matérielle

Le but de l'algorithme de *benchmark* est de mettre en œuvre toutes les fonctions du processeur, via l'utilisation de tous les types d'instruction, de l'ensemble du fichier de registres, etc. De fait, des données sont propagées dans l'ensemble du circuit, permettent de collecter des fautes, et de calculer un SER représentatif. Si l'application ne permet pas d'activer toutes zones du processeur, le SER serait artificiellement bas durant les qualifications.

Une contrainte vient s'ajouter au choix de l'algorithme de *benchmark*, qui est le temps d'exécution de la simulation du SoC. En effet, en vue de pouvoir réaliser des campagnes d'injection de fautes composées de centaines de milliers de simulations, il est crucial de maîtriser ce paramètre.

Les algorithmes CoreMark [21] et Fast Fourier Transform (FFT) ont été choisis pour les qualifications respectives de SERVAL et de SQUAL. Le CoreMark est une référence pour la mesure des performances des processeur car il teste toutes les capacités du processeur. L'algorithme de calcul de la transformée de Fourier est souvent utilisée dans des applications de traitement de signaux analogique. Il est composé de multiplications et ac-

cumulations, tout comme le CoreMark effectuant des produits matriciels. Ces opérations sont très employées dans les applications, en particulier celles basées sur des réseaux de neurones. Le circuit SQUAL possédant deux cœurs est plus lourd à simuler, c'est pourquoi un l'algorithme de FFT, plus léger que le CoreMark, a été choisi. Les deux algorithmes effectuent des calculs sur des données embarquées dans le programme.

Algorithme	Nombre de cycles	Taux de couverture	Taux de réussite
Coremark	571312	66.7%	85.3%
Fast Fourier Transform	107468	67.2%	87.7%

TABLE 3.3 – Algorithmes pour le *benchmark*

Les taux de réussite des algorithmes sont extraits de campagnes d'injection de fautes, cette technique est détaillée dans la Section 3.3. Cette métrique permet d'estimer le taux de masquage logiciel d'une application. Les deux algorithmes possèdent des taux de couverture et de réussite quasiment équivalents, voir Tableau 3.3. L'algorithme FFT, nécessite environ 5 fois moins de cycles, mais reste aussi efficace dans le contexte de *benchmark* pour les qualifications.

Les données servant à alimenter les algorithmes ne varient pas, les résultats attendus restent donc inchangés d'une exécution à une autre. Un contrôle de redondance cyclique (CRC) est utilisé pour détecter des erreurs dans le résultat de chaque algorithme. Le CRC de référence est stocké dans la mémoire programme. Après chaque exécution de l'algorithme le résultat est donc vérifié.

3.2.2 Instruction de capture d'erreur

Afin d'améliorer la détection des erreurs sans pour autant modifier le flot d'exécution, des instructions pièges [65] (ECI - *Error Capturing Instruction*) ont été ajoutées dans le code. Ces dernières se matérialisent par des instructions de *breakpoint* logicielles, (*eBreak*) pour les processeurs RISC-V. Ces instructions permettent de déclencher une exception sans avoir besoin d'argument. Elles sont donc invariantes par translation dans le code, ce qui permet de les copier manuellement dans l'image mémoire après la compilation et même pendant l'exécution.

Les ECIs ont été utilisées sous deux formes. La première consiste à ajouter des zones mortes, ou (*deadcode*), dans le programme qui ne sont pas censées être parcourues par l'exécution. Cette méthode est dérivée des techniques d'obscurcissement de code. Elles se matérialisent par une série d'instructions précédée par un saut après la dernière instruction du *basic bloc*. Ce *basic bloc* ne contient que des instructions piège dans notre utilisation. Ces macros de *deadcode* sont ajoutées explicitement dans le code avant la compilation. La seconde forme d'utilisation des ECIs consiste à copier ce type d'instruction dans les zones mémoires inutilisées au démarrage du programme. Ceci permet d'initialiser une valeur correcte dans les cases mémoires et de couvrir l'ensemble de la mémoire. Seule la pile est amenée à recouvrir les ECIs au fur et à mesure de l'exécution.

Le fait d'insérer ce type d'instruction dans la mémoire du programme augmente le taux de capture des sauts corrompus. Grâce au fait que toutes les cases soient initialisées, cela permet aussi de pouvoir lire l'entièreté de la mémoire sans que d'erreur ECC ne soit

détectée. On peut donc bénéficier de l'intégralité de la mémoire pour évaluer l'hostilité de l'environnement.

3.2.3 Procédure logicielle

La procédure logicielle consiste en une phase d'initialisation au démarrage, puis l'exécution en boucle de l'algorithme tout en vérifiant qu'aucune erreur n'ait été détectée. Elle est implémentée en *bare-metal*, car un système d'exploitation induit une charge supplémentaire qui décuple le temps de simulation. L'image du programme est directement chargée dans la SRAM du SoC lors des expérimentations. Cette mémoire est séparée en une partie programme en lecture uniquement et une partie données incluant la pile. Le tas n'a pas été implémenté, aucune allocation dynamique n'est donc possible. Afin d'optimiser le temps de chargement du programme dans le SoC via JTAG, seulement le programme et les données initialisées à des valeurs non nulles sont chargés.

Initialisation

Tout commence par le démarrage du SoC, le processeur démarre à une adresse située au début de la mémoire SRAM. Dans le cas du SoC multicœur de SQUAL, les deux processeurs commencent par lire leurs identifiants. Le processeur sélectionné à l'avance pour exécuter l'application continue l'exécution tandis que l'autre reçoit l'instruction d'attente d'interruption (WFI). Les opérations suivantes sont réalisées :

— **Initialisation des variables** : Les variables nulles sont initialisées à zéro.
— **Détection de reset** : Au chargement du programme, les registres dans l'UDRB sont initialisés à zéro. Lors du démarrage, le programme vérifie la valeur d'un de ces registres et l'incrémente, ce qui permet de détecter les reset inopportuns.
— **Copie des ECIs** : Les ECIs sont copiées dans les parties non allouées de la mémoire.
— **Interruptions d'erreurs ECC** : Le bloc mémoire est configuré pour générer des interruptions en cas d'erreur simple ou double. Les interruptions sont démasquées au niveau du contrôleur d'interruption et du processeur.
— **IP d'observation** : Les assertions matérielles sont configurées, la trace du processeur est enregistrée en continue dans le buffer circulaire, et les déclencheurs sont armés pour stopper l'enregistrement en cas d'erreur détectée.
— **CRC de référence** : Un CRC est calculé sur les registres de configuration non protégés par ECC, dans notre cas il s'agit en particulier de la configuration de l'IP d'observation, décrite en Section 4.2, pouvant aller jusqu'à 1KB de données. Une valeur de CRC est calculée au démarrage, puis utilisée comme référence.

Surveillance

Cette étape de la procédure est exécutée en boucle dans le cadre des expérimentations. La version adaptée à l'exécution en simulation n'effectue qu'une seule fois cette étape, en se passant des tâches de vérification mémoire qui ne sont pas réalisées à chaque itération de l'algorithme. Les opérations qui composent cette étape sont décrites ci-dessous :

— **Initialisation des assertions matérielle** : Pour l'IP d'observabilité décrite dans la Section 4.2, il est nécessaire d'initialiser des compteurs dédiés à des assertions surveillants l'exécution de l'algorithme.
— **Algorithme de *benchmark*** : L'algorithme de *benchmark* (CoreMark pour SERVAL et FFT pour SQUAL) est exécuté. Un CRC est calculé sur le résultat puis comparé à celui de référence.
— **Vérification des assertions** : Les états des assertions sont vérifiés pour relever la présence d'erreur, et constater si la trace est toujours en cours d'enregistrement.
— **Vérification de la mémoire** : Toutes les secondes d'exécution environ, la mémoire et les registres de configuration sont scrutés. Un CRC est calculé sur les registres sélectionnés pour être surveillés puis comparé avec la référence. La mémoire, possédant un syndrome ECC sur chacune de ses cellules, est lue pour faire ressortir les erreurs.

Durant cette étape de surveillance de l'application, si une erreur est détectée, la fonction d'erreur correspondante est appelée, et l'application est immédiatement stoppée en gelant l'exécution depuis l'IP d'observation. La gestion des erreurs est détaillée dans la Section 3.2.4.

3.2.4 Gestion logicielle des erreurs

Le logiciel est composé de fonctions pour chaque exception et cas d'erreur afin d'identifier la raison de la faute. Le Tableau 3.4 liste les cas d'erreur supporté. Ces cas d'erreur incluent ceux du Tableau 3.2, ainsi que les vérifications de la Section 3.2.3.

Code erreur	Abbreviation	Description
1	SDC	Erreur de calcul
2	BREAKPT	ECI exécutée
3	FAULT	Exception de faute levée par le processeur
4	INTER	Interruption machine inattendue
5	HSU_TRAP	Interruption hyperviseur/superviseur/utilisateur
6	EXIT_TRAP	Fonction *exit* appelée
7	PANIC_TRAP	Fonction *panic* appelée
8	RESET_TRAP	Reset inattendu
9	ECC_DE	Erreur double en mémoire - non récupérable
10	REG_CRC	Erreur dans les registres de configuration
11	TIMEOUT	Timeout détecté par le banc de test
-1	UNK	Raison inconnue - Nécessite une analyse de la trace

TABLE 3.4 – Codes d'erreur logicielle

Dans le cadre de tests en réel, les deux derniers cas sont identifiés depuis la machine hôte. Certains d'entre eux peuvent survenir à cause d'événements transitoires dans le circuit qui provoquent des reset locaux, des glitches sur les alimentations, etc. Ces événements entraînent des conséquences très particulières, impossibles à reproduire en simulation selon la méthode présentée dans la Section 3.3.

Concernant les erreurs en mémoire, si une erreur double est détectée par le gestionnaire ECC, il déclenche une interruption dont le *handler* exporte le code d'erreur correspondant et stoppe l'exécution. Si une erreur simple est détectée, la donnée peut être corrigée, donc

l'interruption n'est pas considérée comme due à une faute, et l'exécution se poursuit. Durant l'interruption liée à une erreur simple, la donnée est lue en appliquant la correction de l'ECC, puis réécrite dans la même case mémoire. De fait, l'inversion binaire est corrigée. Cette étape est nécessaire car le bloc mémoire ne gère pas la correction matérielle automatique.

Un ensemble de fonctions du logiciel permet d'associer les modes de défaillances à des codes d'erreur. Ces fonctions d'erreur effectuent un ensemble d'action dans le but de collecter un maximum d'information, signaler une défaillance, et stopper l'exécution.

— **S'assurer de l'arrêt de l'enregistrement de la trace**. Certains appels à fonction d'erreur ne se font pas depuis une exception (exemples : PANIC_TRAP, EXIT_TRAP, ou encore des sauts corrompus). Il est donc nécessaire de stopper l'enregistrement au plus tôt pour ne pas écraser les données de diagnostic pertinente stockées dans le buffer.
— **Désactiver le bloc de détection d'erreur** pour sauvegarder l'état des assertions.
— Sauvegarder la valeur des registres d'état d'intérêt du processeur dans l'UDRB, en particulier les registres *mstatus*, *mcause*, *mepc* et *mtval*.
— **Exporter le code d'erreur dans l'UDRB**, ainsi que toutes informations de diagnostic disponible (exemple : nombre de reset détectés pour RESET_TRAP et la données corrompue et son adresse dans le cas de ECC_DE).
— **Arrêter l'exécution depuis l'IP d'observation**. Le processeur est donc bloqué en attendant l'intervention externe de l'opérateur pour garantir qu'aucune donnée de diagnostic ne soit altérée par un comportement erratique.

Suite à une erreur, le code d'erreur est copié dans un registre de l'UDRB lu en continu depuis le banc de test par le script de supervision. Ce qui permet à la machine hôte de déclencher la procédure de récupération des données de diagnostic et de relancer l'expérience pour poursuivre le test selon la procédure décrite dans la Section 3.1.3.

3.2.5 Génération des images mémoires

Déclinaisons simulation et expérimentations

Deux déclinaisons de l'image du code sont générées. La première est utilisée en simulation lors des campagnes d'injection de fautes. Les particularités sont que la procédure de surveillance, présentée Section 3.2.3, n'est effectuée qu'une seule fois, et la vérification périodique de la mémoire est volontairement omise. La deuxième image est dédiée aux tests sur carte, la procédure de surveillance est donc exécutée en boucle tant qu'aucune erreur n'est détectée. Lors d'une exposition sous faisceau, l'immense majorité du temps d'exécution est donc consacrée à l'algorithme de *benchmark*.

Paramétrage

Une contrainte importante de similarité entre les deux versions du codes subsiste. En effet, les cas d'erreur qui composent la base de données générée par la campagne d'injection

de fautes, présentée Section 3.3, doivent être alignées avec celles récoltées lors de tests réels, afin de pouvoir appliquer la méthode d'analyse par apprentissage automatique présentée en Section 4.3. Les IPs d'observation du processeurs, présentées Section 4.1 et 4.2, se basent sur l'enregistrement de la trace d'exécution, les images mémoires doivent donc correspondre d'une version de code à l'autre. De plus, toute modification de la version du code implique de recommencer les campagnes d'injection de fautes, il est préférable qu'un minimum de paramètres soient configurables.

Pour répondre à ce cahier des charges, deux techniques ont été appliquées :

— **Instruction *NOP*** : Des instructions *NOP* ont été ajoutées dans le code juste avant la procédure de surveillance et les opérations de vérification. Si une étape doit être omise ou exécutée qu'une seule fois, il suffit de remplacer une instruction *NOP* par un saut vers un label, et de placer ce label à l'endroit souhaité. De fait, d'une déclinaison à l'autre du code, deux instructions diffèrent, le reste des images est identique. Il en est de même s'il faut modifier le déroulement de certaines étapes suite à des tests en réel infructueux. Ce qui laisse une chance de pouvoir modifier le code sans avoir à recommencer les campagnes d'injection de fautes.

— **Variables *volatile*** : Certains paramètres pouvant être amenés à évoluer, comme la période de vérification de la mémoire, se réservent la possibilité d'être ajustés. Pour cela des variables sont volontairement forcées comme variable *volatile*. Elles se retrouvent donc stockées en mémoire dans la section réservée aux données, et peuvent être modifiés de manière unitaire dans l'image du code. Cette technique est également utilisée pour sélectionner le processeur autorisé à poursuivre l'exécution de l'application dans la plateforme SoC de SQUAL basée sur deux cœurs.

Impact mémoire

L'impact de l'image du code sur la mémoire est illustré sur la Figure 3.4. On voit apparaître la section de code (gris), de données (bleu), et les ECIs (rouge). Le Tableau 3.5 fournit les taux de répartition de chaque élément cité et reprend la légende.

(a) Logiciel SERVAL

(b) Logiciel SQUAL

FIGURE 3.4 – Représentation des images mémoires

On peut voir que le code et les données ne représentent que peu de surface mémoire. C'est pour cette raison que la mémoire est entièrement initialisée avec des ECIs et que l'étape de vérification couvre toutes les cases. Ce qui permet d'augmenter considérablement la sensibilité de la mémoire aux SEUs, d'en faire un moyen de mesure du taux d'erreur, et donc d'en déduire l'hostilité du milieu lors des irradiations. Sur une mémoire ayant été qualifiée, connaissant sa section efficace, mesurer le taux d'erreur lors de qualification permet d'estimer le flux de particules.

Section	SERVAL	SQUAL	Color
CODE	18.9 %	17.5 %	Gris
DEADCODE (% de l'algorithme)	7.08 %	67.1 %	Rouge
DATA	1.31 %	2.16 %	Bleu
ECIs (mémoire inutilisée)	79.4 %	80.0 %	Noir
Pile	0.354 %	0.241 %	Vert

TABLE 3.5 – Répartition des zones mémoires sur la Figure 3.4

Temps d'exécution

Les temps d'exécution des différentes opérations sont indiqués dans le Tableau 3.6 pour les logiciels des deux plateformes. Pour obtenir une vérification mémoire toutes les secondes environ, l'algorithme CoreMark est exécuté 200 fois de suite, et l'algorithme SQUAL 1000 fois de suite.

Procédure	Opération	Exécution		Répartition	
		SERVAL	SQUAL	SERVAL	SQUAL
Initialisation	Toutes	4.52 ms	31.4 ms	-	-
Surveillance	Configuration assertions	-	5 μs	-	0.45 %
	Algorithme	5.71 ms	1.08 ms	98.8 %	97.4 %
	Vérification assertions	-	10 μs	-	0.90 %
	Vérification registres	-	213 μs	-	0.02 %
	Vérification mémoire	13.9 ms	14.1 ms	1.20 %	1.27 %

TABLE 3.6 – Répartition du temps d'exécution selon les opérations listée en Section 3.2.3

La répartition des temps d'exécution présentée dans le Tableau 3.6 permet de constater que le temps nécessaire à la vérification de la mémoire est négligeable en comparaison du temps d'exécution des multiples itérations des algorithmes, pour les logiciels des deux plateformes (1.20% pour SERVAL et 1.27% pour SQUAL).

3.3 Simulation du modèle avec injection de fautes

Les SEUs sont reproduis en simulation par inversion binaire symétrique. Cette technique est utilisée dans les cellules logiques séquentielles pour obtenir la réponse du modèle au phénomène de SEU. Une injection de faute possède donc une inconnue sur la localité et une sur le temps. Le fait de lier les conséquences d'une faute au couple cellule mémoire et temps d'injection, constitue une base de données appelée dictionnaire de fautes. Ce dernier contient donc tous les modes de défaillance du modèle. Une campagne d'injection de fautes réunissant toutes les simulations pour la génération du dictionnaire de fautes nécessite beaucoup de ressources et de temps. Dans les campagnes d'injection de fautes présentées dans ces travaux, seul le processeur est pris pour cible.

3.3.1 Environnement de simulation et modèle du SoC

La plateforme présentée dans la Section 3.1 a été développé en langage *Verilog*. Le modèle utilisé pour les simulations en campagne d'injection de faute est un modèle RTL issue directement de la compilation des fichiers sources en langage de description.

Afin de réaliser le circuit sur silicium, plusieurs étapes sont nécessaires. Tout d'abord, les éléments logiques réalisables font partie d'un ensemble de cellules standards, composées de portes logiques et de bascules, permettant de réaliser tout type de circuit de logique séquentielle et combinatoire. Chacune de ces cellules standards possède un modèle fonctionnel et physique. Le modèle fonctionnel permet de composer avec ces cellules afin de réaliser des circuits logiques lors de la synthèse de modèle RTL. Le modèle physique traduit le temps de réaction de la cellule pour déterminer la plage de fréquence de fonctionnement du circuit, l'emprunte physique sur le circuit et la consommation statique et dynamique.

La première étape de réalisation des plateformes de test est la synthèse du modèle RTL afin de composer les circuits logiques avec des cellules standards. Lors de cette phase, les outils de traitement du modèle peuvent ajouter des éléments tels que des buffers ou des bascules afin de répondre aux contraintes physiques des cellules standards. La *netlist* issue de la synthèse ne contient pas encore d'arbre d'horloge. L'étape suivante consiste en le placement et le routage physique des cellules et signaux du circuit. C'est lors de cette phase que les derniers éléments sont insérés dans le design. L'arbre d'horloge, basé sur des buffers, est alors généré selon l'emprunte physique du circuit et les contraintes de temps de propagation qu'elle impose. Le fichier de description matériel modélisant le circuit est alors la *netlist post-layout*.

Les deux *netlists* générées après l'étape de synthèse puis en particulier l'étape de placement et routage, possèdent des caractéristiques plus proches de l'implémentation physique du composant électronique. Néanmoins, les *netlists* sont très lourdes à simuler, les temps d'exécution d'un même nombre de cycles sont allongés d'un facteur 2 à 3. Les signaux hiérarchiques clairement délimités entre les différents blocs du modèle RTL, dont la dénomination est respectée par rapport au fichier source, sont remaniés et renommés de manière non déterministe d'une synthèse à l'autre. Ce qui pose un problème pour placer des points d'observation sur des signaux identifiés dans des zones spécifiques du processeur. A l'inverse, le modèle RTL traduit de manière fidèle la hiérarchie décrite dans les fichiers sources. Ce modèle est plus léger, la simulation est donc plus rapide. Il est possible d'ajouter des sondes en son sein afin de surveiller des signaux durant les simulations.

Dans le cadre de notre étude, on peut considérer qu'un processeur est composé des quatre sous-sembles suivants : les cellules de stockage, les portes de logique combinatoire, l'arbre d'horloge et l'arbre de reset. Les arbres d'horloge et de reset sont principalement constitués de buffers utilisés pour introduire des retards sur les lignes et synchroniser les signaux dans l'ensemble du design, ainsi que pour répartir les courants de commandes. Ces arbres sont générés à partir de la *netlist post-layout* et tiennent compte du placement physique des éléments dans le circuit. De nombreux buffers sont également ajoutés dans le design pour des raisons de synchronicité et faire en sorte que les données soient correctement capturées par les bascules. Comme nous avons pu le voir grâce à l'étude faite sur les registres à décalages dans le Chapitre 2, les buffers sont sujets aux SETs, dont les conséquences dépendent de leur usage. Un SET dans un buffer de donnée ou dans une porte logique combinatoire peut aboutir à un SEU, si une inversion binaire est capturée par une bascule.

En tant que signaux de référence, l'horloge et le reset, sont théoriquement beaucoup plus sensibles aux SETs, mais comme vu dans [9], ces événements sont atténués d'un buffer à l'autre, ce qui tend à diminuer leur section efficace. Les reset inopportuns sont détectés et identifiés par l'application. Les SETs dans l'arbre d'horloge peuvent affecter de manière non prévisible l'application et conduire à des cas d'erreur difficilement identifiables par la suite. Le Tableau 3.7 indique le nombre bascules et de buffers utilisés dans l'arbre d'horloge dans 4 instances différentes du processeur dans les deux SoCs de SERVAL et le SoC de SQUAL.

Test vehicle	SERVAL		SQUAL	
Instance	STD	RAD	STD	RAD
Storage cells	2974	4451	2796 (+TCM : 82535)	2796
Clock cells	332	273	209	109

TABLE 3.7 – Analyse des fautes potentielles dans le SCR1

Cependant, la section efficace d'un buffer d'horloge est environ 10 fois inférieure à celle d'une cellule de stockage en raison de l'atténuation de l'amplitude de l'impulsion et du nombre inférieur de transistor (4 transistors pour un buffer contre 6 pour une cellule SRAM par exemple). Le SER provenant des buffers d'horloge est donc négligeable devant celui des cellules de stockage dans le cas du processeur. Le choix d'un modèle RTL pour notre étude est donc pertinent. Le simulateur utilisé fait partie du logiciel Xcelium [99] développé par CADENCE. La campagne d'injection de fautes est supervisée par l'outil Functional Safety [33] de CADENCE, cette suite de logiciel fournit un outil d'analyse du modèle et gère l'injection de faute durant les simulations.

Afin de surveiller le modèle durant la simulation et de récupérer des informations de diagnostic dans le SoC à la fin de l'exécution, plusieurs éléments ont été mis en place. Chaque simulation est indépendante et est accompagnée d'un fichier de journalisation. Ce fichier est complété par le simulateur via des commandes dans le banc de test ainsi que par l'outil de d'injection de faute. Ce dernier supervise l'utilisation des sondes qui ont été placées dans tous les registres du pipeline afin d'étudier la propagation des erreurs. Le but étant de déterminer le taux de couverture des fautes par sonde ainsi que le délai entre une injection et la propagation jusqu'aux sondes, chacune d'entre elle notifie la première divergence détectée par rapport à la référence. A l'issue de la simulation, l'UDRB est copiée dans le fichier de journalisation ainsi que toutes les informations de diagnostic récoltées par l'IP d'observation. Seules les données accessibles dans le cadre de tests réels sont exploitées, outre les sondes.

3.3.2 Méthode d'injection de fautes

Pour la simulation de circuit de logique synchrone, est indiqué au simulateur un pas de temps de simulation, ainsi que le nombre de points de calcul dans ce pas. Le pas de simulation choisi est de 1 ns, le pas de calcul est de 1 ps, ce qui représente 1000 pas de calcul par pas de simulation. La génération du signal d'horloge se base sur un nombre de pas de simulation, la fréquence de l'horloge est fixée à 100 Mhz, soit une période de 10 ns.

Seuls des SEUs sont injectés dans le modèle. Le principe de l'injection est une inversion binaire symétrique. Une injection dépend de deux paramètres, la cellule cible ainsi

qu'un temps d'injection. D'après le Tableau 3.7, le processeur est composé de 2827 emplacements de SEU potentiel, correspondant à des DFFs. D'après le Tableau 3.3 sur les temps d'exécution des algorithmes de test, 107468 cycles pour SQUAL et 571312 cycles pour SERVAL sont nécessaires pour exécuter la simulation. Si une faute devait être injectée par cycle et par DFF, cela représenterait entre 300 millions et 1 milliard de simulations. Ceci n'est pas réalisable compte tenu de la durée d'une simulation de l'ordre de la minute. Faute de quoi, 400 fautes sont injectées par DFFs en sélectionnant une valeur de temps pseudo aléatoire suivant une loi normale, dont une implémentation est fournie par la bibliothèque scientifique GSL de GNU. Cette configuration représente jusqu'à 1.2 M de simulations, soit environ une semaine pour exécuter la campagne sur 100 licences en parallèle.

Comme le montre le Tableau 3.2.5, plus de 97% du temps d'exécution est consacrée à l'exécution de l'algorithme de *benchmark* pour les deux plateformes, les fautes sont donc uniquement injectées durant la phase d'exécution de l'algorithme.

3.3.3 Génération du dictionnaire de fautes

L'ensemble des SEUs potentiels dans le processeur est couvert lors des campagnes d'injection de fautes. 400 valeurs de temps d'injection sont tirées aléatoirement, ce qui permet d'obtenir des statistiques représentatives selon la Loi des grands nombres. Les issues des simulations ont été regroupées selon le Tableau 3.8. L'erreur est considérée comme masquée si l'algorithme fournit un résultat correct, seuls les autres cas sont investigués.

Issue de l'exécution	**Description**	**CoreMark**	**FFT**
Test échoué	L'algorithme n'a pas pu fournir de résultat correct	14.7 %	12.3 %
Erreur Silencieuse	Corruption des donnée de l'application	40.2 %	37.4 %
Exception	Exception levée - code d'erreur logiciel	38.8 %	38.2 %
Timeout	Délais d'attente dépassé	21.0 %	24.4 %

TABLE 3.8 – Nomenclature des erreurs et répartition

Chaque exception ou cas d'erreur identifié par le logiciel est accompagné d'un code répertorié dans le Tableau 3.4 afin d'informer l'opérateur de la raison de l'arrêt de l'application. La valeur du *timeout* est fixée à 3 fois le temps de la simulation, car une exécution peut être retardée par l'injection d'une faute mais pour autant terminer sur un résultat correct ou une erreur, et non une boucle infinie. L'injection d'erreurs couvre des DFFs de reset du processeur, ce qui génère un redémarrage de l'application avant d'être détectées par l'application. Pour les cas d'erreur, le dictionnaire de fautes est complété avec les données de diagnostic recueillies durant les simulations provenant de l'IP d'observation du SoC. Ce dictionnaire est utilisé en tant que base de données d'entrainement pour modèle de classification à apprentissage automatique présenté dans la Section 4.3. Le Tableau 3.8 présente la répartition des différentes issues des simulations pour les campagnes d'injection de fautes ciblant le cœur d'un SoC exécutant l'application de SERVAL et de SQUAL, avec 400 fautes injectées par cellules.

3.3.4 Étude de la propagation des fautes

La propagation des fautes est étudiée au travers des analyses des simulations via des sondes placées dans tous les registres et cellules du pipeline. Afin de détecter les erreurs une simulation de référence est exécutée sans injection de faute. Les signaux des sondes sont enregistrés dans une base de données par l'outil Xcelium Functional Safety. Lors de campagnes d'injection de fautes, les signaux provenant des sondes sont comparés au cycle près à ceux de référence mettant en lumière chaque divergence. Cette analyse de la propagation des fautes ainsi que des délais de propagation a été exploitée pour identifier des points d'observation dans le processeur pour les développements des IPs présentées dans les Section 4.1 et 4.2. Le but dans ce contexte est de capturer le plus d'information possible en un délais suffisamment court afin de sauvegarder une propagation de faute conduisant à une erreur.

3.4 Conclusion

Au terme de ce chapitre, nous avons présenté les plateformes SoC implémentées sur les deux circuits de test contenant nos IPs dédiées à l'observation des processeurs. Ces plateformes permettent de mettre en œuvre le processeur lors d'expérience sous faisceau de particule et de pouvoir accéder aux données de manière fiable pour collecter les informations issues des IPs. Les plateformes des deux circuits se base sur un bloc mémoire SRAM embarqué, protégé par ECC, pour le programme et les données. L'infrastructure de débogage permet au travers d'un accès JTAG conventionnel d'adresser le système, afin de charger l'application en mémoire, et de récupérer de manière autonome les données de diagnostic en cas d'erreur. Le circuit SERVAL comporte deux plateformes SoC indépendantes, le circuit SQUAL est quant à lui doté d'une plateforme basée sur deux processeurs indépendants. Chaque circuit possède donc deux cœurs indépendants dont un est partiellement implémenté avec des cellules TMR sur les registres sensibles. Ce processeur durci est utilisé à titre de comparaison pour les erreurs observées.

L'application développée dans ce contexte a été conçue dans plusieurs objectifs. Premièrement, elle s'assure de l'intégrité du programme et des données en effectuant de manière périodique une scrutation de l'entièreté de la mémoire. Les erreurs simples sont corrigées au fur et à mesure, et les erreurs doubles sont signalées ce qui met fin à l'exécution car la donnée est irrécupérable. Deuxièmement, elle implémente une fonction d'erreur correspondante à chaque mode de défaillance, de sorte à identifier les cas d'erreur et à récupérer le maximum d'informations de diagnostic. Enfin, l'application exécute un algorithme de *benchmark* afin de solliciter toutes les fonctionnalités du processeur et de pouvoir collecter les potentiels erreurs en son sein.

Pour finir, afin de pouvoir identifier de manière exhaustive tous les modes de défaillance de l'application et de fournir des données sur la propagation des erreurs dans le processeur, des campagnes d'injection de fautes par simulation ont été réalisées. Cette méthode a permis le développement et l'évaluation des IPs d'observation du processeur présentés dans le chapitre suivant.

Chapitre 4

Observation en ligne et capture des erreurs sur SoC

Ce chapitre décrit les deux modules proposées afin d'améliorer l'observabilité de systèmes basés sur microprocesseurs dans le cadre de tests sous radiations. Il fait mention des deux IPs implémentées dans les plateformes SERVAL et SQUAL décrites dans le chapitre précédent. Nous verrons comment ces techniques se positionnent par rapport à la littérature et quelles sont les limites adressées.

Ces deux IPs ont été développées dans l'objectif de capturer des informations qui permettent l'investigation d'une défaillance de l'application. La technique adoptée est d'enregistrer en continu des informations sur le flot d'exécution, et d'être en mesure de détecter une erreur afin d'arrêter l'enregistrement de la trace. De fait, dès lors qu'une défaillance survient, l'historique des dernières instructions est rendu accessible. Cette historique contient alors la signature de l'erreur que nous sommes en mesure d'exploiter pour tenter de déterminer quelle est la cause de la défaillance et d'où provient la faute. Les deux IPs proposées ont été développés afin d'être agnostiques vis-à-vis du processeur et non-intrusives envers l'application.

La première IP est une approche initiale dont les contraintes calendaires ont limité le temps de développement. Elle est donc dotée de fonctionnalités réduites qui ont permis de valider l'approche et qui se sont révélées être fiables et capables de fournir des informations pertinentes. La seconde IP contient un mécanisme de génération de trace basés sur les spécifications RISC-V [86] et un ensemble d'assertions matérielles afin de couvrir un plus large spectre de modes de défaillance. Elle constitue une deuxième approche visant à apporter un aperçu plus détaillé et plus complet du flot d'exécution précédent une erreur, pour adresser des cas dont la signature est plus complexe.

Pour analyser les informations de diagnostic récoltées, nous proposons une technique consistant à établir une correspondance entre le cas d'erreur à analyser et un des cas d'erreurs identifiés dans la campagne d'injection de faute. Un modèle de classification basé sur machine-learning a été utilisé pour cette approche. Les résultats de classification ont permis d'avoir un retour sur le taux d'observation et ouvrent de nombreuses perspectives d'évolution.

4.1 Première approche : Capture de l'historique de registres processeur

Les informations de diagnostic sont collectées suite à une défaillance et permettent l'investigation de la cause de celle-ci. Elles peuvent être constituées de registres d'état du processeur, de trace d'exécution, ou tout autres informations en lien avec l'exécution de l'application. Comme nous avons pu le voir au travers de la littérature dans la Section 1.3, les informations de diagnostic sont pauvres et ne permettent pas de comprendre la manière dont une faute a pu engendrer une erreur bloquante. Néanmoins, dans un article récent [78], les auteurs exploitent les informations concernant les exceptions de faute issues de la sauvegarde de registres d'état du processeur. Cependant, la profondeur de l'analyse est limitée aux capacités d'indentification d'une erreur du processeur en relation avec la dernière instruction ayant été exécutée.

Les articles [75, 77] ont pavé la voie de notre approche. Dans ces travaux, la trace processeur est enregistrée en continu dans un buffer grâce à l'architecture du CoreSight, jusqu'à ce qu'une erreur soit détectée. Par ce moyen, les auteurs ont pu évaluer la sensibilité des instructions du code et constater d'importantes disparités. Cependant, ils reportent qu'un manque de fiabilité a fortement nuit à l'exploitation des résultats, limitant le nombre de cas d'erreur exploitables lors des tests sous faisceau.

Notre approche consiste donc à continuer sur cette voie, mais en proposant un moyen d'enregistrement durci de la trace en continue. Cette IP est couplée à un mécanisme de détection d'erreur pour capturer la trace au moment opportun, ainsi qu'une zone mémoire dédiée à l'utilisateur. Un effort a été fait afin de garantir un accès aux données indépendant de l'application. La contribution a été implémentée dans un vecteur de tests en 28 nm UTBB FDSOI. La Figure 4.1 présente le schéma d'architecture de l'IP d'observation dans sa première version.

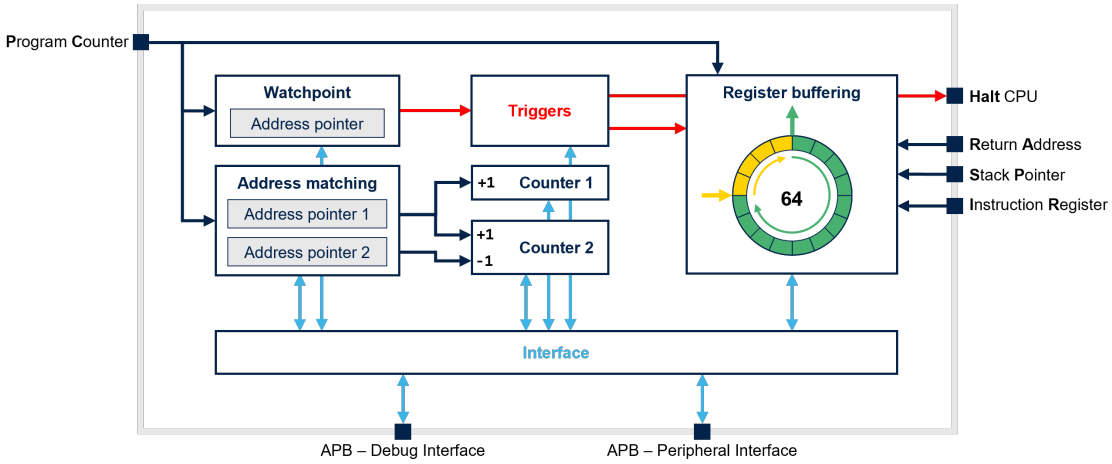


FIGURE 4.1 – Architecture de l'IP d'observation initiale (V1)

4.1.1 Intégration de l'IP dans le SoC

Selon la procédure logicielle détaillée dans la Section 3.2.3, l'IP d'observation est configurée par l'application durant la phase d'initialisation. En revanche, d'après les standards

de test définis par la norme JESD89B de JEDEC [49], décris Section 1.4, les informations doivent être récupérées via une requête de l'utilisateur depuis le banc de test. L'IP est donc accessible de manière autonome depuis l'architecture de débogage.

Pour ce faire, l'IP d'observation est mappée sur le bus de débogage et le bus périphérique, comme illustré sur la Figure 3.2. Elle est accessible depuis le processeur en tant que périphérique, et depuis l'accès JTAG. L'accès depuis le bus de débogage est prioritaire dans l'interface de l'IP. Celle-ci possède une sortie connectée au processeur permettant de bloquer le pipeline et donc d'arrêter l'exécution. Avant tout intervention depuis l'interface de débogage, pour charger le programme ou récupérer des informations de diagnostic, le pipeline est gelé, car un comportement erratique de l'application pourrait venir corrompre des données. L'UDRB est mappée en mémoire dans l'IP d'observation, elle peut donc être adressée depuis l'accès JTAG au même titre que l'IP, sans aucune intrusion envers l'application courante. Ce qui permet faire des lectures répétitives depuis le banc de test FPGA pour surveiller l'application, sans faire d'accès concurrent avec le processeur.

L'IP fonctionne à la même vitesse que le cœur. Elle possède des sondes sur 4 registres de 32 bits du processeur, le registre du pointeur de programme (PC), d'instruction (IR), du pointeur de pile (SP) et d'adresse de retour de fonction (RA). Les sondes permettent d'enregistrer l'historique des valeurs de ces registres. Le choix des registres d'intérêt est détaillé dans Section 4.2.3.

4.1.2 Détection d'erreurs

Dans cette première approche, la détection d'erreur repose sur celle du processeur. Elle est composée des trois blocs suivants :

- ***Watchpoint*** : Le *watchpoint* contient un registre d'adresse comparé en continu avec le PC. Si les deux valeurs correspondent un signal est transmis au bloc *trigger*.
- ***Triggers*** : Ce bloc configurable possède deux sorties. La première permet de bloquer le pipeline, et la seconde d'arrêter l'enregistrement de la trace dans le buffer. Selon la configuration, les deux actions peuvent être déclenchées sur signal du *watchpoint*.
- ***Address matching*** : Ce bloc ainsi que les compteurs 1 et 2, vont de pair, ils permettent de surveiller deux instructions dans un programme supposés être exécutées le même nombre de fois. Il contient deux registres d'adresse comparés en continue avec la valeur du PC. Si la valeur du premier registre correspond à celle du PC, les deux compteurs sont incrémentés (**Counter 1** et **Counter 2**). Si la valeur du second registre correspond à celle du PC, le compteur 2 (**Counter 2**) est décrémenté.
- ***Counter 1*** : La valeur de ce compteur représente le nombre d'accès à la première adresse, qui peut être le début d'une fonction par exemple.
- ***Counter 2*** : La valeur de ce compteur représente le nombre de fois où le processeur a accédé à la première adresse mais pas à la deuxième. Ce compteur permet par exemple de surveiller une fonction supposée être atomique. Cette assertion a été spécifiquement implémentée pour détecter la réentrance d'interruption due à une faiblesse de l'architecture du processeur SCR1.

Cas d'usage

Dans le cadre de la plateforme SERVAL, la configuration des fonctionnalités de détection d'erreur est détaillée par blocs dans les points suivants :

- ***Watchpoint*** : L'adresse configurée dans le registre est celle du vecteur *Machine Trap* afin de détecter toutes les interruptions et exceptions.
- ***Triggers*** : L'action d'arrêter le processeur a été utilisée dans le cadre du développement de l'application, mais dans le cadre d'une utilisation standard, le processeur doit pouvoir poursuivre l'exécution du *handler* d'interruption utilisé comme fonction d'erreur, décrite dans la Section 3.2.4. Donc seule la fonction permettant d'arrêter l'enregistrement de la trace sur signal du *watchpoint* est configurée.
- ***Address matching*** : Les deux registres d'adresse de ce bloc contiennent l'adresse de *boot* pour le premier, et le vecteur de reset pour le deuxième.
- ***Counter 1*** : Ce compteur indique donc le nombre d'accès au vecteur de boot, c'est-à-dire le nombre de resets et d'interruptions ou d'exceptions.
- ***Counter 2*** : Ce compteur indique le nombre de fois où le processeur a accédé à un autre vecteur que celui de reset. Il est donc possible de distinguer d'un comportement erratique s'il s'agit d'appels récursifs infinis d'interruptions ou de resets intempestifs de l'application.

Comme décrit dans la Section 3.1.1, lors d'une exception ou interruption, le processeur effectue un saut sur l'unique vecteur d'exception *Machine Trap*, ce qui stoppe l'enregistrement de la trace. En cas d'interruption légale, la trace est réactivée et le trigger réarmé. Une telle interruption dans notre application ne peut survenir que suite à une erreur simple détectée en mémoire. La détection d'erreur couvre donc tous les cas d'erreur d'exception ou d'interruption du Tableau 3.4.

4.1.3 Capture de la propagation

L'approche repose sur la capture de la propagation de fautes dans le processeur lorsqu'une erreur a généré une exception. Un processeur est un bloc logique conséquent qui traite une bande passante de données très élevée. Ceci impose de lourdes contraintes lorsqu'il s'agit d'observer et de sauvegarder l'historique d'un ensemble de données du flot d'exécution. C'est pour cette raison que des compromis doivent être fait dans le choix des signaux à observer, et de la taille du buffer, afin que l'IP d'observation puisse être réalisable. Nous verrons par la suite au travers de l'analyse des données extraites, une manière d'estimer le taux d'observation sur le processeur selon les données fournies par l'IP, Section 4.3.

Pour déterminer quels signaux du pipeline sont les plus pertinents à observer dans notre approche, nous proposons d'étudier le taux de couverture d'observation des erreurs depuis des sondes placées sur ces signaux. Le délai de propagation d'une faute injectée jusqu'à l'observation d'une divergence sur une sondes est aussi une grandeur prise en compte dans le choix des signaux. Plus ce délai est faible, plus l'information est supposée refléter l'origine de l'erreur et non les conséquences d'un comportement erratique. Les informations dans le pipeline circulent d'un bloc à l'autre, il est donc intéressant de capturer un type de donnée, comme les instructions ou le PC, juste avant le bloc sur le point de l'analyser, susceptible de lever une exception qui arrête la trace. Enfin, compte tenu que l'utilisateur

est le premier à pouvoir porter un regard critique sur la trace enregistrée, il est important que les signaux reflètent une grandeur tangible, qui puisse être interprétée. Par exemple, la cohérence d'une adresse ou d'une instruction peut rapidement être évaluée contrairement à d'autres signaux plus spécifiques.

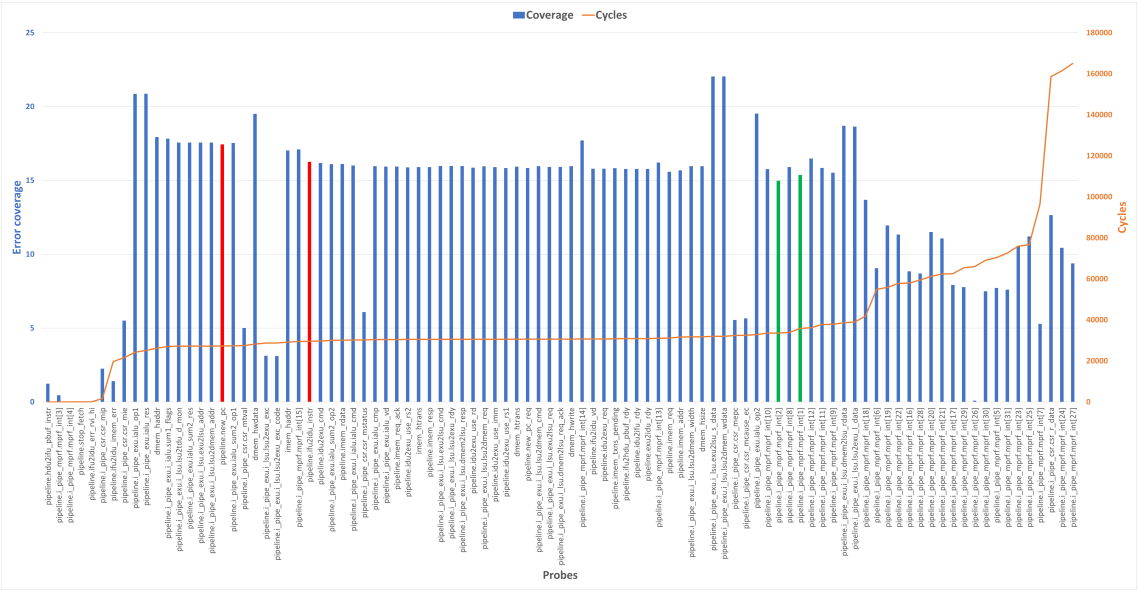


FIGURE 4.2 – Couvertures et délais de propagation de signaux du pipeline du SCR1 - application CoreMark

La Figure 4.2 est issue de l'analyse des simulations de la campagne d'injection de fautes sur la plateforme SERVAL, dont l'application est basée sur le CoreMark. L'échelle de gauche est le taux de couverture de détection des erreurs par rapport au nombre total d'erreurs injectées. L'échelle de droite est le nombre de cycles moyen entre l'injection et la détection de la faute sur la sonde. Le délai moyen de propagation sur chaque sonde n'est pas pris en compte pour sa valeur, mais pour sa position relative par rapport aux autres sondes. On peut qualifier l'inverse de ce délai par le taux de réactivité.

Ci-dessous est présentée la liste des signaux choisis pour être observés et sauvegardés dans le buffer, avec les noms des registres correspondant à la Figure 4.2 :

— ***Program Counter*** (*pipeline.new_pc*) : La valeur du PC est prélevée dans le pipeline, juste après le calcul de la nouvelle valeur dans le bloc IFU.
— ***Instruction Register*** (*pipeline.ifu2idu_instr*) : Le registre d'instruction est récupéré en sortie du bloc IFU, juste avant son décodage par l'IDU.
— ***Stack Pointer*** (*pipeline.i_pipe_mprf_mprf_int[2]*) : Le pointeur de pile provient du fichier des registres.
— ***Return Address*** (*pipeline.i_pipe_mprf_mprf_int[1]*) : L'adresse de retour provient également du fichier des registres.

Le PC et l'IR sont indispensables, ils constituent la trace d'exécution. Les sondes sont placées sur des registres du pipeline fournissant les meilleurs taux de couverture et réactivité pour ces types de donnée. Les registres SP et RA font également partie des sondes ayant un très bon taux de couverture. Ils ont été choisis car les informations fournies sont complémentaires, et leurs caractéristiques dépendent peu de l'application. Nous verrons que pour l'autre benchmark, les taux de couverture et de réactivité des sondes sur le contenu

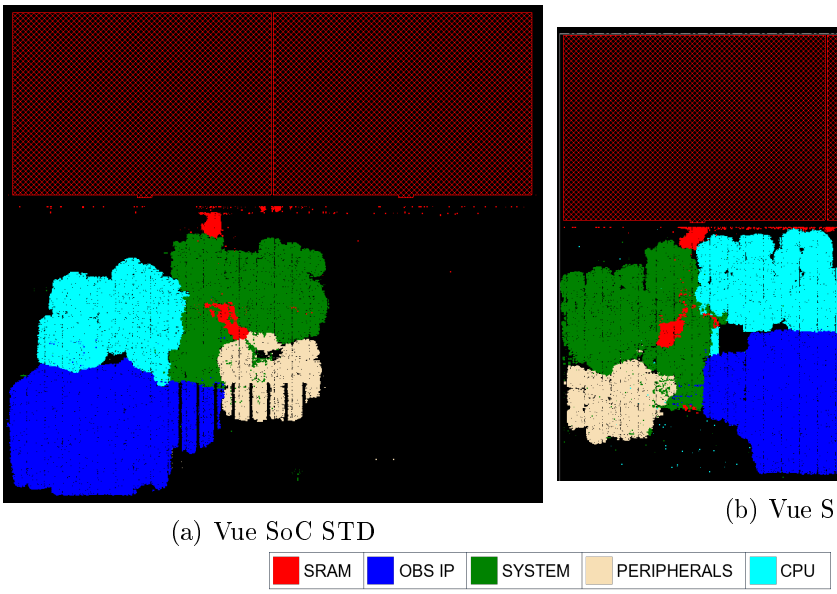
du fichier de registres sont assez différents, Section 4.2.3. Le comportement de l'application influe de manière significative sur la collecte des erreurs dans le fichier de registres, notamment pour les champs qui concernent les appels et les arguments de fonction.

Les quatre signaux (PC, IR, SP et RA) sont donc concaténés et enregistrés dans un buffer circulaire, comme illustré sur la Figure 4.1. Ce buffer a une taille de 1 kB, ce qui lui confère une profondeur de 64 événements de 128 bits chacun. Une copie des 128 bits dans une ligne du buffer est déclenchée si un bit au moins a été modifié. Une fois la mémoire entièrement remplie, les événements les plus anciens sont écrasés par les nouveaux. Cette taille a été imposée par les contraintes de surface disponible sur le silicium. La version de cette IP a été implémentée dans les deux SoCs du vecteur de test SERVAL, dans le but de comparer le comportement du processeur durci et de celui standard. Par la suite, nous proposons deux approches afin de d'optimiser la taille du buffer, Section 5.2.2 et 5.2.3.

4.1.4 Implémentation

Dans le cadre de tests sous radiations, l'implémentation doit faire l'objet d'un durcissement afin de garantir la fiabilité de la mesure et la cohérence des données. Dans l'IP d'observation, tous les registres, incluant ceux de l'UDRB ont été implémentés avec des cellules TMR. Le Tableau 4.1 récapitule l'impact mémoire et le type d'implémentation des registres de l'IP. Le durcissement par conception couvre tous les registres dont la persistance des données est importante. Cette technique n'a pas pu être étendue dans le buffer en raison du manque de surface de silicium. Néanmoins, la technologie 28 nm UTBB FDSOI est durcie par conception et ne souffre pas d'un SER élevé durant les tests. Le buffer de 64 événements enregistre en continue la trace durant les tests et le processeur fonctionne à une fréquence de l'ordre de la centaine de MHz. Ainsi les données ont une persistance de stockage très limitée. Lorsqu'une erreur bloque le système, l'enregistrement et interrompu et le code d'erreur apparait immédiatement dans l'UDRB. L'utilisateur est alors notifié de l'erreur dans l'itération de lecture suivante du registre d'état, et les données sont entièrement récupérées dans les 10 secondes qui suivent l'arrêt de l'enregistrement. La probabilité que les données soient corrompues dans ces conditions reste donc très limitée.

Les Figures 4.3 viennent en complément de la vue GDS, Figure 2.11, elles représentent l'implémentation des SoCs STD et RAD en faisant apparaître les IPs de différentes couleurs. En superposant cette vue avec le GDS il est possible de localiser de manière précise chaque IP sur le Silicium. Cette démarche a de l'intérêt dans le cadre de tests sous rayons X, décrit en Section 2.1.4, où le faisceau peut être concentré sur une surface de l'ordre de celle de quelques cellules. Il est donc possible d'injecter des fautes de manière sporadique en ciblant des IPs ou blocs mémoires. Sur cette figure sont représentées les différentes parties des deux SoCs, incluant les cœurs (version STD - standard / RAD - radiation-hardened), l'ensemble bus système, l'ensemble bus périphérique, la mémoire, et l'IP d'observation.



(a) Vue SoC STD

(b) Vue Soc RAD

(c) Légende

FIGURE 4.3 – Représentation de l'implémentation des deux SoCs de SERVAL

4.2 Seconde approche : Capture de la trace processeur et assertions matérielles

Dans cette approche, deux limitations ont été adressées par rapport aux travaux présentés dans la Section 4.1. Premièrement tous les cas d'erreur du processeur n'étaient pas couverts. Seuls les cas d'exception de fautes étaient couverts, soit 40% des cas d'erreur, selon le Tableau 3.8. Le détecteur d'erreur a été amélioré afin d'implémenter des assertions matérielles configurables proches de l'application. Ces assertions surveillent le comportement de l'application sans aucune intrusion envers elle, ni besoin d'intervention de sa part. Aucune modification de celle-ci n'est donc nécessaire, seules des adresses doivent être récupérées de l'image du code lors de la compilation.

La deuxième limitation s'avère être la profondeur mémoire. Le buffer dans sa version initiale n'implémente pas de tableau mémoire, mais des registres indépendants, implémentés avec des DFFs. Cette configuration n'est pas efficace en termes de compacité d'implémentation. La surface de silicium disponible dans le vecteur de test SERVAL étant réduite, le buffer n'a pas pu être durci. Dans cette deuxième approche, un générateur d'événements de trace optimise les données à enregistrer dans le buffer grâce à une machine à état synchronisée avec le pipeline. Deux registres supplémentaires sont inclus dans la trace, parmi un ensemble de 4 registres sondés. Le buffer est dorénavant basé sur un tableau mémoire plus compact, où chaque case mémoire est protégée par un code correcteur d'erreur.

Dans le vecteur de test SERVAL, deux instances du SoC coexistent, il en est de même pour l'IP d'observation. Dans SQUAL, une instance de l'IP d'observation a été implémentée, elle peut être configurée pour surveiller l'un ou l'autre des deux cœurs. Cette seconde approche propose une version plus aboutie que la version précédente. Des éléments ont été travaillés afin d'augmenter le taux d'observation des fautes survenant dans le processeur. Les techniques mises en œuvre se basent sur celles à l'état de l'art en adressant les limites existantes. Le vecteur de test SQUAL est fabriqué en 28 nm UTBB FDSOI et

est conçu pour faire partie d'application spatiale GEO. La Figure 4.4 présente le schéma d'architecture de l'IP d'observation dans sa seconde version.

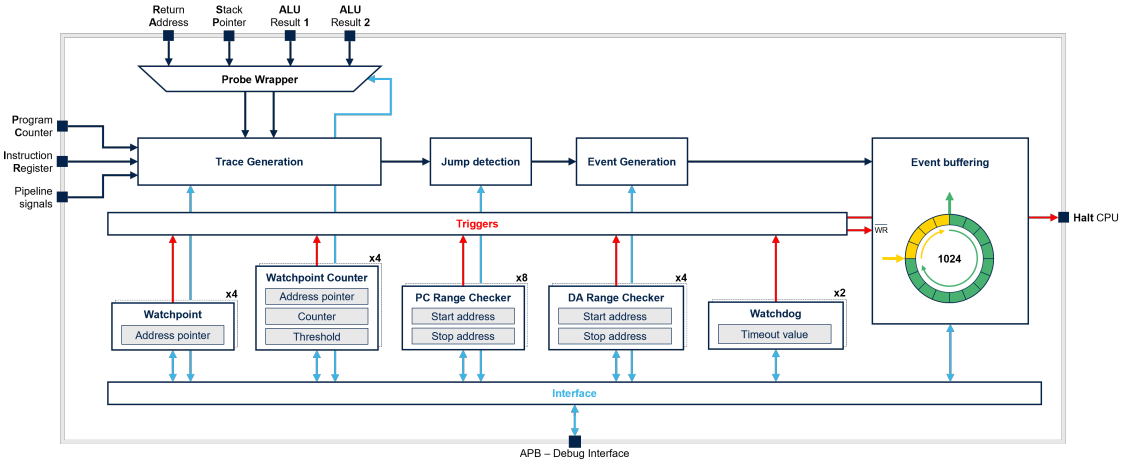


FIGURE 4.4 – Architecture de l'IP d'observation avancée (V2)

4.2.1 Intégration de l'IP dans le SoC

L'IP d'observation n'est mappée que sur le bus de débogage, comme présenté sur le schéma d'architecture de la Figure 3.3. Les améliorations de l'architecture de débogage, décrite dans la Section 3.1.2, permettent au processeur d'accéder à ce bus en tant que maître depuis le bus système. L'IP d'observation peut donc être configurée depuis l'application. L'accès JTAG reste inchangé dans sa structure, dorénavant il gagne en bande passante de manière significative puisque le débogage du cœur ne fait plus partie de la chaîne JTAG d'entrée. Le scénario d'usage de cette nouvelle IP d'observation reste le même, l'application configure l'IP au démarrage, puis vérifie de manière périodique que sa configuration n'ait pas été altérée par des SEUs. Lorsqu'une erreur est détectée dans l'application, l'utilisateur récupère alors les données de manière autonome depuis le port JTAG.

Des sondes sont placées directement dans le cœur afin de récupérer la valeur des registres à surveiller, et de prélever des signaux pour restaurer la trace d'exécution depuis l'IP d'observation. La plateforme SQUAL est basée sur deux cœurs, comme décrit dans la Section 3.1.2. Elle n'a pas été développée dans le but d'évaluer un SoC double coeur mais chaque cœur de manière indépendante. Une seule instance de l'IP d'observation est donc instanciée dans le SoC ainsi qu'un multiplexeur pour sélectionner le cœur à observer. Les sondes des registres et signaux sont donc placées dans les deux processeurs de manière équivalente.

4.2.2 Détection d'erreurs basée sur assertions matérielles

Le taux de couverture de l'IP précédente correspond au taux d'exception de faute uniquement, soit 40% des erreurs, comme indiqué dans le Tableau 3.8. Le bloc de détection d'erreur de cette nouvelle version de l'IP implémente un ensemble d'assertions matérielles configurables pour surveiller plus finement l'application. Ces assertions se basent sur l'observation de l'évolution des valeurs du PC et des adresses lues ou écrites depuis le

processeur. Le bloc de surveillance n'est donc pas intrusif envers l'application et ne requiert aucune action de sa part. Le but de ces assertions est de surveiller les zones d'accès du processeur et le nombre d'accès à certaines fonctions afin de détecter les erreurs puis de fournir des éléments de diagnostic.

Ci-dessous est décrite la fonction de chaque assertion ainsi que la configuration utilisée :

- ***Watchpoints*** : Les *watchpoints* configurables ont le même fonctionnement que celui décrit précédemment, 4 instances ont été implémentées.
- ***Watchpoint Counters*** : Les *Watchpoint Counters* contiennent, comme les *watchpoints*, un registre d'adresse comparé en continue avec le PC, un registre de valeur seuil et un pour le compteur. Si le PC correspond à l'adresse indiquée dans le registre, un compteur est incrémenté. Dans la cas où la valeur de ce compteur franchit celle du seuil, un signal est transmis au bloc *Trigger*. Quatre assertions de ce type ont été implémentées.
- ***PC Range Checkers*** : Le bloc *PC Range Checkers* contient un ensemble de 8 unités de comparaison de la valeur du PC, afin de définir des zones d'exclusion et d'inclusion. Chacune de ces unités de comparaison contient 2 registres d'adresse (borne supérieure et borne inférieure), si le PC est hors de cette fourchette de valeur, un signal est levé en sortie. Si le bloc *PC Range Checkers* détecte que le PC est hors d'une zone autorisée, un signal est transmis au bloc *Trigger*.
- ***DA Range Checkers*** : Le bloc *Data Address (DA) Range Checkers* fonctionne de la même manière que le bloc précédent. Il définit des zones d'inclusion d'adresses dans le SoC grâce à 4 unités de comparaison comportant chacune deux registres d'adresse (borne supérieure et borne inférieure). Si un accès mémoire provenant du cœur est hors d'une zone autorisée, un signal est transmis au bloc *Trigger*.
- ***Watchdogs*** : Les deux *Watchdogs* sont deux compteurs de front d'horloge possédant chacun une valeur seuil stockée dans un registre. Si la valeur du compteur dépasse celle de la valeur seuil, un signal est envoyé au bloc *Trigger*.
- ***Triggers*** : Ce bloc est hérité de celui de la précédente IP. Les signaux provenant des assertions sont mappés via une matrice de configuration vers les deux déclencheurs afin d'arrêter l'enregistrement et/ou d'arrêter l'exécution.

Cas d'usage

Dans le cadre de la plateforme SQUAL, la configuration des assertions matérielles est détaillée par blocs dans les points suivants :

- ***Watchpoints*** : Implémentés au nombre de 4, ils surveillent les 4 points d'entrées possibles lors du boot, à savoir les vecteurs de *Reset*, *Machine trap*, *Hypervisor*, et *Supervisor*, listés dans les types d'interruption, Tableau 3.4. Les *watchpoints* apportent des informations dans le cas d'un comportement erratique n'ayant pas offert la possibilité au logiciel d'exporter de code d'erreur.
- ***Watchpoint Counters*** : Implémentés au nombre de 4, les *Watchpoint Counters* sont placés sur des instructions *nop* insérées dans des fonctions de l'algorithme FFT, en particulier celles du calcul des puissances, et des coefficients. Ces blocs permettent notamment de capturer des cas d'erreurs de calcul ou de *timeout*, lorsque le nombre d'itérations dépasse une valeur attendue. Ceci peut être dû à

des données corrompues, ou des sauts de PC légaux dans des zones où les données ne sont pas initialisées.

— ***PC Range Checkers*** : Ce bloc définit des zones d'inclusion qui entourent l'application de *benchmark* uniquement, excluant toutes les fonctions d'initialisation ou d'erreur.
— ***DA Range Checkers*** : Les zones d'accès définies comme étant autorisées sont l'ensemble des adresses correspondant au code, aux données, ainsi qu'à l'IP d'observation. Un accès hors de ces zones est considéré comme illégal et est signalé au bloc *Trigger*.
— ***Watchdogs*** : Dans l'application, les deux *watchdogs* sont seulement utilisés à titre informatif pour mesurer le temps d'exécution du *benchmark* et des étapes de vérification de la procédure de surveillance. En effet, certaines exécutions sont retardées mais pour autant fournissent des résultats corrects. De plus, tout blocage de l'application est détecté depuis le banc de test qui opère le SoC.
— ***Triggers*** : Dans la procédure logicielle présentée dans la Section 3.2.3, le *Triggers* est configuré pour arrêter l'enregistrement sur détection d'une erreur. L'exécution de la fonction d'erreur peut se poursuivre afin de collecter le maximum d'information de diagnostic et d'arrêter ensuite l'exécution.

La configuration du bloc de détection d'erreur nécessite de compiler une première fois le code. L'image est ensuite analysée pour en extraire les adresses d'intérêt, le code est alors à nouveau compilé avec les valeurs de configuration mise à jour. Lors de l'exécution itérative de la procédure de surveillance, explicitée en Section 3.2.3, la vérification des assertions consiste à contrôler les compteurs des *watchdogs*, et l'état des assertions. En effet, il est possible que des assertions soient levées quand bien même le résultat de l'algorithme soit correct. Ces cas sont rares mais considérés comme des erreurs et font l'objet d'investigation. Durant la phase d'initialisation des assertions, les *Watchpoint Counters* ainsi que les *Watchdogs* sont remis à zéro avant la nouvelle exécution de l'algorithme. En cas d'interruption légale, à savoir une erreur simple en mémoire dans cadre de notre application, les assertions sont réinitialisées et les déclencheurs sont réarmés. Ce bloc de détection d'erreur couvre donc toutes les issues possibles énoncées dans le Tableau 3.8, les taux de couverture seront présentés dans le Chapitre 5.

4.2.3 Capture de la trace d'exécution

Dans cette approche, la profondeur d'enregistrement ainsi que la fiabilité des données sont les problématiques qui ont été adressées. La gestion de la capture de la trace est assurée par quatre blocs configurables depuis l'interface. Ces blocs sont décrits dans les points suivants, leur intégration dans l'IP est détaillée sur la Figure 4.4.

— ***Event Buffering*** : Un tableau mémoire de 16 kB est désormais géré dans le bloc *Event Buffering*, ce qui représente une profondeur de 1024 éléments de trace de 4 registres de 32 bits. Grace à l'échantillonnage des registres synchronisé avec le pipeline, le bloc génère une trace cohérente et donc moins d'événements que la première approche. La taille du buffer a quant à elle été multipliée par 16 par rapport à l'IP précédente. Par conséquent, les conditions tendent à se rapprocher des conditions critiques décrites dans [75, 77]. Les auteurs constataient alors que les traces étaient incohérentes et inexploitables. C'est pourquoi chaque cellule mémoire du tableau est dotée d'un code correcteur d'erreur SECDED, permettant de

corriger automatiquement les erreurs simples lors de la lecture de la trace depuis le banc de test, et de signaler les erreurs doubles. La profondeur mémoire permet de capturer un taux d'erreur plus important, en couvrant le délai de propagation des fautes dû à l'architecture du processeur ou au comportement de l'application par rapport à la couverture matérielle. Le tableau mémoire est géré comme un buffer circulaire. Dans le cas d'une congestion mémoire, les nouveaux événements écrasent les plus anciens.

— ***Probe Wrapper*** : Un ensemble de 4 registres est disponible pour être enregistré, deux seulement peuvent faire partie de la trace. Les registres supplémentaires sélectionnés ne contiennent pas que des valeurs étant interprétables par un utilisateur mais qui pour autant comportent des informations très pertinentes au regard du flot d'exécution. La technique d'analyse de la trace proposée dans la Section 4.3 sait tirer le meilleur parti de ces données. A l'inverse, dans le cadre d'un débogage d'une application, la même configuration que la première approche pourra être choisie donnant accès au pointeur de pile et à l'adresse de retour.

— ***Trace Generation*** : Ce bloc implémente une machine à état qui se base sur des signaux du pipeline pour synchroniser la capture des valeurs des registres afin de générer une trace cohérente. Cette trace est initialement composée du PC et de l'instruction lui correspondant. Les signaux sont prélevés au niveau du bloc IFU, qui constitue la première étape du pipeline, avant donc que les autres blocs aient pu lever des exceptions de fautes quant à la cohérence des données. D'autres signaux provenant de l'unité d'exécution sont ajoutés à la trace pour améliorer l'observabilité sur le flot d'exécution, le choix est justifié par la suite.

— ***Jump detection*** : Ce bloc effectue un filtrage sur les événements de trace récoltés afin de ne garder que ceux issus d'instructions de branchement. Cette technique permet de couvrir un nombre d'instructions plus important pour un même nombre d'événements stockés. Le code entre les instructions peut être restauré grâce au code compilé. Néanmoins, si une faute corrompt un saut, il est tracé. Cependant, cette technique est moins à même de capturer une propagation d'erreur car la fréquence d'échantillonnage des registres est drastiquement réduite. Elle n'est utilisée que dans les phases de débogage de l'application. Une fréquence d'échantillonnage plus élevée des registres est préférable au vu de la capacité du buffer d'événements et dans l'objectif d'observer l'activité du processeur avec une granularité fine.

— ***Event Generation*** : Les données sont compactées par ce bloc afin de générer un vecteur de 128 bits à enregistrer dans le buffer circulaire. Compte tenu de la taille de la mémoire, les bits inutiles du PC sont retirés et remplacés par un comptage du nombre de cycle d'horloge séparant chaque événement. Ceci constitue une datation relative d'un événement à l'autre.

Le choix des registres à surveiller a donc été reconsidéré en adressant des zones plus spécifiques de l'architecture du processeur. D'après la Figure 4.5, deux autres registres fournissent des taux de capture et de réactivité parmi les plus élevés. Le seul inconvénient est qu'il n'est pas possible d'interpréter la cohérence des données récoltées du point de vue de l'utilisateur. Seuls des outils d'analyses croisées permettent d'extraire des caractéristiques discriminantes afin de détecter des fautes, la méthode utilisée est présentée en Section 4.3.

Comme décrit dans la Section 3.1.1, le bloc EXU du processeur possède deux ALUs. La première reçoit deux registres de 32 bits, et supporte tous les opérandes sur des nombres entiers, les résultats de calcul affluent donc par le registre résultat de cette unité. La seconde ALU ne supporte que les additions, elle n'est composée que de logique combinatoire, le résultat est donc rendu immédiatement. Cette unité est utilisée pour calculer les sauts de

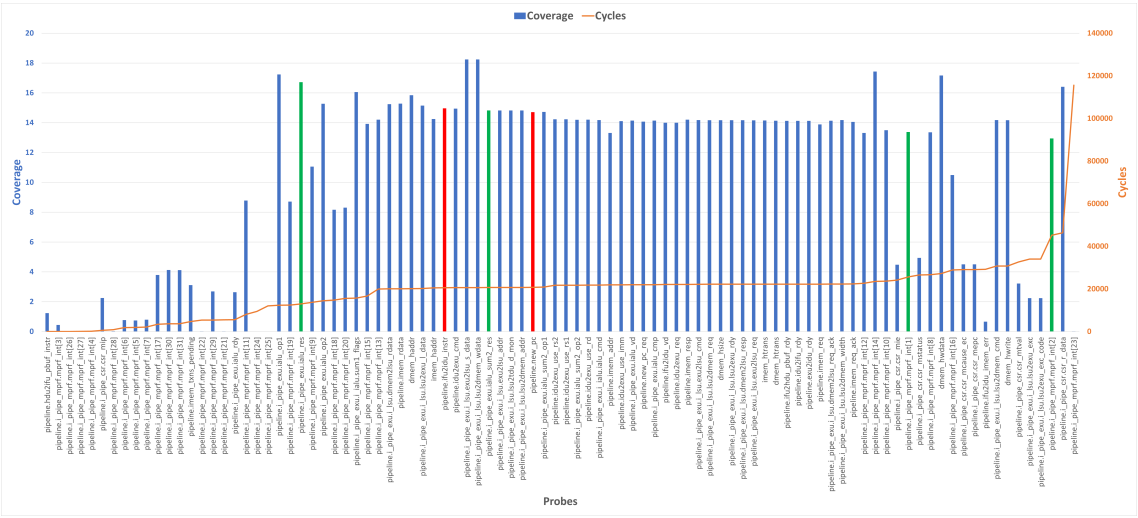


FIGURE 4.5 – Couvertures et délais de propagation de signaux du pipeline du SCR1 - application FFT

PC. Des sondes ont été placées sur ces deux registres de résultats ainsi que ceux hérités de l'IP précédentes. Les points suivants détaillent leurs emplacements dans le pipeline :

- ***ALU 1 Result* (ALU1)** (*pipeline.i_pipe_exu.ialu_res*) : Le résultat de l'ALU 1 est récupéré directement en sorti du bloc, l'échantillonnage est synchronisé par le module *Trace Generation*.
- ***ALU 2 Sum result* (ALU2)** (*pipeline.i_pipe_exu.ialu_sum2_res*) : Le résultat de l'ALU 2 est aussi récupéré en sorti du bloc, l'échantillonnage par le module *Trace Generation* permet d'avoir des informations sur la prochaine adresse du PC en cas d'instruction de branchement.
- ***Stack Pointer* (SP)** (*pipeline.i_pipe_mprf_mprf_int[2]*) : Le pointeur de pile.
- ***Return Address* (SA)** (*pipeline.i_pipe_mprf_mprf_int[1]*) : L'adresse de retour.

Les deux registres sondés comportant les résultats des deux ALUs possèdent des informations n'ayant aucune corrélation entre eux ni avec les registres d'instruction et du PC. Parmi les 4 registres ALU1, ALU2, SP et RA, seuls deux peuvent être enregistrés en même temps que la trace dans le buffer. Lors de session de débogage, le choix se portera sur les registres SP et RA permettant à l'utilisateur de suivre la cohérence du flot d'exécution et l'utilisation de la mémoire. Tandis que, lors des campagnes d'injection de fautes ou des expérimentations sous faisceau, les registres ALU1 et ALU2 seront choisis afin de fournir une trace plus riche en informations pour l'outil d'interprétation automatique de la trace.

4.2.4 Implémentation

La taille du tableau mémoire implémenté pour le buffer de trace correspond à une des valeur maximales configurable dans la panoplie des SoCs *low-end* basés sur RISC-V proposés par SiFive Core Designer[20]. Dans le cadre du développement d'un vecteur de test, nous avons choisi la taille maximale implémentable, les contraintes calendaires n'ayant pas permis de développer l'ensemble des méthodes d'analyse des traces utilisées pour estimer une valeur optimale. Néanmoins, comme pour l'approche précédente, une taille optimale est

évaluée grâce à la technique présentée en Section 4.3 permettant d'analyser l'ensemble des traces récoltés lors de la campagne d'injection de fautes et d'estimer un taux d'observation.

Le Tableau 4.1 récapitule les ressources mémoires nécessaires à l'implémentation de cette IP d'observation et le type d'implémentation utilisé. Les pointeurs d'adresse, cités dans ce tableau, représentent l'ensemble des registres de configuration des assertions matérielles du bloc de détection d'erreur. La persistance des données est très élevée dans ces registres, c'est pourquoi leur contenu est vérifié de manière périodique dans la procédure de surveillance.

Elément	**Nombre de cellules**		**Implémentation**	
	V1	**V2**	**V1**	**V2**
IP d'observation	245 b	2075 b	Cellules TMR	Cellules durcies
UDRB	512 b	512 b	Cellules TMR	Cellules durcies
Buffer de trace	1024 B	17,536 B	Cellules standards	Mémoire standard avec ECC

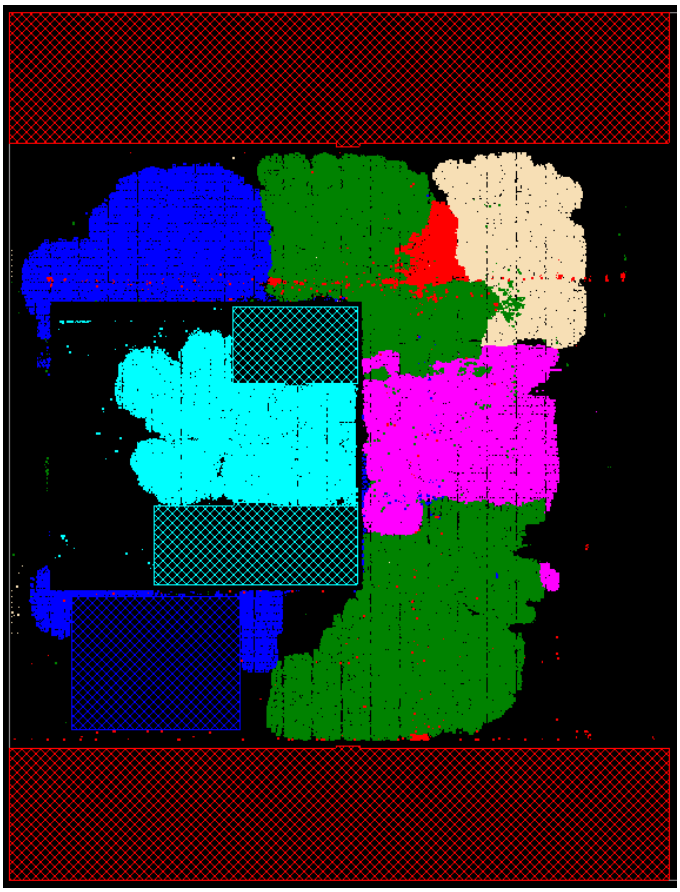
TABLE 4.1 – Impact mémoire des IPs d'observation de SERVAL (V1) et SQUAL (V2)

La Figure 4.6 illustre l'implémentation du SoC de SQUAL. Sur cette figure sont représentées les différentes parties du SoC, en incluant les cœurs (version STD - standard avec TCM / RAD - radiation-hardened), l'ensemble bus système, l'ensemble bus périphérique, la mémoire, et l'IP d'observation. On remarquera dans l'IP d'observation l'apparition du bloc mémoire (rectangle en bas - zone bleu), et que cette IP accapare une surface plus importante que celle de la V1, principalement à cause des registres de configuration nécessaires au détecteur d'erreur.

4.3 Interprétation automatique de la trace du processeur

Les deux approches présentées dans les Sections 4.1 et 4.2 sont basées sur l'enregistrement en ligne d'informations sur le flot d'exécution. En cas d'une erreur détectée par le processeur ou les assertions matérielles, l'enregistrement est interrompu, conservant de fait, l'historique des dernières instructions exécutées avant une défaillance. Dans le cas d'une erreur capturée par l'IP d'observation, les informations de diagnostic peuvent contenir l'apparition évidente d'un SBU, la propagation d'une faute aboutissant à la divergence du flot d'exécution, la corruption silencieuse de données, etc. Les données de trace sont très difficilement interprétables par un opérateur, en plus de représenter un volume très significatif.

Le but de l'analyse des informations de diagnostic est de remonter à l'emplacement de la faute originelle. Cette approche est basée sur la même idée directrice que celle de la technique présentée dans le Chapitre 2 qui consiste à retrouver l'élément logique ayant été affecté dans le circuit. L'identification de l'élément logique d'où provient la faute permet, lors des qualifications sous radiations d'un processeur, d'évaluer le taux d'erreur de chaque zone. Ceci peut permettre deux choses en lien avec les sujets adressés dans la bibliographie. Premièrement, il serait possible de comparer l'estimation de la criticité des registres du pro-



(a) Vue SoC

(b) Légende

FIGURE 4.6 – Représentation de l'implémentation du SoC de SQUAL

cesseur faite grâce à des campagnes d'injection de fautes [11, 62, 94, 96] avec des résultats d'expérimentations sous faisceau. Deuxièmement, l'efficacité de la triplication partielle [57, 94] pourrait être améliorée en surveillant d'où proviennent les erreurs résiduelles.

Le principe de cette approche repose sur le fait d'analyser et de mapper des traces capturées avec des cas du dictionnaire de fautes, afin de déduire l'origine de la faute. Le dictionnaire de fautes, présenté en détail dans la Section 3.3.3, couvre de manière assez exhaustive, tous les cas d'erreur potentielles, accompagnés des informations de diagnostic disponibles. Ce faisant, il est possible de mapper les informations diagnostic extraites de tests réels avec ceux du dictionnaire. Les points d'injection de fautes, couvrant l'ensemble des registres du processeur, ont été regroupés autant que possible par registres. En d'autres termes, les classes d'intérêt correspondent aux registres du processeur, exceptés pour les signaux internes qui forment des classes indépendantes.

4.3.1 Classification basée sur des algorithmes de machine-learning

Les classes identifiées sont donc les registres du processeur ou des cellules mémoires. Elles représentent donc un sous-ensemble du dictionnaire de fautes correspondant aux cas

d'injection capturés par l'IP. Le but de l'algorithme de classification est de trouver à quelle classe appartient une nouvelle trace capturée en cas d'erreur. Dans l'absolu, une méthode classique pourrait être utilisée pour cette tâche. L'idée aurait été de trouver une métrique de corrélation entre une trace capturée et les différents sous-ensembles du dictionnaire de fautes. La classe ayant le taux de corrélation le plus élevé pourrait être considérée comme étant issue d'une faute dans la même cellule ou registre. Un algorithme de classification par apprentissage automatique adopte finalement la même démarche d'identification de caractéristiques propres à chaque classe afin de proposer une classe d'appartenance à un nouvel élément.

Les algorithmes de classification par machine-learning (ML) sont désormais très accessibles grâce à des bibliothèques Python. Notre cas d'usage correspond d'ailleurs aux cas typiques d'utilisation de ce type d'algorithme de classification avec apprentissage supervisé [36]. De plus, les méthodes d'entrainements sont très optimisées du point de vue de la précision obtenue par le modèle et du temps d'exécution. Les bases de données d'entrainement de SERVAL et SQUAL sont déjà prêtes à être utilisées en l'état. Le modèle peut être entraîné et évalué sur ces bases de données, puis sauvegardé sous forme de fichier. Ce modèle peut ensuite être restauré, puis utilisé pour traiter de manière efficace les données récoltées sur des résultats de test sous faisceau lors du dépouillement.

Le dictionnaire forme une base de données suffisamment conséquente pour entrainer un modèle de ML. La campagne d'injection, décrite en Section 3.3.3, comprend 400 injections de faute par cellule. 47 classes ont été identifiées, ce qui implique qu'une injection de fautes dans les registres ou cellules correspondants résulte en une erreur capturée par l'IP d'observation. Le masquage logiciel affecte les injections de fautes selon le timing et l'emplacement, par conséquent le résultat d'une inversion binaire dans un registre sensible n'est pas déterministe. Ceci explique que le nombre d'événements par classe ne dépend pas du nombre d'injection. Le graphique de la Figure 4.7 détaille les classes en fonction de la zone matérielle et du nombre de cas identifiés pour les plateformes SERVAL et SQUAL.

Au total, la plateforme SERVAL a capturé moins de cas d'erreur que la plateforme SQUAL, 56,455 cas contre 79,966. Cette différence est due à la capacité de détection d'erreur des IPs d'observation décrites dans les Sections 4.1 et 4.2. La distribution du nombre de cas par classe diffère d'une plateforme à l'autre. Les deux algorithmes de benchmark, détaillés dans la Section 3.2.1, en sont la cause. Le CoreMark nécessite plus de ressources mémoires pour être exécuté, notamment plus de registres du fichier de registres (MPRF), ce qui se traduit par une répartition des cas plus étalée sur l'ensemble des classes. La répartition du nombre de cas par classe est également très hétérogène. Ceci est une conséquence de la sensibilité des registres du processeur, certains sont plus enclin à générer des erreurs bloquantes selon leur fonctionnalité. Néanmoins nous verrons dans la Section 5.2.3 que la précision de modèle par classe ne dépend pas du nombre de cas.

4.3.2 Evaluation et choix des modèles

Dans cette étude, nous avons utilisé la bibliothèque Scikit-Learn [87] pour exploiter les algorithme de ML. Cette bibliothèque contient notamment des modèles de classification avec apprentissage supervisé. Parmi ces modèles, les plus enclins à effectuer une classification dans notre contexte, sont les modèles ensemblistes basés sur des méthodes de moyennage. Ce choix se base sur la représentation de la carte des choix du modèle, Figure

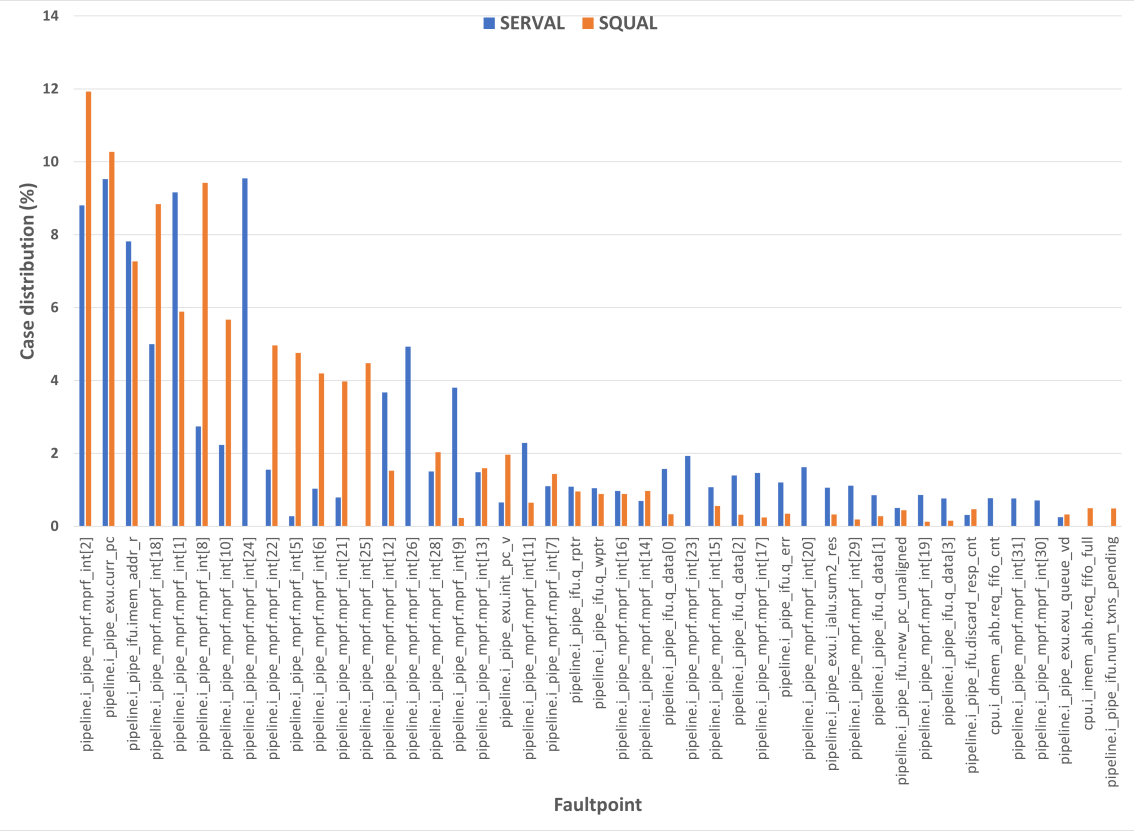


FIGURE 4.7 – Répartition des cas d'erreur par registre après injections de faute

4.8, issue de [87].

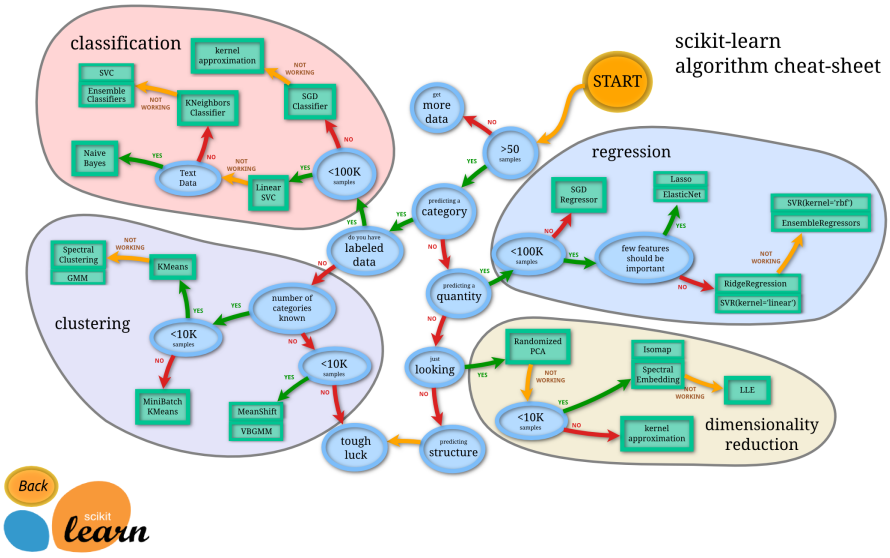


FIGURE 4.8 – Carte des choix du modèle de machine-learning [87]

Le principe de ces modèles est de construire plusieurs estimateurs et de moyenner leurs prédictions. Parmi ce type de modèle plusieurs algorithme de classification sont disponibles, *Bagging meta-estimator* [13, 60], *Forests of randomized trees* [14] et *K-Nearest Neighbors* [41].

Classification model	Platform	
	SERVAL	SQUAL
RandomForestClassifier	84.1 %	82.9 %
ExtraTreesClassifier	82.2 %	82.4 %
BaggingClassifier	85.9 %	86.0 %
KNeighborsClassifier	79.2 %	80.8 %

TABLE 4.2 – Evaluation des modèles de classification avec apprentissage supervisé

Le Tableau 4.2 présente les précisions des modèles obtenues suite aux entraînements sur les bases de données des plateformes SERVAL et SQUAL. Afin de mesurer le taux de précision du modèle, 25 % de la base de données a été utilisée pour évaluer les performances du modèle. Les quatre modèles sont très proches en terme de précision de classification, mais le modèle *BaggingClassifier* est utilisé par la suite pour analyser les données.

La précision obtenue du modèle sur la classification des erreurs dans une base de données par registres d'injection est une métrique très intéressante qui traduit le taux d'observation de l'IP sur le processeur. L'évolution de cette précision en fonction de la diminution d'événements disponibles dans les buffers des deux versions de l'IP d'observation, a permis d'estimer leurs tailles optimales.

4.4 Conclusion

Dans ce chapitre nous avons présentés notre méthode d'observation de la propagation des erreurs dans le processeur en se basant sur la capture de la trace d'exécution. Cette technique a été implémentée sous forme d'une IP intégrée dans un SoC. Deux versions de l'IP ont été développées, et implémentées dans les plateformes SoC des circuits SERVAL et SQUAL.

Cette méthode d'observation se base sur l'enregistrement en continue dans un buffer embarqué de la trace d'exécution complétée par la valeur de certains registres du pipeline. Ces registres ont été choisi grâce à l'étude de la propagation des erreurs dans le processeur au travers des campagnes d'injection de fautes par simulation. Le taux et le délai de propagation d'une erreur à une sonde sur un registre est une approche qui a permis de choisir les points d'observation les plus pertinents. La méthode d'observation repose également sur des mécanismes de détections d'erreurs composés d'assertions matérielles configurables. Le but de ces assertions est de couvrir le plus large spectre possible de modes de défaillance tout en minimisant l'impact sur les ressources. Suite au déclenchement d'une assertion, le bloc de détection d'erreur déclenche l'arrêt de l'enregistrement, conservant ainsi dans le buffer de trace un historique des instructions fautives. Les résultats d'évaluations des taux de couverture et de capture des erreurs par simulation sont présentés dans le Chapitre 5.

Afin d'exploiter au mieux les traces d'exécution collectées par chaque IP lors des expérimentations, un modèle de classification à apprentissage automatique a été entraîné sur une base de données constituée des cas d'erreur issues de la campagne d'injection de fautes par simulation. Ce dictionnaire de fautes est constitué de tous les cas d'erreur ayant provoqués le déclenchement de l'IP composés des points d'injection et des traces d'exécution fournies par l'IP. Les taux de précision obtenus du modèle de classification

sont présentés dans le Chapitre 5.

Le circuit SERVAL a pu être expérimenté sous radiations. Les résultats sont également présentés dans le Chapitre 5 et permettent de valider l'approche en établissant une cohérence entre la zone d'injection et le registre fautif estimé par le modèle de classification.

Chapitre 5

Résultats de simulations et d'expérimentations

Ce chapitre présente les résultats issus des simulations et des expérimentations des deux axes majeurs de la thèse, regroupant la classifieur d'événements et les deux IPs d'observation des processeurs, présentés respectivement dans les Chapitres 2 et 4. Ce chapitre a pour but de mettre en lumière les bénéfices des contributions et d'identifier leurs limites.

Le classifieur d'erreur, implémenté dans deux circuits de tests, a été expérimenté sous faisceau de protons pour le circuit SERVAL, et sous faisceau ions lourds pour le circuit SQUAL. L'identification des événements provoquant de multiples erreurs dans les registres à décalage révèle la pertinence de la classification en ligne compte tenu du mode de fonctionnement du testeur. Des événements survenus au sein de l'IP de classification décèlent les limites de la technique de durcissement des registres.

En ce qui concerne les techniques d'observation des processeurs, les résultats des campagnes d'injection de fautes par simulation font tout d'abord l'objet d'une étude statistique sur la réponse des IPs développées aux cas d'erreur provoqués. Cette approche évalue le taux de couverture des différents modes de défaillance, ainsi que le taux de capture de la propagation des erreurs sur les sondes des registres du processeur. Le modèle de classification par apprentissage automatique permet une analyse plus poussée des données de diagnostic capturées, au regard du dictionnaire de faute généré par la campagne de simulation. Cette analyse est exploitée pour estimer une métrique du taux d'observation et proposer une approche afin d'optimiser la taille des buffers de trace. Enfin, le circuit SERVAL, dans lequel est implémenté la première version de l'IP, a été utilisé pour des test sous impulsions électromagnétiques, sous particules Alpha et sous rayons X focalisés. Les résultats sont présentés et discutés afin d'évaluer la cohérence de l'approche.

5.1 Classification d'événement dans les registres à décalage

Les circuits SERVAL et SQUAL ont été testés avec les particules correspondantes aux milieux dans lesquels ces applications sont supposées évoluer, suivant les spécifications JEDEC [49] et ESCC [30]. Le circuit SERVAL est un circuit de tests d'applications dans le domaine automobile, il aurait dû être testé sous faisceau de neutrons. Cependant, la crise sanitaire a contraint l'accès aux établissement d'irradiation aux neutrons [43, 56, 95]. Nous avons donc utilisé une équivalence protons-neutrons autorisée par la norme JESD89B [49]. Ce circuit a finalement été irradié sous faisceau de protons au Paul Scherrer Institut (PSI) [74]. Le circuit SQUAL a quant à lui été conçu pour qualifier une technologie spatiale GEO, ses tests se sont déroulés sous faisceau d'ions lourds dans l'établissement RADiation Effects Facility (RADEF) [83]. Les résultats des campagnes de tests sont présentés dans les Section 5.1.1 et 5.1.2.

5.1.1 Tests sous protons - Plateforme SERVAL

La campagne de test sous protons a été réalisée selon la procédure décrite dans la Section 2.1.4. Le plan de test inclut des fréquences de fonctionnement de 50, 100 et 200 MHz, ainsi que des tests en température à 30, 135 et 165°C. Les résultats ne permettent pas de mettre en lumière de bénéfice dans la précision de mesure apporté par le module de classification en ligne des événements car très peu d'erreurs ont été collectées. Seul un événement sur toute la campagne, impliquant tous les registres à décalage de plusieurs pièces, est un événement multiple. Le nombre d'événements simples s'élève à 148 sur l'ensemble des conditions de test et des registres à décalage. Les erreurs simples qui arrivent de manière sporadique ont été correctement identifiées comme des SBUs dans les DFFs. Le seul événement multiple de 793 erreurs d'affilées a été identifié comme provenant de l'arbre d'horloge. Néanmoins, le comportement du module de classification a pu être vérifié lors de tests sous protons.

5.1.2 Tests sous ions lourds - Plateforme SQUAL

Le circuit SQUAL a été qualifié sous ions lourds pour un environnement spatial GEO à RADEF [83]. Le plan de test comportait différentes conditions de tension d'alimentation (minimale, maximale et nominale), de tension de polarisation (nulle, maximale et nominale), de fréquence de fonctionnement, et de type de particules pour chacun des 4 registres présentés dans le Tableau 2.2. Nous présentons les résultats de la configuration la plus sensible (tension d'alimentation minimale et tension de polarisation maximale), pour deux fréquences de fonctionnement (25 MHz et 375 MHz) des registres comportant les cellules de référence (SELFF-8T). De nombreux éléments ont été collectés suivant ces conditions, les résultats sont présentés sous forme d'histogrammes selon la largeur des événements regroupés par type de particule et par fréquence de fonctionnement sur les Figures 5.1, 5.2, 5.3, 5.4, 5.5, et 5.6. Chaque événement est classifié soit comme un événement simple correspondant à un SEU dans une bascule, soit comme un événement multiple associé à un SET dans l'arbre d'horloge. La largeur des événements correspond donc au nombre d'inversion binaire constaté grâce à l'échantillonnage de la valeur du compteur d'erreur. Les tests se déroulent sous ions Argon (^{18}Ar), Fer (^{26}Fe), Krypton (^{36}Kr), Neon (^{10}Ne), Oxygen (^{8}O), et Xenon (^{54}Xe), d'après les cocktails d'ions fournis par l'établissement RADEF listés sur

la Figure 2.9.

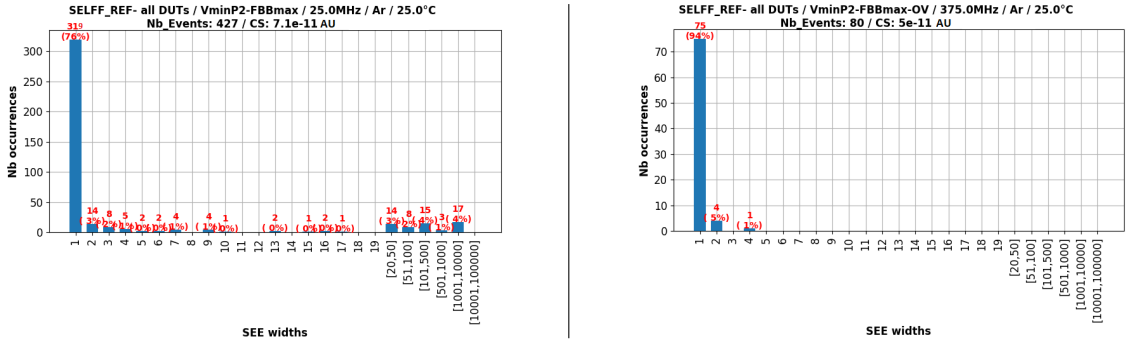


FIGURE 5.1 – Distribution de la largeur des événements lors de tests sous ions ^{18}Ar de SQUAL de l'IP SELFF-8T à 25°C - 25 MHz (à gauche) et 375 MHz (à droite)

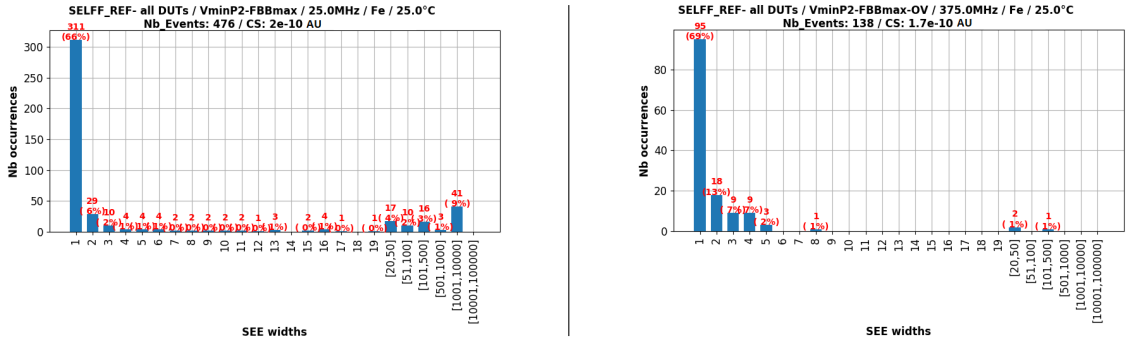


FIGURE 5.2 – Distribution de la largeur des événements lors de tests sous ions ^{26}Fe de SQUAL de l'IP SELFF-8T à 25°C - 25 MHz (à gauche) et 375 MHz (à droite)

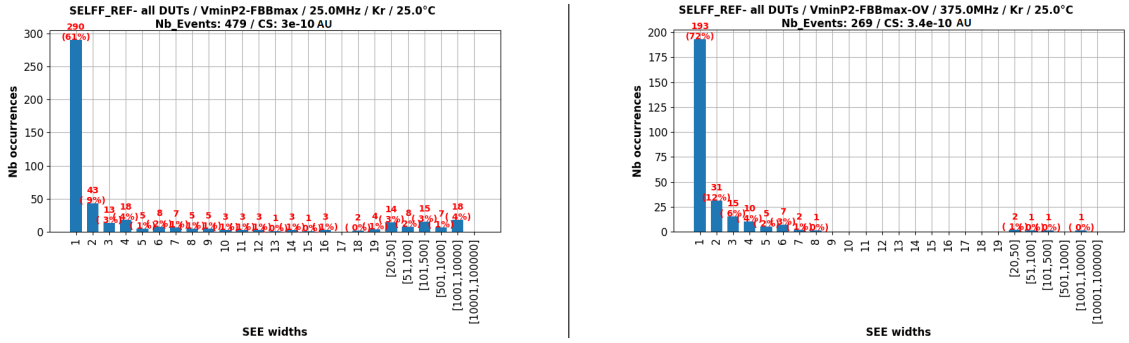


FIGURE 5.3 – Distribution de la largeur des événements lors de tests sous ions ^{36}Kr de SQUAL de l'IP SELFF-8T à 25°C - 25 MHz (à gauche) et 375 MHz (à droite)

On peut voir sur les Figures 5.1, 5.2, 5.3, 5.4, 5.5, et 5.6 que les événements multiples sont nombreux. Sur ces figures, la section efficace a été encodée en unité arbitraire (AU), la fluence n'est pas non plus indiquée. Le Tableau 5.1 récapitule la distribution des événements multiples par rapport aux événements collectés par type de particule selon la fréquence de fonctionnement. En regroupant toutes les erreurs collectées par l'IP SELFF-8T, **41 %** des événements à 25 MHz et **34 %** des événements à 375 MHz sont des événements multiples. En somme, **38 %** des événements collectés sont des événements multiples identifiés par la module de classification.

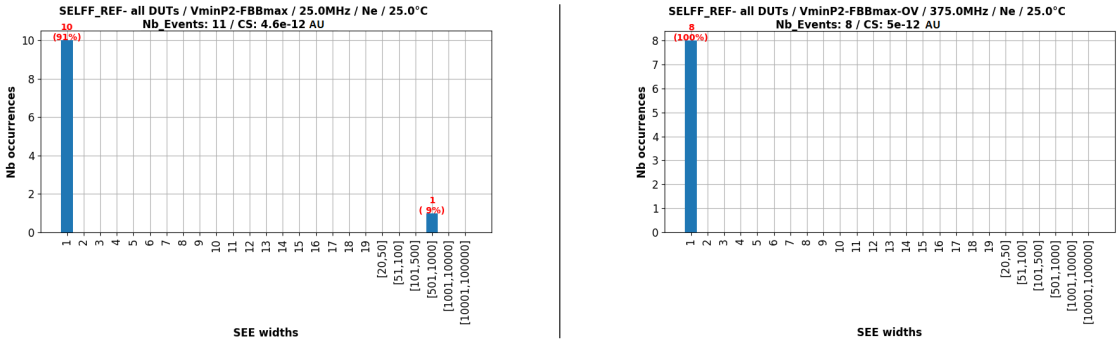


FIGURE 5.4 – Distribution de la largeur des événements lors de tests sous ions ^{10}Ne de SQUAL de l'IP SELFF-8T à 25°C - 25 MHz (à gauche) et 375 MHz (à droite)

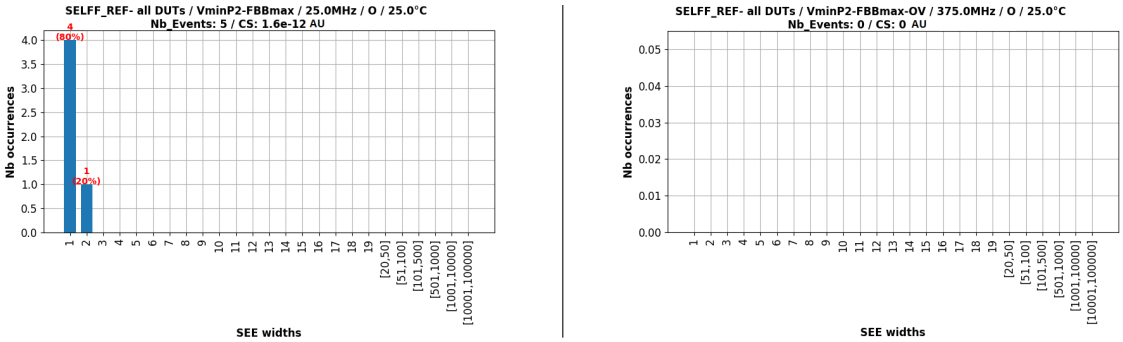


FIGURE 5.5 – Distribution de la largeur des événements lors de tests sous ions ^{8}O de SQUAL de l'IP SELFF-8T à 25°C - 25 MHz (à gauche) et 375 MHz (à droite)

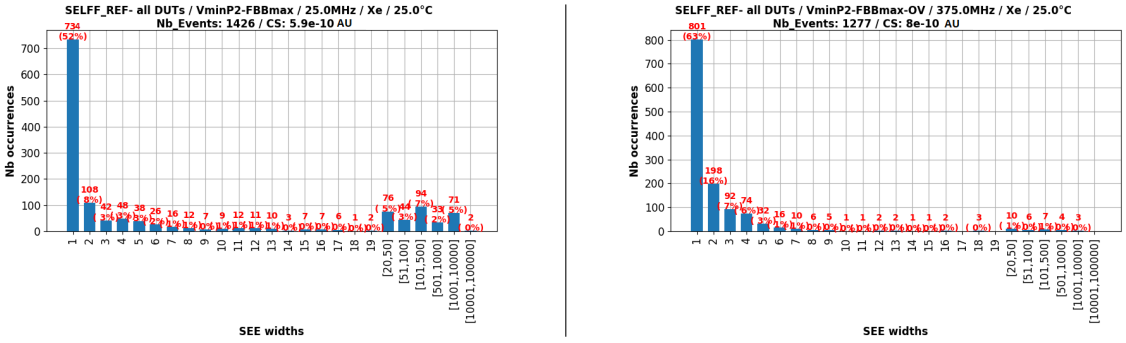


FIGURE 5.6 – Distribution de la largeur des événements lors de tests sous ions ^{54}Xe de SQUAL de l'IP SELFF-8T à 25°C - 25 MHz (à gauche) et 375 MHz (à droite)

5.1.3 Analyse pratique de la technique

Précédemment, dans le contexte industriel de ces travaux [34, 61], pour estimer le nombre d'événements provenant des bascules et des buffers d'horloge, la méthode employée consistait à exploiter les fichiers de journalisation. Ces fichiers sont générés par une routine Python depuis la machine hôte, qui reçoit du testeur tous les incréments du compteur d'erreurs et ajoute une datation. De fait, il est possible d'identifier à postériori les événements simples provenant de SEUs dans les bascules, des événements multiples causés par un SET dans l'arbre d'horloge, grâce à leur datation. Dorénavant, le testeur surveille également l'évolution des compteurs d'erreur de type SEU et SET de l'IP de classification.

Particules	25 MHz	375 MHz
^{18}Ar	25 %	6 %
^{26}Fe	35 %	31 %
^{36}Kr	39 %	28 %
^{10}Ne	9 %	0 %
^{8}O	20 %	-
^{54}Xe	48 %	37 %
TOTAL	41 %	34 %

TABLE 5.1 – Récapitulatif des distributions des événements multiples lors de test sous ions lourds de l'IP SELFF-8T du circuit SQUAL selon les particules et la fréquence

Le problème majeur de la technique précédente est qu'elle reposait uniquement sur la période d'échantillonnage qui est de l'ordre de la milliseconde pour notre testeur. Pour une fréquence de fonctionnement du DUT de 25 MHz, le nombre de coups d'horloge qui sépare deux lectures est de l'ordre de grandeur de la taille du registre, soit plusieurs dizaines de milliers. Les incréments sporadiques simples sont donc considérés comme des événements simples. Alors que les incréments multiples, ou se poursuivant sur deux lectures consécutives, sont considérés comme des événements multiples. Cependant une incertitude apparaît lorsque la fréquence de fonctionnement augmente, allant jusqu'à plusieurs centaines de MHz. Ceci n'est plus le cas grâce à la classification en ligne dont la granularité de la mesure ne varie pas selon la fréquence de fonctionnement. Cette technique permet donc d'effectuer des tests à haute fréquence sans dégrader la précision de la mesure.

Outre cela, les fichiers de journalisation nécessitaient d'être analysés en détails afin d'extraire ce genre de métriques. Cette analyse est faite à postériori de la campagne de test lors du dépouillement des données. La technique précédente ne permettait donc pas d'avoir un aperçu en temps réel de la progression du nombre d'événements. Cependant l'IP de classification offre cet aperçu direct du nombre d'événements survenus dans le registre selon leur provenance. Lors du dépouillement des données, il est possible d'avoir accès à la largeur des événements comme nous avons pu l'exploiter pour les résultats de SQUAL. Cette analyse a aussi été mise en œuvre pour vérifier la cohérence du fonctionnement de l'IP de classification de événements. Nous avons d'ailleurs pu relever, lors des tests sous ions lourds de SQUAL, l'apparition d'un SBU sur un compteur de cette IP, malgré leur implémentation avec des cellules TMR. Cette méthode de durcissement n'a pas suffi à garantir l'intégrité des données durant les tests. La persistance des données dans les bits de poids fort est telle que plusieurs éléments de stockage d'une bascule TMR ont pu être corrompus.

5.1.4 Perspectives d'amélioration

D'après les résultats présentés dans la Section 5.1, trois points peuvent être améliorés. Le premier point fait suite à des failles de fiabilité constaté lors des tests sous protons, discutés Section 5.1.1. Le second point s'adresse au besoin de qualifier d'autres technologie, notamment des technologies pouvant être plus sensibles aux événements multiples dans les DFFs. Enfin, la dernière amélioration consisterait à sauvegarde de l'historique de la largeur des événements classifiés comme des événements multiples. La somme de la largeur des événements et des événements simples doit être égale au nombre total d'événements, ce

qui permet de faire une vérification croisée des résultats.

Amélioration de la fiabilité

Les cellules tripliquées n'ont pas suffi à garantir l'intégrité des données lors des tests. Une inversion binaire a été constatée sur un compteur de SEUs d'une IP. Ceci s'explique par la persistance très importante des données sur ces compteurs. Au fil de l'exposition sous faisceau, les SEUs s'accumulent dans les cellules tripliquées jusqu'à ce que plusieurs éléments de stockage au sein d'une cellule soient en erreur, la donnée est alors perdue.

Pour remédier à ce problème, nous proposons d'utiliser des cellules logiques séquentielles à correction automatique, comme vues dans la bibliographie [54, 55]. Ces cellules sont capables de rafraîchir les éléments de stockage redondant si une divergence est détectée. Si ce type de cellule n'est pas disponible dans le portfolio, une technique visant à rafraîchir de manière périodique chaque registre constituerait une alternative. Une machine à état pourrait déclencher cette procédure toutes les 100 ms à 1 s afin de diminuer artificiellement la persistance des données.

Adaptation à d'autres technologies

L'utilisation d'une autre technologie sensible aux MBUs compromettrait les hypothèses initiales. Pour pallier ce problème nous proposons premièrement de travailler sur l'arbre d'horloge. Celui-ci pourrait être modifié afin de maîtriser le nombre de cellules minimum par buffer d'horloge. Il serait ainsi possible d'augmenter le seuil à la limite du minimum de cellules par buffer d'horloge. Une nouvelle classe MCU pourrait être créée dont le nombre d'événements se situerait entre celui des SEUs et des SETs. Le Tableau 5.2 détaille l'ensemble des classes et les règles de classification, selon le nombre, $K_{FFs/buffer}$, de cellules par buffer d'horloge.

Type d'événement	Condition de classification
SEU	$N_{bit-flips} = 1$
MCU	$1 < N_{bit-flips} < K_{FFs/buffer}$
SET	$K_{FFs/buffer} <= N_{bit-flips}$

TABLE 5.2 – Règle de classification des événements SEU, MCU et SET pour un rafraîchissement complet du registre

Sauvegarde du nombre de fautes par événements

La sauvegarde du nombre de fautes permettrait d'estimer l'étage du buffer d'horloge ayant été touché dans le cas d'un événement SET, ou de connaître le nombre de bits en erreur dans le cas d'un MBU. Evaluer la répartition des étages de l'arbre d'horloge d'apparition des événements SETs permettrait de savoir si les impulsions sont propagées ou atténuées d'un buffer à l'autre. Une étude de la largeur des SETs dans les technologies 28 nm UTBB FDSOI [9] se base sur ce phénomène de propagation. Les auteurs expliquent que des SETs dans les buffers ont été simulés afin de choisir le type de buffer le plus

enclin à propager une impulsion et réaliser leur structure de test. Connaître la largeur des événements multiples permet de retrouver la valeur du compteur total d'erreur et de faire une vérification croisée du résultat. Cela apporterait également des informations sur les signatures des événements sans avoir à échantillonner les données selon l'ancienne méthode. En cas de susceptibilité d'une technologie aux MCUs, il est important de pouvoir fournir une statistique d'apparition.

5.2 Observation des CPUs - Évaluation de l'approche par simulation

L'analyse des résultats des campagnes d'injection de fautes permet d'évaluer l'efficacité des IPs d'observation sur un ensemble exhaustif de cas d'erreurs. Cette évaluation comprend l'estimation des taux de couverture selon les modes de défaillances, et du taux de capture de la propagation des erreurs dans les buffers de trace. Une première approche permet de proposer un compromis sur la taille optimale du buffer en se basant sur la capacité du buffer de trace à couvrir la période de propagation d'une faute jusqu'à l'arrêt de l'enregistrement. Une seconde approche basée sur un modèle de classification par apprentissage automatique exploite les cas d'erreur capturés par l'IP du dictionnaire de fautes pour remonter au registre ayant été affecté par un SEU. La précision globale obtenue donne une estimation du taux d'observation des erreurs dans le processeur. En réduisant artificiellement la profondeur du buffer de trace, l'évolution de la précision globale a été utilisée comme seconde approche pour optimiser la taille du buffer.

5.2.1 Couverture des modes de défaillances

Dans cette section, nous allons évaluer les taux de capture des erreurs des deux IPs d'observation selon les différents modes de défaillance de l'application. Ces résultats sont présentés dans les Tableaux 5.3 pour la plateforme SERVAL et 5.4 pour la plateforme SQUAL. Les campagnes couvrent de manière exhaustive l'ensemble des SEUs potentiels dans les cellules de stockage du cœur, ainsi que toute la durée de l'exécution de l'application via des tirages aléatoires des instants d'injection des fautes. Dans les résultats, apparaissent premièrement les taux de répartition des modes de défaillance identifiés en lien avec les Tableaux 3.8 et 3.4. Les taux de répartition des modes de défaillances sont calculés par rapport au nombre d'exécutions ayant échouées, plus précisément lorsque l'algorithme n'a pas abouti sur un résultat correct. Cette répartition initiale est présentée dans le Tableau 3.8 pour les deux plateformes. Le taux de capture des IPs selon les différents modes de défaillances est le nombre de cas où une des assertions a déclenché l'arrêt de l'enregistrement, par rapport au nombre de cas identifiés correspondants à ce mode de défaillance.

Concernant le Tableau 5.3, les résultats montrent que les modes de défaillance les plus fréquents sont les SDC et les *fault exceptions*. Les SDCs ne sont pas détectables par l'IP dans sa première version. Le résultat erroné est constaté à la fin de l'exécution de l'algorithme, la profondeur du buffer ne couvre pas un temps d'exécution aussi long permettant de capturer des informations du la propagation de la faute, le taux de couverture est donc nul. Les *fault exceptions*, selon les causes listées dans le Tableau 3.2, représentent le deuxième cas d'erreur le plus fréquent. Ce mode de défaillance est capturé par l'IP. Les seuls cas n'ayant pas été couverts sont ceux dus à un saut légal du flot d'exécution directement dans la fonction correspondante à ce type d'erreur. Il en est de même pour les cas non capturés de *breakpoint*. Le vecteur d'exception n'ayant pas été appelé, l'IP n'a pas pu déclencher. Les cas d'interruptions illégales (machine, hyperviseur, superviseur, ou utilisateur) sont très rares, et sont aussi en partie dus à des sauts erronés dans le code. Il en est de même pour la fonction *exit* car elle n'est pas implémentée dans l'algorithme. On pourrait envisager d'appeler cette fonction dans le cas de violation de propriétés du code, comme des dépassements d'index, des états de FSM non prévus, ou encore des divisions par zéro, afin d'augmenter le taux de couverture des cas de SDC. Les cas de reset sont

causés par des SEUs dans les bascules de synchronisation du signal de reset ou celles du signal de remise à zéro de la valeur du PC. Ces cas sont détectés par le logiciel au moment du boot. Enfin, les *timeout* sont des cas difficiles à appréhender car ils sont souvent dus à un saut légal dans le code, mais dans une zone du programme où les variables n'ont pas été correctement initialisées. Dans certaines situations, il en résulte que l'exécution peut rester bloquée dans une boucle un temps indéterminé. Parmi les 10 % de *timeout* capturés, nous avons remarqué quelques cas où une exception a été levée, a fait déclencher l'IP, mais la fonction d'erreur n'a pas eu le temps de terminer son exécution et de mettre fin à la simulation avant la limite de délai d'exécution. Une telle situation constitue un artéfact de simulation qui ne se serait pas produit lors de tests sur carte.

Run issue	**Distribution**	**Coverage**
SDC	40.1 %	0 %
Breakpoint	2.94 %	99.6 %
Fault	31.7 %	99.2 %
Interrupt	0.04 %	0 %
HSU_Trap	0.11 %	95.7 %
EXIT_Trap	0.40 %	97.6 %
Reset_Trap	3.61 %	99.9 %
Timeout	21.0 %	13.0 %

TABLE 5.3 – Distribution des modes de défaillance et taux de couverture de la plateforme SERVAL

Concernant le Tableau 5.4, on retrouve des taux similaires de répartition des cas de défaillance que celui présenté dans le Tableau 5.3. Ce taux est aussi calculé par rapport au nombre d'exécution ayant échouées, présenté dans le Tableau 3.8. En plus du taux de couverture global de l'IP par mode de défaillance, ont été détaillés les taux de couverture de chaque assertion. On retrouve donc le taux de couverture cumulé des *watchpoints*, des *watchpoint counters*, ainsi que celui de l'ensemble des *PC range checkers* et des *Data range checkers*.

Certaines assertions ont des couvertures qui se chevauchent, en particulier les *watchpoints* placés sur les vecteurs d'exception et les *PC range checkers* qui considèrent les exceptions, interruptions et fonctions d'erreur en zone d'exclusion. Les *watchpoints* permettent de savoir si l'exception a été accédée suite à une erreur détectée par le processeur, ou suite à un saut corrompu mais légal. La somme des taux de couverture des exceptions ne rend donc pas compte du taux de couverture global de l'IP par cas d'erreur.

Les bénéfices de cette IP pour la partie détection d'erreur sont que deux cas d'erreur très représentés, non traités par la première version, sont adressés. Il s'agit des cas de SDC et de *timeout*. Les cas de *timeout* sont couvert à 80.6 %, tandis que les SDCs ne sont couvert qu'à 11.4 %. Alors que les *PC range checkers* sont les assertions qui fournissent le taux de couverture le plus élevé, sur les autres modes de défaillance, les deux derniers cités sont capturés en plus grand nombre par les *watchpoint counters* et les *Data range checkers*.

Ce comportement confirme l'hypothèse que les cas de *timeout* sont majoritairement dus à des sauts dans des zone des boucles du codes où les données ne sont pas initialisées. Ce faisant, *watchpoint counters* sont aptes à capturer ce genre d'erreur puisqu'ils sont placés dans des boucles de l'algorithme FFT et comptent le nombre de passages de l'exécution. Ils constituent les assertions les plus proches du code dans le bloc de détection d'erreur. Ces

Run issue	Distribution	Coverage	WPT	WCNT	PC-RK	D-RK
SDC	37.5 %	11.4 %	0 %	68.2 %	12.2 %	25.2 %
Breakpoint	6.07 %	98.7 %	100 %	0.10 %	100 %	100%
Fault	26.9 %	99.9 %	99.5 %	4.11 %	100 %	99.7 %
HSU_Trap	0.12 %	100 %	0 %	0 %	100 %	0 %
EXIT_Trap	0.79 %	62.2 %	35.1 %	64.3 %	37.4 %	4.68 %
Reset_Trap	4.30 %	3.54 %	90.6 %	0 %	96.2 %	7.55%
Timeout	24.4 %	80.6 %	16.8 %	75.0 %	25.5 %	34.5 %

TABLE 5.4 – Distribution des modes de défaillance et taux de couverture de la plateforme SQUAL

assertions ont pu également capturer des cas de SDC. Le Tableau 5.5 quant à lui récapitule les métriques de capture de chaque assertions en ne considérant que celle qui ont déclenché le *Trigger*, c'est à dire celle qui ont détecté l'erreur en premier. La somme des pourcentages d'une ligne vaut dans ce cas 100 %.

Run issue	WCNT	PC-RK	D-RK
SDC	68.1 %	6.74 %	25.2 %
Breakpoint	0.10 %	99.1 %	0.80 %
Fault	4.10 %	49.6 %	46.3 %
HSU_Trap	0 %	100 %	0 %
EXIT_Trap	63.2 %	33.3 %	3.51 %
Reset_Trap	0 %	96.2 %	3.77 %
Timeout	67.0 %	9.04 %	24.0 %

TABLE 5.5 – Taux de capture des assertions

Dans le Tableau 5.5, le bénéfice des *watchpoint counters* apparaît clairement. Les erreurs de type SDC et *timeout* sont en effet majoritairement capturées par ce type d'assertion. La surveillance des zones mémoires, via les *Data range checkers*, des accès en lecture/écriture du processeur est complémentaire avec celle des *watchpoint counters* pour les SDCs et les *timeout*. Les *Data range checkers* permettent également de capturer des *fault exceptions* avant même les *PC range checkers*. Effectivement, les requêtes d'accès erratiques de données (*load*/*store*) sont capturées avant que le *PC range checkers* ne capture l'accès à l'exception de faute. Les *breakpoints* sont quant à eux dus à l'exécution d'une ECI en mémoire hors de la zone de code ou dans un piège. L'exception est capturée par le *PC range checkers* si l'ECI se situe dans une section de *deadcode* au sein de l'application. Sinon le fait que le PC se positionne en dehors d'une zone correspondante à l'application déclenche les *PC range checkers* avant même la levée de l'exception.

Pour terminer, l'évaluation des mécanismes de détection montre que la première version de l'IP assure une couverture globale des erreurs de **41.2 %**, et la seconde version de **57.5 %**.

5.2.2 Capture de la propagation

Selon les cas d'erreur, en plus du taux de couverture des assertions, le délai de déclenchement de l'IP pour arrêter l'enregistrement a une importance quant aux données qui

sont capturées dans le buffer. Au regard de la capture de la propagation d'une faute, le fait que l'enregistrement couvre toute la période, depuis l'apparition d'une faute sur l'une des sondes, jusqu'à son arrêt, garantit la pertinence des données sauvegardées. Le contenu du buffer n'est dans ce cas pas écrasé avec des données issues d'un comportement erratique de l'application, ce qui avait été constaté dans [91]. Nous proposons donc d'évaluer ce taux de capture en considérant tous les cas d'erreur couverts par les IPs dans les campagnes d'injection de fautes respectives. Une erreur est considérée comme capturée par le buffer si celui-ci enregistre une trace contenant des divergences par rapport à la référence, c'est-à-dire que des erreurs se sont propagées jusqu'aux sondes du buffer. La deuxième et dernière condition est que la première divergence enregistrée sur une des sondes du buffer n'ait pas été écrasée par de nouveaux événements. Les Figures 5.7 et 5.7 illustrent les taux de capture de la propagation des erreurs par mode de défaillance selon un nombre décroissant d'événements disponibles dans le buffer des deux plateformes. L'allure des courbes suggère une optimisation de la taille du buffer, notamment pour l'IP de la plateforme SQUAL qui possède une mémoire assez conséquente. La taille optimale des buffers est de nouveau discutée dans le paragraphe suivant depuis une autre approche. Seuls les modes de défaillance ayant un taux de distribution et de couverture significatifs sont représentés. Il s'agit des *breakpoint*, *fault exception*, *timeout* pour la plateforme SERVAL, ainsi que des SDCs pour la plateforme SQUAL.

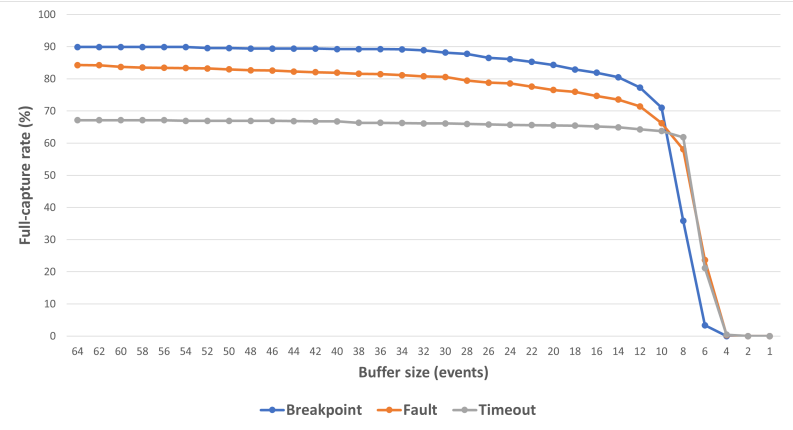
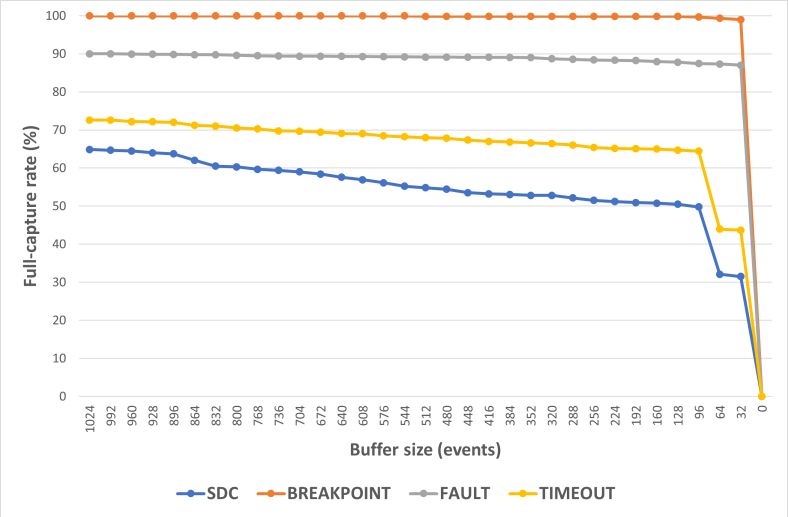


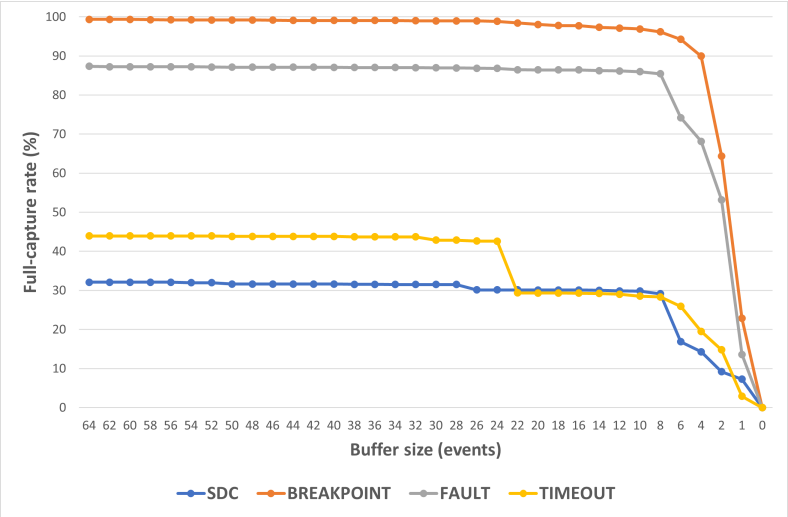
FIGURE 5.7 – Taux de capture complète de la propagation sur plateforme SERVAL

La profondeur du buffer de la plateforme SERVAL est de 64 événements. L'IP a donc des taux de capture complète de propagation de fautes de 89.9 % pour les *breakpoint*, 84.3 % pour les *fault exceptions* et 67.1 % pour les *timeout*. C'est en diminuant le nombre d'événements disponibles dans le buffer que sont obtenues les courbes représentées sur la Figure 5.7. On peut voir que les taux de couverture diminuent jusqu'à un point d'inflexion situé à environ 16 événements sauvegardés. En deçà de ce point, les taux de couverture chutent de manière exponentielle. En conservant 95 % du taux de capture globale avec le buffer initial, seulement 24 événements seraient nécessaires contre 64. Ce qui représente un gain mémoire de 62.5 % pour une perte de 5 % des capacités de capture de l'IP.

La profondeur du buffer da la plateforme SQUAL est de 1024 événements. La génération d'événements de trace est basée sur la trace d'exécution du processeur, plutôt que sur les changements dans les valeurs des registres comme pour l'IP dans sa première version. Les taux de capture complète de propagation de fautes sont de 100 % pour les *breakpoint*, 90.0 % pour les *fault exceptions*, 72.6 % pour les *timeouts* et 64.9 % pour les SDC. De même que précédemment, on observe des points d'inflexion à partir desquels les taux de



(a) Scale : 1024 to 0 events



(b) Scale : 64 to 0 events

FIGURE 5.8 – Taux de capture complète de la propagation sur plateforme SQUAL

capture complète chute de manière exponentielle. Un premier point est situé aux alentours des 96 événements pour les cas d'erreur SDC et *timeout*, et un second aux alentours de 10 événements pour les *breakpoints* et les *fault exceptions*. Les taux de capture sont donc très hétérogènes selon les modes de défaillance. En se plaçant à 95 % du taux de capture global de cette IP, 448 événements suffisent dans le buffer. On obtient donc un gain mémoire de 56.3 % pour une diminution de seulement 5 % des performances de capture globale de la propagation.

Si on considère un buffer de 64 événements pour cette seconde version de l'IP, les taux restent supérieurs à ceux de la première version, avec 93 % de capture complète des *breakpoints*, 87.3 % pour les *fault exceptions*, 44.0 % pour les *timeouts* et 32.1 % pour les SDCs. Grâce à la profondeur accrue du buffer et à la méthode de génération des événements de trace, la nouvelle IP d'observation fournit de meilleurs résultats de capture et de couverture des modes de défaillance. Cependant, l'analyse des données récoltées par le buffer, présentée dans la Section 5.2.3, révèle qu'une telle profondeur n'est pas nécessaire

dans le cas des deux versions de l'IP.

Pour résumer, le taux de capture complète des erreurs détectées avec la configuration actuelle est de **75.1 %** pour la première version de l'IP d'observation et de **82.0 %** pour la seconde version.

5.2.3 Analyse approfondie de la trace

Comme détaillé dans la Section 4.3.2, l'apprentissage du model de classification a été effectué sur le dictionnaire de fautes généré depuis les données des campagnes de simulation. Pour rappel, les classes correspondent aux points d'injection regroupés par registres. Dans cette partie, tous les modes de défaillance supportés par les IPs sont pris en compte.

Précision globale

Le but de ce modèle est d'estimer de quel registre ou cellule provient la faute. On peut donc considérer que la précision moyenne de ce modèle sur les différentes classes correspond au taux d'observation que fournit l'IP sur le flot d'exécution du processeur. Les précisions moyennes obtenues sont très proches entre les deux IPs d'observation, **85.9 %** pour la plateforme SERVAL et **86.0 %** pour la plateforme SQUAL, comme le présente le Tableau 4.2.

Ce qui signifie que sur 41.2 % et 57.5 % des cas d'erreur, l'ensemble module d'observation et modèle de classification est capable de retrouver le registre à l'origine de la faute pour 85.9 % et 86.0 % des erreurs respectivement pour les plateformes SoC de SERVAL et SQUAL.

Optimisation mémoire des buffers de trace

Sur les Figures 5.9 et 5.10 sont représentés les taux de précision global du modèle de classification *BaggingClassifier* en fonction du nombre d'événements dans le buffer de trace pour les deux versions de l'IP. Le but est de constater l'évolution du taux de précision lorsqu'on diminue le nombre d'événements dans le buffer, ce qui revient à diminuer la profondeur d'enregistrement qui est fournie au modèle de classification.

Les Figures 5.9 et 5.10 montrent qu'un très faible nombre d'événements suffisent au modèle afin de classifier les cas d'erreur par registre affecté. Pour la plateforme SERVAL, seulement 16 événements dans le buffer suffisent à obtenir 99 % de la précision globale. Il en est de même pour le buffer de la plateforme SQUAL. Donc en tolérant une diminution de 1 % de la précision globale on obtient un gain mémoire de 75 % et 98.4 %, pour finalement obtenir 81.4 % et 85.2 % de précision respectivement sur les plateformes SoC de SERVAL et SQUAL.

Une profondeur de buffer importante permet donc de capturer la propagation initiale de la faute ayant provoquée une erreur. L'aperçu de cet historique permet à un opérateur d'avoir plus de détails pour comprendre l'événement qui s'est produit. Cependant, Les 16

derniers événements de trace capturés suffisent à ce que le modèle de classification trouve toutes les informations nécessaires à l'interprétation de la trace pour fournir un taux de précision quasiment maximal.

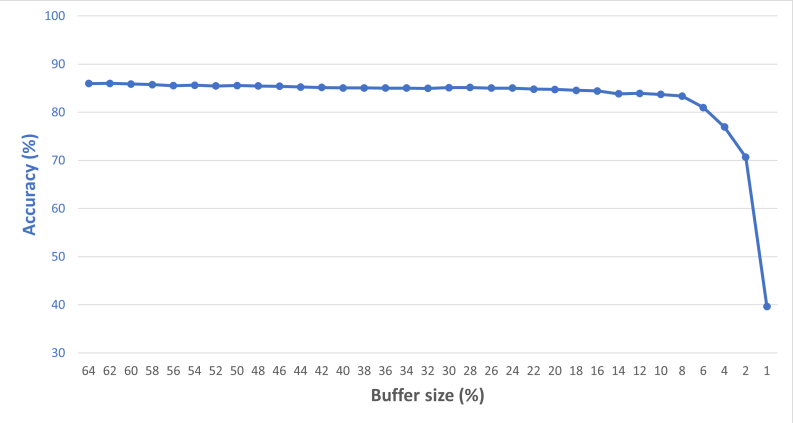


FIGURE 5.9 – Estimation de la taille optimale du buffer de la plateforme SERVAL

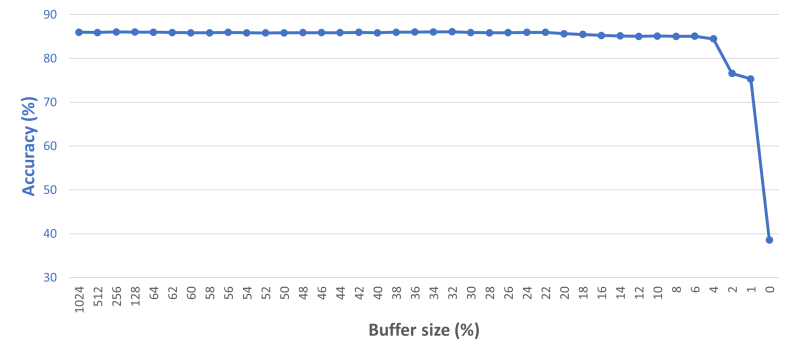


FIGURE 5.10 – Estimation de la taille optimale du buffer de la plateforme SQUAL

Répartition de des cas et de la précision par classe

La précision n'est pas constante sur l'ensemble des classes. La signature des différentes fautes injectées n'est donc parfaitement identifiable depuis les points d'observation. Ces motifs de propagations dépendent également du comportement de l'application au moment où l'événement survient, modulant ainsi le taux de masquage logiciel.

Les Figure 5.11 et 5.12 illustrent le taux de répartition des cas de l'entrainement (barres verticales bleues) ainsi que la précision obtenue par classe (courbe orange) pour les deux plateformes. On remarque sur les deux figures que le nombre de cas par classe n'est pas un paramètre prépondérant de la précision.

La Figure 5.11 illustre que les données de la plateforme SERVAL permettent d'obtenir un taux de précision plus homogène sur l'ensemble des classes par rapport à la plateforme SQUAL, Figure 5.12. On remarque notamment sur cette dernière figure que la confusion entre les deux registres *pipeline.i_pipe_exu.curr_pc* et *pipeline.i_pipe_ifu.imem_addr_r*, qui pourtant sont représentés par un grand nombre de cas, fait grandement baisser la précision. Cette approche peut être complémentaire à celle basé sur les statistiques de

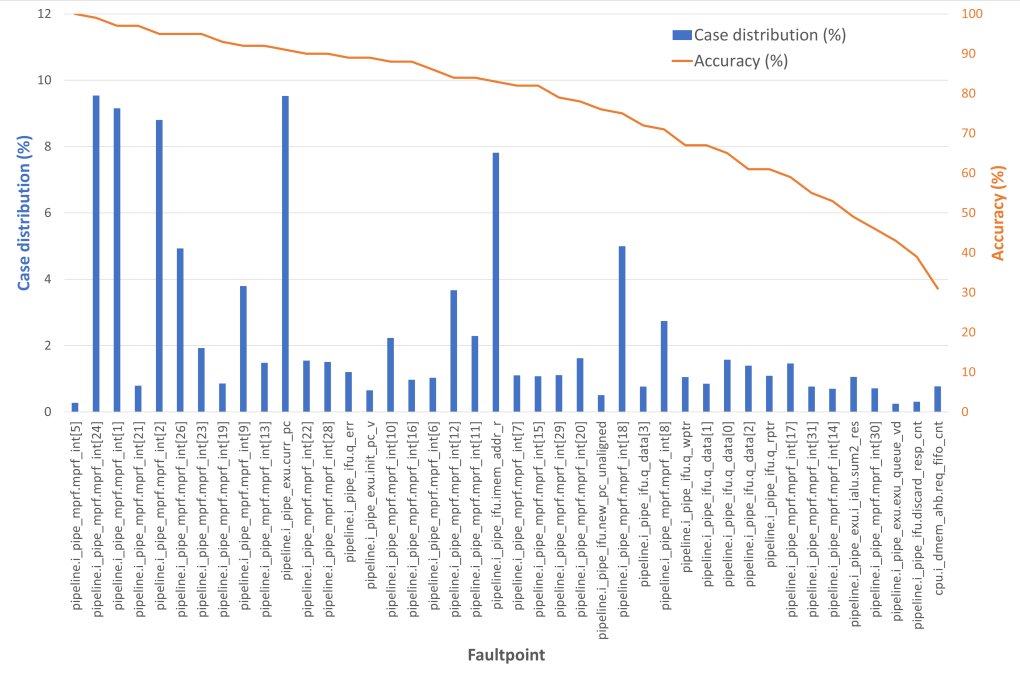


FIGURE 5.11 – Répartition des cas d'erreur par classe et précision du modèle de classification pour la plateforme SERVAL

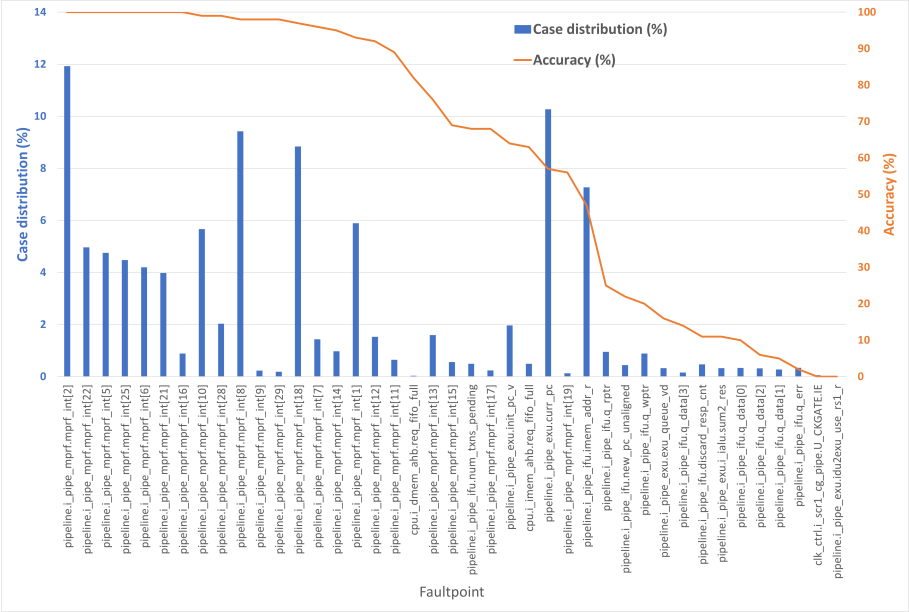


FIGURE 5.12 – Répartition des cas d'erreur par classe et précision du modèle de classification pour la plateforme SQUAL

propagation, présentée dans les Section 4.1.3 et 4.2.3, pour établir le choix des sondes dans le cœur.

5.3 Expérimentations des plateformes SoCs

Dans cette Section, seuls des résultats concernant la plateforme SERVAL sont présentés. Aucune expérimentation sous faisceau de neutrons ou d'ions lourds n'a pu avoir lieu à ce jour afin de mettre en œuvre les SoCs. Seuls les SoCs la plateforme SERVAL ont pu être expérimentés sous impulsions électromagnétique (EMFI), sous particules Alpha, et sous impulsion de rayons X, dans le but de valider la cohérence de l'approche présenté dans la Section 4.1.3.

5.3.1 Injection de fautes par impulsions électromagnétiques

Une session de tests a été réalisée sur la plateforme SERVAL afin de vérifier le comportement de l'ensemble de l'environnement, comprenant la carte fille avec le circuit de test sur son support, la carte mère basée sur FPGA, le script de supervision sur la machine hôte, ainsi que l'application exécutée par le SoC. Le support d'opérabilité du circuit est décrit dans le Chapitre 2, et l'environnement des plateformes SoC fait l'objet du Chapitre 3. L'architecture de l'installation est détaillée sur la Figure 2.3.

Cette session de test a donc eu pour but de valider le comportement de l'application dans tous les modes de défaillance. De nombreux cas d'erreur ont été provoqués pour essayer de couvrir la totalité des modes de défaillance. Pour obtenir de l'appareil générant les impulsions électromagnétiques des effets qui puissent être exploitables, il a fallu procéder à une phase d'étalonnage. Les étapes sont décrites par la suite.

La Figure 2.5 illustre l'installation et le positionnement du ChipSHOUTER pour injecter des erreurs dans le circuit monté sur la carte. La sonde est directement positionnée contre le composant à tester, ce qui supprime un paramètre de positionnement qui aurait été difficile à maîtriser compte tenu de l'installation, et permet une diffusion plus concentrée du flux magnétique sur le circuit. Différentes sondes basées sur des noyaux de ferrite de diamètre 1 mm et 4mm sont disponibles, on les remarque en haut à droite de la photo. Il a été constaté lors de ces tests que la sonde de diamètre 4 mm émet un flux magnétique plus intense et plus diffus. Cette perturbation ne produisait quasiment que des reset. Nous avons donc préféré la sonde de diamètre 1 mm, même si celle-ci était plus difficile à positionner. Sans un bon positionnement, aucun effet n'est produit sur le circuit.

Pour cette expérience, nous avons donc essayé de positionner la sonde en face d'un des deux SoCs à tester, grâce à la vue GDS du circuit Figure 2.11, et au marqueur de la broche n°1 du circuit. La tension du ChipSHOUTER a été configurée au maximum soit 500 V, et la largeur d'impulsion est restée sur une valeur médiane par défaut de 80 ns. Nous avons donc procédé au scan du circuit afin de trouver une zone sensible et de pouvoir injecter des erreurs. Une fois le positionnement effectué, nous avons cherché une tension minimale à laquelle le circuit répondait toujours aux impulsions pour essayer de provoquer le minimum de perturbations, mais tout en restant efficace. Le but est d'obtenir un effet proche de celui d'un SEU. Une fois le réglage effectué, on obtient un cas d'erreur toutes les 5 à 10 impulsions environ.

Lors de ces expérimentations, les deux SoCs du SERVAL ont été testés. Ils étaient alimentés à la tension nominale de 0.9 V et fonctionnaient à une fréquence de 400 MHz.

Les résultats obtenus sont résumés dans le Tableau 5.6. Les pourcentages dans les cases correspondent à la répartition du nombre de cas par rapport au nombre d'événements. Les erreurs mémoires (ECC_DE) ont été exclues de ce compte, dans l'optique de pouvoir comparer la répartition obtenue avec celles de la campagne de simulation.

Run issue	**STD SoC**	**RAD SoC**
SDC	63 (45.3 %)	31 (23.1 %)
Breakpoint	14 (10.1 %)	19 (14.2 %)
Fault	46 (33.1 %)	50 (37.3 %)
Interrupt	0	0
HSU_Trap	1 (0.72 %)	17 (12.7 %)
EXIT_Trap	2 (1.44 %)	0
Reset_Trap	0	0
Timeout	13 (9.35 %)	17 (12.7 %)
ECC_DE	29	8
Total	168	142

TABLE 5.6 – Résultats ChipSOUTER sur le SoCs de SERVAL @(0.9 V et 400 MHz)

Le Tableau 5.6 montre que tous les cas d'erreur ont été couverts à l'exception des interruptions illégales. D'autres interruptions et exceptions machines ont eu lieu comme les *fault exceptions* et les *breakpoints*. Un nouveau cas d'erreur est apparu dans ce tableau, celui des doubles erreurs mémoire (ECC_DE), la seule interruption légale. Par rapport aux résultats du Tableau 5.3, les taux de répartition des SDCs et des *fault exceptions* sont assez similaires. Le processeur RAD quant à lui présente moins de SDCs mais plus de *breakpoints*, de *fault exceptions*, et d'interruptions illégales (HSU_Trap).

Faultpoint	**Cases**	**Accuracy**	**Hardened**	**STD**	**RAD**	**TOTAL**
pipeline.i_pipe_exu.curr_pc	5377	0.91	0	40	40	80
pipeline.i_pipe_ifu.imem_addr_r	4411	0.83	1	4	11	15
pipeline.i_pipe_ifu.q_data[1]	483	0.69	0	4	1	5
pipeline.i_pipe_ifu.q_rptr	615	0.61	0	1	3	4
pipeline.i_pipe_mprf.mprf_int[12]	2073	0.84	0	2	1	3
pipeline.i_pipe_ifu.q_data[0]	888	0.66	0	1	2	3
pipeline.i_pipe_ifu.q_data[3]	432	0.69	0	1	2	3
pipeline.i_pipe_mprf.mprf_int[1]	5170	0.97	1	1	1	2
pipeline.i_pipe_exu.i_ialu.sum2_res	599	0.51	0	1	1	2
pipeline.i_pipe_mprf.mprf_int[19]	486	0.93	1	1	1	2
pipeline.i_pipe_exu.idu2exu_use_rs1_r	8	-	0	2	0	2
pipeline.i_pipe_mprf.mprf_int[24]	5387	0.99	1	0	1	1
pipeline.i_pipe_mprf.mprf_int[2]	4971	0.95	1	0	1	1
pipeline.i_pipe_mprf.mprf_int[8]	1548	0.71	0	1	0	1
pipeline.i_pipe_mprf.mprf_int[11]	1292	0.85	0	1	0	1
pipeline.i_pipe_mprf.mprf_int[20]	914	0.78	1	1	0	1
pipeline.i_pipe_ifu.q_data[2]	789	0.61	0	0	1	1
pipeline.i_pipe_ifu.q_wptr	592	0.64	0	0	1	1
pipeline.i_pipe_mprf.mprf_int[14]	394	0.54	0	0	1	1
pipeline.i_pipe_ifu.discard_resp_cnt	175	0.44	0	0	1	1

TABLE 5.7 – Résultats d'inférence du modèle de classification lors des tests avec le ChipSHOUTER présentés dans le Tableau 5.6

Le Tableau 5.7 présente les résultats après l'analyse du modèle de classification. Ce tableau récapitule l'ensemble des classes, le nombre de cas utilisés pour l'entraînement du modèle, la précision du modèle par classe, l'implémentation du registre, le nombre de fautes par classe pour le processeur STD, pour le RAD ainsi que le nombre total d'événements. La

colonne correspondante à l'implémentation (*hardened*) indique si le registre concerné a été durci par TMR dans le cœur RAD. On remarque que le registre qui apparaît le plus est le textitpipeline.i_pipe_exu.curr_pc, ce registre n'a pas été tripliqué dans la version RAD, la répartition est équivalente entre les deux cœurs, ce qui est donc cohérent. Concernant les registres suivants, on voit que malgré la triplication, des erreurs continuent à être détectées. Il est impossible de tirer de conclusion à ce propos, le comportement physique des perturbations générées par l'impulsion électromagnétique du ChipSHOUTER est très loin de celui des perturbations liées aux fautes induites par les radiations. C'est d'ailleurs pour cette raison que nous avons procédé à d'autres type de tests, dont les résultats sont présentés par la suite.

5.3.2 Validation sous particules alpha

Des tests sous particules Alpha ont été orchestrés selon la procédure décrite dans la Section 2.1.4. Ces tests ont été réalisés par nos soins dans le laboratoire de l'entreprise, le cumul d'heure d'exposition a donc pu être ajusté au besoin. Cependant, les conditions de tests sont très défavorables à l'obtention d'un taux d'erreur suffisant pour collecter des événements. En effet, la technologie 28 nm UTBB FD-SOI est durcie par conception, elle est donc très résistante aux SEEs. De plus, les particules Alpha ne pénètrent que très peu dans la matière, raison pour laquelle il est nécessaire de retirer le boîtier du composant et de poser la source contre la *die* du côté de la surface active pour obtenir des événements. Le taux d'erreur reste malgré tout très faible, la tension est réduite à la tension minimale de fonctionnement afin de voir apparaître des événements. Dans ce cas, les tableaux mémoires, à savoir ceux utilisés pour les SRAMs des SoC, collectent régulièrement des erreurs qui sont corrigées par le processeur dès leur détection. C'est pourquoi la scrutation périodique de la mémoire est nécessaire.

Afin de pouvoir observer le maximum d'erreurs lors de ces tests, la tension d'alimentation du composant utilisée est la plus basse possible [50]. Il est aussi nécessaire de diminuer la fréquence à 50 MHz pour obtenir un fonctionnement stable. La tension minimale est trouvée de manière empirique, on démarre l'application à la tension nominale de 0.9 V, puis on diminue la tension jusqu'à l'arrêt de celle-ci, le seuil se trouve aux alentours de 0.55 V. On a donc choisi une tension minimale avec une certaine marge afin de garantir la stabilité, malgré la température qui peut varier durant l'expérience. Ce point de fonctionnement est hors des spécifications, d'où la nécessité d'utiliser une méthode empirique, en plus du fait que cette tension minimale peut varier selon la dispersion. Malgré tout, très peu d'événements ont été observés dans le processeur.

Deux sessions de tests ont été réalisées. La première a été effectuée assez tôt durant ces travaux, le logiciel manquait de maturité. L'épreuve du ChipSHOUTER ne fait pas ressortir d'erreur mémoire simple en raison de l'ampleur de la perturbation générée. La correction d'erreur en mémoire n'était pas encore opérationnelle, la majorité des cas récoltés durant la session 1, dont les résultats sont présentés dans le Tableau 5.8, sont donc dus à des corruptions de données en mémoire. La deuxième session de tests sous particules Alpha a été réalisée après la validation de la plateforme sous impulsions électromagnétique. La correction d'erreur étant fonctionnelle, nous avons pu constater qu'environ 1 à 5 erreurs mémoires étaient détectées par seconde. La scrutation mémoire permet de corriger les cases en erreur de manière exhaustive et périodique, ce qui garantit l'intégrité des données durant les tests. Cependant, la période de temps que le processeur passe à corriger les

erreurs devient relativement significative par rapport à l'exécution de l'algorithme. Les résultats de la session 2 sont présentés dans le Tableau 5.9.

IP	**STD**	**RAD**	**TOTAL**
VDD (V)	0.6	0.6	-
FBB (V)	0.9	0.9	-
F (MHz)	50	50	-
BeamTime	31h	25h	56h
Events	190	144	334
SDC	67 (35.3 %)	37 (25.7 %)	104 (31.1 %)
Breakpoint	2 (1.05 %)	4 (2.78 %)	6 (1.80 %)
Fault	86 (45.3 %)	92 (63.9 %)	178 (53.9 %)
Timeout	35 (18.4 %)	11 (7.64 %)	46 (13.8 %)

TABLE 5.8 – Résultats Alpha sur le SoCs de SERVAL @(0.6 V et 50 MHz) - Session 1 (ECC non-opérationel)

Le Tableau 5.8 présente les résultats de test sous Alpha des SoCs du SERVAL, mais sans vérification, ni correction des erreurs en mémoire. Les cas d'erreur sont donc quasiment tous dues à des instructions ou des données corrompues en mémoire. Toutefois, l'intégrité des données n'ayant pas été garanti durant l'exposition, les hypothèses ne sont donc pas vérifiées, l'analyse automatique de la trace n'a donc pas de pertinence sur cette session. Par ailleurs, aucune interruption n'a été détectée, ce genre d'événement ne peut résulter de données corrompues en mémoire.

IP	**STD**	**RAD**
VDD (V)	0.55	0.55
FBB (V)	0.9	0.9
F (MHz)	50	50
BeamTime	16h	3h
Events	17	0
SDC	2 (11.8 %)	0
Breakpoint	0	0
Fault	6 (35.3 %)	0
Interrupt	1 (5.88 %)	0
Timeout	8 (47.1 %)	0

TABLE 5.9 – Résultats Alpha sur le SoCs de SERVAL @(0.60 V et 50 MHz) - Session 2

Le Tableau 5.9 présente les résultats de la deuxième session de tests sous particules Alpha. Durant cette dernière session, très peu d'événements suite à des fautes dans le processeur ont été récoltés. Les nombre de cas d'erreur obtenus ne permet pas d'étudier la statistique de répartition. Aucun événement n'a été détecté dans le processeur RAD en 3 heures d'exposition et seulement 1 événement par heures est obtenu sur le processeur STD. Nous proposons donc d'analyser en détails chaque cas d'erreur capturés par l'IP listé dans le Tableau 5.10 qui contient le diagnostic depuis les informations du logiciel et de la trace d'exécution. Chaque cas correspond à une exception ou une interruption, les conditions sont donc favorables à ce que l'origine de l'erreur ait pu être capturée dans le buffer. Cependant, après analyse des traces, toutes les erreurs se sont produites dans l'exécution du *handler* d'interruption, la trace d'exécution n'est donc pas pertinente. En effet, lorsqu'une erreur est détectée, une interruption est levée, ce qui arrête l'enregistrement de la trace. Le *handler*

identifie la raison de l'interruption, corrige l'erreur en mémoire, redémarre l'enregistrement de la trace et arme de nouveau le *watchpoint*. De fait, la trace ne contient pas d'information en lien avec l'erreur

Run issue	**Cause**	**Location**
Fault exception	Illegal instruction	ECC single error function
Interrupt	Illegal instruction	Interrupt handler
Fault exception	Illegal instruction	ECC single error function
Fault exception	Illegal instruction	ECC double error function

TABLE 5.10 – Analyse des cas d'erreur des SoCs de SERVAL sous Alpha - Session 2

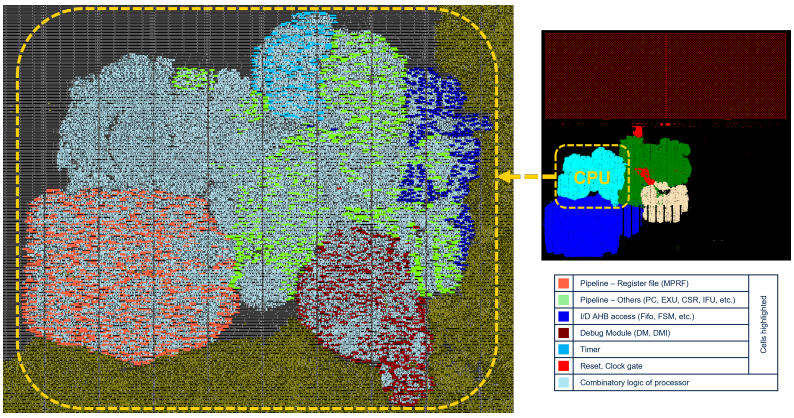
Malgré les conditions de tests en basse tension, le taux d'erreur du processeur reste beaucoup trop faible pour obtenir une statistique correcte permettant de valider la cohérence des résultats obtenu par l'analyse automatique de la trace. La méthode de validation envisagée consistait à obtenir un nombre conséquent d'événements sur chaque processeur. Si en comparant les points d'apparition des erreurs dans les deux processeurs nous avions constaté que les registres protégés du processeur RAD n'étaient pas affectés, nous aurions pu confirmer la cohérence des résultats avec l'implémentation du circuit. C'est pourquoi nous avons organisé une autre campagne de tests pour valider l'approche, les résultats sont présentés dans la section suivante.

5.3.3 Injection de fautes par impulsions de rayons X focalisés

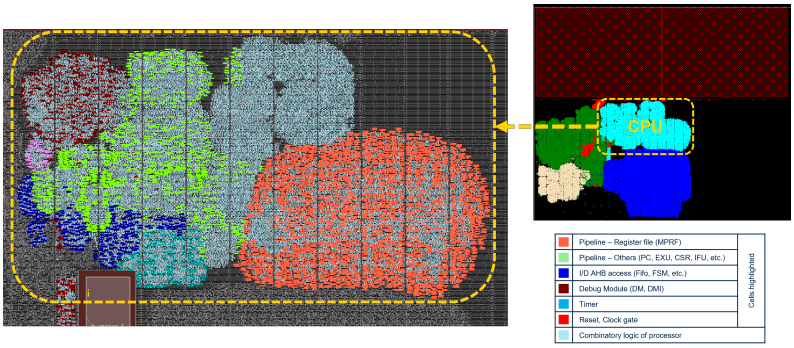
Ces expérimentations ont été réalisées à l'ESRF à Grenoble [31], sur la ligne ID09. Cette ligne fournit un faisceau pulsé de rayons X d'une période de 74 ps, ce qui représentent 2.10^8 protons pour un total de 15 keV. Il est aussi possible d'obtenir une impulsion de 1 μm, ce qui équivaut à 11 impulsions à 74 ps. Le faisceau a un diamètre de 25 μm. Nous avons donc procédé à des scans du circuit par balayage en 'S' afin de couvrir toutes la surface cible avec le faisceau. Par obstruction, il est possible de restreindre le faisceau à 5 μm de diamètre, mais ce réglage n'a pas été utilisé car les scans auraient duré beaucoup trop longtemps.

Dans les processeurs, nous avons pu détourer la zone du fichier de registres (MPRF) dont la délimitation est assez claire sur le GDS, Figure 2.11 grace à l'identification des zones du pipeline sur la Figure 5.13. On constate effectivement que la zone du MPRF est très dense en termes de registres, très localisée, et peut être contenue dans un rectangle dans lequel les scans peuvent être effectués. Effectivement, cette zone s'est avérée être suffisamment sensible pour récolter des événements après quelques scans, et suffisamment localisée pour être ciblée sans déborder sur les autres parties du pipeline. Deux cibles ont donc été définies dans les processeurs, l'ensemble du cœur et le MPRF seulement, leurs surfaces et le paramétrage des scans est détaillé dans le Tableau 5.11. Le faisceau de 25 μm, soit $9.817 \times 10^{-4}\ mm^2$, a été utilisé pour les expérimentations, puisqu'un nombre assez significatif d'impulsions et de scans ont été nécessaires pour provoquer des événements et que la surface est suffisamment restreinte pour adresser les cible définies.

Comme déjà évoqué dans la Section 2.1.4, les impulsions du faisceau à rayons X ont un effet ionisant sur le circuit. Ce qui a pour conséquence dégrader le temps de réponse des cellules jusqu'à la mise en défaut définitive du circuit sur une échelle de temps correspondante au moins à celle de l'expérience. Les circuits fabriqués en technologie 28 nm UTBB



(a) STD SoC



(b) RAD SoC

FIGURE 5.13 – Détails des zones des processeurs des deux SoCs de SERVAL

Element	**Area**	**Scan settings**	
		Lines	Columns
STD CPU	$52.23 \times 10^{-3}\ mm^2$	11	16
STD CPU - MPRF	$10.94 \times 10^{-3}\ mm^2$	5	7
RAD CPU	$56.33 \times 10^{-3}\ mm^2$	19	20
RAD CPU - MPRF	$59.94 \times 10^{-3}\ mm^2$	6	10

TABLE 5.11 – Surface des cibles dans les SoCs de SERVAL - Expériences sous rayons X à l'ESRF [31]

FD-SOI sont conformes aux normes spatiales en termes de dose supportée (50 krad), or, des mesures ont montré que la dose reçue par le circuit SERVAL lors des expérimentations pouvaient atteindre plusieurs centaines de krad. Cependant, le but est donc de pourvoir collecter le maximum d'événements tout en maintenant le circuit fonctionnel le plus longtemps possible. Etant donné que la dose reçue modifie le comportement des transistors, résultant en une modification de la synchronisation des signaux, pour conserver un fonctionnement stable après plusieurs scans, la fréquence a été divisée par 2, soit 200 MHz.

Plusieurs paramètres sont à ajuster concernant le faisceau et la méthode de scan. Il est possible de chevaucher les impacts, de multiplier impulsions avant de passer à la position suivante et changer la largeur de l'impulsion (74 ps ou 1 μs). Ces paramètres ont été ajustés durant les expériences selon le comportement du composant. Afin de positionner le faisceau sur le composant, les opérateurs de la line ID09 ont calibré le réticule de la caméra afin que le faisceau soit centré sur ce réticule au niveau du circuit. Un repère sur le boitier du circuit était positionné au centre, nous avons donc fait l'hypothèse que celui correspondait

aussi au centre de la *die*. Cette hypothèse était soutenue par le fait que le circuit comporte un grand nombre de broches, et qu'un positionnement très précis de la *die* dans le boîtier est nécessaire pour que les connections puissent être faites entre les broches externes et les *pads* de la *die*. Cette hypothèse a pu être vérifiée grâce aux tableaux mémoires des deux SoCs, très sensibles aux impacts du fait de la grande densité de cellules mémoires.

Suite aux premiers scans, nous avons constaté qu'utiliser le chevauchement des tirs n'était pas bénéfique en termes d'événements générés, par contre cela induisait rapidement une forte dose, nous obligeant à changer de composant de test. Le chevauchement a été réduit à 5 μm car la répartition de l'intensité du flux sur un axe passant par le centre du faisceau a l'allure d'une fonction gaussienne. Le fait d'utiliser des impulsions de 1 μs a permis d'augmenter l'agressivité du faisceau envers le circuit, mais cela engendre apparemment une dose supérieure à 10 impulsions par position. Nous avons donc convergé vers les réglages suivant après de nombreux essais qui nous ont permis d'optimiser le nombre d'événements collectés par scan :

— Scanner par pas de 20 μm, soit 5 μm de chevauchement sur la position précédente ;
— Impulsions d'un largeur de 74 ps, soit le minimum fourni par la ligne ID09 ;
— 10 impulsions par position ;
— Tension d'alimentation du circuit nominale, soit 0.9 V ;
— Fréquence de fonctionnement de 200 MHz, soit la moitié de la fréquence nominale.

Malgré les efforts d'optimisation des paramètres, il a été très difficile d'obtenir des événements sur le processeur RAD. Celui-ci s'est avéré plus sensible à la dose que le STD, et moins sensible aux événements du fait de l'implémentation en cellules TMR de certains registres. Nous pensons que cette sensibilité accrue à la dose vient de la disparité des types de cellules utilisés pour l'implémentation du cœur et donc des temps de réponses. Lors de la conception du circuit, l'outils de placement/routage a plus été contraint par les écarts de temps de réponse des cellules et n'a sûrement pas pu conserver autant de marge pour le processeur STD que pour le RAD. Ce qui pourrait expliquer la différence de comportement des deux cœurs face à la dose modifiant la synchronicité des signaux du circuit. Tous les cas d'erreurs ont été pris en compte durant les expérimentations. Les cas ont été classés selon les cibles pour les deux processeurs : soit le processeur complet, soit uniquement le fichier de registres, . Lors des scans répétitifs, nous avons remarqué une reproductibilité des événements d'un scan à l'autre. Les cas concernés ont donc été regroupés afin de faire apparaître un motif de répétition dans les résultats.

SoC	**SoC STD**		**SoC RAD**		**SoC STD (Reproducible scans)**
Area scanned	**CPU**	**MPRF**	**CPU**	**MPRF**	**CPU**
Events	40	8	17	8	23
SDC	23	6	8	2	16
Fault	7	2	2	0	6
Timeout	4	0	3	0	1
ECC_DE	6	0	0	0	0
Unknown	0	0	4	6	0

TABLE 5.12 – Résultats des scans - SoCs de SERVAL sous rayons X @(0.9 V et 200 MHz)

Le Tableau 5.12 récapitule l'ensemble des cas obtenus durant les tests. Les résultats

ont été classés par cible pour les deux SoCs. La colonne de droite, *STD (Reproducible scans)*, est un sous-ensemble des cas du SoC STD, il s'agit d'une suite de 5 scans ayant provoqués quasiment les mêmes effets aux mêmes endroits. Les cas d'erreur ayant été capturés par l'IP d'observation lors de cette session de tests sont des cas de *fault exception*. Dans certains cas, il fut impossible de récupérer les données de diagnostic du SoC, ceci est certainement dû au fait que l'infrastructure de débogage a été affectée par les injections de fautes. Des données de diagnostic auraient néanmoins pu être récupérées en bloquant l'exécution du processeur et en effectuant un reset local pour restaurer un état stable de la partie de débogage.

Faultpoint	**Cases**	**Accuracy**	**Hardened**	**SAF**	**RAD**	**TOTAL**
pipeline.i_pipe_mprf.mprf_int[2]	4971	0.95	1	2	0	2
pipeline.i_pipe_ifu.imem_addr_r	4411	0.83	1	1	1	2
pipeline.i_pipe_mprf.mprf_int[18]	2822	0.75	1	0	1	1
pipeline.i_pipe_mprf.mprf_int[12]	2073	0.84	0	1	0	1
pipeline.i_pipe_mprf.mprf_int[13]	838	0.91	0	1	0	1
pipeline.i_pipe_mprf.mprf_int[17]	826	0.59	0	1	0	1
pipeline.i_pipe_ifu.q_err	678	0.90	0	1	0	1

TABLE 5.13 – Résultats d'inférence des cas d'erreur capturés lors des scans CPU - SoCs de SERVAL sous rayons X @(0.9 V et 200 MHz)

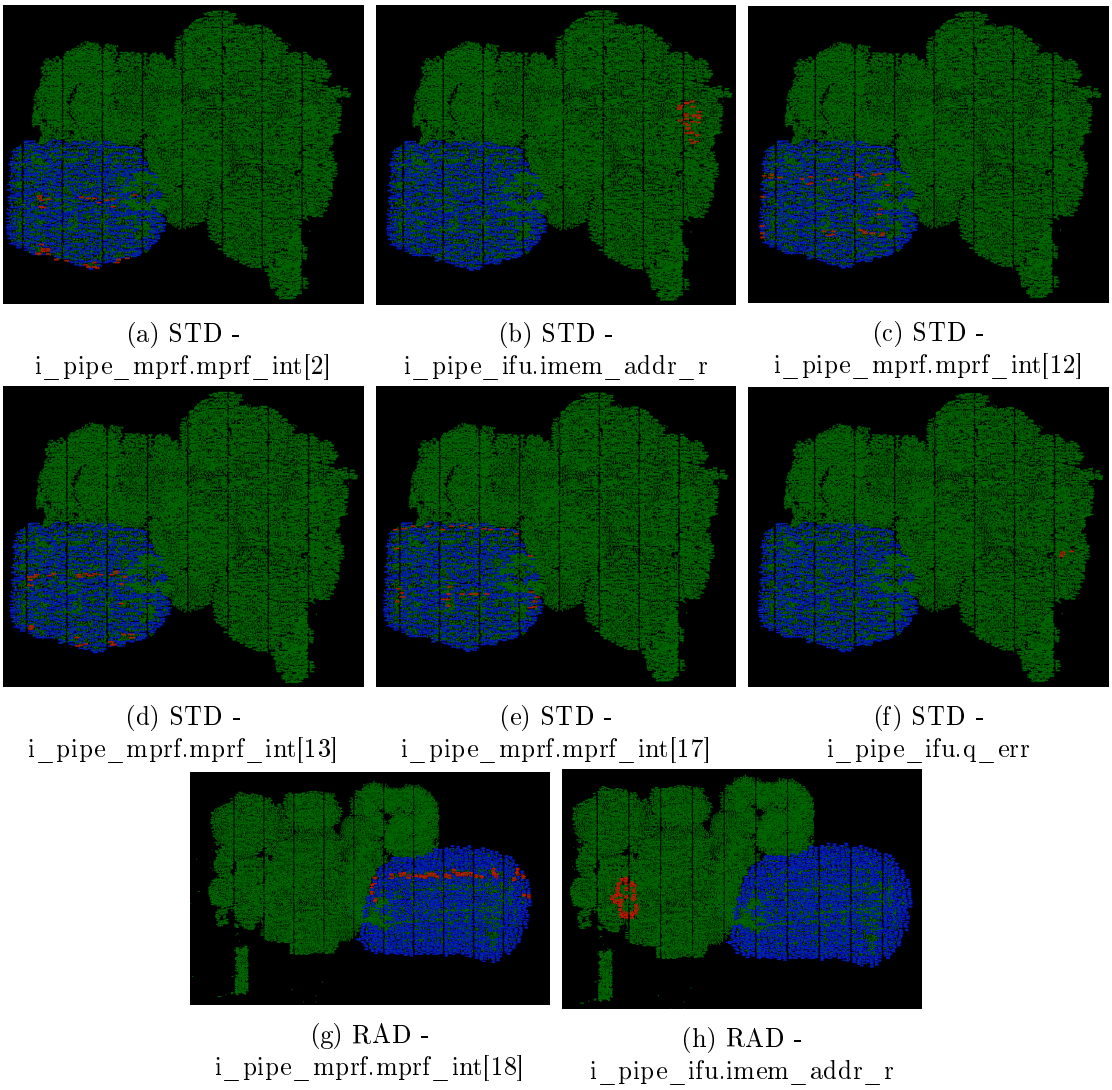
Faultpoint	**Cases**	**Accuracy**	**Hardened**	**SAF**	**RAD**	**TOTAL**
pipeline.i_pipe_mprf.mprf_int[2]	4971	0.95	1	2	0	2

TABLE 5.14 – Résultats d'inférence des cas d'erreur capturés lors des scans MPRF - SoCs de SERVAL sous rayons X @(0.9 V et 200 MHz)

Les résultats d'inférence du modèle de classification présentés dans le Tableau 5.13 concernent les scans des processeurs deux SoCs du circuit SERVAL. Le Tableau 5.14 contient seulement les résultats des scans des zones des fichiers de registres de ces deux processeurs. Nous n'avons pu récolter qu'une quantité très faible d'événements en raison des difficultés rencontrées lors des expérimentations. Les caractéristiques de la technologie, robuste au SEEs et sensible à la dose, croisées avec celles des effets des rayon X sur le circuit, à savoir un dépôt significatif de dose, ont pour conséquence de rendre peu probable l'apparition des SEUs que nous souhaitons collecter lors de cette expérience. On peut néanmoins constater que durant les scans des processeurs, des erreurs ont été détectées dans un certain ensemble de registres, Tableau 5.13, alors qu'en restreignant la zone des scans aux MPRFs, les éléments récoltés ne proviennent plus que ce sous-ensemble de registres du pipeline.

Les images de la Figure 5.14 illustrent les registres du processeur, listés dans les Tableau 5.13 et 5.14, suspectés d'avoir collecté des SEUs. Sur ces images, on retrouve l'ensemble du processeur en vert (STD ou RAD selon la légende), les registres du MPRF en bleu, ainsi qu'un registre en particulier en rouge dont le nom est dans la légende. On remarque premièrement que les bits des registres ne sont pas groupés, mais répartis de manière assez aléatoire. Deuxièmement les registres illustrés du processeur RAD sont tripliqués, ils possèdent donc trois fois plus de cellules mémoires et représentent une surface plus conséquente.

Le Tableau 5.15 quant à lui regroupe les inférences par ordre d'apparition des cas avec le numéro de l'itération du scan correspondant. On remarque que le même registre apparaît plusieurs fois de suite sur des scans différents. L'implémentation de ces registres est aussi illustrée parmi les images de la Figure 5.14.



(a) STD - i_pipe_mprf.mprf_int[2]
(b) STD - i_pipe_ifu.imem_addr_r
(c) STD - i_pipe_mprf.mprf_int[12]
(d) STD - i_pipe_mprf.mprf_int[13]
(e) STD - i_pipe_mprf.mprf_int[17]
(f) STD - i_pipe_ifu.q_err
(g) RAD - i_pipe_mprf.mprf_int[18]
(h) RAD - i_pipe_ifu.imem_addr_r

FIGURE 5.14 – Détail des zones des processeurs affectées lors des scans sous rayons X des deux SoCs de SERVAL

Scan	**Faultpoint**
1	pipeline.i_pipe_mprf.mprf_int[2]
2	pipeline.i_pipe_mprf.mprf_int[2]
3	pipeline.i_pipe_mprf.mprf_int[2]
4	pipeline.i_pipe_mprf.mprf_int[2]
5	pipeline.i_pipe_mprf.mprf_int[17]
6	pipeline.i_pipe_mprf.mprf_int[13]

TABLE 5.15 – Inférence scans MPRF - rayons X des SoCs de SERVAL @(0.9 V et 200 MHz)

5.3.4 Bilan des expérimentations

Les expérimentations ont été très instructives du point de vue des plateformes, et des contribution dans le cadre de chaque méthode de test. Dans l'ensemble, les tests sous EMFI et sous Alpha ont permis de configurer et d'optimiser la plateforme et l'environnement

logiciel, pour veiller à ce que les hypothèses soient vérifiées et que des résultats pertinents puissent être collectés. Les tests sous impulsions de rayons X focalisés mettent en lumière une certaine cohérence et efficience de l'ensemble de l'approche.

Lors des tests sous EMFI, l'ensemble des cas d'erreurs a pu être couvert. Par itérations, ces expériences ont permis de faire ressortir les différents modes de défaillance et d'améliorer le diagnostic logiciel. Ces tests peuvent être effectués sans contrainte et au besoin, ce qui en fait un outil extrêmement pratique pour la mise au point et les premières étapes de validation du circuit. De fait, il est possible de cumuler de nombreux cas d'erreurs, mais il n'en est pas moins que les perturbations physiques de l'EMFI sont très loin de celles des particules et ne permettent ni d'obtenir de SER équivalent, ni de cibler une zone restreinte du circuit. Ces tests ont constitué les premières expérimentations des plateformes, et fait apparaître des erreurs de gestion mémoire qui n'avaient pas été appréhendées en simulation en rapport avec l'initialisation à de valeurs aléatoires des cellules SRAM à la mise sous tension. Cette piqûre de rappel a mis en lumière l'absolue nécessité d'une validation exhaustive en simulation et en réel.

Les tests sous particules Alpha ont représenté une alternative très pratique. Nous avions accès à un laboratoire sur site pour effectuer des tests avec une source radioactive. Tout en suivant quelques mesures de précautions, nous avons été en mesure de réserver plusieurs semaines pour mener des tests. Ces expériences nécessitant de déplacer l'ensemble du matériel, ont été très formatrices du point de vue pratique, car elles ont permis d'être pleinement opérationnel pour des tests futurs dans des établissements où le temps est compté. Cependant les particules Alpha n'ont pas permis de récolter un nombre suffisant d'événements liés au processeur sur nos circuits de test. Les conditions hors spécifications de basse tension et basse fréquence nécessitaient une surveillance accrue de l'application pour s'assurer qu'elles ne soient pas la cause des défaillances. La mémoire est quant à elle très sensible, elle souffrait donc d'un SER élevé dans ces conditions, ce qui a perturbé le peu d'événements que nous avons récoltés. Une limitation matérielle qui oblige une action du processeur pour corriger une case mémoire en erreur a contraint l'application de passer un temps significatif dans le *handler* d'interruption dans le but de corriger les erreurs. Par conséquent, les quelques cas d'erreur capturés ont surpris l'application dans ce *handler* d'interruption, les rendant inexploitables car le dictionnaire de faute ne couvre pas cette section de code.

Finalement, des résultats dont la pertinence a pu mettre en lumière la cohérence de l'approche ont été obtenus durant les expérimentations réalisées à l'ESRF sous rayons X. Grâce au faisceau pulsé et focalisé sur une surface suffisamment petite pour n'adresser qu'un sous-ensemble de registres de chaque bloc du processeur, nous avons pu cibler des zones en particulier et comparer avec les résultats de l'inférence du modèle de classification. Les difficultés de mise en place de l'expérience et de réglage du faisceau ont été surmontées et quelques événements ont pu être récoltés sur le processeur STD. Le processeur RAD quant à lui, s'est révélé être plus sensible que le STD à la dose.

5.3.5 Perspective d'amélioration de l'observation

Contrôle du flot d'exécution

Comme vu dans la Section 1.3, de nombreuses techniques existent dans la littérature pour contrôler le flot d'exécution et détecter des erreurs comme des divergences du flot ou des corruptions de données. Ces techniques pourraient être implémentées afin d'augmenter le taux de couverture des erreurs. Ceci permettrait d'adresser plus de cas de corruptions silencieuses de données et de *timeout*, mais aussi d'augmenter la réactivité de détection sur l'ensemble des erreurs. Les bénéfices résultants seraient une augmentation du nombre de cas d'erreur collectés par expérience ou par campagne de simulation, ainsi qu'une réduction de la taille optimale du buffer de trace grâce au gain en réactivité. L'optimisation du buffer consiste à garder un taux d'observation élevé avec une taille minimale, afin de réduire l'impact sur les ressources. Plusieurs techniques pourraient être adoptées dans ce cadre :

Contrôle des instructions de branchement

Le fait de contrôler les instructions de branchement, plus précisément vérifier si l'adresse de l'instruction suivante en cas de saut correspond à la valeur attendue [28, 39, 80], est une technique efficace et ne nécessitant que peu de ressources. Une technique équivalente basée sur logiciel [8, 84] permet de contrôler l'ordre d'exécution des *basic blocs*, cette technique implique quant à elle une baisse des performances et une modification du flot d'exécution.

Signature de code et contrôle des données

Le contrôle de l'intégrité des données apporterait une couverture supplémentaire pour la détection des erreurs silencieuses. Des techniques ont été développées afin de vérifier en ligne l'apparition d'erreurs, comme la méthode SWIFT [85] basée sur la duplication des données et étapes de calcul, ou encore le calcul de la signature de code depuis un bloc matériel externe [28]. Le calcul de la signature impose néanmoins une table des signatures de référence, ce qui implique un tableau mémoire supplémentaire à protéger de manière active par scrutation périodique.

Géométrie des événements de trace

Les résultats obtenus grâce à la méthode de ML, présentés dans la Section 5.2.3, montrent que la taille du buffer de la seconde version de l'IP d'observation peut être largement réduite, sans pour autant affecter la taux d'observation. En revanche, on peut en déduire que pour augmenter le taux d'observation, il faudrait plutôt chercher à collecter plus de signaux pertinents, listés sur les Figures 4.2 et 4.5.

Amélioration de la fiabilité

Une implémentation basée sur des cellules mémoires TMRs ou durcies permet de diminuer drastiquement le SER. Cependant, malgré un taux d'erreur faible, les données à forte persistance sont toutefois soumises à des erreurs. Comme nous avons pu le remarquer lors de tests sous alpha, des erreurs ont pu apparaître dans l'UDRB de la plateforme SERVAL, bien que les registres aient été implémentés avec des cellules TMRs. Maîtriser le taux d'erreur est essentiel, sans quoi, il est impossible de réagir à temps aux SEUs, et la redondance perd son intérêt. Il faut donc pouvoir rafraîchir les cellules de stockage pour éliminer les erreurs résiduelles masquées par la redondance. Pour combler ce manque, deux techniques sont envisageables, comme celles proposées dans la Section 5.1.4. Le principe serait d'utiliser des cellules logiques séquentielles redondantes qui se rafraîchissent en cas de SEU [54, 55], ou d'implémenter une machine à état pour déclencher un rafraîchissement périodique de chaque registre par copie mémoire.

Conclusion générale

Au terme de ces cinq chapitres, les deux axes majeurs de cette thèse ont été abordés à savoir la classification de SEUs pour le registre à décalage, et les IPs d'observation des processeurs. Ces contributions à l'état de l'art ont pu être mises à l'épreuve pour évaluer leurs bénéfices. L'objectif général de ces contributions est d'augmenter l'observabilité de systèmes logiques en fournissant un support à l'investigation des erreurs dues aux radiations. Dans le contexte de la qualification du comportement des circuits sous radiations, les techniques développées ont été implémentées sur des circuits de test et expérimentées sous faisceau de protons à PSI, d'ions lourds à RADEF et sous faisceau focalisé et pulsé de rayons-X à l'ESRF. Les deux circuits ont été fabriqués en technologie 28 nm UTBB FD-SOI dans l'usine de STMicroelectronics à Crolles. Les deux circuits, SERVAL et SQUAL, sont conçus pour qualifier respectivement une technologie terrestre et une technologie spatiale GEO.

Dans le cadre des qualifications de cellules logiques séquentielles, la technique s'intègre à une démarche d'estimation du taux d'erreur lors de tests accélérés sous faisceau de particules. Ce besoin industriel est accompagné d'exigences quant à la réalisation du registre à décalage qui constitue une structure de test dédiée, concernant son implémentation et le profil de mission imposant des tests à hautes fréquences de fonctionnement. Le module développé permet d'identifier la provenance des événements qui surviennent dans la structure de test grâce à une analyse en ligne de la signature des erreurs. Ce module permet de fournir un suivi direct du nombre d'événements, ce qui représente un atout lors des campagnes de test. Des tests sous protons et ions lourds sur des technologies respectivement terrestres et spatiales en 28 nm UTBB FD-SOI ont mis en lumière les bénéfices de la technique. Également exposé au faisceau, le module a rencontré un SBU lors des qualifications sous ions lourds, ce qui a permis d'identifier les limites de la méthode d'implémentation durci par TMR. Ces limites sont adressées dans les futures implémentation tout comme les types d'événements identifiables par le module. Nous proposons également une amélioration de la méthode de classification pour supporter les tests de technologies plus sensibles pouvant être sujettes aux MCUs.

Les qualifications utilisées pour évaluer le taux d'erreur de cellules logiques séquentielles servent en pratique à estimer de taux d'erreurs de circuit contenant ce type de cellules. Les cellules logiques séquentielles contiennent des informations qui peuvent être critiques dans un circuit. Nous traitons en particulier le cas des systèmes basés sur microprocesseurs dans lesquels le taux d'erreur prépondérant est celui des bascules utilisées pour implémenter les registres qui le composent. Les systèmes basés sur microprocesseurs sont présents dans de nombreuses applications notamment des applications dont la sûreté de fonctionnement est critique, en particulier pour la vie de personnes. Ces systèmes ont donc été utilisés pour valider les règles de conception mise en œuvre pour élaborer les

circuits. Plus précisément, certaines méthodes de conception sont utilisées pour améliorer la fiabilité en diminuant le SER des circuits dans les parties sensibles, comme la triplication partielle. La démarche de qualification des SoCs n'est cependant pas aussi directe que pour les registres à décalage étant donné la complexité logique et le comportement de l'application. Nous avons donc proposé donc dans ce contexte une IP, intégrée dans le SoC, capable de fournir des informations de diagnostic permettant l'investigation de la cause de l'erreur. Les données accessibles reposent sur l'enregistrement en continue dans un buffer embarqué de la trace d'exécution complétée par les valeurs de certains registres du cœur. Elles incluent aussi la sauvegarde depuis l'application des registres d'état du processeur et d'informations à propos de l'erreur identifiée, dans une mémoire protégée et dédiée à l'utilisateur. La collecte de la trace est effectuée de manière non intrusive envers l'application, et l'architecture de l'IP est agnostique vis-à-vis du processeur. L'intégrité des informations disponibles est garantie par l'implémentation durcie de l'IP. L'accès aux informations est quant à lui assuré par l'infrastructure de débogage adressée par un lien conventionnel JTAG de manière autonome envers le reste du système.

Notre approche se base sur la capture de la propagation des erreurs dans le processeur grâce à l'enregistrement en continue des informations de diagnostic dans un buffer de trace embarqué. L'IP intègre des mécanismes capables d'arrêter l'enregistrement si une erreur est détectée par des assertions matérielles configurables. Cette combinaison, composée du buffer de trace et de déclencheurs, permet de conserver les dernières instructions exécutées avant qu'une erreur ne soit détectée, capturant ainsi la propagation de la faute dans la trace d'exécution. Nous avons développé deux versions de l'IP basées sur la technique de la collecte de trace combinée avec des assertions matérielles et des déclencheurs.

La première constitue une approche initiale, elle est dotée de fonctionnalités réduites qui ont pu être implémentées rapidement afin de suivre les contraintes calendaires de développement. Cette IP a été intégrée dans le circuit SERVAL. Ce circuit ayant été disponible assez rapidement durant le déroulement de ces travaux, nous avons pu réaliser des tests sous EMFI, et des tests radiatifs sous faisceau de particules alpha et sous faisceau de rayons-X focalisés et pulsés. Les résultats montrent que cette IP fournit des informations pertinentes dont l'analyse approfondie révèle la localité de la faute originelle. L'ensemble des éléments de collecte et de récupération des données s'est également avérée fiable, et a permis d'accéder au système dans tous les cas d'erreurs rencontrés.

La seconde version de l'IP est basée sur un bloc de génération de trace respectant les spécification RISC-V, elle est également dotée d'un ensemble d'assertions matérielles plus complet et offrant une meilleure capacité de détection d'erreur. Cette IP a été développée afin d'adresser les limites en termes de profondeur d'enregistrement, capture des erreurs et de couverture des modes de défaillance de la première IP. Les assertions matérielles plus élaborées sont capables de détecter un spectre plus large d'erreur. Le buffer de trace d'une taille conséquence géré par un mécanisme de génération de trace synchronisé avec le pipeline couvre un nombre plus important d'instructions. L'étude des modes de défaillances, grâce à une campagne d'injection de fautes par simulation couvrant de manière exhaustive tous les registres du processeur, a mis en évidence les taux de couverture des cas d'erreur et les taux de capture complète de la propagation d'erreur des deux IPs.

La campagne d'injection de fautes a aussi permis de constituer un dictionnaire de fautes composé des différentes signatures d'erreur associés à un point d'injection, semblable à un SEU dans une bascule. Nous avons tiré parti de cette base de données grâce à un modèle de classification à apprentissage automatique pour retrouver, lors de l'analyse de

données issues des expérimentations, d'où provient la faute dans le microprocesseur. Cette approche a pu être validée grâce aux tests sous faisceau de rayons-X focalisés et pulsés, les résultats montrent une cohérence entre la zone ciblée et le registre suspecté d'avoir subi une SBU. En identifiant la localisation des fautes dans le microprocesseur, il est dorénavant possible de confronter l'estimation de la sensibilité des registres de la campagne de simulation avec celle mesurée lors de qualification radiative. De même, l'efficacité de la TMR partielle sur les registres du cœur et l'ensemble du flot de conception du circuit peut être validée expérimentalement.

Perspectives

Premièrement il est prévu de conduire des tests sous EMFI et sous particules alpha de la plateforme SoC de SQUAL afin de valider l'ensemble du banc de test avant les campagnes d'irradiation. L'objectif est de vérifier le fonctionnement du SoC, de l'application, de l'IP d'observation, ainsi que de l'ensemble des moyens permettant d'opérer le circuit depuis la machine hôte au travers la carte FPGA.

Dans le but d'approfondir la démarche de validation de l'IP d'observation, de nouveaux tests sont prévus à l'ESRF sur la ligne ID09 pour essayer de collecter de nouveaux cas d'erreur dans les plateformes SoC. La même démarche se poursuit et consiste à confronter les zones ciblées par le faisceau focalisé et pulsé de rayons-X aux localisations estimées des fautes grâce au modèle de classification à apprentissage automatique. Des tests sous faisceau LASER pulsé auraient également pu être conduit chez PULSCAN [81] mais l'occasion ne s'est pas présentée durant cette thèse. Les possibilités d'injection de faute de manière localisée sont équivalentes à celle de la ligne ID09 grâce à un faisceau LASER focalisé. Ces tests pourraient adresser notamment le circuit SQUAL dont le banc de test sera bientôt opérationnel.

Des expérimentations sous ions lourds pour SQUAL et sous neutrons pour SERVAL sur les plateformes SoCs permettraient potentiellement de collecter de nombreux cas d'erreur avec une irradiation de l'ensemble de la plateforme. Collecter des cas en nombre sur les deux cœurs dont l'un est partiellement durci par TMR permettrait de comparer les statistiques d'apparition des événements dans les deux processeurs et de vérifier la cohérence des résultats en fonction de l'implémentation.

Lors des tests menés durant ces travaux, nous avons identifié des éléments dont les résultats suggèrent qu'ils peuvent être perfectionnés. Concernant la classification d'événements, l'implémentation durcie par TMR a été mise à l'épreuve des ions lourds lors des tests sous faisceau. La forte persistance des données a malgré tout conduit à des SBUs sur les compteurs. L'utilisation de cellules à correction automatique ou de rafraîchissement périodique des compteurs serait suffisamment efficace pour garantir l'intégrité des données durant les tests. Pour adresser des technologies plus sensibles sujettes aux MCUs dans les bascules, le module de classification en ligne serait en mesure supporter cette nouvelle classe moyennant des contraintes de structure de l'arbre d'horloge. Cette méthode pourrait être complétée avec l'utilisation d'un buffer circulaire pour stocker la largeur des événements multiples identifiés.

Le développement des deux versions de l'IP d'observation a été contraint par le planning de conception des circuits. De même, la méthode d'analyse des données par le modèle de classification à apprentissage automatique n'a été exploitée qu'après les réalisations

des circuits. Les résultats de l'apprentissage sur les bases de données constituées lors des campagnes d'injection de fautes par simulation, ont montrés que le placement des signaux et la taille du buffer de trace d'exécution pouvaient être perfectionnés. En effet, grâce à cette méthode, il est possible de faire le lien entre la localisation d'une faute injectée et la pertinence des informations récoltées grâce à l'IP d'observation. La précision du modèle par classe est devenue la métrique de référence pour établir les points d'observation et la profondeur de l'enregistrement.

Les expérimentations des plateformes SoC sous particules alpha et sous rayons-X ont permis de récolter des cas d'erreur dont certains n'étaient pas supportés par l'IP d'observation. Une amélioration de la partie de détection d'erreur permettrait d'adresser un spectre plus large de mode de défaillance et augmenter en proportion le nombre de cas exploitables lors des expérimentations. Il serait possible de modifier l'application pour inclure des techniques logicielles de contrôle du flot d'exécution et des données par duplication des étapes de calcul. Pour les versions ultérieures de l'IP, il serait bénéfique de supporter des techniques de contrôle du flot d'exécution par vérification des adresses des instructions de branchement, ou des techniques de contrôle de la signature du code.

De la même manière que le module de classification des événements dans les registres à décalage, l'IP d'observation a été sujette aux SBUs dans les registres avec une forte persistance des données. Les mêmes techniques pourraient être proposées pour assurer de l'intégrité des données durant les expérimentations.

Publications

Conférences internationales

Présentation orale virtuelle, récompensée deuxième meilleure contribution étudiante :
Sébastien Thomet, Serge De-Paoli, Jean-Marc Daveau, Valérie Bertin, Fady Abouzeid, Philippe Roche, Fakhreddine Ghaffari and Olivier Romain, **FIRECAP : Fail-Reason Capturing hardware module for a RISC-V based System-on-a-Chip**, *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) | 978-1-6654-1609-2/21/$31.00 ©2021 IEEE | DOI : 10.1109/DFT52944.2021.9568317*

Présentation de poster virtuelle :
Sébastien Thomet, Serge De-Paoli, Fakhreddine Ghaffari, Jean-Marc Daveau, Valérie Bertin, Fady Abouzeid, Olivier Romain, Philippe Roche, **Fail-Reason Capturing hardware module for a RISC-V based System-on-a-Chip**, *2021 21th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*

Présentation de poster en présentiel :
Sébastien Thomet, Serge De-Paoli, Fakhreddine Ghaffari, Fady Abouzeid, Olivier Romain, Philippe Roche, **CLASS : on-Chip Lightweight Accurate SEU/SET event claSSifier**, *2019 19th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*

Journal scientifique

Soumission en attente d'un retour :
Sébastien Thomet, Fakhreddine Ghaffari, Serge De-Paoli, Jean-Marc Daveau, Fady Abouzeid, Olivier Romain, **Observation framework of errors in microprocessors with machine-learning location inference of radiation-induced faults**, *ELSEVIER Microelectronics Reliability*

Bibliographie

[1] *A Case Study of Toyota Unintended Acceleration and Software Safety - Prof. Phil Koopman*. https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf. Accessed: 2021-10. 2014.

[2] M. Abramovici et al. « A reconfigurable design-for-debug infrastructure for SoCs ». In : *2006 43rd ACM/IEEE Design Automation Conference*. 2006, p. 7-12. DOI : 10.1145/1146909.1146916.

[3] *Advanced Trace and Debug IP - SiFive Insight*. https://www.sifive.com/soc-ip/sifive-insight. Accessed: 2021-11.

[4] *Alpha source ^{241}Am - Eckert & Ziegler*. https://www.ezag.com/fileadmin/ezag/user-uploads/pdf/isotope/5_industrial_sources.pdf. Accessed: 2021-10.

[5] *Artist's impression of Solar influence on Earth - NASA*. https://sci.esa.int/s/wRVgbXw. Accessed: 2021-10. 2021.

[6] J.L. Autran et Daniela Munteanu. *Soft errors - from particle to circuits*. 1re éd. CRC Press, 2015.

[7] José Rodrigo Azambuja et al. « A Fault Tolerant Approach to Detect Transient Faults in Microprocessors Based on a Non-Intrusive Reconfigurable Hardware ». In : *IEEE Transactions on Nuclear Science* 59.4 (2012), p. 1117-1124. DOI : 10.1109/TNS.2012.2201750.

[8] José Rodrigo Azambuja et al. « HETA: Hybrid Error-Detection Technique Using Assertions ». In : *IEEE Transactions on Nuclear Science* 60.4 (2013), p. 2805-2812. DOI : 10.1109/TNS.2013.2246798.

[9] Capucine Lecat Mathieu-de Boissac et al. « Single-Event Transient Space Characterizations in 28-nm UTBB SOI Technologies and Below ». In : *IEEE Transactions on Nuclear Science* 68.1 (2021), p. 21-26. DOI : 10.1109/TNS.2020.3033590.

[10] Capucine Lecat-Mathieu de Boissac et al. « Influence of Supply Voltage and Body Biasing on Single-Event Upsets and Single-Event Transients in UTBB FD-SOI ». In : *IEEE Transactions on Nuclear Science* 68.5 (2021), p. 850-856. DOI : 10.1109/TNS.2021.3071963.

[11] Cyril Bottoni et al. « Partial triplication of a SPARC-V8 microprocessor using fault injection ». In : *2015 IEEE 6th Latin American Symposium on Circuits Systems (LASCAS)*. 2015, p. 1-4. DOI : 10.1109/LASCAS.2015.7250415.

[12] Fred A. Bower, Daniel J. Sorin et Sule Ozev. « Online Diagnosis of Hard Faults in Microprocessors ». In : *ACM Trans. Archit. Code Optim.* 4.2 (2007), 8–es. ISSN : 1544-3566. DOI : 10.1145/1250727.1250728. URL : https://doi.org/10.1145/1250727.1250728.

[13] Leo Breiman. « Pasting Small Votes for Classification in Large Databases and On-Line ». In : *Machine Learning* 36 (juill. 1999), p. 85-103. doi : 10.1023/A:1007563306331.

[14] Leo Breiman. « Random Forests ». In : *Machine Learning* 45 (oct. 2001), p. 5-32. doi : 10.1023/A:1010933404324.

[15] *CERN - Conseil Européen pour la Recherche Nucléaire - Geneva - Switzerland.* https://home.cern/. Accessed: 2021-10.

[16] Chia-Hsiang Chen, Phil Knag et Zhengya Zhang. « Characterization of Heavy-Ion-Induced Single-Event Effects in 65 nm Bulk CMOS ASIC Test Chips ». In : *IEEE Transactions on Nuclear Science* 61.5 (2014), p. 2694-2701. doi : 10.1109/TNS.2014.2342872.

[17] P. Cheynet et al. « Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors ». In : *IEEE Transactions on Nuclear Science* 47.6 (2000), p. 2231-2236. doi : 10.1109/23.903758.

[18] Eduardo Chielle et al. « Evaluating Selective Redundancy in Data-Flow Software-Based Techniques ». In : *IEEE Transactions on Nuclear Science* 60.4 (2013), p. 2768-2775. doi : 10.1109/TNS.2013.2266917.

[19] *ChipSHOUTER® - NewAE Technology Inc.* https://www.newae.com/products/NAE-CW520. Accessed: 2021-10.

[20] *Concepteur en ligne de SoC basé sur processeur RISC-V - SiFive.* https://www.sifive.com/core-designer. Accessed: 2021-11.

[21] *CoreMark CPU Benchmark - Embedded Microprocessor Benchmark Consortium.* https://www.eembc.org/coremark/. Accessed: 2021-10.

[22] *CoreMark CPU Benchmark - Embedded Microprocessor Benchmark Consortium.* https://www.python.org/. Accessed: 2021-10.

[23] *Coresight Debug and Trace - ARM.* https://developer.arm.com/ip-products/system-ip/coresight-debug-and-trace. Accessed: 2021-11.

[24] *CubeSats - ESA.* https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/CubeSats. Accessed: 2021-11.

[25] *Cyclotron of Louvain-la-Neuve (CYCLONE) - UCLouvain - Belgium.* uclouvain.be/en/research-institutes/irmp/crc/applications-technologiques.html. Accessed: 2021-10.

[26] Corrado De Sio et al. « SEU Evaluation of Hardened-by-Replication Software in RISC- V Soft Processor ». In : *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2021, p. 1-6. doi : 10.1109/DFT52944.2021.9568342.

[27] *Debug and Trace Applications - MIPI Alliance.* https://www.mipi.org/specifications/debug. Accessed: 2021-11.

[28] Boyang Du et al. « Online Test of Control Flow Errors: A New Debug Interface-Based Approach ». In : *IEEE Transactions on Computers* 65.6 (2016), p. 1846-1855. doi : 10.1109/TC.2015.2456014.

[29] Vasant Easwaran et al. « A unique non-intrusive approach to non-ATE Based cul-de-sac SoC debug ». In : *2014 27th IEEE International System-on-Chip Conference (SOCC)*. 2014, p. 336-339. doi : 10.1109/SOCC.2014.6948950.

[30] *ESCC Basic specification No. 25100 - Single event effects test method and guidelines.* http://escies.org/escc-specs/published/25100.pdf.

[31] *Europeam Synchrotron Radiation Facility (ESRF) ID09 Beamline - Grenoble - France.* https://www.esrf.fr/home/UsersAndScience/Experiments/CBS/ID09.html. Accessed: 2021-10.

[32] Harry Foster. *IC/ASIC functional verification trend report.* Rapp. tech. 2020 Wilson Research Group functional verification study, 2020.

[33] *Functional Safety - CADENCE.* https://www.cadence.com/ko_KR/home/solutions/automotive-solution/functional-safety.html. Accessed: 2021-10.

[34] Gilles Gasiot et al. « Experimental Soft Error Rate of Several Flip-Flop Designs Representative of Production Chip in 32 nm CMOS Technology ». In : *IEEE Transactions on Nuclear Science* 60.6 (2013), p. 4226-4231. doi : 10.1109/TNS.2013.2284546.

[35] *GEO, MEO, and LEO - How orbital altitude impacts network performance in satellite data services - Via Satellite by SES.* https://www.satellitetoday.com/content-collection/ses-hub-geo-meo-and-leo/. Accessed: 2021-10.

[36] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 1st. O'Reilly Media, Inc., 2017. isbn : 1491962291.

[37] R. Ginosar. « Survey of processors for space ». In : *European Space Agency, (Special Publication) ESA SP* 701 (jan. 2012).

[38] *Global Embedded Processor Debug Interface - Nexus 5001 Forum™ Standard.* https://nexus5001.org/nexus-5001-forum-standard/. Accessed: 2021-11.

[39] M. Grosso et al. « An on-line fault detection technique based on embedded debug features ». In : *2010 IEEE 16th International On-Line Testing Symposium.* 2010, p. 167-172. doi : 10.1109/IOLTS.2010.5560215.

[40] Sofiane Guissi. *Everything You Need to Know about FDSOI Technology – Advantages, Disadvantages, and Applications of FDSOI.* Rapp. tech. 2018.

[41] Trevor Hastie et Robert Tibshirani. « Discriminant adaptive nearest neighbor classification and regression ». In : *Advances in neural information processing systems.* 1996, p. 409-415.

[42] *Heavy ion cocktails available at RADEF - University of Jyväskylä - Finland.* https://www.jyu.fi/science/en/physics/research/infrastructures/accelerator-laboratory/radiation-effects-facility/heavy-ion-cocktails. Accessed: 2021-10.

[43] *ISIS ChipIR - Rapid testing of Single Event Effects with atmospheric-like neutrons - Oxford - United Kingdom.* https://www.isis.stfc.ac.uk/Pages/ChipIR.aspx. Accessed: 2021-10.

[44] JEDEC. *JESD234 - Test standard for the measurement of proton radiation single event effects in electronic devices.* https://www.jedec.org/standards-documents. 2013.

[45] JEDEC. *JESD57A - Test procedures for the measurements of single-event effects in semiconductor devices from heavy ion irradiation.* https://www.jedec.org/standards-documents. 2017.

[46] JEDEC. *JESD89-1B - Test method for real-time soft error rate.* https://www.jedec.org/standards-documents. 2021.

[47] JEDEC. *JESD89-2B - Test method for alpha source accelerated soft error rate.* https://www.jedec.org/standards-documents. 2021.

[48] JEDEC. *JESD89-3B - Test method for beam accelerated soft error rate.* https://www.jedec.org/standards-documents. 2021.

[49] JEDEC. *JESD89B - Measurement and reporting of alpha particle and terrestrial cosmic ray induced soft errors in semiconductor devices*. https://www.jedec.org/standards-documents. 2021.

[50] H. Jiang et al. « Designing soft-error-aware circuits with power and speed optimization ». In : *2018 IEEE International Reliability Physics Symposium (IRPS)*. 2018, P-SE.5-1-P-SE.5-5. DOI : 10.1109/IRPS.2018.8353692.

[51] Fernanda Kastensmidt et Paolo Rech. *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN : 3319143514.

[52] Ho Fai Ko et Nicola Nicolici. « Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging ». In : *2010 15th IEEE European Test Symposium*. 2010, p. 62-67. DOI : 10.1109/ETSYM.2010.5512781.

[53] Fotis Kostarelos, George Charitopoulos et Dionisios N. Pnevmatikatos. « Less is more: Increasing the scope of hardware debugging with compression ». In : *2017 Panhellenic Conference on Electronics and Telecommunications (PACET)*. 2017, p. 1-4. DOI : 10.1109/PACET.2017.8259958.

[54] Sandeep Kumar et Atin Mukherjee. « A Highly Robust and Low-Power Real-Time Double Node Upset Self-Healing Latch for Radiation-Prone Applications ». In : *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.12 (2021), p. 2076-2085. DOI : 10.1109/TVLSI.2021.3110135.

[55] Sandeep Kumar et Atin Mukherjee. « A Self-Healing, High Performance and Low-Cost Radiation Hardened Latch Design ». In : *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2021, p. 1-6. DOI : 10.1109/DFT52944.2021.9568359.

[56] *LANSCE - Los Alamos Neutron Science Center - United States*. https://lansce.lanl.gov/facilities/wnr/flight-paths/ice-house/. Accessed: 2021-10.

[57] Clark Lawrence T. et al. « A Soft Error Hardened by Design Microprocessor Implemented on Bulk 12 nm FinFET CMOS ». In : *2021 21th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*. 2021, 5 pp.-.

[58] *Learn More About FD-SOI*. https://www.st.com/content/st_com/en/about/innovation---technology/FD-SOI/learn-more-about-fd-soi.html. Accessed: 2021-10.

[59] A. Lindoso et al. « A Hybrid Fault-Tolerant LEON3 Soft Core Processor Implemented in Low-End SRAM FPGA ». In : *IEEE Transactions on Nuclear Science* 64.1 (2017), p. 374-381. DOI : 10.1109/TNS.2016.2636574.

[60] Gilles Louppe et Pierre Geurts. « Ensembles on Random Patches ». In : *Machine Learning and Knowledge Discovery in Databases*. Sous la dir. de Peter A. Flach, Tijl De Bie et Nello Cristianini. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 346-361. ISBN : 978-3-642-33460-3.

[61] Victor Malherbe et al. « Investigating the single-event-transient sensitivity of 65 nm clock trees with heavy ion irradiation and Monte-Carlo simulation ». In : *2016 IEEE International Reliability Physics Symposium (IRPS)*. 2016, SE-3-1-SE-3-5. DOI : 10.1109/IRPS.2016.7574639.

[62] Wassim Mansour et Raoul Velazco. « An Automated SEU Fault-Injection Method and Tool for HDL-Based Designs ». In : *IEEE Transactions on Nuclear Science* 60.4 (2013), p. 2728-2733. DOI : 10.1109/TNS.2013.2267097.

[63] P. Marshall et al. « Autonomous bit error rate testing at multi-gbit/s rates implemented in a 5AM SiGe circuit for radiation effects self test (CREST) ». In : *IEEE Transactions on Nuclear Science* 52.6 (2005), p. 2446-2454. DOI : 10.1109/TNS.2005.860740.

[64] *Microprocessor Development Tools - LAUTERBACH.* https://www.lauterbach.com/frames.html?home.html. Accessed: 2021-11.

[65] G. Miremadi et al. « Two software techniques for on-line error detection ». In : *1992 FTCS - The Twenty-Second International Symposium on Fault-Tolerant Computing.* Los Alamitos, CA, USA : IEEE Computer Society, 1992, p. 328-335. DOI : 10.1109/FTCS.1992.243568. URL : https://doi.ieeecomputersociety.org/10.1109/FTCS.1992.243568.

[66] Jani Mäkipää et Olivier Billoint. « FDSOI versus BULK CMOS at 28 nm node which technology for ultra-low power design? » In : *2013 IEEE International Symposium on Circuits and Systems (ISCAS).* 2013, p. 554-557. DOI : 10.1109/ISCAS.2013.6571903.

[67] J. M. Mogollón et al. « Metrics for the Measurement of the Quality of Stimuli in Radiation Testing Using Fast Hardware Emulation ». In : *IEEE Transactions on Nuclear Science* 60.4 (2013), p. 2456-2460. DOI : 10.1109/TNS.2013.2241079.

[68] J. M. Mogollón et al. « Real time SEU detection and diagnosis for safety or mission-critical ICs using HASH library-based fault dictionaries ». In : *2011 12th European Conference on Radiation and Its Effects on Components and Systems.* 2011, p. 705-710. DOI : 10.1109/RADECS.2011.6131367.

[69] B. Nicolescu et R. Velazco. « Detecting soft errors by a purely software approach: method, tools and experimental results ». In : *2003 Design, Automation and Test in Europe Conference and Exhibition.* 2003, 57-62 suppl. DOI : 10.1109/DATE.2003.1253806.

[70] N. Oh et E.J. McCluskey. « Error detection by selective procedure call duplication for low energy consumption ». In : *IEEE Transactions on Reliability* 51.4 (2002), p. 392-402. DOI : 10.1109/TR.2002.804735.

[71] *Open On-Chip Debugger - Provides on-chip programming and debugging support with a layered architecture of JTAG interface and TAP support.* https://openocd.org/. Accessed: 2021-10.

[72] L. Parra et al. « A New Hybrid Nonintrusive Error-Detection Technique Using Dual Control-Flow Monitoring ». In : *IEEE Transactions on Nuclear Science* 61.6 (2014), p. 3236-3243. DOI : 10.1109/TNS.2014.2361953.

[73] Luis Parra et al. « Efficient Mitigation of Data and Control Flow Errors in Microprocessors ». In : *IEEE Transactions on Nuclear Science* 61.4 (2014), p. 1590-1596. DOI : 10.1109/TNS.2014.2310492.

[74] *Paul Scherrer Institut (PSI) - Villigen - Switzerland.* https://www.psi.ch/en. Accessed: 2021-10.

[75] M. Peña-Fernandez et al. « Online Error Detection Through Trace Infrastructure in ARM Microprocessors ». In : *IEEE Transactions on Nuclear Science* 66.7 (2019), p. 1457-1464. DOI : 10.1109/TNS.2019.2921767.

[76] M. Peña-Fernandez et al. « PTM-based hybrid error-detection architecture for ARM microprocessors ». In : *Microelectronics Reliability* 88-90 (2018). 29th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis (ESREF 2018), p. 925-930. ISSN : 0026-2714. DOI : https://doi.org/10.1016/j.microrel.2018.07.074. URL : https://www.sciencedirect.com/science/article/pii/S0026271418306279.

[77] M. Peña-Fernandez et al. « The Use of Microprocessor Trace Infrastructures for Radiation-Induced Fault Diagnosis ». In : *IEEE Transactions on Nuclear Science* 67.1 (2020), p. 126-134. DOI : 10.1109/TNS.2019.2956204.

[78] M. Peña-Fernández et al. « Microprocessor Error Diagnosis by Trace Monitoring Under Laser Testing ». In : *IEEE Transactions on Nuclear Science* 68.8 (2021), p. 1651-1659. DOI : 10.1109/TNS.2021.3067554.

[79] M. Pignol. « DMT and DT2: two fault-tolerant architectures developed by CNES for COTS-based spacecraft supercomputers ». In : *12th IEEE International On-Line Testing Symposium (IOLTS'06)*. 2006, 10 pp.-. DOI : 10.1109/IOLTS.2006.24.

[80] M. Portela-Garcia et al. « On the use of embedded debug features for permanent and transient fault resilience in microprocessors ». In : *Microprocessors and Microsystems* 36.5 (2012). Special Issue on Design of Circuits and Integrated Systems, p. 334-343. ISSN : 0141-9331. DOI : https://doi.org/10.1016/j.micpro.2012.02.013. URL : https://www.sciencedirect.com/science/article/pii/S0141933112000300.

[81] *PULSCAN - Optical and Electronic Solutions for Testing and Failure Analysis - Gradignan - France*. https://www.pulscan.com/. Accessed: 2021-10.

[82] Heather Quinn et al. « Software Resilience and the Effectiveness of Software Mitigation in Microcontrollers ». In : *IEEE Transactions on Nuclear Science* 62.6 (2015), p. 2532-2538. DOI : 10.1109/TNS.2015.2496342.

[83] *RADiation Effects Facility (RADEF) - University of Jyväskylä - Finland*. www.jyu.fi/science/en/physics/research/infrastructures/accelerator-laboratory/radiation-effects-facility. Accessed: 2021-10.

[84] G.A. Reis et al. « SWIFT: software implemented fault tolerance ». In : *International Symposium on Code Generation and Optimization*. 2005, p. 243-254. DOI : 10.1109/CGO.2005.34.

[85] George A. Reis, Jonathan Chang et David I. August. « Automatic Instruction-Level Software-Only Recovery ». In : *IEEE Micro* 27.1 (2007), p. 36-47. DOI : 10.1109/MM.2007.4.

[86] *RISC-V Specifications - RISC-V International©*. https://riscv.org/technical/specifications/. Accessed: 2021-11.

[87] *Scikit-Learn - Machine Learning in Python*. https://www.python.org/. Accessed: 2021-10.

[88] *SCR1 Microcontroller Core - Syntacore*. https://syntacore.com/page/products/processor-ip/scr1. Accessed: 2020-11.

[89] Norbert Seifert et al. « Soft Error Rate Improvements in 14-nm Technology Featuring Second-Generation 3D Tri-Gate Transistors ». In : *IEEE Transactions on Nuclear Science* 62.6 (2015), p. 2570-2577. DOI : 10.1109/TNS.2015.2495130.

[90] V.P. Srini. « Special Feature: Fault Diagnosis of Microprocessor Systems ». In : *Computer* 10.1 (1977), p. 60-65. DOI : 10.1109/C-M.1977.217500.

[91] Hai Tao SUN. « First failure data capture in embedded system ». In : *2007 IEEE International Conference on Electro/Information Technology*. 2007, p. 183-187. DOI : 10.1109/EIT.2007.4374434.

[92] *TaskGroup RISC-V Nexus Trace*. https://github.com/riscv-non-isa/tg-nexus-trace. Accessed: 2021-11.

[93] *The RISC-V Instruction Set Manual*. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf. Accessed: 2020-11.

[94] Rodrigo TRAVESSINI et al. « Processor core profiling for SEU effect analysis ». In : *2018 IEEE 19th Latin-American Test Symposium (LATS)*. 2018, p. 1-6. DOI : 10.1109/LATW.2018.8347235.

[95] *TRIUMF Neutron Irradiation Facility - Canada's particle accelerator center*. www.triumf.ca/neutron-irradiation-facility. Accessed: 2021-10.

[96] R. VELAZCO, S. REZGUI et R. ECOFFET. « Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection ». In : *IEEE Transactions on Nuclear Science* 47.6 (2000), p. 2405-2411. DOI : 10.1109/23.903784.

[97] Su WEI, Fan TONGSHUN et Du MINGFANG. « Research for digital circuit fault testing and diagnosis techniques ». In : *2009 International Conference on Test and Measurement*. T. 1. 2009, p. 330-333. DOI : 10.1109/ICTM.2009.5412926.

[98] James R. WERTZ et al. « Methods for Achieving Dramatic Reductions in Space Mission Cost ». In : 2011.

[99] *Xcelium Logic Simulation - CADENCE*. https://www.cadence.com/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/xcelium-simulator.html. Accessed: 2021-10.

[100] *Xilinx Kintex-7 FPGA KC705 Evaluation Kit*. https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html. Accessed: 2021-10.

[101] H. ZHANG et al. « Frequency Dependence of Heavy-Ion-Induced Single-Event Responses of Flip-Flops in a 16-nm Bulk FinFET Technology ». In : *IEEE Transactions on Nuclear Science* 65.1 (2018), p. 413-417. DOI : 10.1109/TNS.2017.2779785.