THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Electronique

École doctorale I2S — Information Structures et Systèmes

IES — Institut d'Electronique et des Systèmes (UMR 5214)

Radiation reliability analysis of FPGA-based systems: testing methodologies and analytical approaches

Présentée par Gaëtan BRICAS
Le 25 novembre 2022

Sous la direction de Jérôme BOCH

Devant le jury composé de

M. Luca STERPONE, Professeur, École polytechnique de Turin	Rapporteur
M. Luis ENTRENA, Professeur, Université de Madrid	Rapporteur
M. Jérôme BOCH, Professeur, Université de Montpellier	Directeur de thèse
M. Georgios TSILIGIANNIS, Dr., Ingénieur, Think Silicon	Examinateur et co-encadrant
M. Charles DETEMMERMAN, Ingénieur, Thales Alenia Space	Examinateur
M. Luigi DILILLO, Chercheur, Université de Montpellier	Examinateur
M. Philippe CHRISTOL, Professeur, Université de Montpellier	Examinateur

UNIVERSITÉ DE MONTPELLIER

Acknowledgment

This thesis project has been an incredible adventure, full of pitfalls and unforeseen events, but above all it has been an exciting exploration in the world of electronics and radiation, full of great encounters and enriching experiences. This project would never have been possible without the support of my supervisors, my colleagues, my friends and my family.

I would like to thank Antoine Touboul, who initiated this project, for his guidance, support and advice, without which I would not have reached this point today. Always with humor, Antoine transmitted his passions to me and brought me a lot on a scientific and human level. Unfortunately, our adventure together has been far too short. I hope that he would have been proud of the outcome of the project he gave birth to.

Thanks to Georgios Tsilligiannis, for his unconditional support and the time he gave me despite the distance and his obligations. I thank him for never letting me down and for sharing his experience and expertise that served as a compass throughout the project.

I would like to acknowledge my supervisor Jérôme Boch, for his commitment, his confidence and his support at all levels. His advice proved to be essential to improve the presentation and the valorization of our work.

Thank you to Alain Michez for his teachings and for putting me on the path of electronics and radiation. Our meeting seven years ago was a major turning point in my professional life.

I would like to thank Tadec Maraine for his continuous support, for his precious help during the X-ray experiments in the PRESERVE facility and for his solutions to all problems. Many thanks to the rest of the Radiac team for welcoming me and sharing their expertise on the effects of radiation, especially Frederic Saigné and Frederic Wrobel. Thanks to Julie, Daylet and Jean-Francois for their administrative support.

Thanks to all my colleagues, Kimmo, Salvator, Israel, Catherine, Ygor, Hoang, Arthur, Samir, Thibaut, Alain, Vincent, Hiba, Cleiton, Matthieu, Alejandro, Ismaël, Lucas, Douglas, and André for their presence and their good mood.

Thanks to my brother Samuel, for the time he spent helping me with all the programming aspects of this project. His teachings and expertise in Java code were crucial to the success of this project.

Thank you to my sweet Roxane, for her love and her daily presence that have brightened my steps since high school.

Thank you to my long-time friends for accompanying me and entertaining me during these last fifteen years.

Finally, I would like to thank my family, my grandmother, my two brothers, Alexandre and Samuel, and my parents Nicolas and Elisabeth for giving me the most beautiful childhood that one can dream of and for having transmitted to me their values which made me what I am today.

Resume

Au cours du dernier siècle, l'utilisation de l'électronique dans les sociétés modernes n'a cessé d'augmenter. Durant ces dernières décennies, la croissance du secteur de l'électronique s'est accélérée ; l'électronique affecte maintenant quasiment tous les secteurs de la société. Les systèmes de contrôle, traditionnellement mécaniques, sont peu à peu remplacés par des systèmes électrifiés dans de nombreuses applications industrielles, notamment celles liées au transport, à l'automobile et à l'aviation. D'autre part, l'humanité s'est rendue de plus en plus dépendante de systèmes informatiques tels que le système bancaire, les systèmes de télécommunication et de contrôle du réseau électrique, qui s'appuie essentiellement sur des datacenters et sur des systèmes décentralisés tels que les satellites. La question de la fiabilité des systèmes électroniques est alors devenue un enjeu essentiel de notre ère.

En 1962, la première étude prédisant le fait que les radiations pouvaient générer des anomalies dans les composants à base de semi-conducteur fut publiée. Depuis, deux principaux types d'effets ont été identifiés. D'un côté, les effets de dose totale ionisante, causés par l'interaction avec de multiples particules chargés (protons, particule alpha, ions lourds) ou des photons de hautes énergies (rayons X, rayons Gamma) qui induisent une dégradation progressive des propriétés électrique du composant. De l'autre, les effets singuliers (SEE), générés par l'interaction avec une particule unique, déposant suffisamment d'énergie dans le volume de silicium pour inverser l'état d'un transistor, pouvant engendrer différents types de défaillances. Initialement, ces sources de défaillances étaient uniquement prises en compte pour le domaine du spatial et pour les équipements militaires. En effet, les niveaux de radiation dans l'espace sont beaucoup plus hauts que sur terre à cause des particules émises par le soleil et le reste de l'univers, tandis que l'environnement terrestre est en grande partie protégé par la magnétosphère (répulsion des particules chargées) et de l'atmosphère terrestre (interactions avec les particules d'air). À partir des années 90, ces considérations se sont étendues aux altitudes de l'avionique et enfin au niveau du sol. En effet, les particules venant de l'espace qui parviennent à pénétrer le bouclier magnétique de la terre, entrent en collision avec les atomes de l'atmosphère et créent une pluie de particules secondaires qui peuvent également provoquer des défaillances dans les systèmes électroniques, notamment à hautes altitudes. Aujourd'hui, les progrès technologiques réalisés en physique des semi-conducteurs ont permis d'atteindre des niveaux d'intégration jusqu'à l'échelle nanométrique. De par cette réduction d'échelle technologique et de par la complexité croissante des systèmes électroniques, la contrainte radiative est maintenant prise en compte, non seulement pour les applications liées à des forts niveaux de radiations (satellites, aviations, centrale nucléaire) mais également pour tous les systèmes électroniques mettant en jeu la sécurité des êtres humains (automobile, transport, médical).

Les progrès réalisés dans le domaine de la microélectronique ont également entraîné une large digitalisation du secteur de l'électronique. De nombreuses fonctions, historiquement gérées par des circuits analogiques, sont progressivement implémentées sur des circuits intégrés numériques, souvent plus performant et plus flexibles que leurs homologues analogiques. Dans l'industrie spatiale par exemple, de nombreuses fonctions autour du traitement du signal, du filtrage et de la formation de faisceau, sont désormais réalisées par des calculateurs numériques.

Pour répondre à la demande croissante de bande passante, de nombreux systèmes digitaux requièrent à la fois un haut niveau de parallélisation des opérations et un certain niveau de spécialisation auxquels les CPU et GPU standards ne peuvent répondre. Traditionnellement, des circuits intégrés pour application spécifiques (ASIC) sont conçus dans ce but. Cependant, les coûts non-récurrents d'ingénierie impliqués dans le développement et la fabrication d'un nouveau circuit intégré sont considérables et tendent à augmenter avec la réduction d'échelle technologique.

En 1985, un nouveau type de composant fut créé, les Field Programmable Gate Array (FPGA). Les FPGAs sont des circuits intégrés qui peuvent être reprogrammés après fabrication pour implémenter tout type de circuits numériques. Leur principe de fonctionnement repose sur une matrice de blocs logiques programmables et d'un réseau d'interconnexions configurables utilisé pour relier les blocs logiques entre eux. La fonctionnalité du FPGA est spécifiée en téléversant un fichier binaire dans une mémoire spécifique du composant. Chaque cellule de cette mémoire sert alors à définir la fonctionnalité d'un des blocs logiques ou à activer l'une des interconnexions de la matrice de routage. Malgré des performances généralement plus faibles, les FPGAs représentent aujourd'hui une réelle alternative aux ASICs grâce à leur coût réduit pour les faibles volumes de production, leur faible délai de mise sur le marché et leur reprogrammabilité. Cependant, cette reprogrammabilité s'accompagne d'une sensibilité accrue aux radiations. En effet, les cellules mémoire utilisées pour stocker la configuration du composant peuvent être affectées par des événements singuliers ou être dégradées par des effets de dose ionisante selon le type de technologie utilisé. La corruption de cette mémoire de configuration peut alors entraîner une modification de la topologie du circuit implémenté et impacter sévèrement la fiabilité du système. Des techniques de durcissement au niveau du processus de fabrication ou au niveau du layout peuvent être appliquées pour surmonter cet inconvénient, mais les composants durcis aux radiations souffrent généralement d'un retard technologique significatif et de coûts beaucoup plus élevés par rapport aux composants commerciaux (COTS). Les FPGA COTS restent alors une alternative viable, même pour une utilisation dans des environnements à haute contrainte radiative. Des techniques de mitigation au niveau du design du circuit doivent alors être adéquatement appliquées pour renforcer la fiabilité du système. La fiabilité du composant face aux radiations doit alors être évaluée à travers des campagnes de tests radiations utilisant des faisceaux de particules accélérées ou des sources radiatives dans des installations spécialisées.

Dans ce contexte, ce travail de thèse se focalise sur des méthodologies de test pour analyser la sensibilité aux radiations des systèmes basés sur FPGA. De par leur flexibilité, l'analyse de fiabilité sur ces composants est particulièrement complexe, car leur sensibilité aux radiations est entièrement conditionnée par le circuit qui y est implémenté. En effet, elle dépend d'une part de la sensibilité intrinsèque des ressources du composant (à la fois aux effets de dose ionisante et aux effets singuliers) et d'autre part à la manière dont les différentes perturbations induites peuvent impacter la fonctionnalité du système. Les méthodologies présentes dans l'état de l'art ont montré un certain nombre de limitations pour faire le lien entre la sensibilité intrinsèque du FPGA et celle du design implémenté. L'objectif de cette thèse est d'améliorer les méthodologies de tests radiations pour surmonter ces difficultés. Les méthodologies proposées se basent sur une approche benchmark : des structures de test spécifiques sont développées et implémentées sur le composant pendant les tests radiations. Cette approche fournit alors à la fois une bonne visibilité sur les mécanismes de défaillances internes et des moyens de comparer quantitativement la

sensibilité aux radiations entre FPGA de différents fabricants et de différentes familles. Cette approche fournit également aux concepteurs, des outils essentiels pour analyser la sensibilité de leurs circuits et pour améliorer leur fiabilité. L'approche benchmark proposée est également complétée par une méthodologie, basée sur un outil logiciel, permettant de prédire la sensibilité aux effets singuliers sur un circuit arbitraire à travers une analyse en profondeur de la netlist du circuit.

La première méthodologie proposée dans cette étude se focalise sur l'analyse de la dérive des propriétés électriques des FPGA en fonction du niveau de dose ionisante absorbé. Ces effets de doses sont en fait causés par les charges électriques libres générées dans les matériaux lors de l'interaction avec un photon de haute énergie ou une particule chargée. Une partie des charges générées se retrouve piégée dans les oxydes du composant, venant ainsi altérer les champs électriques et la mobilité des porteurs dans le canal des transistors. L'accumulation de ces charges (dominée par les charges positives) vient donc modifier les caractéristiques électriques du transistor (tension de seuil et transconductance). Ces effets se traduisent au niveau du FPGA par une augmentation de la consommation globale du composant, mais également par une dégradation des temps de propagation des portes logiques. Pour caractériser ces dégradations du temps de propagation, la méthodologie proposée emploie des structures de tests spécifiques permettant de mesurer indépendamment la dégradation en temps de propagation pour chaque type de ressource du FPGA (LUT, CARRY, DSP, PIPs). Le temps de propagation à travers ces structures est évalué grâce à une structure de test interne, utilisant les oscillateurs programmables du FPGA, permettant une implémentation rapide, précise et peu coûteuse. Cette méthode de test est appliquée à trois familles de FPGA lors de test sous faisceau de rayons X pour démontrer les bénéfices apportés. L'extraction de la dégradation des temps de propagation, indépendamment pour chaque type de ressource, permet d'une part une meilleure standardisation des résultats (en s'affranchissant de l'influence des paramètres d'implémentation sur les résultats de test) et une meilleur réutilisabilité : en combinant les dérives individuelles, un designer peut estimer la dérive de temps de propagation subit par les chemins critiques de son design en se basant uniquement sur le nombre et le type de ressources qui les composent.

L'autre méthodologie proposée s'intéresse à l'impact des effets singuliers sur la fiabilité des systèmes implémentés sur FPGA. Sur les technologies FPGA, on recense principalement trois mécanismes de défaillances :

- **Les SET** : des impulsions de courant générées dans la logique combinatoire ou sur les arbres de distribution de signaux, qui peuvent se propager sous la forme d'impulsions de tension à travers les portes logiques et potentiellement corrompre l'état d'une ou plusieurs bascules.
- **Les SEU** : le contenu d'une cellule mémoire est inversé, pouvant corrompre les données stockées. Les SEU peuvent également affecter la mémoire de configuration du FPGA lorsque celle-ci est basée sur une technologie SRAM. De tels événements peuvent alors entraîner une modification structurelle du circuit implémenté, en désactivant une piste utilisée, en activant une connexion indésirable entre deux nœuds distincts ou encore en modifiant l'équation logique réalisée par les LUT ou les autres blocs spécialisés.

- **Les SEL** : un chemin de conduction est créé entre la source d'alimentation et la masse, entrainant une élévation de la consommation, un échauffement et généralement une interruption de la fonctionnalité de toute une partie du FPGA.

La fiabilité d'un circuit implémenté sur FPGA est donc particulièrement difficile à analyser, car celle-ci nécessite à la fois de bien identifier la sensibilité intrinsèque pour chaque type de ressource du FPGA, de comprendre l'influence de différents paramètres du circuit sur la sensibilité relative à ces différents effets, sur la propagation des effets transitoires et sur le masquage logique des erreurs. L'analyse de cette sensibilité aux effets singuliers repose aujourd'hui essentiellement sur des tests sous faisceau de particules accélérées (neutrons, protons, ou ions lourds). Face à la complexité des phénomènes mis en jeu, deux principales approches ont émergé. La première consiste à utiliser des structures de test monomorphiques pour extraire la sensibilité intrinsèque de chaque ressource. Cependant, ces structures de tests offrent généralement une diversité très faible dans les paramètres de circuits, masquant ainsi des mécanismes essentiels mis en jeu dans des circuits beaucoup plus complexe. La deuxième approche, souvent utilisée dans un but de validation, consiste à réaliser ces tests en implémentant directement le circuit utilisé par l'application finale. Cette approche peut néanmoins présenter différents inconvénients de par la complexité des circuits testés. En effet, lorsque le nombre d'état du circuit est trop important, chaque état de circuit ne peut être suffisamment exposé au cours d'une campagne de tests pour être suffisamment représentatif des toutes les potentielles défaillances qui peuvent apparaître au cours de la mission. De plus, la complexité du circuit tend à réduire la proportion des erreurs qui peuvent être effectivement observées depuis l'extérieur du composant, réduisant alors drastiquement la statistique obtenue avec ce type de test. La méthodologie proposée dans cette étude cherche alors à venir compléter ces deux approches en proposant de nouvelles structures et de nouvelles techniques de tests pour améliorer à la fois la compréhension des mécanismes de défaillance mis en jeu et la capacité à estimer la fiabilité d'un circuit. Cette méthodologie s'appuie sur le principe du benchmark : un jeu de circuits, représentatif des applications réelles, est spécifiquement développé pour les tests radiations. Ce jeu de circuit peut alors être implémenté sur différentes familles de FPGA pour comparer leur sensibilité aux radiations dans des conditions réelles d'utilisation tout en fournissant des métriques permettant d'appréhender la sensibilité relative entre les différents types de ressources. Le benchmark développé dans cette étude s'articule autour des opérateurs arithmétiques. Les opérateurs arithmétiques sont des structures très largement utilisées dans tous les systèmes liés au traitement du signal, au traitement de l'image, à la cryptographie, ou à l'intelligence artificielle. Ces opérateurs peuvent alors être implémentés de manières très différentes, utilisant différents types de ressources ou différentes topologies de circuits amenant à plusieurs compromis entre performance et consommation. En constituant un benchmark de différentes implémentations d'un opérateur arithmétique, la sensibilité relative aux radiations peut être analysée pour chaque implémentation pour identifier l'influence des paramètres de design sur la sensibilité aux radiations. Ainsi, des recommandations pour améliorer la tenue aux radiations peuvent être identifiés et étendues à des circuits plus complexes. La sensibilité aux radiations apparaît alors comme un nouveau facteur pouvant amener les designers à reconsidérer le choix d'implémentations de leurs circuits. Pour contrôler la fonctionnalité de ces opérateurs lors des tests radiations, une structure interne d'auto-test est utilisée. Celle-ci permet de conduire les tests radiations avec des cartes de développement du commerce et une instrumentation

limitée, simplifiant et réduisant ainsi les coûts de l'installation de test. Pour évaluer l'efficacité et les avantages apportés par cette méthodologie, différentes campagnes de tests sous faisceau de particules accélérées (neutrons et protons) ont été conduites sur différentes familles de FPGA. Les résultats de ces expériences montrent notamment que la sensibilité relative entre les différentes composantes du benchmark peut être très différents entre les différentes technologies testées, renforçant ainsi l'idée que les recommandations visant à améliorer la fiabilité des circuits doivent être adaptés à chaque type de FPGA. Pour les FPGA SRAM, une forte dominance des défaillances liées aux corruptions de la mémoire de configuration a pu être observée. De par la complexité du réseau de routage programmable, la susceptibilité d'un circuit à ce type d'erreur est très difficile à prédire. Face à cette difficulté, de nouveaux outils d'analyse ont émergé tel que l'injection de faute. L'injection de faute consiste à reproduire artificiellement l'effet d'un SEU sur la mémoire de configuration en introduisant des erreurs dans le bitstream programmé sur le FPGA. En injectant indépendamment un nombre significatif de fautes dans la mémoire de configuration tout en contrôlant la fonctionnalité du circuit, le nombre de bits de configuration susceptibles de perturber le fonctionnement d'un circuit donné peut être identifié. En multipliant ensuite ce nombre par la section efficace d'une cellule mémoire, la sensibilité du circuit à ce type d'événement peut être estimée. Des campagnes d'injection de faute ont été systématiquement menées pour chaque composante du benchmark et les résultats comparés à ceux issues des tests sous faisceau de particules. Une très bonne corrélation des résultats a pu être observée, confirmant ainsi la capacité de l'approche proposée à estimer la sensibilité d'un circuit arbitraire aux corruptions de la mémoire de configuration. Néanmoins, le processus d'injection de fautes présente certains inconvénients. D'une part, la durée d'une campagne de test peut prendre plusieurs jours ou semaines pour obtenir des résultats statistiquement significatifs. D'autre part, ce type d'approche, tout comme les tests sous faisceau de particules, ne fournit qu'une estimation de la sensibilité globale du circuit et ne permet pas de gagner en visibilité sur les zones les plus sensibles du circuit et sur les potentielles modifications qui peuvent être apportés pour améliorer sa fiabilité.

Dans la dernière partie de cette étude, une nouvelle méthodologie d'analyse de fiabilité des circuits implémentés sur FPGA SRAM a été explorée. Cette méthodologie s'appuie sur l'analyse détaillée de la netlist du circuit. Un logiciel est développé, permettant de parcourir la netlist du circuit pour extraire l'ensemble des bits de configuration susceptibles de modifier son comportement. En se basant sur les résultats d'ingénierie inverse du bitstream d'un FPGA et sur des tests d'injection de fautes localisés, le modèle de défaillance du composant a pu être établie. Celui-ci identifie pour chaque bit de configuration, le type de modifications architecturales engendrés par son inversion ainsi que les conditions dans lesquelles cette inversion résulte effectivement en une modification d'un ou plusieurs signaux du circuit. En se basant sur ce modèle de défaillance, le logiciel développé parcourt l'ensemble des branches du circuit pour identifier les bits de configuration critiques pour le système. Une des contributions majeures apportée par l'approche développée est la prise en compte de la charge de travail du circuit pour évaluer la criticité de chacun des bits de configuration. L'état de l'ensemble des signaux du circuit et leur évolution au cours d'un scénario définie par l'utilisateur est extrait par simulation comportementale. Ces données sont utilisées d'une part pour identifier, pour chaque bit de configuration, les instants dans la simulation où la modification structurelle engendrée génère effectivement une erreur sur l'un des signaux du circuit et d'autre part, pour identifier les instants

où ces erreurs se propagent à travers le circuit pour atteindre l'une des sorties primaires du circuit. Cette méthode a été appliquée à l'ensemble des composantes du benchmark testé par injection de fautes et sous faisceau de particules. Les résultats montrent alors que le logiciel et non seulement capable d'estimer rapidement et précisément la sensibilité du circuit, mais également d'identifier ses zones sensibles. Ce logiciel pourrait alors fournir au designer, des outils précieux pour évaluer, dès les premières phases de design, la sensibilité globale de leurs circuits et leurs principales vulnérabilités permettant alors de sélectionner judicieusement les techniques de mitigation à appliquer et les mesures à prendre pour améliorer la fiabilité.

Acronyms and definitions

ADC- Analog to Digital Converter

API- Application Programming Interface

ASIC- Application Specific Integrated Circuit

ATMR- Approximate Triple Module Redundancy

ATPG- Automatic Test Pattern Generation

BEL- Basic Element of Logic

BIST- Built-In Self-Test

BRAM- Block RAM

BTMR- Block Triple Module Redundancy

CAD- Computer Aided Design

CED- Current Error Detection

CME- Coronal Mass Ejections

CMIC- Configuration Memory Integrity Check

CMOS- Complementary Metal Oxide Silicon

COTS- Commercial Off The Shelf

CRAM- Configuration Memory

CRC- Cyclic Redundancy Check

DA- Distributed Arithmetic

DAC- Digital to Analog Converter

DD- Displacement Damage

DICE- Dual Inter-Locked Cell

DRAM- Dynamic Random-Access Memory

DSP- Digital Signal Processing

DTMR- Distributed Triple Module Redundancy

DUT- Device Under Test

DWC- Duplication With Comparison

ECC- Error Correction Code

EDA- Electronic Design Automation

EDAC- Error Detection and Correction Code

EEPROM- Electrically-Erasable Programmable Read-Only Memory

ELT- Enclosed Layout Transistors

FF- Flip-flop

FI- Fault Injection

FIR- Finite Impulse Response

FIT- Failure In Time

FPGA- Field Programmable Gate Array

FSM- Finite State Machine

GCR- Galactic Cosmic Rays

GEO- Geosynchronous orbit

GTMR- Global Triple Module Redundancy

HDL- Hardware Description Language

ICAP- Internal Configuration Access Port

IP- Intellectual Property

JTAG- Joint Test Action Group

LEO- Low Earth Orbit

LET- Linear Energy Transfer

LFA- Linear Frame Address

LHC- Large Hadrons Collider

LSB- Least Significant Bit

LTMR- Local Triple Module Redundancy

LUT- Look-Up Table

LVTTL- Low Voltage Transistor-Transistor Logic

MBU- Multiple Bit Upset

MEO- Medium Earth Orbit

MMCM- Mixed Mode Clock Manager

MOSFET- Metal Oxide Silicon Field Effect Transistor

MSB- Most Significant Bit

MTTF- Mean Time To Failure

MUX- Multiplexer

NCD- Native Circuit Description

NRC- Non-Recurring Costs

NRE- Non-Recurring Engineering

PCB- Printed Circuit Board

PFA- Physical Frame Address

PLD- Programmable Logic Devices

PLL- Phase Locked Loop

RAM- Random-Access Memory

RILC- Radiation Induced Leakage Current

ROM- Read Only Memory

RTL - Register Transfer Level

SAA- South Atlantic Anomaly

SBU- Single Bit Upset

SEB- Single Event Burnout

SEC-DEC- Single Error Correction and Double Error Detection

SEE- Single Event Effects

SEFI- Single Event Functional Interrupt

SEGR- Single Event Gate Rupture

SEL- Single Event Latchup

SEM- Soft Error Mitigation

SEP- Solar Energetic Particles

SET- Single Event Transient

SEU- Single Event Upset

SoC- System on Chip

SOI- Silicon-On-Insulator

SRAM- Static Random-Access Memory

STA- Static Timing Analysis

STI- Shallow Trench Isolation

STMR- Selective Triple Module Redundancy

TID- Total Ionizing Dose

TMR- Triple Module Redundancy

TNID- Total Non-Ionizing Dose

UART- Universal Asynchronous Receiver/Transmitter

AXI- Advanced eXtensible Interconnect

CERN- European Organization for Nuclear Research

FPGA architecture terminology *(Xilinx 7 series):*

Fabric: the internal configurable structure of the FPGA. It refers to the interconnection matrix and the configurable logic blocks, often used in contrast to specialized blocks (PLL, BRAM, DSP, etc.).

Tile: this is the elementary building blocks into which the FPGA structure is divided. There are several tile types depending on the resources it contains. The most common tile contains two configurable logic blocks.

Configurable Logic Block (CLB): a collection of two slices.

Slices: a collection of 4 LUTs, 1 CARRY4, 8 flip-flops, 3 logic multiplexers and 25 routing multiplexers. There are two types of slices, SLICEL and SLICEM. SLICEM type is supplemented with additional features: LUT used as dual port memory (LUTRAM) or as shift-register (SRL).

Site: A group of related elements and their connectivity. Regular sites contain one slice and other site type may contain specialized blocks such as PLL, DSP, BRAM, IOs, etc.

BEL: Basic Elements. BELs are the smallest, indivisible, representable component in the fabric of an FPGA. There are two kinds of BELs, Logic BELs (LUT, flip-flop, Carry4, DSP etc.) and Routing BELs (intra-slice multiplexers).

Extra-slice routing resource: used to drive signals from site to site, mainly composed of nodes and PIPs.

Intra-slice routing resource: used to drive signals inside a slice. Also called routing multiplexers.

Nodes: Physical representation of metal tracks separating two PIPs or site pins. They are composed of wires.

Wire: a segment of metal connection.

PIP: programmable interconnection point. Configurable connection contains in switchboxes.

Switchbox: A configurable routing matrix used to connect different extra-slice nodes together.

TABLE OF CONTENT

ACKNOWLEDGMENT **1**

RESUME **2**

ACRONYMS AND DEFINITIONS **8**

TABLE OF CONTENT **11**

INTRODUCTION **16**

1. RADIATION: ENVIRONMENTS AND EFFECTS ON ELECTRONICS **19**

1.1. Radiation environment 19

1.1.1. Space radiation environment 19

1.1.1.1. Galactic cosmic rays 19

1.1.1.2. Solar energetic particles 20

1.1.1.3. Trapped radiations 21

1.1.1.4. Radiation environment modeling tools 22

1.1.2. Atmospheric and terrestrial radiation environments 23

1.1.3. Artificial radiation environments 24

1.1.3.1. Nuclear power plants 24

1.1.3.2. Medical radiation environments 24

1.1.3.3. Particle accelerators 25

1.1.3.4. Radiation sources 26

1.2. Radiation effects on electronics 27

1.2.1. Radiation-matter interactions 27

1.2.1.1. Photon-matter interactions 27

1.2.1.2. Nuclear interactions 28

1.2.1.3. Coulombic interactions 29

1.2.1.4. Energy transfers in matter 30

1.2.2. Cumulative effects 31

1.2.2.1. Total Ionizing Dose 31

1.2.2.2. Total Non-Ionizing Dose 34

1.2.3. Single event effects 35

1.2.3.1. Single Event Transient 36

1.2.3.2. Single Event Upset 36

1.2.3.3. Single Event Latch-up 37

1.2.3.4. SEE sensitivity metrics 38
1.3. Conclusion 39
2. FPGA ARCHITECTURE 40
2.1. Definition and principle 40
2.1.1. Advantages and drawbacks 40
2.1.2. Development workflow 41
2.2. Architecture description 43
2.2.1. Overview 43
2.2.2. Configurable logic blocks 44
2.2.3. Configurable Routing matrix 45
2.2.4. Specialized Blocks 46
2.2.5. Configuration memory cell technologies 47
2.2.5.1. Static RAM 47
2.2.5.2. Flash memory 48
2.2.5.3. Antifuse 49
2.3. Radiation effects on FPGAs 49
2.3.1. Radiation effects on configuration memory cells 49
2.3.1.1. SRAM 49
2.3.1.2. Flash 50
2.3.1.3. Antifuse 51
2.3.1.4. Summary 51
2.3.2. Total Ionizing Dose effect – Parametric degradation 51
2.3.3. Single event effects 52
2.3.4. Summary of radiation effect on FPGAs 55
2.4. Radiation hardening 57
2.4.1. Process based hardening 58
2.4.2. Layout based hardening 58
2.4.3. Circuit based hardening 60
2.4.4. Memory hardening 62
2.5. Conclusion 64
3. FPGA TESTING METHODOLOGIES FOR TID EFFECTS ASSESSMENT 65
3.1. From testing methodologies to Benchmarking 65
3.1.1. State of the art of testing methodologies 65
3.1.2. Extension of degradation evaluation to all resources 66

3.1.3. Benchmarking structures 67
3.2. Test setup 68
3.2.1. Propagation delay measurement 68
3.2.2. Device selection 70
3.2.3. X-ray Generator and parameters 72
3.3. Radiation test results 73
3.3.1. Propagation delay degradation results 73
3.3.2. Thermal effect consideration 79
3.3.3. Power consumption 82
3.3.4. Design margins consideration 84
3.4. Conclusion 84

4. FPGA TESTING METHODOLOGIES FOR SEE ASSESSMENT 86
4.1. State of the art methodologies 86
4.1.1. SEE testing challenges 86
4.1.2. Configuration memory sensitivity evaluation 87
4.1.3. Primitive level sensitivity evaluation 89
4.1.4. Final application testing 92
4.1.5. Radiation test benchmarking 93
4.2. Benchmarking requirements 93
4.3. Benchmark selection 94
4.3.1. Hardware implementation of arithmetic operators 95
4.3.1.1. Binary addition 95
4.3.1.2. Ternary adder 96
4.3.1.3. Binary multiplier principle 97
4.3.1.4. Carry-Save Multiplier 97
4.3.1.5. Speed optimized multiplier 98
4.3.1.6. Area optimized multiplier 100
4.3.1.7. Booth encoding optimization 100
4.3.1.8. Constant multiplier optimization 101
4.3.1.9. Selected multiplier implementations 101
4.3.1.10. Finite Impulse Response filter 102
4.3.1.11. Distributed arithmetic 104
4.4. Built-in self-test 105
4.4.1. Test pattern selection 105

4.4.2. BIST architecture 106
4.4.3. Error formatting 107
4.4.4. Test setup and procedure 108
4.5. Experimental results 109
4.5.1. First campaign: influence of timing constraints 110
4.5.1.1. Setup details 110
4.5.1.2. Results 111
4.5.2. Second campaign: improvements and extensions 116
4.5.2.1. Test setup modifications 116
4.5.2.2. Results 117
4.5.3. Third campaign: filter's structural parameters 121
4.5.3.1. Test setup 121
4.5.3.2. Results 123
4.6. Fault injection 127
4.6.1. State of the art methodologies 127
4.6.2. Fault injection procedure 128
4.6.3. Experimental results 130
4.6.3.1. Statistical analysis 130
4.6.3.2. Application to the second test campaign benchmark 131
4.6.3.3. Application to the third test campaign 132
4.7. Conclusion 133

5. SEE SUSCEPTIBILITY EVALUATION TOOL 136

5.1. State of the art SEE susceptibility prediction tools 137
5.1.1. Bitstream reverse engineering 137
5.1.2. Vulnerability analysis 138
5.1.3. SEE susceptibility prediction tools 139
5.1.4. CAD tools and APIs for fine-grained circuit manipulation and analysis 140
5.2. Failure model establishment 143
5.2.1. Decoded bitstream database 143
5.2.2. Localized fault injection 144
5.2.3. Switchboxes 145
5.2.3.1. Failure model 145
5.2.3.2. Activation conditions 148
5.2.4. Intra-slice routing multiplexers 148

5.2.4.1. Failure model 149
5.2.4.2. Activation conditions 150
5.2.5. Logic multiplexer 151
5.2.6. Carry Logic 152
5.2.7. Look Up Tables 152
5.2.7.1. Activation conditions 153
5.2.7.2. Propagation conditions 155
5.2.8. Flip-flops 157
5.2.9. Shift Register LUT 157
5.2.9.1. Failure modes 158
5.2.9.2. Propagation conditions 158
5.2.10. Block RAM 158
5.2.10.1. Failure modes 158
5.2.10.2. Activation conditions 159
5.2.10.3. Propagation conditions 159
5.2.11. Unsupported primitives 159
5.3. Netlist analysis and critical bits extraction 160
5.3.1. Data structures 160
5.3.2. Workload extraction 163
5.3.3. Circuit navigation algorithms 164
5.4. Evaluation of the tool capacities 169
5.4.1. Exhaustive fault injection 169
5.4.2. Comparison with experimental results 171
5.4.3. Execution time 175
5.5. Conclusion 176
GENERAL CONCLUSION 178
PUBLICATIONS 181
REFERENCES 182

Introduction

During the last century, the use of electronics has been increasing in modern societies. In the last decade, this growth of the electronics sector has greatly accelerated and now affects almost all sectors of societies. The replacement of traditional mechanical controls by electrified systems is taking place in many sectors of industry and transportation, especially in the automotive and avionics industries. The human being, is also more and more dependent on computer systems, such as the banking systems, the telecommunication systems, the electrical network, etc. which rely on data centers and decentralized computers such as satellites. The question of the reliability of electronic systems has thus become one major issue of our era.

In 1962, the first study predicting that radiation could cause anomalies in semiconductor-based components was published [1]. Since then, two main types of effects have been identified. Total Ionizing Dose (TID) effects, caused by the interaction with multiple charged particles or high energy photons, inducing a progressive degradation of the component parameters, and Single Event Effects (SEE), generated by the interaction with a single particle depositing enough energy to corrupt the content of a memory elements. At first, these sources of failure were considered only in the military field and for space systems. Indeed, the radiation levels in space are much higher than on earth due to the particles emitted by the sun and the rest of the universe, while the ground environment is relatively protected by the earth magnetosphere and atmosphere. From the 90's, these considerations have been extended to the avionic altitudes and then to the ground level. Particles coming from space that manage to penetrate the magnetic shield of the earth, collide with the atoms of the atmosphere and create a shower of secondary particles that can also cause failures in electronic systems, especially at high altitudes. Today, advances in semiconductor physics have enabled integration levels of transistors down to the nanoscale. Due to this technological scaling and to the increasing complexity of electronic systems, the radiative constraint is now considered, not only for applications with high radiation levels such as space, avionics and nuclear facilities, but also for applications where the human life is involved.

The progress in the field of microelectronics has also led to a wide digitalization of the electronics sector. Many functions, historically carried by analog circuits are progressively being implemented in digital integrated circuits, often more powerful and flexible than their analog counterparts. In the space industry for example, many functions around signal processing, filtering and beam forming are now carried by digital processors. To meet the growing demand in the telecommunication bandwidth, many digital systems require both a high level of parallelization and a certain level of specialization that standard CPUs and GPUs cannot meet. Traditionally, Application Specific Integrated Circuits (ASICs) have been designed for this purpose. However, the Non-Recurring Costs (NRC) involved in developing and manufacturing a new IC are considerable and tend to increase with technology scaling [2].

In 1985, a new type of component was created, the Field Programmable Gate Array (FPGA). FPGAs are integrated circuits that can be reprogrammed after manufacture, to implement an arbitrary digital circuit. Their main principle of operation lies in an array of programmable logic blocks and a network of reconfigurable interconnects used to map the logic blocks together. The FPGA functionality is specified by loading a binary file in a specific internal memory. While generally

having lower performances, FPGAs provide a real alternative to ASICs, due to their low NRC, short time-to-market, and reprogrammability. However, this reprogrammability comes at the cost of a higher sensitivity to radiation. Indeed, the memory cells storing the component configuration can be affected by single event effects or dose effects depending on the type of used technology. Corruptions of this configuration memory can then lead to modifications of the implemented circuit topology and severely impact the system reliability. Process and layout-based radiation hardening techniques can be applied to overcome this drawback, but radiation hardened components generally suffer from a significant technological time lag and higher prices with respect to Commercial Off The Shelf (COTS) components. COTS FPGAs then remain a viable option even for highly radiative environments, provided that circuit level mitigation techniques are properly applied. The reliability of these components must then be evaluated through extensive radiation tests.

In this context, the work of the thesis focuses on testing methodologies to analyze the radiation sensitivity of FPGA-based systems. Due to their flexibility, the reliability analysis on these components is a challenging task as the radiation sensitivity is entirely conditioned by the implemented system. Indeed, it depends on the one hand on the intrinsic sensitivity of the component (to both TID and SEEs) and, on the other hand, on the way the different induced perturbations can impact the operation of the system. State-of-the-art methodologies have shown a number of limitations in bridging the intrinsic sensitivity of the FPGA and the one of the implemented designs. The objective of this thesis is to improve radiation testing methodologies to overcome these limitations. The proposed methodologies are based on a benchmarking approach: specific test structures are developed and implemented on the component during radiation tests. This benchmarking approach provides at the same time a good visibility on the internal failure mechanisms and quantitative comparisons of the radiation sensitivity across FPGAs from different vendors or families. It also provides designers with tools to assess the sensitivity of their design and improve their reliability. This benchmarking approach is also supplemented with a software-based methodology developed to predict the SEE susceptibility of an arbitrary circuit through an in-depth analysis of the circuit netlist.

In a first chapter, the different radiation environments are presented with a general description of the radiation effects on electronics.

In the second chapter, the FPGA architecture is detailed along with a more in-depth description of radiation effects and failure mechanisms specific to these components.

In the third chapter, a test methodology to assess the TID-induced degradations is proposed. Its main contribution is to extend the evaluation of parametric degradations to all logical and routing resources of the component. For this purpose, specific benchmarking structures have been developed. A new technique to measure the propagation delay in real time and with limited external instrumentation is also proposed. This technique is based on the reprogrammable feature of the clock generators embedded in the device. X-ray radiation tests have been performed on three FPGA families to highlight the benefits of this methodology.

In the fourth chapter, a test methodology to assess the sensitivity to single event effects is proposed. This methodology lies between the two traditional accelerated particle beam testing approaches (primitive level testing and final application testing) by proposing a sensitivity

evaluation at a higher level of granularity. The basic idea is to instantiate a set of dedicated benchmarking structures, simple enough to provide a good testability while sufficiently complex to provide a good representativity of the circuits effectively implemented on FPGAs, in particular by instantiating all types of logic resources of the component with a wide diversity in the circuit's architecture. The benchmarks selected in this study are based on arithmetic operators. By using different implementations of the same arithmetic functions with a large diversity in the circuit parameters (fan-in and fan-out of logic gates, logic levels between flip-flops), and in the use of resources (LUTs, flip-flops, carry logic, DSPs), the radiation tests fulfill a multifaceted purpose. First, the test results provide extensive information to identify and understand the different failure mechanisms and their predominance; second, it allows to qualitatively evaluate the impact of different types of resources on the system sensitivity. In addition, test results can be used to quantitatively compare the sensitivity of different implementations of the same logic function and to evaluate the effectiveness of mitigation solutions. Finally, it provides a set of guidelines for designers to improve the reliability of FPGA-based systems. Neutron and proton beam tests have been performed, as well as emulation-based fault injections, to demonstrate the advantages of this approach.

The main limitation of radiation testing lies with the difficulty to extrapolate the results of tests performed with a given implemented circuit to estimate the sensitivity of any other circuit. The last chapter addresses these limitations by proposing a new software-based approach to predict the susceptibility of circuits implemented on SRAM based FPGA. This analytical approach parses the physical netlist of the circuit and explores the different nets and logical resources that compose it to extract all the configuration bits that are critical for the system operation. The main contribution of the proposed methodology is to integrate the workload of the circuit, extracted from logic simulation, to analyze the propagation of errors and thus filter among the set of potentially critical configuration bits, those that actually modify the output signals of the system. The approach is validated by confronting its results with those obtained with fault injection and proton tests.

1. RADIATION: ENVIRONMENTS AND EFFECTS ON ELECTRONICS

Ionizing radiations are present everywhere on earth and in our solar system. These radiations can be of natural origin, coming from the radioactivity of the earth's soils, from the nuclear fusion processes within the sun or from the star's implosion at the other end of the cosmos, but they can also be of artificial origin when used for medical purposes, scientific experiments or energy production. The increasing use of electronics in critical systems is forcing more and more electronics engineers to consider the radiation effects on the equipment they develop. To ensure their reliability, a good knowledge is required regarding the nature of these phenomena: the composition of the different radiative environments, the physical interactions between particles and matter and the effects that these interactions can have on electronic components. This section presents the fundamental elements to apprehend these topics with an emphasis on dimensions related to FPGA technologies (subject of this study).

1.1. RADIATION ENVIRONMENT

Radiation is a form of energy released by atoms that propagates through electromagnetic waves or particles. For electromagnetic waves, when interacting with matter, their short wavelength, vector of the quantity of energy they carry, defines their capacity to ionize the atoms. There are two types of ionizing electromagnetic waves, X-rays for energies between 40eV and 400keV and Gamma rays for energies over 400keV. As for particles, protons, electrons, neutrons, alpha particles and heavier ions are mainly considered for their interaction with electronics. All these types of radiation can be generated naturally or artificially and their relative density and energy can vary significantly from one environment to another. Reliability considerations for electronics are therefore highly dependent on the type of environment in which the system being developed will operate. In this section are presented the most constraining radiative environments for electronics: space, terrestrial atmosphere and artificial radiation environments (nuclear power plant and particle accelerators).

1.1.1. SPACE RADIATION ENVIRONMENT

In our solar system, spacecrafts are exposed to a complex radiative environment with three main sources: Galactic and extragalactic Cosmic Rays (GCR), Solar Energetic Particles (SEP) and trapped particles.

1.1.1.1. GALACTIC COSMIC RAYS

According to the hypothesis emitted in 1949 by the Italian physicist Enrico Fermi, galactic radiations are mainly generated and accelerated from supernovas. These radiations are composed of charged particles with 87% of protons, 12% of alpha particles and 1% of heavier ions [3]. As shown in Figure 1, the GCR flux perceived in our solar system is modulated by the activity of the sun. When the sun is the most active, the GCR flux received is the lowest. The energy of these particles can reach several millions of GeV. The density of heavy ions decreases with their weight as described in Figure 2 but even the heaviest ions remain a concern for spacecraft reliability as they can have a more dramatic effect on electronics

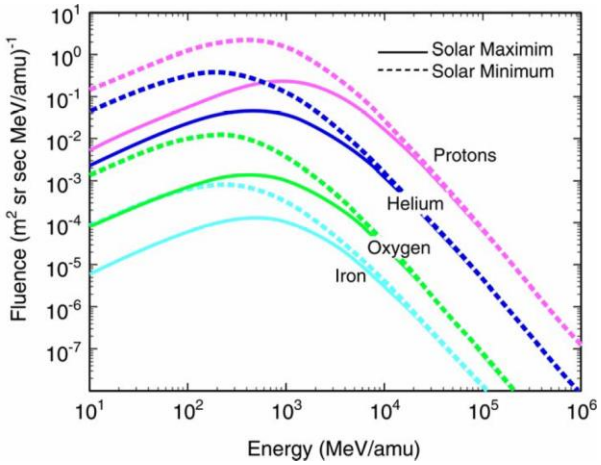


Figure 1: GCR energy spectra during solar minimum and maximum, from [3].

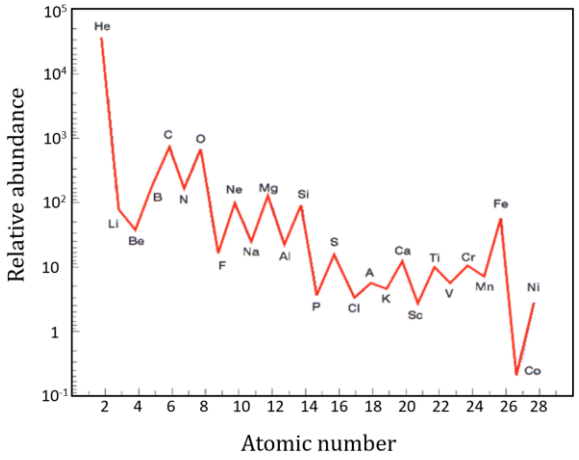


Figure 2: Relative abundance of heavy ions in perceived GCR, from [4].

1.1.1.2. SOLAR ENERGETIC PARTICLES

Solar energetic particles are continuously released by the sun through its activity. SEPs include protons, heavy ions, electrons, neutrons, gamma rays and X-rays. The three main phenomena of particle ejections from the sun are: solar winds, solar flares, and Coronal Mass Ejections (CME). The solar wind is a continuous flow of plasma of electrons and protons. Most of the particles emitted by solar winds are of relatively low energy and are mostly deflected or trapped by the Earth's magnetic field. Solar flares and Coronal Mass ejections are random events due to the reconfiguration of the solar magnetic field and their frequency is strongly linked to the activity of the sun which follows a cycle of about 11 years as shown in Figure 3.

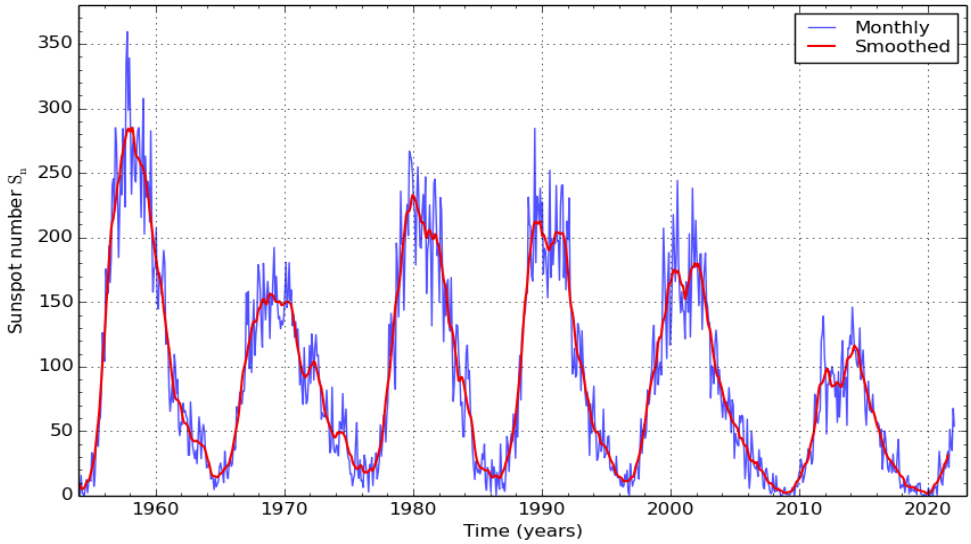


Figure 3: Solar activity from 1749 to 2022, from [5].

Solar flares mostly release electrons but also protons, alpha particles heavier ions and electromagnetic radiation. CMEs are larger events that release more protons but less heavy ions with respect to solar flares. The energy of the particles emitted by the sun remains generally of lower energies than the GCR and their presence in the Earth's orbital environment is therefore greatly impacted by the Earth's magnetic field.

1.1.1.3. TRAPPED RADIATIONS

The earth has a magnetosphere that results mainly from the interaction of the solar wind with the earth's geomagnetic field. Charged particles of cosmic or solar origin interact with the magnetic field lines near the earth and tend to follow them. These deviated particles get trapped in what is called the Van Allen belts. The ability of the magnetic field to deflect the charged particle is directly related to their mass and energy. These radiation belts are therefore composed mainly of protons and electrons. The particles trapped in these radiation belts are clearly separated into two different belts: the outer belt, mainly composed of electrons and the inner belt mainly composed of protons [6]. The density of these particles in the terrestrial orbital environment is described in Figure 4.

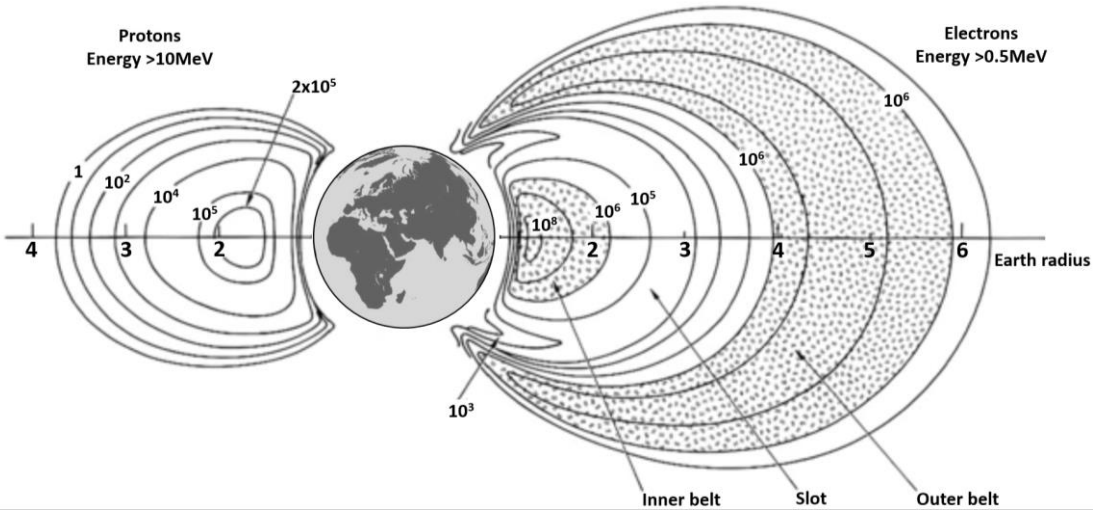


Figure 4: Time averaged radiation belt omnidirectional fluxes (cm^{-2}) for protons (left) and electrons (right) as a function of the distance from earth, from [6].

As the Earth's magnetic field is tilted around 11 degrees from the rotation axis, the radiation belts do not align exactly with the Earth's surface. As a result, the altitude of the inner belt drops significantly (around 200-800km) in a specific region located over South America off the coast of Brazil as shown in Figure 5. This phenomenon, called the South Atlantic Anomaly (SAA), induce a level of radiation significantly higher than anywhere else on Earth orbit for this altitude. Spacecraft paths are therefore specifically tailored to minimize the exposure time to these radiation belts.

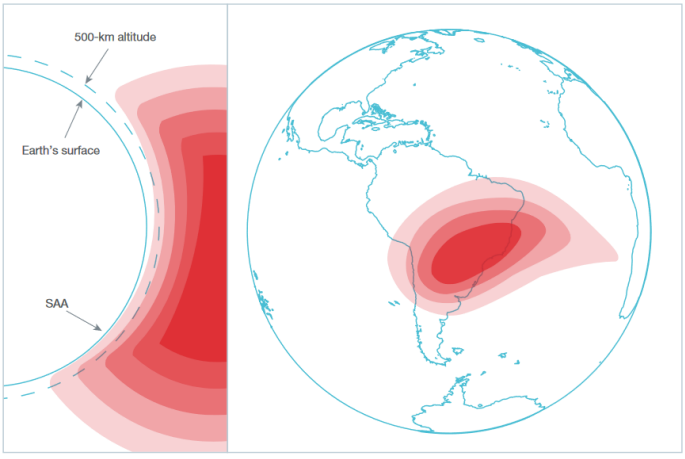


Figure 5: Inner radiation belt ingress at the South Atlantic Anomaly, from [7].

1.1.1.4. RADIATION ENVIRONMENT MODELING TOOLS

To evaluate the reliability of the electronics embedded in space systems, it is necessary to consider the flux and energy spectrum of each type of radiation that will be received by the spacecraft during the entire mission. The radiation received by a spacecraft is very dependent on its position and trajectory in the solar system. For a satellite, it is necessary to consider the entire trajectory followed from the launch to the final orbit. Traditionally, there are three main near-earth space mission orbits classified by the altitude:

- Low Earth Orbit (LEO) for altitudes up to 2000km from the Earth's surface. It includes Earth observation satellites (meteorology, climate studies, disaster management, crop monitoring, oceanography), some telecommunication satellites (more and more with recent large constellations like *Starlink* and *OneWeb*) but also the international space station.
- Medium Earth Orbit (MEO) covers altitudes from 2000km to 35 780km. This orbit is mainly used for navigation satellites (GLONASS, GPS, Galileo) but also for some telecommunication satellites.
- Geosynchronous orbit (GEO) is at 35 780km on which the satellite moves around earth in 24 hours. They can be used for telecommunication and weather monitoring.

Due to the complex influence of earth magnetic field and the sun activity on the radiation fluxes in our solar system, computer tools are required to properly estimate the type and the fluxes of radiation that the spacecraft will undergo. During the last decades, a lot of models such as AE9, AP9, ESP, CREME96 and GCR ISO have been developed to model the radiation environment for each type of radiation source. Today, software such as OMERE (TRAD [8]) are gathering the main state-of-the-art radiation environment models and allowing, among other things, to compute the radiation flux and energies received by a spacecraft for a given orbit or trajectory (Figure 6).

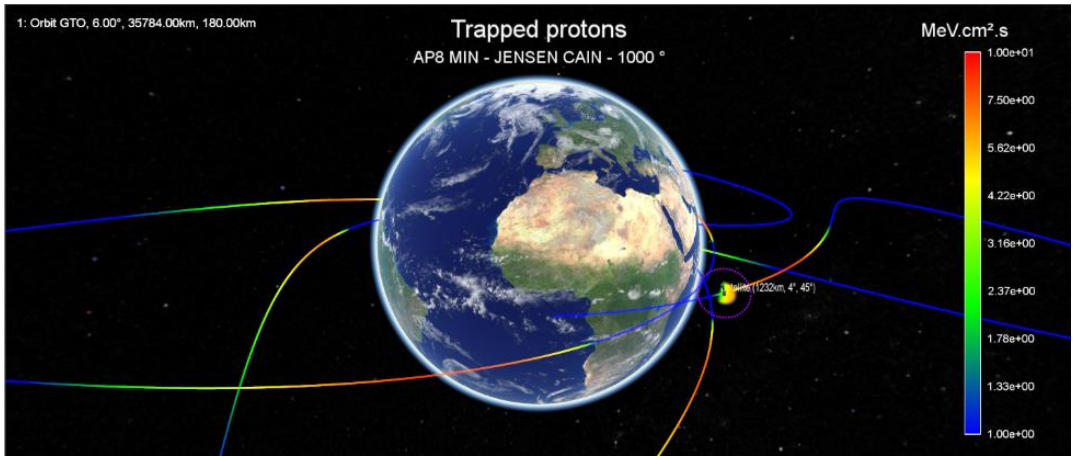


Figure 6: Screen shot of OMERE software modeling the trapped protons environment for a given mission profile [8].

1.1.2. ATMOSPHERIC AND TERRESTRIAL RADIATION ENVIRONMENTS

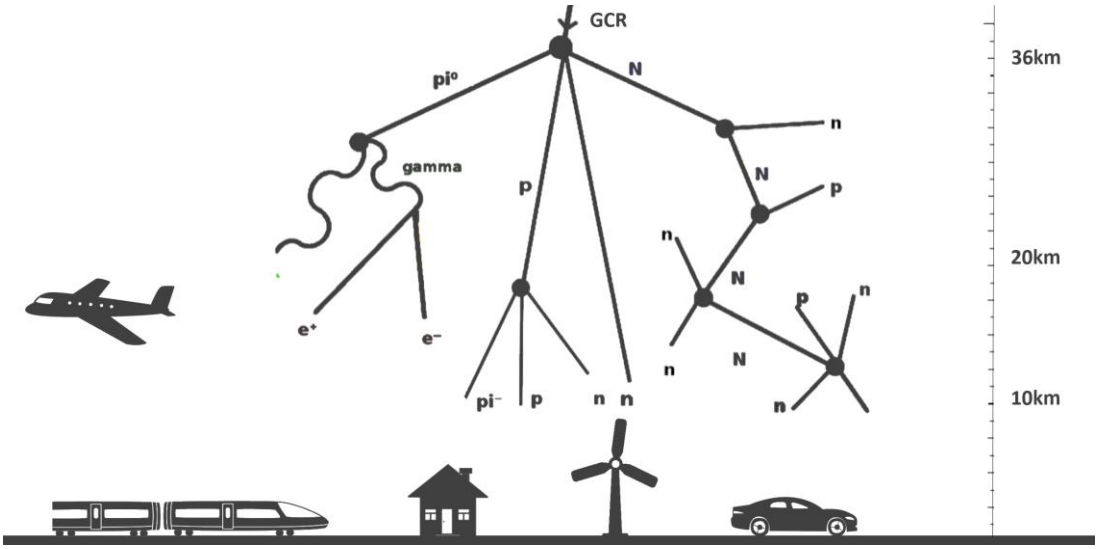


Figure 7: Cosmic particles (GCR) interacting with atmosphere molecules resulting in a shower of secondary particles.

High energy galactic and solar particles that are not blocked by the Earth's magnetosphere interact with the molecules in the atmosphere resulting in a shower of secondary, lower-energy particles such as protons and neutrons [9] as shown in Figure 7. Similarly, these secondary particles can also interact with the molecules of the atmosphere. The energetic spectrum of these particles as a function of the altitude is described in Figure 8. The flux of these secondary particles increases with the altitude reaching a maximum at an altitude around 20km while the flux at ground level is attenuated by a factor of 500 compare to the peak flux. These secondary particles are therefore a major concern for on-board electronic systems in aircrafts [10], but even if the probability of failure induced by these secondary particles is much lower at ground level, it remains a major concern for all systems that are safety critical. With the massive deployment of electronic devices in most sectors of daily life, more and more electronic systems have to deal with the radiative constraints. In automotive, for example, more and more crucial functions are handled by electronics system, from the engine control unit to the brake control module, and this digitalization is expected to grow as human handled functions are gradually being replaced by autonomous systems. The concern for radiation is reflected in many areas of transportation, but also in the medical industry, in the military field and even in data centers where the large number of electronic components used in parallel drastically increases the probability of observing a radiation-induced error.

Another significant source of ionizing radiation in microelectronics comes from radioactive impurities present in the package materials. Mainly, uranium and thorium and their associated daughter isotopes can emit alpha particles as a result of spontaneous breakdown of the nuclei [11]. On the same principle, Boron atoms used as P dopants in many components can be another source of radiation induced failures. Indeed, during the doping process with Boron atoms, even if the major part of the implanted atoms are ^{11}B isotopes, a small portion of ^{10}B isotopes can be implanted. When exposed to low energy neutrons, this isotope is unstable and its nucleus can break apart releasing alpha particles and gamma photons [12]. These emitted particles can potentially induce failures to electronics due to their proximity to the active silicon area.

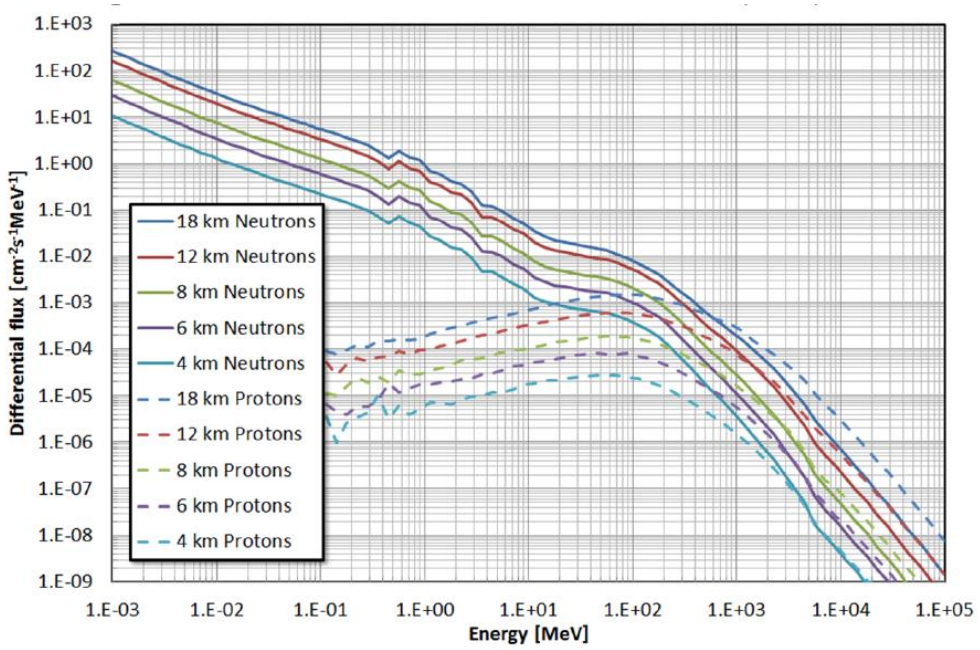


Figure 8: Simulated spectrum of secondary particles in the earth atmosphere from [13].

1.1.3. ARTIFICIAL RADIATION ENVIRONMENTS

As mentioned earlier, radiations are not only generated naturally, humanity has also exploited radioactive phenomena in many sectors ranging from energy production (nuclear power plants), to medical (imaging, radiotherapy), military (nuclear-powered submarines and aircraft carriers) and scientific research (particle accelerators). The electronic systems used in these domains may also be subject to high levels of radiation. The radiation environments, specific to each type of facility must thus be carefully defined.

1.1.3.1. NUCLEAR POWER PLANTS

The reliability of electronic systems in nuclear power plants is a major issue for the proper functioning of the plant and for the safety of the operators. Indeed, a strong radiative constraint is present not only during the normal operation of the plant but also during the creation of fuels, the treatment and storage of radioactive waste and during the dismantling of the plant. During normal operation of a nuclear power plant, the radiation present in the nuclear reactor and the spent fuel containment area are mainly gamma rays and neutrons [14]. The dose rate can reach up to 1 Grad/h and the flux up to 10^{14} neutrons.cm^{-2}.s^{-1} for energies below 1 MeV. The level of hardening of electronic systems is therefore strongly dependent on the criticality of the system function and its position in the plant. Microelectronics used in detection and monitoring equipment installed inside the high radiation areas are exposed to high level of gamma rays and neutrons. For the most critical systems, radiation hardened devices might be required and/or periodically replaced.

1.1.3.2. MEDICAL RADIATION ENVIRONMENTS

In the medical field, ionizing radiations are used mainly for three functions. X-ray imaging, proton therapy and radiation sterilization (e-beam, RX, or gamma ray). For X-ray imaging, the maximum dose that a patient could absorb for a full body scan does not exceed 2rad(SiO_2)[7]. Likewise, proton therapy, used for cancer treatment, uses very localized proton beams and therefore represents very low equivalent dose levels. Electronic devices implanted in the human

body are not sensitive to such low dose levels. However, repeated use of the equipment on many patients can cause long-term accumulation of high dose levels to the internal electronics. Most electronics components are usually protected by a sufficiently thick layer of metal to prevent their degradation, but image sensors are necessarily exposed to a significant dose level over time. In this case, even radiation-hardened imagers will need to be occasionally replaced. On the other hand, sterilization of surgical instruments, radiation is commonly used to degrade the DNA of potential bacteria and viruses. The doses used to completely sterilize a sample can reach 5Mrad(SiO_2), a particularly high dose level for most electronic systems. Irradiated components can then be deactivated during irradiation to limit charge trapping and thus mitigate Total Ionizing Dose (TID) effects.

1.1.3.3. PARTICLE ACCELERATORS

To investigate radiation effects on electronics, a common approach is to expose the electronic devices to the expected radiation source in an accelerated manner using radiation test facilities. Several such facilities exist across the world offering different types of particles, flux and energies.

A frequently used example to illustrate the radiation environments in a radiation facility is the one around the Large Hadrons Collider (LHC) at CERN. The radiation environment of the LHC and its injection lines is composed of different particles over a wide energy spectrum. The ionizing dose levels around the LHC can be very high, up to tens of krad(SiO_2) per year in the most radiative areas. These radiation levels are mainly due to the generation of secondary particles by collision debris from proton collisions and from the interaction of the particle beam with the residual gas inside the vacuum pipes. As an example, the energy spectra in the tunnel area are presented in Figure 9. As a result, electronic devices installed in the accelerator complex are exposed to a mixed-particle radiation environment.

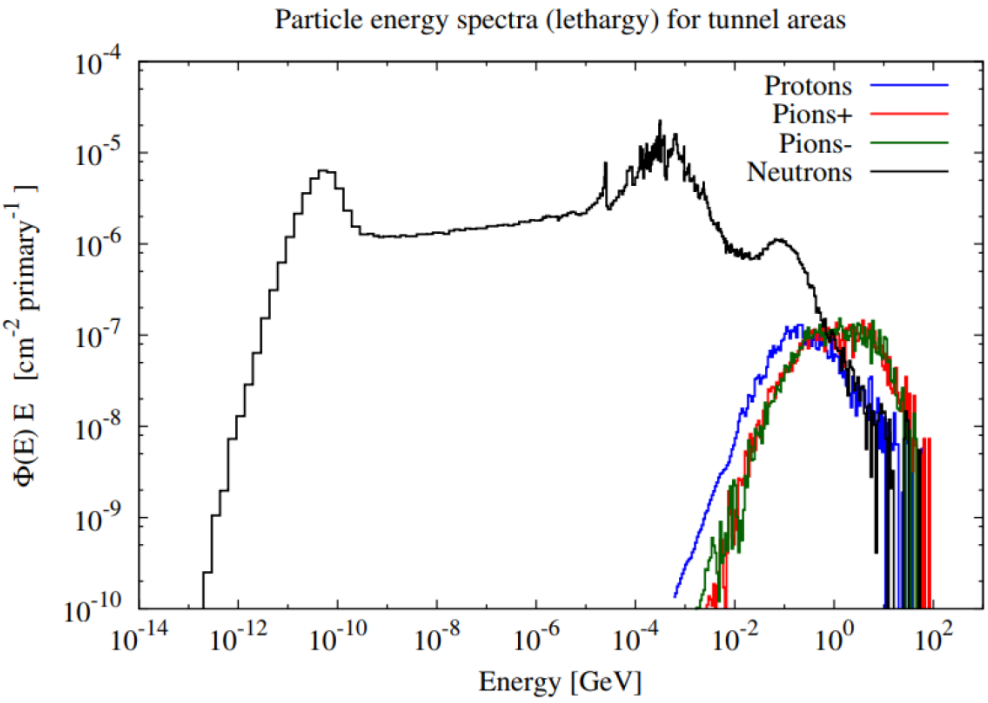


Figure 9: Simulated particle energy spectra of the tunnel areas for nominal LHC operation (normalized to one proton-proton collision), from [15].

In this study, the radiation tests have been performed in two different facilities. The first tests were performed with neutrons at the ChipIr neutron beam line at ISIS facility (UK). To generate neutrons, a high-energy proton beam (800MeV) accelerated in a synchrotron is blasted on a Tungsten target as shown in Figure 10. The collisions generate neutrons with an atmospheric-like energy spectrum as shown in Figure 11.

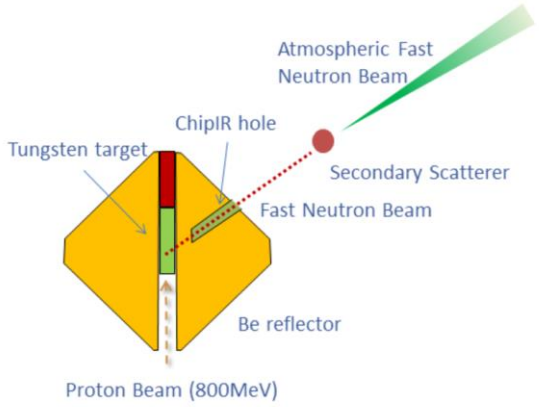


Figure 10: Schematic of atmospheric neutrons beam production principle, from [16].

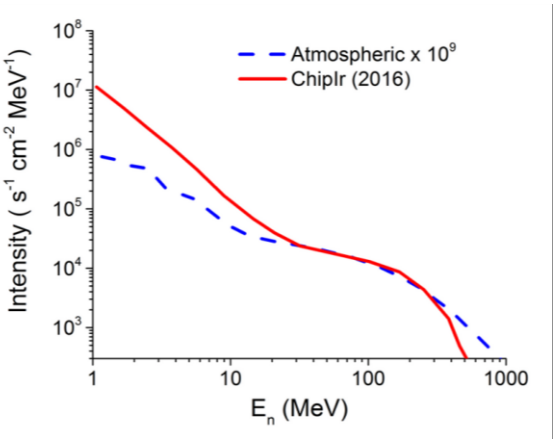


Figure 11: Neutron energy spectrum generated by the ChipIr beam line, from [16].

The second set of tests were performed with protons at PARTREC facility from the University of Groningen (the Netherlands). This facility uses a large superconducting cyclotron to generate beams of ions ranging from protons to oxygen with proton energies up to 184MeV and providing flux up to 1.10^8 protons.cm^{-2}. The energy of the particle in this type of facility can be controlled by using degraders; materials of different thickness that can be inserted into the beam to reduce the beam energy.

1.1.3.4. RADIATION SOURCES

To analyze the dose effects, high-energy photons (X-rays and Gamma rays) are commonly used. In this study, the X-ray generator from PRESERVE Platform [17] at the Montpellier University was used. To generate X-rays, this type of generator (described in Figure 12) uses a filament, heated by injecting a strong electric current. The heated wire emits electrons from its surface which are then accelerated by a strong electric field generated by the high voltage anode (around 320kV) in a vacuum chamber. The accelerated electrons collide with the anode, usually made of high-z metal such as Tungsten. The collision between the electrons and the metal atoms generates X-rays that pass through the wall of the vacuum chamber. The X-ray beam then pass through a collimator to control the beam size before interacting with the devices under tests. The generator used in this study can generate up to 12 rad(SiO_2)/s.

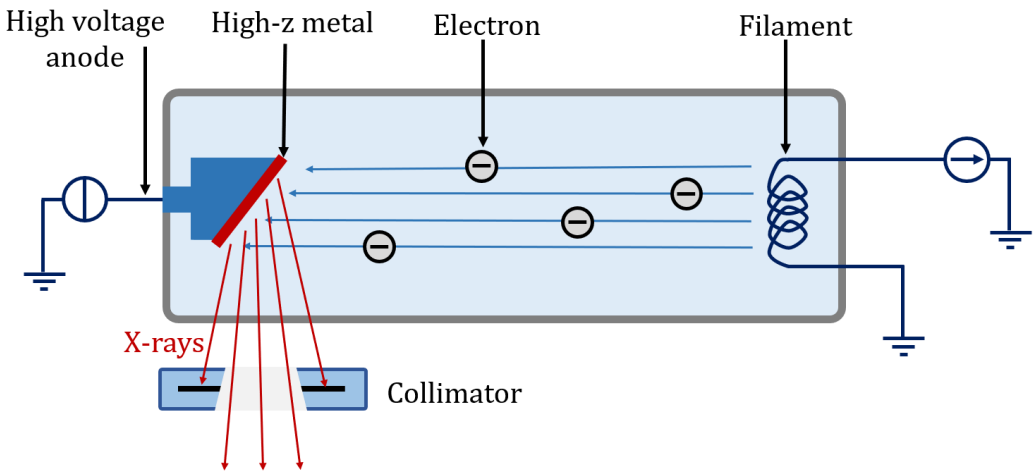


Figure 12 : Schematic description of an X-ray generator.

1.2. Radiation effects on electronics

In this part, an introduction to radiation effects on electronics will be presented with an emphasis on effects affecting digital CMOS technologies, as it is used by all modern FPGAs (subject of this study). Radiation effects can be divided into two main types: cumulative effects which describe the progressive degradation of component parameters induced by successive interaction with a large number of ionizing particles, and Single Event Effects (SEE) which are generated by interaction with a single particle.

Before diving into the radiation effects on electronics, it is important to briefly present the main physical phenomena involved in the interaction between radiation and matter.

1.2.1. Radiation-matter interactions

The interactions between radiation and matter are strongly dependent on the type of particles, their mass and their energy. Based on the type of interactions with matter, radiations can be divided into three groups: photons, which can ionize atoms, neutrons, which, due to their electrical neutrality, interact with atoms only via nuclear interactions, and charged particles which can have nuclear interactions but also coulombic interactions due to their electrical charge.

1.2.1.1. Photon-matter interactions

Four main types of interactions are induced by photons:

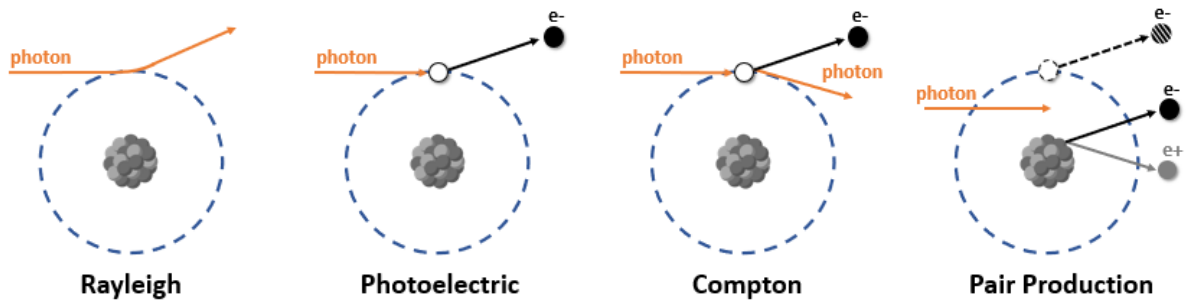


Figure 13: Main photon induced ionizations of atoms.

- **Coherent scattering (Rayleigh):** the photon is scattered by atomic electrons without excitation of the impacted atom. The energy of the photon is conserved, and only its direction is modified.
- **Photoelectric effect**: the photon transfers all of its energy to an electron of the atom. The electron is pulled out from its orbital with a kinetic energy equal to the difference between the energy of the incident photon and the binding energy of the electron.
- **Compton effect:** this effect occurs when the energy of the photon is much greater than the binding energy. In that case, the photon loses part of its energy and gets scattered to pull out the electron from its orbital.
- **Pair production:** for even higher energies, when the photon passes close to the atom nucleus, its energy can be transformed into a pair of electron and positron. This pair production process can be accompanied by the ejection of an electron from the atom if the photon passes in its field (triplet production).

The relative dominance between these four effects depends on the incident energy of the photons as shown in Figure 14. In silicon, at low energy, photoelectric effects dominate. Between 60keV and 15MeV Compton effects dominate and pair production dominates for photon energy above 15MeV. As an example, gamma rays emitted by cobalt 60 (extensively used for TID testing) are found almost exclusively on two energy peaks at 1.17MeV and 1.33MeV, which corresponds to the Compton effect. As described in the rest of the manuscript, X-ray irradiations have been performed in this study, for which the photoelectric effect dominates.

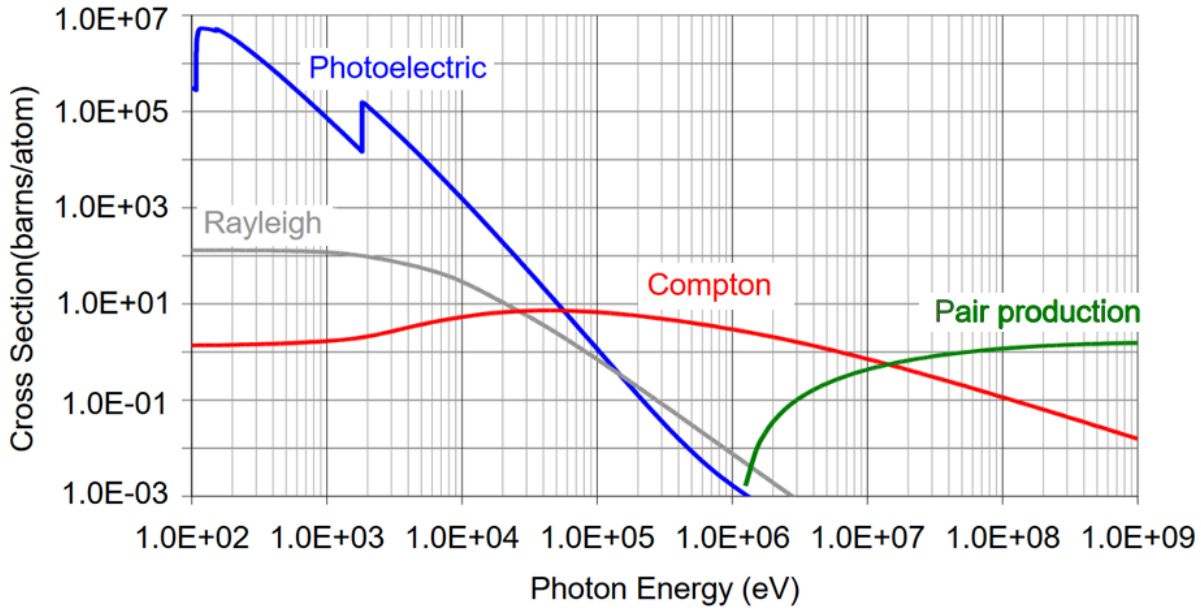


Figure 14: Relative proportion of photoelectric effects in silicon as a function of photon energy, from [18].

1.2.1.2. NUCLEAR INTERACTIONS

Particles with sufficient energies can interact with the atom's nucleus through four main types of interactions as shown in Figure 15:

- **Absorption:** this interaction can occur for thermal neutrons. The nucleus absorbs the neutron without disintegrating and emits a photon to release excess energy.

- **Elastic scattering:** the particle is scattered by collision with the nucleus. The kinetic energy lost in the collision is transferred to the nucleus that gets in motion. The global kinetic energy is conserved.
- **Inelastic scattering:** Beyond transferring part of its energy to the nucleus to get it in motion (recoil), part of the particle energy is also lost to excite or ionize an electron of the atom. The global kinetic energy is not conserved.
- **Non-elastic scattering:** the collision fractures the nucleus into smaller nuclei. This reaction can also be accompanied by the emission of one or more photons and neutrons.

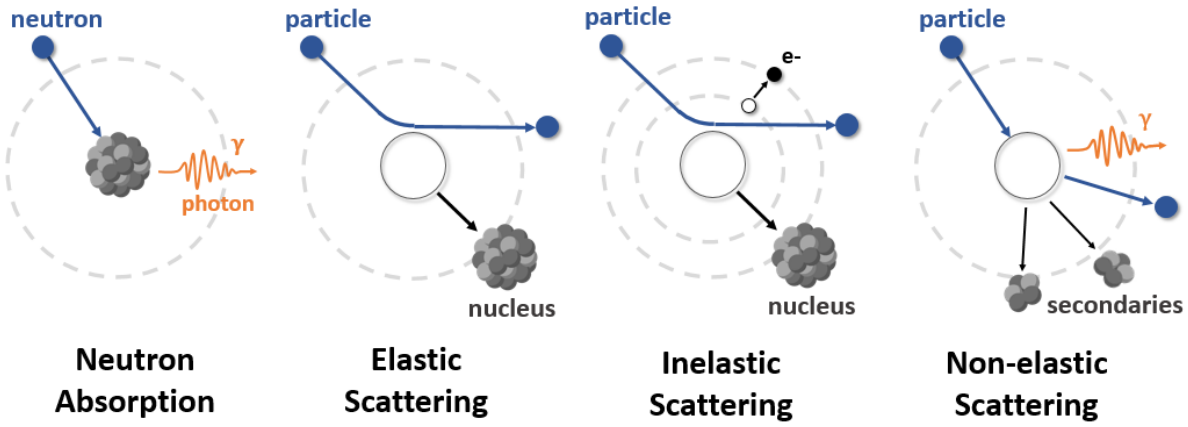


Figure 15: Nuclear interactions.

1.2.1.3. COULOMBIC INTERACTIONS

Charged particles can also interact with matter through three types of Coulombic interactions:

- **Bremsstrahlung:** (or braking radiation) when an electron passes near the nucleus of an atom, the electron loses kinetic energy due to the attraction of the nucleus electric field. This loss of kinetic energy is converted into photons.
- **Ionization:** when a charged particle passes near an orbital electron, the electric field of the particle can pull out the electron from its orbit thus generating an electron-hole pair.
- **Excitation:** Similarly, the charged particle passing near an electron can attract the electron on an orbital of higher energy. By returning to its initial orbital, the electron releases a photon.

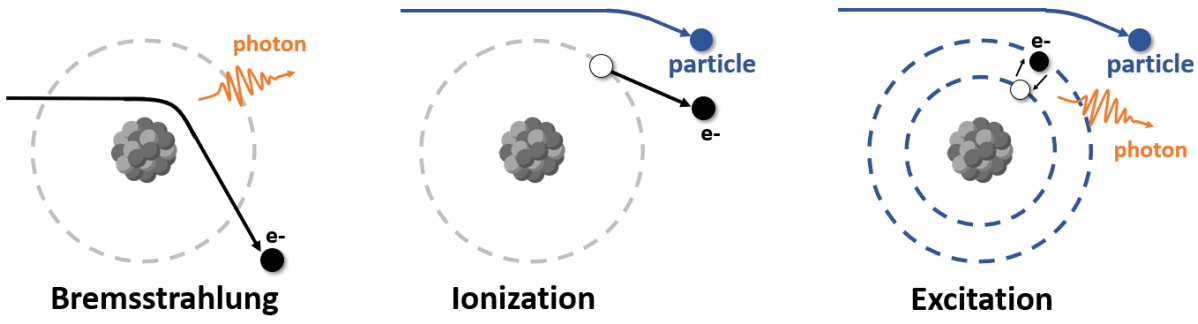


Figure 16: Coulombic interactions.

1.2.1.4. ENERGY TRANSFERS IN MATTER

When a particle (other than photons) penetrates a material, it gradually transfers its energy to the surrounding matter through the various interactions mentioned above. The type of interactions involved will depend on the energy and the charge of the particle. When entering a material, a charged particle such as a proton or a heavy ion will first transfer energy by direct ionization of the surrounding matter. To quantify this energy transfer, the Linear Energy Transfer (LET) is used. This variable is defined by equation (1).

$$LET = -\frac{1}{\rho}\frac{dE}{dx} \tag{1}$$

Where ρ is the density of the material, dE is the energy loss is the material and dx is the unit of path length. The LET is usually given in $MeV.cm^2.mg^{-1}$.

As the particle slow down, its energy gets closer to the binding energy of the surrounding electrons, the probability of interaction by ionization increases, thus increasing the linear energy transfer until it reaches the Bragg peak as shown in Figure 17. This Bragg peak corresponds to the maximum linear energy deposition before the particles come to rest. As the particle continues to slow down, the nuclear interaction probability increases and becomes predominant at low energies. The nuclear interactions can generate secondary particles which can in turn continue to ionize the matter as described in Figure 19. The primary particle will continue its path until the total loss of its energy. Figure 18 shows the proportion of energy deposition between ionization and nuclear interaction as a function of the particle energy.

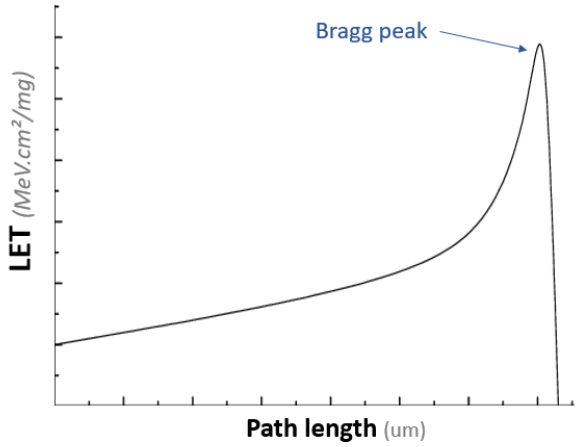


Figure 17: Evolution of linear energy transfer in the path length.

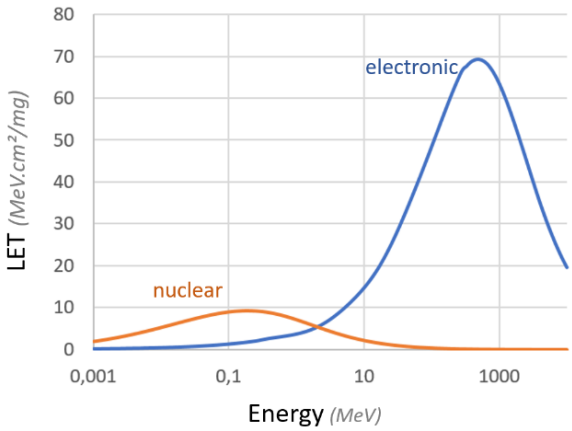


Figure 18: Energy deposition of Xenon ion in silicon according to SRIM [19].

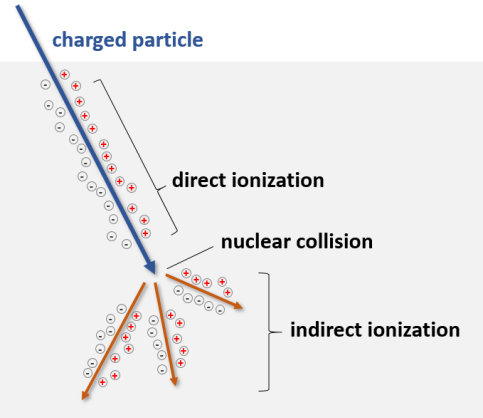


Figure 19: Energy deposition by a charged particle.

To be noted that the heavier the particle, the higher the LET. With the same initial energy, the path length in the material is thus longer for the lightest ions. When performing radiation tests with heavy ions, the penetration through the component package and silicon layers (for flip-chip packages) or through the back end of line (contacts, insulating layers, bonding sites) must be considered to ensure that the particle can reach the active zone of the component at a given energy. Regarding neutrons, as they cannot perform direct ionization, they transfer their energy to the material only through nuclear interactions and indirect ionization.

Photons that penetrate a material interact with the atoms through one of the different interactions mentioned above (photoelectric, Compton, pair production) depending on their energy. The higher the energy of the photon, the lower the probability of interaction and therefore the greater the penetration. The penetration of a beam of photons of intensity I in matter is defined by the Beer-Lamber law (2).

$$\frac{I}{I_0} = B.\exp\left(-\frac{\mu}{\rho}x\right) \quad (2)$$

With I_0 the incident intensity, μ the attenuation coefficient of the material, ρ the density of the material, x the depth of penetration into the material and B a factor that takes into account the photon scattering when the material thickness to be penetrated is important.

1.2.2. Cumulative effects

The cumulative effects caused by radiation on electronic components can be divided into two subcategories, Total Ionizing Dose effects (TID) and Total Non-Ionizing Dose effects (TNID).

1.2.2.1. Total Ionizing Dose

Ionizing dose effects are caused by the interaction of charged particles or high-energy photons with dielectric materials contained in electronic components such as SiO_2. The total ionizing dose (TID) is defined as the total energy deposited by ionization per unit mass. When a high-energy photon or charged particle passes through the component, the ionization process generates electron-hole pairs. A part of the generated pairs recombines almost instantaneously but a certain ratio of these generated pairs will survive the initial recombination due to the local electric field that tends to separate opposing charges. The non-recombined pairs then migrate in opposite directions according to the electric field. In an electrical insulator such as SiO_2, there are intermediate energy levels located within the band gap that are due to defects in the crystal lattice.

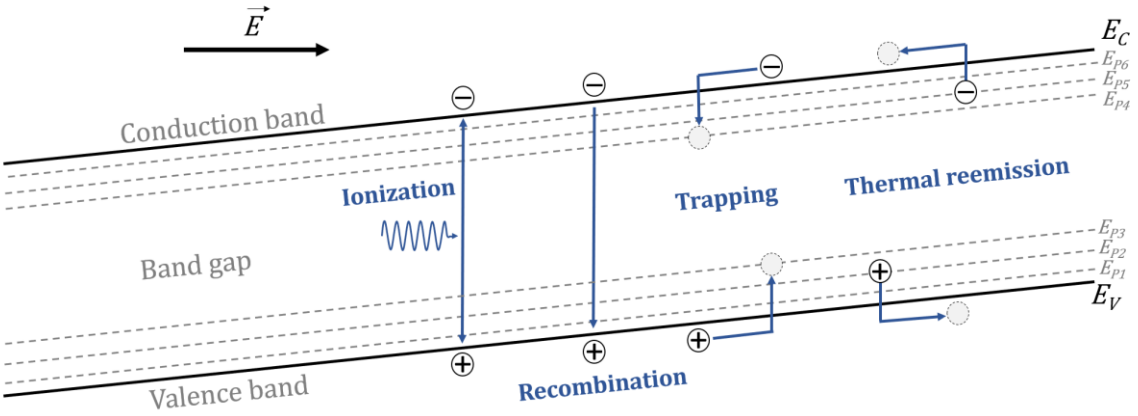


Figure 20: Band diagram representing the trapping and release mechanisms in a dielectric material.

These energy levels are called traps. Indeed, part of the electrons and holes generated by ionization, migrating by the action of the electric field can get blocked in these energy levels. The energy depth of the trap and the temperature of the component define the probability that the trapped electron or hole releases through thermal agitation [20]. The electron-hole pair generation, charge trapping and thermal reemission phenomenon are illustrated in Figure 20. As the mobility of holes is lower in SiO_2 than the one of electrons, the number of holes trapped inside the oxide overwhelms the number of trapped electrons which results in an accumulation of positive charges in the oxide. On a MOSFET device, this charge accumulation in the isolation oxides acts directly on the electrostatic field inside the transistor channel and will thus directly impact the threshold voltage of the transistor. Due to the difference in the mesh size between SiO_2 and Si, defects in the crystal lattice are more important near the Si/SiO2 interface.

The majority of the trapped charges will be located close to this interface and therefore have a greater impact on the electrostatic field in the channel. Note that the polarization of the transistor during the irradiation also plays an important role as it defines the direction and the intensity of the electric field in the SiO_2. Beside its influence on the initial recombination rate, a positive

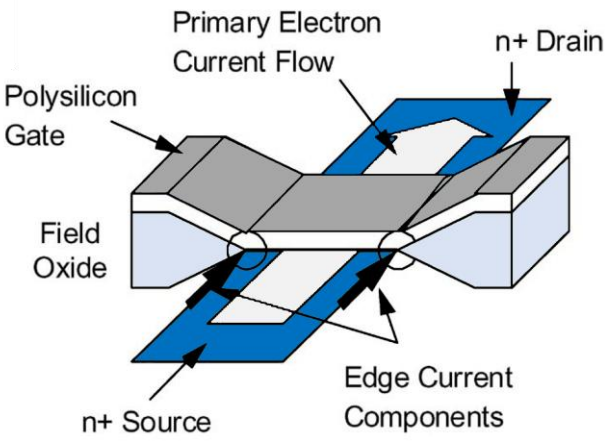


Figure 21: Parasitic conduction path on transistor edges, from [33].

polarization tends to push the holes towards the interface which tend to amplify the trapping phenomenon. A second phenomenon comes in addition when the generated free holes react with the Si-H bonds at the Si/SiO_2 interface. By breaking this bond, new interface defects are generated which translate in additional energy levels in the band gap. In the drain-source channel, the charges will move intermittently on these newly created energy levels leading to a decrease in the overall mobility of charges. Finally, a third TID effect appears on the recent CMOS technologies: the radiation induced leakage current (RILC). Shallow Trench Isolation Oxides (STI) used to prevent electric current leakage between adjacent semiconductor device are much thicker and poorer quality oxide than gate oxides. Holes trapped in STI oxides can create a parasitic conduction path on the transistor edges as shown in Figure 21 as well as a leakage path between the n+ source/drain regions of adjacent NMOS transistors, resulting in an increased source-drain current in the transistor off state (V_G=0V) and mitigating the intended inter-device isolation [21].

In summary, there are three TID phenomena affecting the MOSFET characteristics: the accumulation of trapped holes in the insulation oxide inducing a negative threshold voltage drift (on both NMOS and PMOS transistors), the generation of new interface states inducing a decrease of the transconductance as described in Figure 22, and the increase of the leakage current induced by trapped holes in the STI.

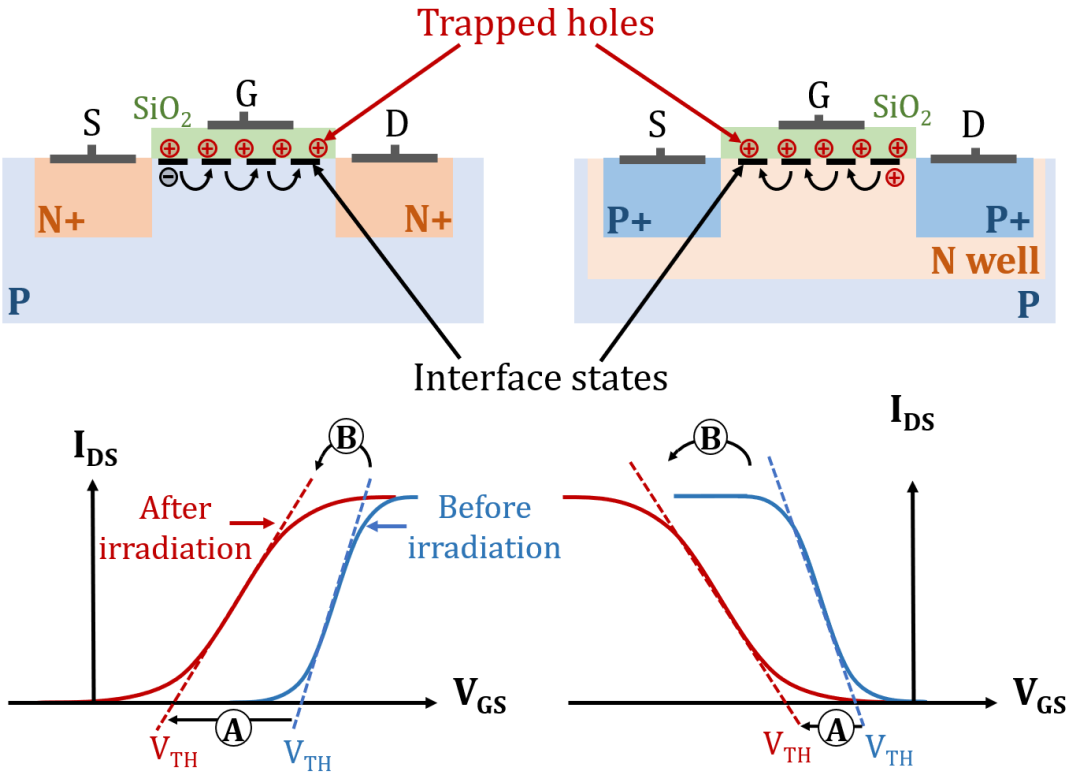


Figure 22: Main TID effects on NMOS (left) and PMOS (right) transistors: negative threshold voltage drift induced by accumulation of holes in the SiO2 layer (A), decrease of the transconductance due to interface states (B).

The radiation induced parametric drift at transistor level can be translated in many different ways depending on the type of systems considered. For example, on a CMOS image sensor, the increase in leakage current will be predominant and will cause a whitening of the recorded image; analog circuits may suffer a distortion of their spectral response and a decrease in performance. Memory elements may also suffer from TID induced degradations. For instance, Dynamic Random-Access

Memory cells (DRAM), are usually composed of a tiny capacitor, which retains the binary information by storing or not a certain amount of electrical charge and an access transistor, used for the read and write operations as shown in Figure 23. This volatile memory cell type needs to be refreshed periodically to overcome the charge leakage. TID induced degradation on the access transistor can result in an increase of the leakage current, translated by a reduction of the retention time of the cell as shown in Figure 24 [22]. If the retention time gets lower than the refresh rate, data corruption may appear. Similarly, Flash memory cells and the associated peripheral circuitry (charge pump) can be corrupted by TID. This effect will be described in section 2.3.1.

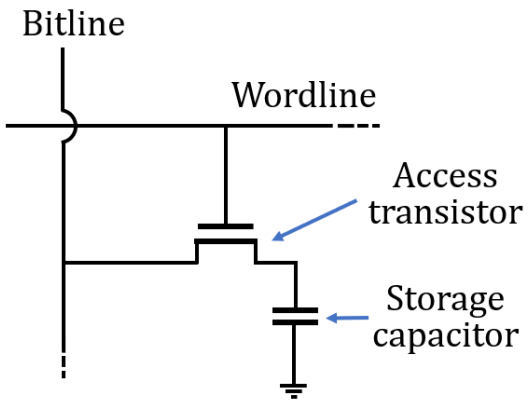


Figure 23: DRAM cell structure. The storage capacitor can either be charged or discharged to represent a logical 1 and 0 respectively.

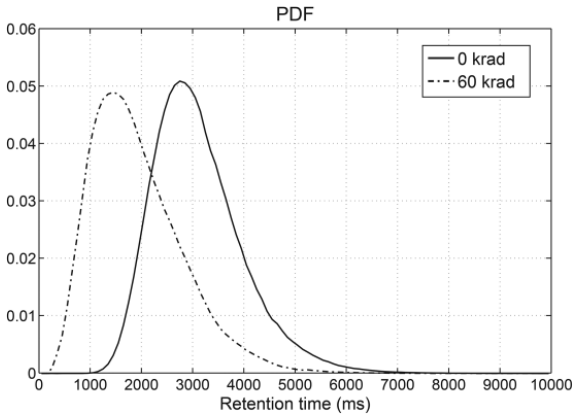


Figure 24: Probability density function of retention time before and after radiation exposure (Co60) of a commercial 0.14µm SDRAM, from [22]

As for digital circuits, these transistor level effects will be mainly translated by a drift of the propagation delay of the logic gates and by an increase of the power consumption. In another note, for flash memory cells, the threshold voltage drift of the floating gate can be translated in a permanent corruption of the stored information. The effects typically FPGAs are concerned (the components studied in this work) and the consequence on system reliability will be described in a chapter 2.

1.2.2.2. Total Non-Ionizing Dose

TNID effects (also called displacement damage) are caused by high-energy particles that, when penetrating a component, can collide with atoms of the crystal lattice and displace the nucleus of the impacted atom. These atomic displacements will break the regularity of the crystal lattice by

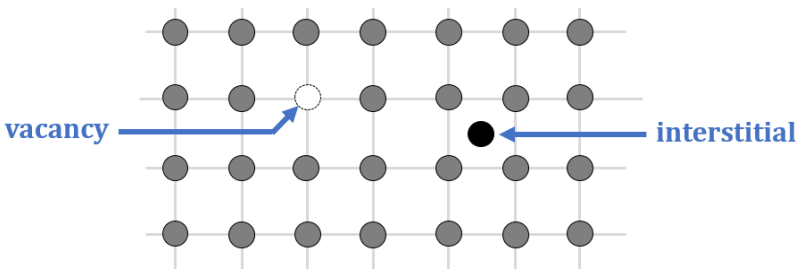


Figure 25: Crystal lattice defects.

creating vacancies (missing atom from its normal lattice position) and interstitial (atom located in a non-lattice position) as shown in Figure 25.

The defects generated in the crystal lattice result in the creation of intermediate energy levels in the band gap. For MOS devices, these energy levels will mainly cause a decrease in carrier mobility and an increase in leakage current. The effects induced by the displacement damage will not be further discussed in this manuscript.

1.2.3. SINGLE EVENT EFFECTS

As explained in the previous sections, the passage of a high-energy particle in silicon produces a localized ionizing track. The electric field and the local density of charges in the component determine the displacement of these charges which will locally modify the electric currents through two main phenomena, drift and diffusion, as shown in Figure 26.

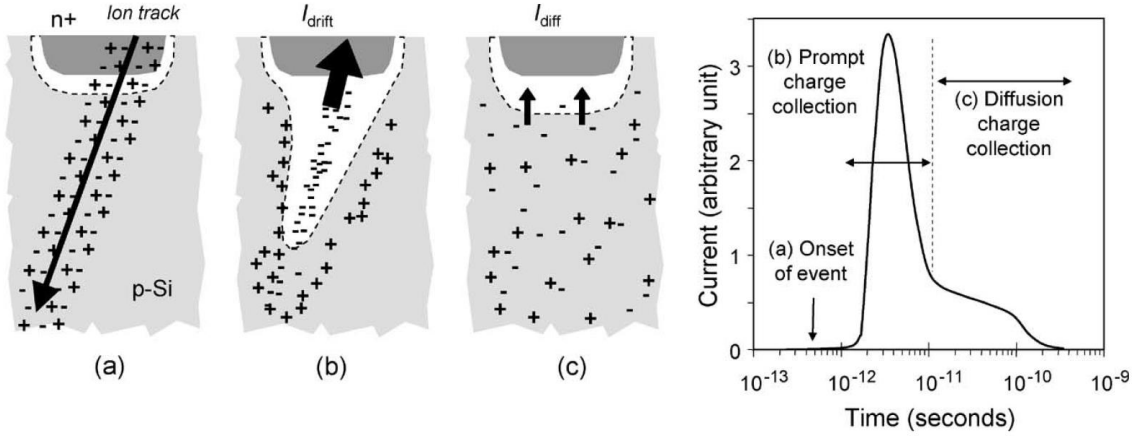


Figure 26: Charge collection phenomenon in a reverse-biased junction, from [23].

The carriers of the ionizing track are firstly dragged by the electric field extending the depletion region deeper in the semiconductor with a funnel shape. The remaining charges then diffuse inside the semiconductor to equilibrate the charge density. This charge collection phenomenon generates current pulses in the surrounding transistors. The critical charge Q_{CRIT} defines the amount of collected charge required to change the state of the transistor. With technology scaling, this critical charge tends to decrease, and the transistors are also closer to each other, the charges generated by the passage of a single particle thus have an increased probability to upset several transistors simultaneously.

Single **E**vent **E**ffects (SEE) are classified into subcategories depending on the type of circuit and the way they can be affected. This classification can vary depending on the type of electronic component considered. The main categories for CMOS digital devices are described below.

1.2.3.1. Single Event Transient

Single Event Transients (SET) are characterized by the generation of a current spike in the node of a circuit. Depending on its amplitude and width, this current spike can then propagate across the nets and the logic gates of the circuit. The logic gates crossed can have a broadening or narrowing effect on the propagation of the current pulse as well as an amplifying or decreasing effect depending on the type of gate and the design parameters. The SET can potentially impact the circuit functionality if it is captured by a sequential element. In others words, if the SET reaches the input of a latch or flip-flop with sufficient amplitude during the rising edge of the clock signal, an erroneous logic value is stored by the latch until the next clock cycle as described in Figure 27.

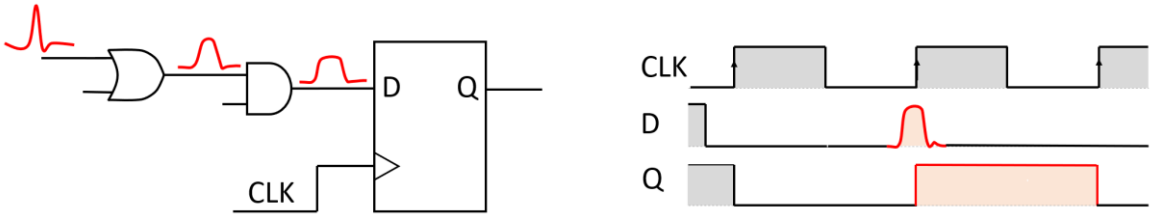


Figure 27: The generated SET electrically propagates through the logic gates before reaching the end-point flip-flops. If the SET reach the flip-flop at the rising edge of the clock, the error is captured and maintained until the next clock cycle.

The SET can propagate through several nets and can eventually generate logic error in different latch elements at the same time. The number of affected sequential elements can increase drastically when the affected nodes impact global nets like the clock or reset signal. To be noted that the SETs generated in the combinatorial logic are asynchronous, so the probability to get captured by a sequential element increases linearly with the frequency of the clock signal (i.e. the number of rising edges per second).

1.2.3.2. Single Event Upset

Single **E**vent **U**pset (SEU) is a soft error generated in a latch or a memory cell. A SEU is created when a radiation-induced current pulse is generated inside the latch or memory cell circuit. For instance, the content of DRAM memory cells (described in section 1.2.2.1) can be corrupted by single-event upset in or near either the storage capacitor or the source of the access transistor [24]. Another example of the upset mechanism of the content of a Static Random-Access Memory (SRAM) memory cell is described in Figure 28. If the current pulse is strong enough to flip the state of one of the inverters during a certain amount of time, the mirror inverter will feed the erroneous value back to the input of the affected inverter thus sustaining the error until the next write operation.

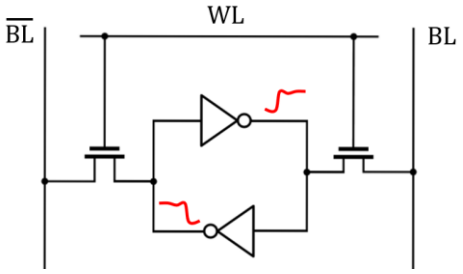


Figure 28: SRAM cell upset. If sufficient charges are generated along the particle track, the resulting current pulse can temporarily charge or discharge one of the inverters and propagate to the cross-coupled inverter thus latching the erroneous stored value.

The mechanism is quite similar for edge-triggered D-flip-flops. Figure 29 shows the behavior of this type of latch element. When the clock signal is low, the input signal is transferred and stored in the master cell while the output value is maintained in the slave cell. When the clock signal goes high, the value previously stored in the master cell is transferred to the slave cell.

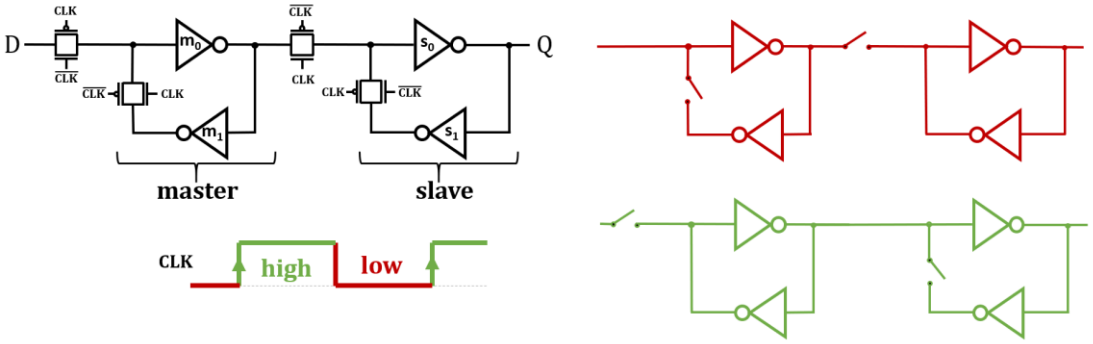


Figure 29: Edge triggered D-flip-flop behavior.

Depending on the time at which the current spike is generated and the transistor affected, the error signature may be different. Current spikes occurring during the rising edge of the clock signal will force a change state of the flip-flop that will necessarily be transferred to the rest of the circuit. Current spikes generated in the master cell when the clock is at logic level 1 and the one affecting the slave cell when the clock is at logic level 0 also generate an error that is maintained until the next clock cycle [25]. However, as the error is generated after the rising edge of the clock, it can be temporally masked if it doesn't reach the end point flip-flops before the next rising edge. The probability that this type of error propagates depends on the moment at which the error is generated and the timing margins on the path that separates it from the end point flip-flop. To be noted that with recent CMOS technologies, a single particle can generate SEU on more than one bit at the same time. This phenomenon is called Multiple Bit Upset (MBU) as opposed to Single Bit Upset (SBU).

1.2.3.3. Single Event Latch-up

Single Event Latchup (SEL) occurs when a particle deposits enough energy to trigger the parasitic thyristor formed by the arrangement of an NMOS and PMOS transistor (in a CMOS technology) as shown in Figure 30.

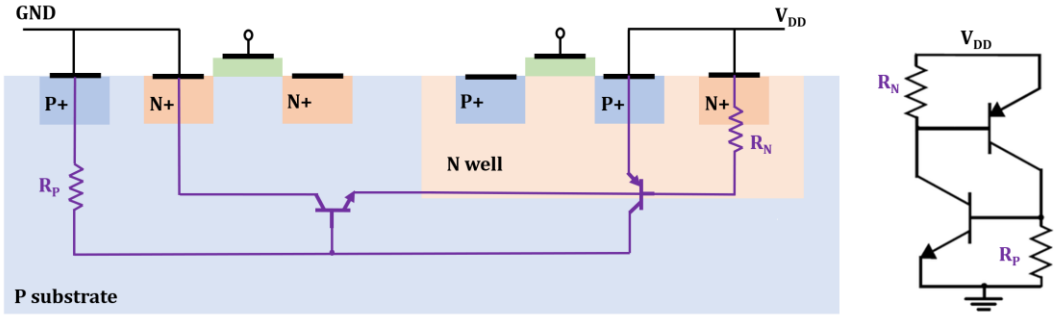


Figure 30: Parasitic thyristor in CMOS technology.

When the thyristor is triggered, it forms a regenerative feedback loop producing a low-impedance path between the power supply rail and the ground [26]. This regenerative feedback maintains the high current state and can only be removed by powering down the device. The severity of the

consequences of an SEL can vary from a simple interruption of the component's operation (recovered after power cycle) to the total destruction of the component due to an important heat generation by the induced short circuit. To mitigate the effects of SELs on an integrated circuit, the basic solution is to monitor externally the supply current of the circuit and to perform a power cycle when a significant current peak is detected.

High-energy particles can cause other types of permanent damage such as Single Event Gate Rupture (SEGR) and Single Event Burnout (SEB) in high voltage devices, however, these effects are out of the scope of this work.

In another hand, Single Event Functional Interrupt (SEFI) is a widely used terminology to define a soft error, caused by a single particle, that causes the component to reset, lock-up, or otherwise malfunction in a detectable way. This type of event does not correspond to a physical phenomenon but is often associated with an SEU or an SET capture in a control bit or register or SET in global routes (global reset). The nature of SEFIs is specific to the type of component considered.

1.2.3.4. SEE SENSITIVITY METRICS

To characterize the SEE sensitivity of a circuit, a memory cell, or a component, the cross section concept is generally used. The cross section (σ) corresponds to the probability that an interaction occurs between a particle and the investigated device. Cross section can be seen as the effective area that a device represents to a particle. Based on accelerated particles experiments, the cross section of a given device can be computed by dividing the total number of observed events by the total fluence of particles it has been exposed to (3).

$$\sigma = \frac{N_{event}}{\Theta} \tag{3}$$

Other metrics, providing a quantitative measure of a device reliability over time in a given environment can be used, such as the Mean Time To Failure (MTTF) or Failure In Time (FIT). To compute these metrics, the cross section of the device has to be evaluated for each particle type and energies to which it will be exposed in real operation conditions. For terrestrial or avionics applications, facilities providing a neutron flux with an atmospheric like spectrum are generally used. As the energy spectrum is similar to the real conditions one, the MTTF can be estimated by simply dividing the obtained cross section by the expected neutron flux in real operation conditions (4).

$$MTTF = \frac{\sigma}{\theta} \tag{4}$$

This MTTF (often expressed in hours) then represent the average time that would be expected between each failure. The FIT, representing the number of expected failures per billion hours of execution is then obtained using equation (5).

$$FIT = \frac{1}{MTTF} \times 10^9 = \sigma \times \theta \times 10^9 \tag{5}$$

However, for spatial applications, the flux and energy spectrum of protons and heavy ions strongly vary according to the considered orbit. To get an estimation of the failure contribution of protons, the cross section of the device must be evaluated for a representative part of the proton energy spectrum. The proton cross section evolution as a function of the energy is generally correlated to

a reliability model such as a Weibull distribution. Similarly, the SEE contributions of heavy ions is analyzed by computing the cross section with ions of different LET.

To compute the FIT contribution for both particle types, the obtained cross sections must be integrated with the expected proton spectra (6) and heavy ions spectra (7) of the considered environment.

$$FIT = \int \sigma(E) \cdot \theta(E) \cdot dE \times 10^9 \quad (6)$$

$$FIT = \int \sigma(LET) \cdot \theta(LET) \cdot dLET \times 10^9 \quad (7)$$

1.3. Conclusion

Radiation can induce different types of effects on microelectronics devices ranging from the progressive degradation of its performance, the corruption of the data passing through and stored in the component and the complete destruction of the component. The type of effects induced is determined probabilistically by the type of particle and its energy as shown in Figure 28. In a simplified way, and considering only natural radiations, protons and electrons produce TID effects and displacement damage while high-energy protons can also generate SEEs. Atmospheric neutrons generate SEEs through nuclear interactions. Heavy Ions, due to their high LET, are mainly considered for SEE which, although being less frequent in the space environment, can have much more severe effects (MBU, SEL). For space systems, the dose effects are mainly caused by protons and electrons, but X-rays and Gamma rays artificially generated on earth are extensively used to investigate TID effects on electronic components.

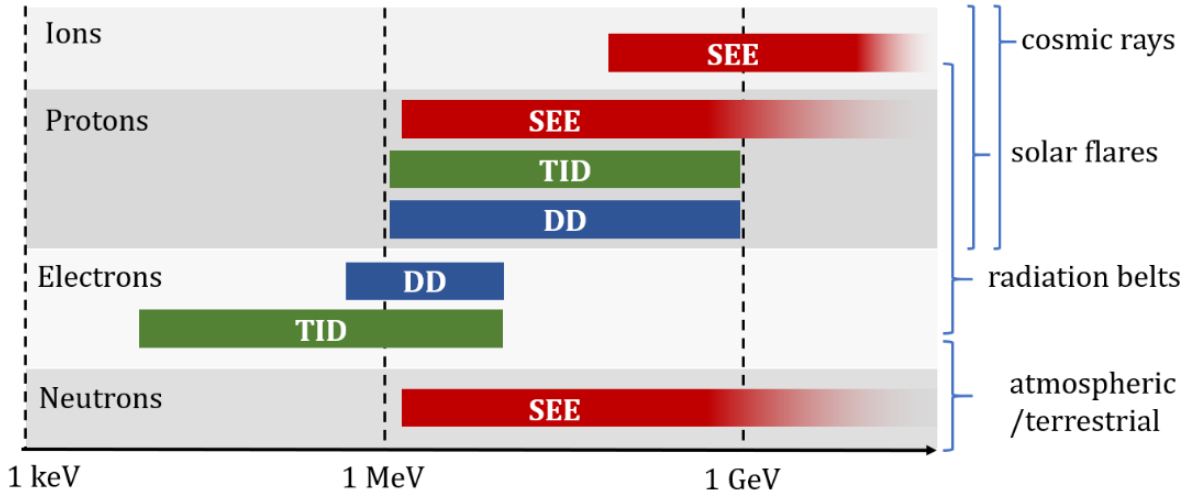


Figure 31: Simplified classification of radiation effects per type of natural radiation as a function of incident energy, inspired from [27].

TID effects affect, in a roughly uniform way, the whole component (any device containing oxides) and progressively degrade its electrical properties, while SEEs occur in a unique way and on a specific location, randomly on the chip and potentially generate soft errors. On complex components such as FPGAs (the subject of this study), these soft errors can propagate throughout the circuit potentially leading to a system failure. The structure, operation and radiation behavior of these components are described in the next chapter.

2. FPGA ARCHITECTURE

The work presented in this manuscript focuses on the reliability of programmable electronic components in radiative environments. In the previous section, different radiative environments have been described. In this section, the components studied in this work are introduced: Field Programmable Gate Array (FPGA). In a first instance, their principle of operation and their architecture will be described. Then, the different failure mechanisms induced by radiation on these components will be presented, along with a presentation of the main hardening techniques used to improve their reliability.

2.1. DEFINITION AND PRINCIPLE

Field Programmable Gate Arrays (FPGA) are integrated circuits that can be reprogrammed after being manufactured, to implement an arbitrary digital circuit. Their main principle of operation lies in an array of programmable logic blocks and a network of reconfigurable interconnects that allows the mapping of the logic blocks together. The FPGA functionality is specified by loading a binary file in a specific internal memory. Each bit of this memory is used either to define the functionality of a logic block or to activate a specific connection between routing tracks. In contrast to standard microprocessors, which can only execute logical operations in a sequential manner, FPGAs can implement digital functions with a high level of parallelization just like Application Specific Integrated Circuits (ASIC), with the difference that their functionality is not permanently defined in production stages but can be reprogrammed on the field.

2.1.1. ADVANTAGES AND DRAWBACKS

Inheriting from PLDs (Programmable Logic Devices), the first FPGA was invented in 1985 by the co-founders of *Xilinx*. Since their introduction, the capacity, speed, complexity and flexibility of these components have been steadily increasing. Although the performance of FPGAs is generally lower than their ASIC counterparts, FPGAs are increasingly being used, not only for prototyping but also to replace ASICs in a wide range of applications. This growing use is due, on one hand, to their increasing performance and capacity and on the other hand to their low Non-Recurring Costs (NRC), short time-to-market, and their reprogrammability. Indeed, the possibility to reprogram the component "on the field", provides the opportunity to modify, correct and update the system functionality after its deployment. By bypassing all the engineering steps related to the IC layout and to the manufacturing process (masks creation, calibration, etc.), most of the Non-Recurring Engineering (NRE) costs and production delays are eliminated. However, the unit cost of an FPGA component becomes higher than an ASIC once the production volume reaches a certain threshold, as illustrated in Figure 32.

In recent years, the use of FPGAs is growing, especially for real-time systems where response time plays a crucial role: telecommunications, automotive industry, aerospace, medicine, audio, finance, computer science, video, security but also more recently in specific computer applications, such as cryptocurrency mining and deep learning algorithms. In the space industry, the use of FPGA in satellite payload is becoming more and more attractive due to the low production volume and to the extension of the payload lifetime it provides by updating its functionalities to follow the evolution of telecommunication protocols.

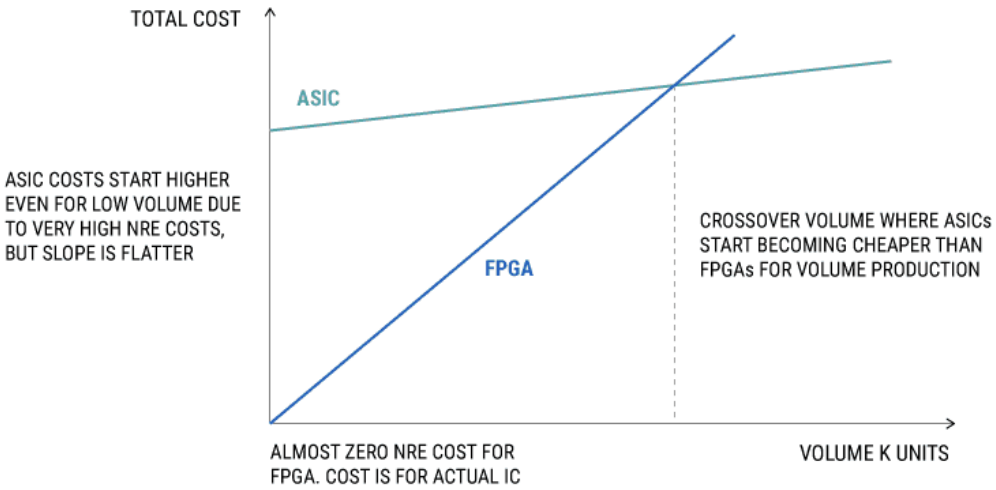


Figure 32: FPGA vs ASIC cost analysis, from [28].

The sensitivity of commercial FPGAs to radiation effects and the high cost of radiation-hardened FPGAs are the main obstacles to their widespread use. The comprehension of degradation and failure mechanisms induced by radiation, the development of test methodologies and mitigation techniques are key factors that play an important role for the wide use of FPGAs in the space industry among others that are concerned by radiation effects.

2.1.2. Development workflow

To configure an FPGA, the user must follow a multi-step development flow, comparable to the digital ASICs development flow, as described in Figure 33.

The first step to design a digital circuit with an FPGA is to describe the intended functionality using a Register Transfer Level format (RTL) via a hardware description language such as VHDL or Verilog. The concept of RTL design is to describe the behavior of a circuit in terms of signals sent or data transferred between registers and the logical operations performed on these signals. With the increasing complexity of digital systems, new design methodologies, called High Level Synthesis (HLS) have appeared to describe the system functionality with a higher level of abstraction by using programming languages such as C/C++. This high-level description is then automatically translated into RTL.

Next comes the **synthesis** process in which the design tool interprets the provided RTL description and translates it into a logic circuit that matches the described functionality. The resulting logic circuit is built by instantiating the lowest level elements of logic that compose the FPGA fabric (called primitives). The synthesis result is thus entirely dependent on the targeted FPGA architecture. The development flow of digital ASICs uses a similar process although the design of the circuit is done by instantiating standard cells. These standard cells, generally provided by the foundry libraries, are logic gates of varying degrees of complexity, provided with a physical layout (at transistor level) tailored to the technological process used.

The development workflow continues with the **placement** step where each primitive element of the previously synthesized logic circuit is assigned a precise position in the FPGA matrix. Since an FPGA chip contains many replicas of each type of primitive, the placement process is driven to

limit as much as possible the area overhead and the physical distance between logic gates linked by a common net.

The placement step is followed by the **routing** step which consists in physically linking the logic gates together. To do this, specific routing tracks of the routing matrix are selected and activated to build the physical connections. This part of the flow is driven to minimize the propagation delay of each net. For ASIC counterparts, the placing and routing stages are replaced by a physical layout stage of the IC: standard cells are physically placed and routing tracks are drawn on the dedicated chip area.

Finally, the binary configuration file (**bitstream**) is generated by setting all the configuration bits that must be activated in the FPGA configuration memory to match the circuit architecture implemented in previous steps. This bitstream is then programmed in the FPGA configuration memory.

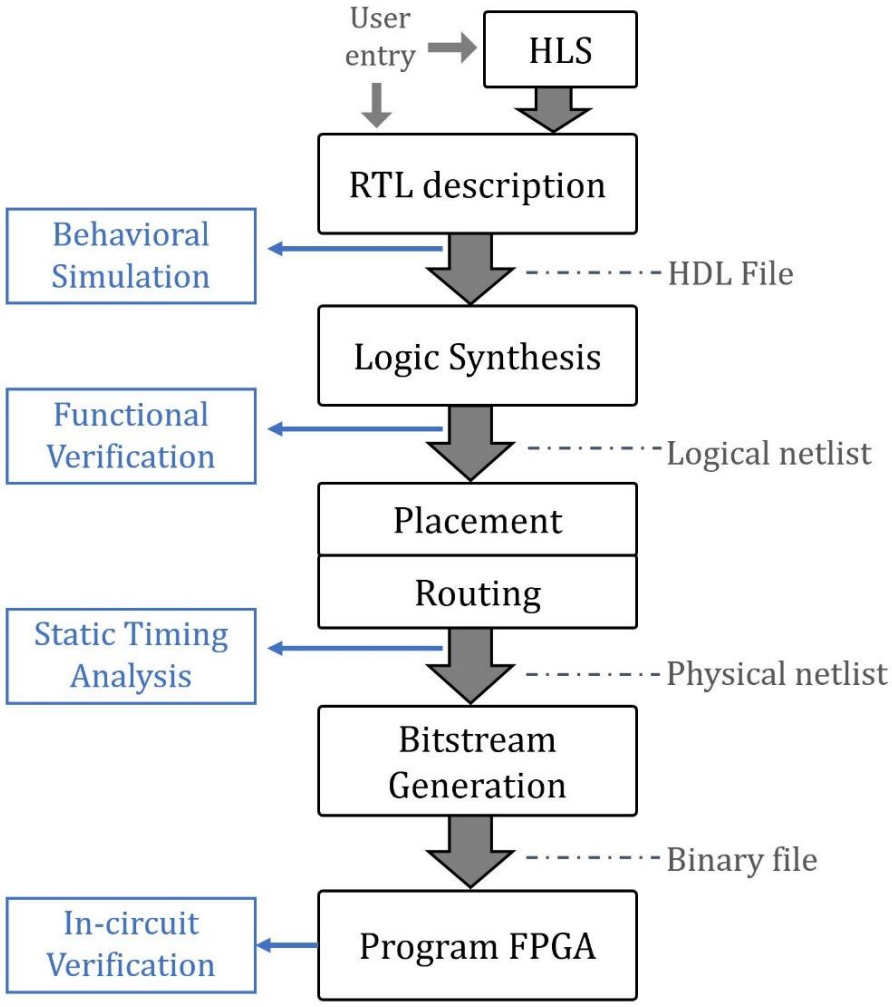


Figure 33: Development workflow for FPGA-based design.

Each step of this development flow can be customized and driven by the designers by using constraints and directives. The development workflow also integrates simulation and verification tools that allow to validate the functionality of the design at each step. The simulation can be performed on the RTL description, on the post synthesis circuit and on the implemented circuit.

Each type of simulation grants access to different levels of circuit analysis: RTL simulation simply provides insight on the functionality of the circuit while physical simulation, based on Static Timing Analysis (STA), can consider all the timing constraints that can alter the behavioral operation of the circuit.

2.2. Architecture description

2.2.1. Overview

The flexibility of FPGAs is achieved thanks to the combination of Configurable Logic Blocks (CLB) and a routing matrix as shown in Figure 34. These logic blocks, designed to handle a wide diversity of Boolean and sequential functions, are replicated on the entire chip area. By combining several blocks together via the programmable routing matrix, any type of digital function can be implemented.

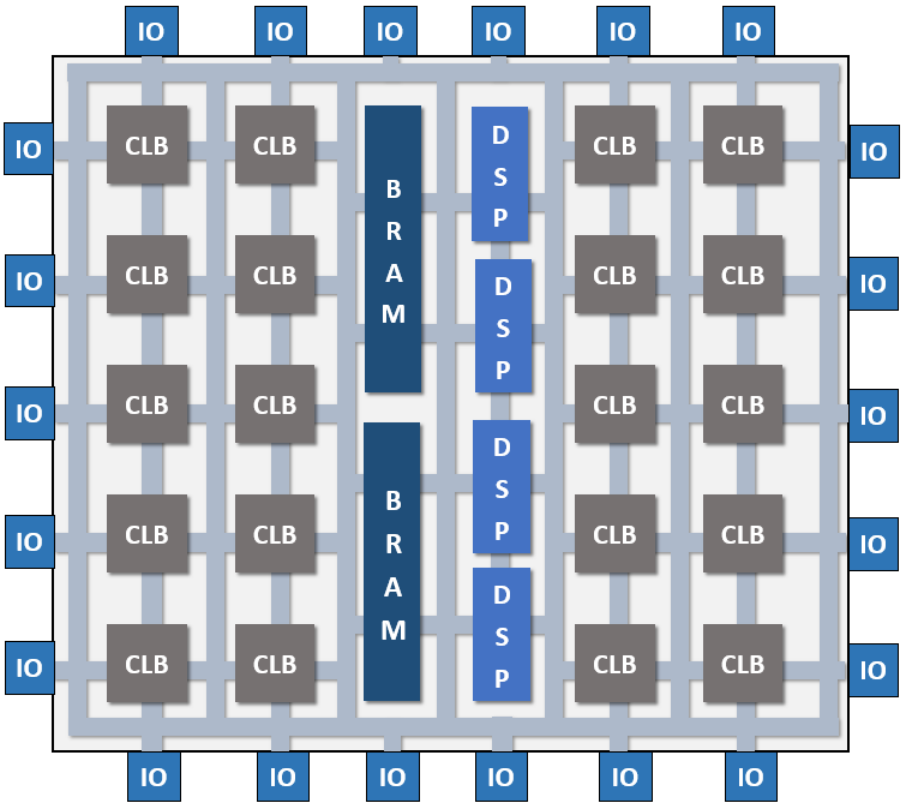


Figure 34: FPGA architecture overview: Configurable Logic Block (CLB), Input/output (IO) pads, block RAM (BRAM) and other specialized blocks such as Digital Signal Processing blocks (DSP) are connected together through a programmable routing matrix.

The structure of the logic blocks and the routing matrix have significantly evolved since the invention of the first FPGA and there are still notable differences between the FPGAs of different families and different vendors. The level of granularity of the elementary logical resources used in the CLBs is a main criterion defining the trade-off between area, performance and power consumption of the component. Even if the choice of this trade-off (LUT size, number of LUT per CLB, global routing scheme) significantly evolved in the last decades, the main CLB structures and concepts are now closely shared between all modern FPGAs. The flexibility brought by a low-level granularity in the CLB comes at the cost of a strong decrease in performance compared to ASIC

counterparts. To limit this performance gap, most modern FPGAs now integrate a higher level of granularity by integrating specialized blocks to efficiently implement commonly used functions such as memory blocks and arithmetic operators.

The detailed architecture description below and the used terminology are based on the architecture of *Xilinx* FPGAs. With certain differentiations the described architectures are very close to what other vendors implement as well.

2.2.2. CONFIGURABLE LOGIC BLOCKS

The purpose of a CLB is to provide the basic computation and storage elements used in digital logic systems with a relatively low level of granularity. In *Xilinx 7 series* FPGAs, a CLB is a pair of slices, and each slice is composed of Look-Up Tables (LUT), Multiplexers (MUX) and D-type flip-flops (FF).

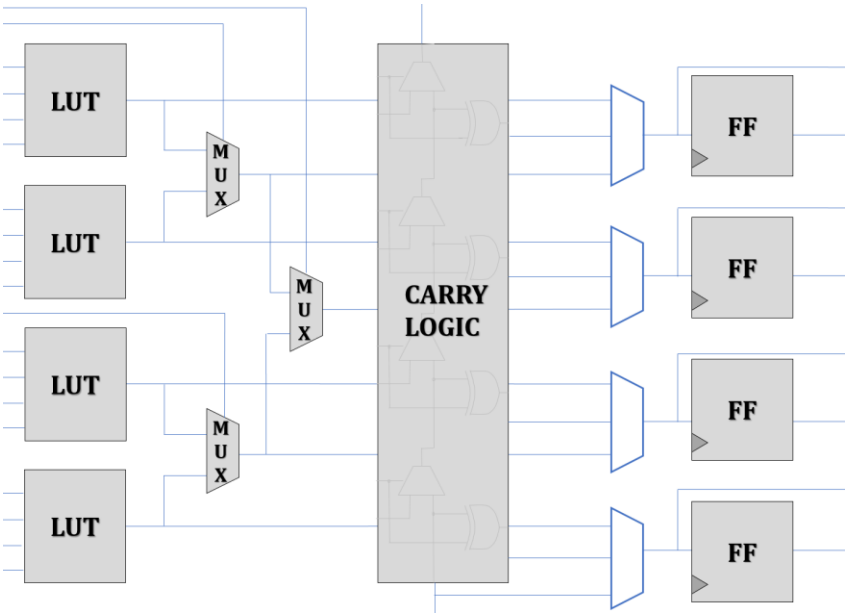


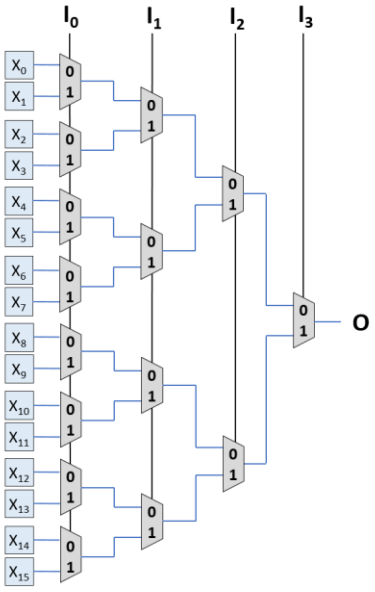
Figure 35: Schematic representation of a slice inspired by the one use by *Xilinx 7 series* FPGAs.

A Look-up table is a combinatorial logic cell that can be configured to generate the desired output for each input value. LUT is the core of the configurable logic as it can be programmed to emulate the function of any logic gate. An LUT is actually a Read Only Memory (ROM) which is initialized with the results of the truth table of the intended logical function. The address bus is used as the gate input and the contents of the ROM are multiplexed to the output. A basic LUT architecture with 4-inputs is described in Figure 36.

Any N-input LUT requires 2^N configuration memory bits to store its functionality. The FPGA market is now dominated by FPGAs with LUT size ranging from four to six inputs as these LUT sizes have shown the best trade-off between area and routing [29]. Several optimization features have been added over the years, like combining two LUTs that share the same inputs to either operate as a logic gate with two outputs or as a bigger LUT. Some FPGA technologies also integrate LUTs that can be dynamically reconfigured to operate as a dual port RAM (LUTRAM) or shift registers (SRL) [30].

As shown on Figure 35, slices integrate multiplexers in order to create wider logic operators by combining the LUTs' outputs. However, arithmetic operations (additions, multiplication, etc.)

cannot be implemented efficiently using only LUTs and such multiplexers. Indeed, to compute any type of arithmetic operation, a carry bit must be propagated between the bits of increasing weight which requires a significant overuse of routing resources and high propagation delays. To improve the implementation of this type of application, slices now integrate dedicated logic gates and routing paths to efficiently propagate the carries across bit weights and across slices.



I_3	I_2	I_1	I_0	O
0	0	0	0	X_0
0	0	0	1	X_1
0	0	1	0	X_2
0	0	1	1	X_3
0	1	0	0	X_4
0	1	0	1	X_5
0	1	1	0	X_6
0	1	1	1	X_7
1	0	0	0	X_8
1	0	0	1	X_9
1	0	1	0	X_{10}
1	0	1	1	X_{11}
1	1	0	0	X_{12}
1	1	0	1	X_{13}
1	1	1	0	X_{14}
1	1	1	1	X_{15}

Figure 36: 4-inputs Look-up table architecture.

Finally, slices integrate D-type flip-flops to register the results the combinatorial functions. These flip-flops can be configured to define the type of clock edge it is referring to, the polarity of the reset signal, the use or not of clock enable signals as well as the initial value.

2.2.3. Configurable Routing matrix

One key issue for FPGA performance is the organization of the global routing architecture, which is the macroscopic allocation of routing segments to link all the logical resources together, minimizing the propagation time and maximizing the flexibility of the network. Two main types of global routing architectures are used in modern FPGAs: hierarchical and island-style.

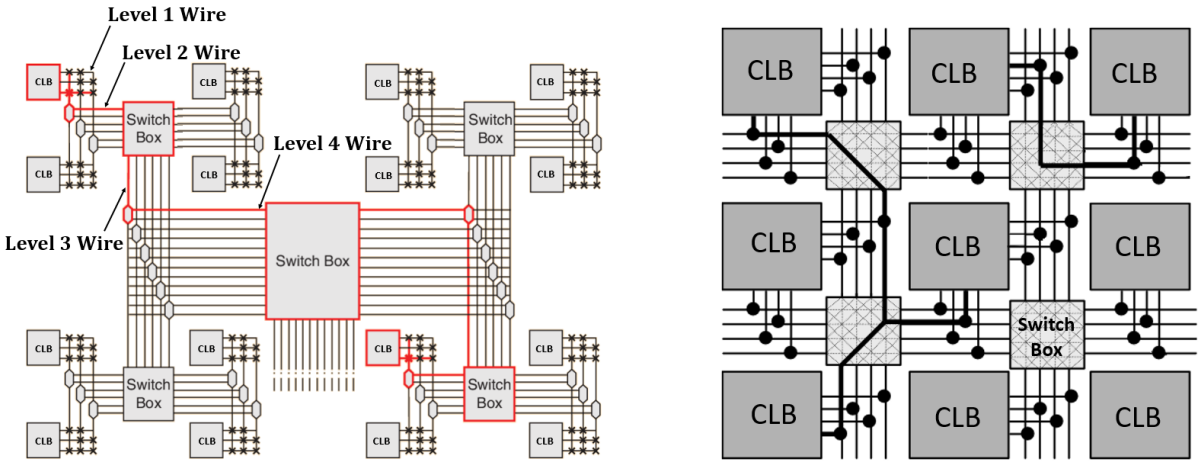


Figure 37: Global routing architectures, from [31].

Hierarchical routing architecture separate FPGA logic block into distinct groups. Connections between blocks within the same group can be made using the lowest level of wire segments, but connections with blocks in other groups require the traversal of one or more hierarchical level of routing segment. Even if this architecture offers more predictable routing delays, its lack of flexibility can lead to many disadvantages for some applications. Indeed, if the hierarchy of the design does not match the one of the routing architectures, it may suffer from large timing penalties. For this reason, most recent commercial FPGA use island-style global routing architecture. In island-style FPGAs, logic blocks are arranged in two-dimensional grids with routing resource evenly distributed throughout the mesh. Vertical and horizontal wire segments of different length are employed and connected through programmable switchboxes offering a higher flexibility and a better average timing performance.

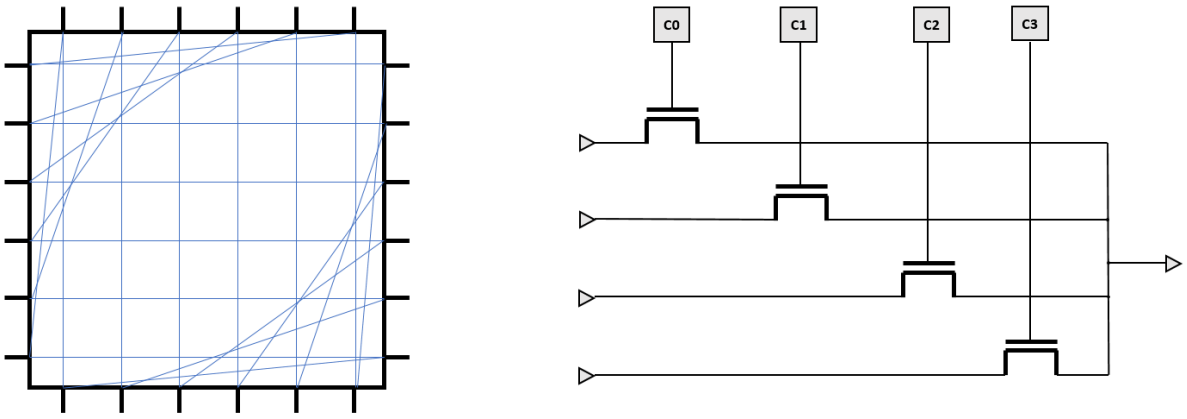


Figure 38: Subset type switch box (left) using routing multiplexer (right).

Switchboxes are built to connect wire segments together using **P**rogrammable **I**nterconnexion **P**oints (PIP). To reduce their footprint, only a subset of all the possible connection between wires of the same switchbox are typically available as represented on Figure 38. For each wire, a certain set of the other wires can be connected to it through a routing multiplexer. These routing multiplexers (Figure 38) are built by pass-transistors activated by configuration memory bits. The detailed structure of a routing multiplexer is usually not accessible by users but more information on this structure for *Xilinx 7 Series* FPGA is provided in chapter 5.

Another important element of the routing matrix is the buffer, either used to route high fanout and global signal such as clocks, reset and clock-enable signals. Most FPGAs also use specific routing tracks for global signals to limit the time skew across the chip. To communicate with the outside world, FPGAs use input/output banks whose parameters can be configured: direction (input/output/bidirectional), voltage, logic level standards (LVCMOS/LVTTL), impedance (digitally controlled impedance), slew rate, termination type (Single-ended/differential) etc.

2.2.4. Specialized Blocks

As mentioned earlier, to overcome the timing and consumption penalty induced by the FPGA fabric flexibility, many coarse grain resources have been added to efficiently implement widely used Intellectual Properties (IPs):

- Block RAM (BRAM): a memory array which can be accessed with different data size and depth, as single port memory or double port memory. Some FPGA families even integrate a

dedicated logic to use the BRAM as FIFO while providing the required synchronization logic for asynchronous use.

- Digital Signal Processing blocks (DSP): a computing unit allowing to implement a certain set of arithmetic operations. The most advanced FPGAs integrate DSPs allowing to perform addition, multiplication, division, subtraction, accumulation, and various bitwise operations with several configurable levels of pipelining as shown in Figure 39.

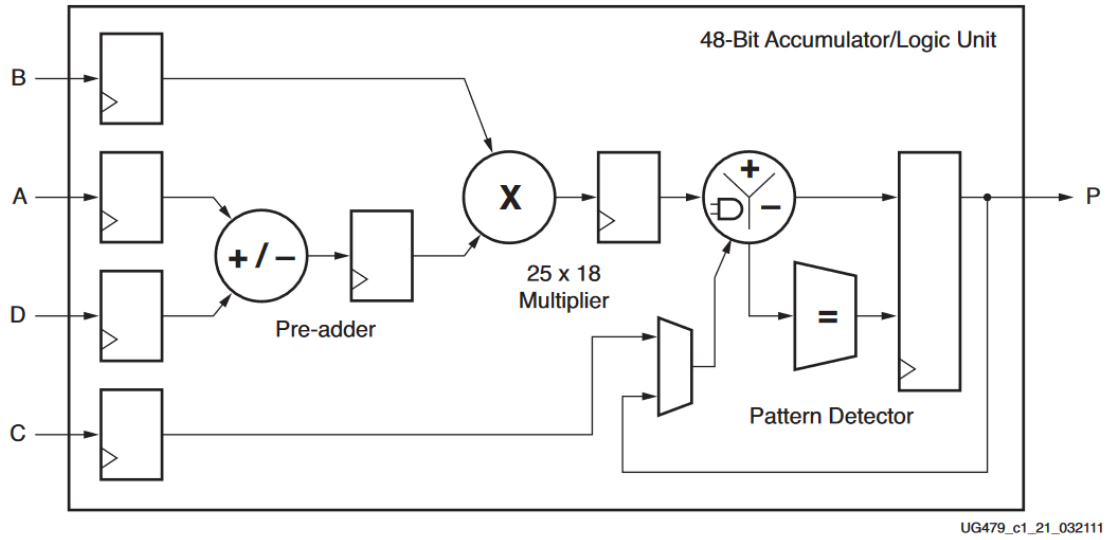


Figure 39: Digital signal processing block in *Xilinx* FPGAs (7series and later) from [32].

- Phased Locked Loop (PLL) for clock generation and synchronization.
- Analog to Digital Converters (ADC) and Digital to Analog Converters (DAC).
- AES Encryption/Decryption modules.
- High-speed transceivers.
- Ethernet MAC, PCIe Gen4 among others.

Each generation of FPGA integrates new dedicated modules to enable higher performance interfaces through a wide diversity of communication protocols. In addition, System on Chip (SoC) FPGAs integrate microprocessors tightly coupled with the FPGA fabric through high speed buses, further advancing the capabilities of these devices.

2.2.5. CONFIGURATION MEMORY CELL TECHNOLOGIES

As explained previously, all programmable logic resources and programmable routing points are configured by memory elements that define their functionality. Depending on the technology of these memory elements FPGAs are divided in three categories: Static Random Access Memory (SRAM), Flash and Antifuse. Each type of technology brings its own advantages and drawbacks, whether in terms of manufacturing process, performance, power consumption and component flexibility.

2.2.5.1. STATIC RAM

SRAM cells uses latching circuitry to store the information bit as a voltage level (GND or VCC). The most common SRAM structure uses six MOS transistors as shown in Figure 40. Four transistors form two cross-coupled CMOS inverters and two transistors are used to control the access to the cell content during read and write operations.

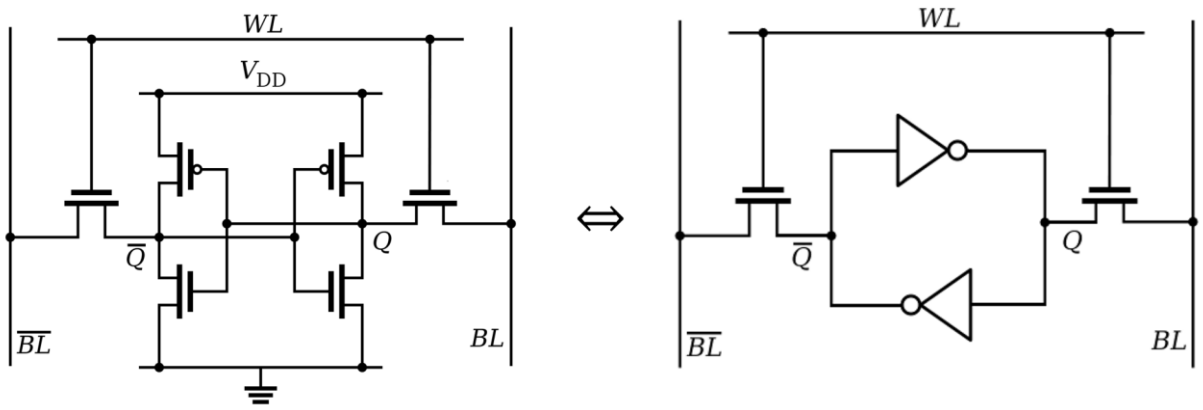


Figure 40: SRAM cell using six transistors (6T) by forming a latch with two interleaved CMOS inverters.

SRAM offers a simple and very fast data access and does not require a refresh circuit to maintain the data. However, SRAM cells are volatile, meaning that the data is lost when power is removed. SRAM based FPGAs thus require an external memory to store the bitstream and load it in the FPGA configuration memory at power up. Thanks to this simple and fast access to data content, SRAM based FPGA benefit from additional features like using the LUT content as user writeable memories [33], BRAM content initialization through bitstream and partial reconfiguration [34] in which only a part of the implemented design can be reconfigured while the rest of the design can continue running.

2.2.5.2. FLASH MEMORY

Flash memory cells are a type of Electrically-Erasable Programmable Read-Only Memory (EEPROM) that use a floating-gate transistor. Floating-gate transistor, described in Figure 41, are N-type transistors with an additional floating gate buried in the middle of an oxide, between the channel and the grid. The data is stored by trapping electrons on this floating gate through a tunneling effect by applying high voltage on the control gate or through hot carrier injection. Electrons trapped in the floating gate repel the electrons in the channel preventing the current from passing from source to drain even when the control gate is ON, which will be detected as a logic '0'. If the floating gate is not charged, the current may flow between source and drain, which will be detected a logic '1'.

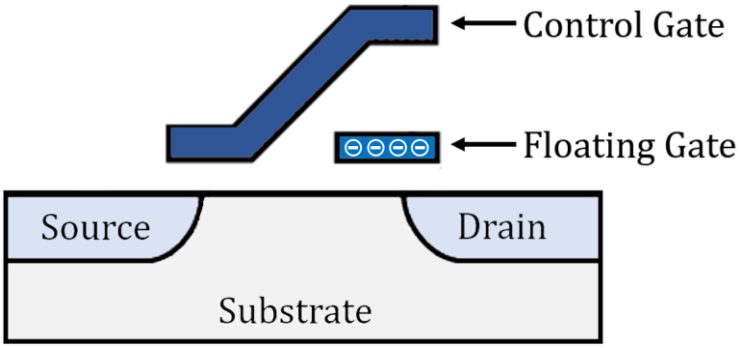


Figure 41: Cross section view of a flash cell: a floating gate is buried in the middle of the oxide. Data is stored by trapping or releasing electrons in the floating gate.

This memory cell technology is non-volatile, which implies that the information is kept after power down. This means that flash-based FPGAs does not need external memory to store the bitstream, thus reducing the required printed-circuit board area and canceling the bitstream loading latencies and energy overheads at power up. Unlike SRAM cells, the nature of this memory cell technology prevents them to be used dynamically by the implemented system. Therefore, LUTs cannot host writable memory functions, and BRAM must be implemented with SRAM cells, thus requiring additional stages to initialize their content from the flash memory. Nonetheless, flash-based devices benefit from many power-saving advantages with lower static and dynamic power consumption as well as advanced sleeping modes [35].

2.2.5.3. ANTIFUSE

An antifuse is a normally open circuit dipole that can be permanently converted into an electrically conductive path by applying a certain voltage level. Antifuse can be built using either a very thin oxide barrier that can be converted into conductive channel by dielectric breakdown or amorphous silicon (low conductivity) that can be turned into polycrystalline silicon (high conductivity) through the action of a strong electric field. By nature, antifuse based FPGAs are one-time programmable: once programmed, the antifuse state is permanent. As a result, they do not benefit from all the advantages brought by reprogrammability. In return, they provide some benefits: lower power consumption and shorter propagation delays for programmable routing points.

2.3. RADIATION EFFECTS ON FPGAS

In section 1.2, general radiation effects on electronics have been presented. In this section, the radiation effects and failure mechanisms specific to FPGA based systems are further discussed. The physical radiation effects related to the configuration memory will be first presented before describing the effects on the rest of the component for both TID and SEEs along with a description of the associated failure mechanisms.

2.3.1. RADIATION EFFECTS ON CONFIGURATION MEMORY CELLS

When it comes to radiation tolerance, the key difference between FPGAs and ASICs lies in the sensitivity of the configuration memory to both TID and SEEs. Indeed, the type of technology used to store the FPGA configuration has a major impact on the radiation tolerance of the component.

2.3.1.1. SRAM

For SRAM technologies, the main reliability issue lies with their high sensitivity to SEU [36]. Indeed, as described in section 1.2.3, when an SRAM cell holds a value, two transistors from the inverter pair are *ON* while the two remaining are *OFF*. This means that there are always two SEU sensitive nodes in the cell: if a particle strikes one of these nodes, transistors in the “off” state can be switched “on”, thus flipping the stored value. As for TID, even if SRAM cells can be theoretically corrupted by an excessive TID induced standby current increase or by a reduced read and write voltage margins [37], SRAM cells are generally considered as incorruptible by TID as most modern SRAM devices can reach TID tolerance over 1Mrad(Si) [37]–[39]. This is particularly true for FPGAs for which other on chip circuitry such as the programming or power-on reset circuitry have a much lower TID tolerance [40], [41]. Nevertheless, some studies have demonstrated that the

accumulated dose on SRAM cells can cause an important increase of their SEU sensitivity [42]–[44].

2.3.1.2. FLASH

As opposed to SRAM-based FPGAs, flash cells are known to be strongly sensitive to TID. Indeed, as described in section 2.2.5.2, the data retention in flash cells is carried out through the accumulation of electric charges in the floating gate, locally modifying the threshold voltage of the transistor to prevent or not the current from passing from source to drain. According to [45], [46], three basic mechanisms are responsible for the TID-induced corruption of the flash cell:

1) Electric charges generated in the surrounding oxides and injected into the floating gate through the action of the local electric field. This effect decreases the number of charges in the floating gate as the charges attracted by the electric field (holes) are opposed to the one stored on the grid (electrons) and will therefore recombine.
2) Electric charges generated and trapped in the surrounding oxides. As for any MOS device, this phenomenon results in an accumulation of positive charges (due to their lower mobility).
3) Carriers in the floating gate received sufficient energy from the radiation to escape the potential well through photoemission, thus reducing the number of charges in the floating gate.

These mechanisms will alter the local threshold voltage drift of the floating gate transistor, thus potentially corrupting the stored information. Beside these floating gate related effects, the weakest point of the flash-based devices generally come from the peripheral circuitry including decoders and buffers but mostly from voltage multiplications circuitry (charge pumps) providing the high voltages required for carrier injection in the floating gate (write operation). In several studies [47], [48], this charge pump circuitry has been reported to be the first cause of failure due to inability to perform write operations.

Regarding single event effects, the main observed phenomenon on floating gate cells is a drift of the threshold voltage induced by a particle passing through or near the cell. Generally, this drift is observed only on programmed cells (floating gate storing electrons) but not on erased cells (no charge stored)[49]. The observed V_{TH} drift, due to a charge loss in the floating gate is linearly proportional to the particle LET [45]. Two mechanisms have been proposed in the literature to explain this SEE induced charge loss. The first one assumes the creation of a transient conductive path through the tunnel oxide that discharge the floating gate [49], [50]. The second model considers the generation of energetic carriers by the impinging radiation that fluxes in and out the floating gate through tunneling currents [51]. Anyway, if the threshold voltage drift exceeds the reading voltage of the reading circuitry, the stored information is corrupted. To be noted that the cell is still functional and can be reprogrammed, only the stored data is lost. For most flash technologies, low LET particles such as protons have a very low probability of generating an upset, flash cells are considered immune to SEU for these particles [52], [53]. For heavy ions, the upset probability is not negligible and increases with the ion LET. As the natural presence of these ions is much lower, the SEUs in the configuration memory of a flash-based FPGAs are generally not considered in their reliability studies.

2.3.1.3. ANTIFUSE

Antifuse offer the best resistance to the radiation effects as these memory cell are fairly considered as immune to both TID [54] and SEE.

2.3.1.4. SUMMARY

The main radiation effects for each type of memory cell are summarized in TABLE I. In a simplified way: SRAM are sensitive to SEU but not to TID while Flash are sensitive to TID and not to SEU. The failure mechanisms associated with radiation effects on configuration memory cells are described in the next section.

TABLE I: MAIN RADIATION EFFECTS ON THE THREE MEMORY CELL TYPES USED BY FPGAS

Memory cell	SRAM	Flash	Antifuse
TID	• **No data corruption** • Increased power consumption • Increased SEU sensitivity	• **High sensitivity leading to permanent cell corruption** • **Sensitive charge pumps circuitry**	• **Not sensitive**
SEE	• **High SEU sensitivity**	• **Low SEU sensitivity**	• **Not sensitive**

2.3.2. TOTAL IONIZING DOSE EFFECT – PARAMETRIC DEGRADATION

Beyond the intrinsic sensitivity of the configuration memory, the rest of the FPGA fabric also exhibits TID induced effects. As introduced in section 2.2.2, TID effects induce 3 main types of degradation on CMOS transistors parameters:

- Negative threshold voltage drift induced by the accumulation of holes in the gate oxide.
- Decrease of the transconductance due to interface states in the Si/SiO_2 interface.
- Increase of the leakage current due to the creation of parasitic conduction paths on the transistor edges.

These effects at the transistor level result mainly, at the system (entire FPGA) level, in an **increase of the power consumption** and a **degradation of the propagation delay of the logic gates**. This degradation of the propagation delay is even more pronounced for programmable pass transistors controlled by a floating gate (flash memory cells). Indeed, the TID induced threshold voltage drift on these flash cells (described in the previous section) decrease the conductivity of the pass transistor thus increasing its propagation delay [55]. These transistors degradations can also influence the shape of the signals that pass through the logic gates, whether in terms of rise time, fall time or duty cycle. These timing degradations can result in system failure if the drifts exceed the timing margins taken during design stages. Indeed, during Static Timing Analysis (STA), the propagation delays of each net is analyzed to ensure that the propagated signal is correctly stabilized before the next clock rising edge when reaching the end-point flip-flop. This timing analysis takes into account the influence of manufacturing process variations, clock skew, and temperature variations to ensure that the timing margins applied are sufficient to ensure the

circuit functionality in all conditions. Timing degradations induced by TID effects can override these margins on the most critical paths of the design and cause system failures due to faulty signal capture by the flip-flops. When the parametric degradation of transistors reaches a certain point, their ability to commutate can be compromised, potentially leading to complete failure of the device.

2.3.3. SINGLE EVENT EFFECTS

A key point to understand and study the radiation sensitivity of FPGA based systems is to distinguish the intrinsic sensitivity of the component and the one of the systems implemented on it. Each primitive resource that composes the FPGA fabric has its own sensitivity to SEE and the reliability of the implemented design results from a complex combination of the individual sensitivities of each primitive instantiated and their manifestation on the system operation.

The first step is to define the type of SEE each type of basic resource (primitives) are sensitive to. SEE can be classified in five main groups:

- SEUs in the configuration memory (described in section 2.3.1)
- SEUs in the user flip-flops and other memory elements (BRAM)
- SETs mainly generated and propagated in the combinatorial logic (but more generally any resource that uses transistors)
- SEL, that can be generated anywhere in the device
- SEFI, mainly related to global resets and configuration capabilities

The second step is to identify how each type of errors can alter the behavior of the system.

A bitflip in the configuration memory can have a wide range of effects on the system behavior depending on the type of primitive it is linked to and its contribution to define its functionality. The main effects are illustrated in Figure 42: configuration bits related to programmable routing point can create open faults on an existing net or bridges between two distinct nets. Routing multiplexer can also modify the originally routed input for another one. The bits defining the LUT logic equation can alter the intended function when flipped, but the error is propagated to the output only if the corresponding input combination is applied. There are many other types of failures due to SEU in the configuration memory, such as bitflips on bits defining the functionality of specialized blocks (PLL, DSP, SerDes etc.). However, these are very specific to each FPGA family and all FPGA families have their own programmable architecture whose exact bitstream composition is proprietary information. As a result, without reverse engineering of the bitstream composition, the real effect of each configuration bit cannot be identified and precise failure models cannot be established. A more detailed description of the failure model for *Xilinx 7 series* FPGA is described in a chapter 5.

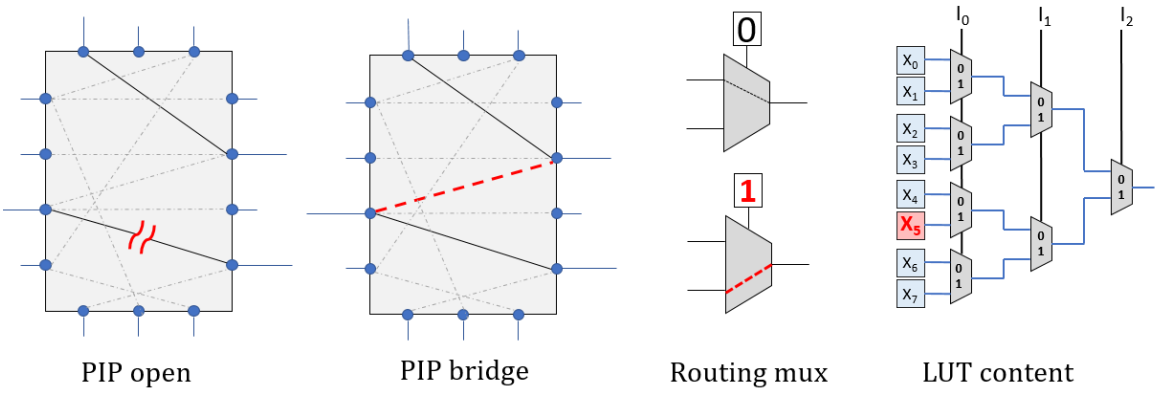


Figure 42: Possible Effects of SEUs on configuration memory.

Note that the SEUs on the configuration memory are permanent effects. The bitflip persists until the configuration memory is rewritten. These errors are therefore very important to consider as they can cause long downtime and thus greatly reduce the availability of the system.

On the contrary, the state of flip-flops affected by an SEU is restored at the next clock cycle. The error can nevertheless spread to the rest of the circuit and cause similar system failures. On the other hand, SEUs in the user memory blocks are revealed when the affected memory point is read and the error persist until the memory point is rewritten.

Single Event Transient are much more complex to consider. They can be generated on any logic gate or pass transistor in the device and their propagation cannot be predicted accurately without proprietary information of the electrical properties of the FPGA primitives or extensive SET propagation tests. Indeed, when the SET propagates through logic gates, the current pulse, a double-sided function, can be subject to narrowing or broadening effects as well as amplifying or attenuating effect depending on the electrical properties of the logic gates through which it propagates [25], [56]–[58]. These effects have a direct impact on their probability of being captured by a flip-flop and therefore on the potential failures that this may cause.

To analyze the impact of SEUs and SETs on the functionality of the system, it is necessary to understand how and when the errors can propagate to the outputs or critical nodes of the system. Indeed, logical errors are subject to logical masking and temporal masking effects during their propagation. Logic masking occurs when an error at the input of a logic gate does not alter the output logic state. Figure 43 shows the logic masking conditions for an AND gate.

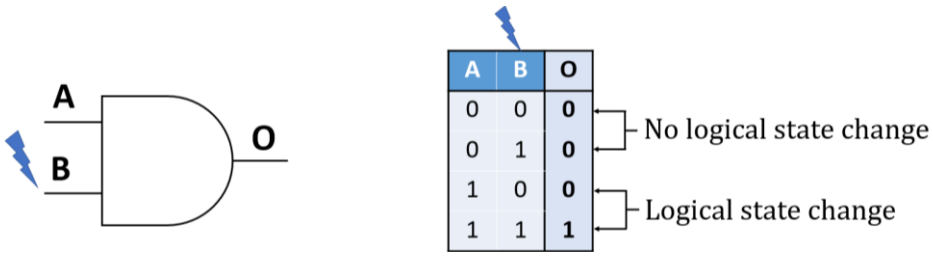


Figure 43: Logical error masking on AND gate. Error on input B is masked when input A is low as B state has no impact on the output value for this input combination.

Time masking occurs when a logic error propagates to the input of a flip-flop outside the capture window. The capture window of a flip-flop corresponds to the time interval around the rising edge of the clock where the input signal is acquired, latched and transferred to the output. To design a

reliable synchronous circuit, one must ensure that the input signal is stable for a small amount of time prior (setup time) and after (hold time) the rising edge of the clock. If the input signal state changes during this capture window, the flip-flop may enter a metastable state, during which the value and settling time of its output state cannot be predicted. If an error propagates to the input of a flip-flop, the error will only be captured if the erroneous value reaches the flip-flop input pin with a sufficient width and amplitude during the capture window.

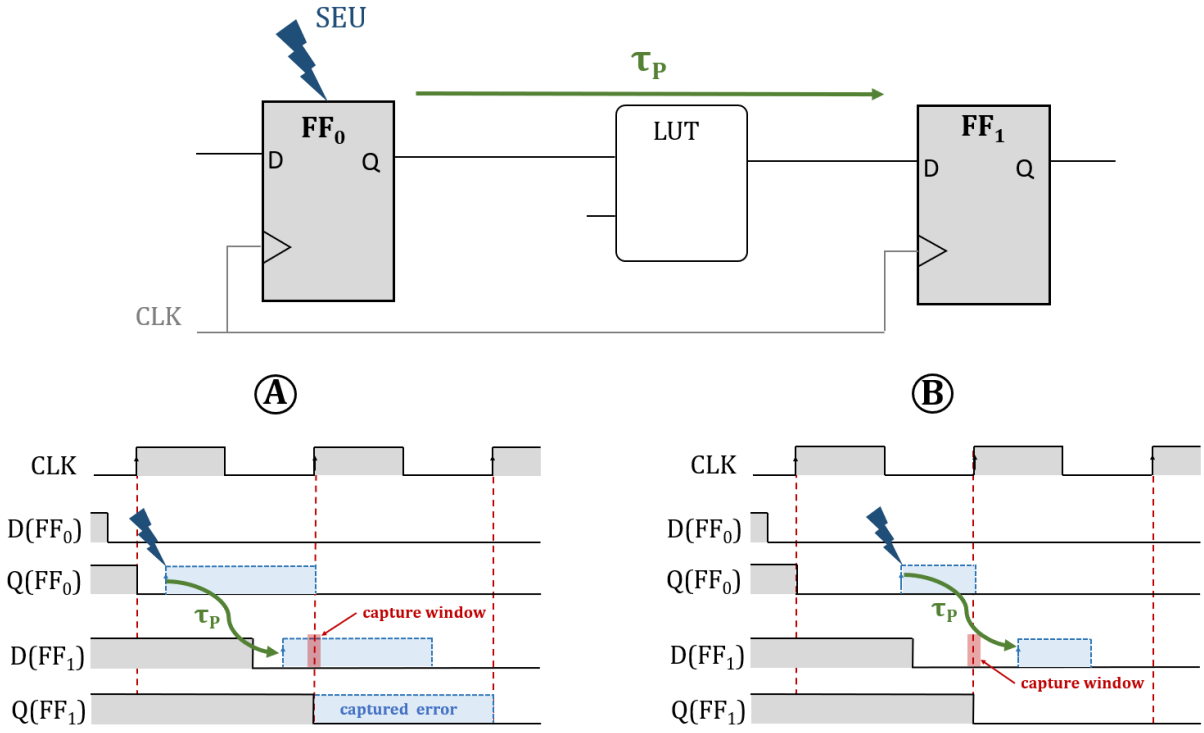


Figure 44: Propagation and capture of SEU generated on a flip-flop. SEUs generated early enough after the clock rising edge (A) are captured while SEU generated later (B) are temporally masked.

As mentioned in section 1.2.3.2, SEU affecting flip-flops can be generated at any time. Once generated, the output value is corrupted until the next rising edge of the clock. If not logically masked, the error propagates to the next flip-flop as described in Figure 44. Depending on the time at which the SEU is generated, the error can reach the next flip-flop before or after the next clock cycle. If the error arrives before the rising edge of the clock, the error fall inside the capture window and is captured. On the other side, if the error reaches the flip-flop after the rising edge of the next clock cycle the error is temporally masked. Only SEUs generated in the time interval $T_{clk} - \tau_p$ after each clock rising edge (with T_{clk}, the clock period and τ_p the propagation delay between the two flip-flops) are captured. The probability of capture is thus defined by (8):

$$P_{FF->FF}^{capture} = (T_{clk} - \tau_p)/T_{clk} \quad (8)$$

This means that for a given propagation delay, the higher the clock frequency and the closer to the maximum operational frequency, the lower the capture probability [25].

On the opposite, the capture probability of SETs is proportional to the clock frequency. Indeed, the current pulses can be generated at any time with an amplitude and width that are independent from the clock frequency, the probability that they reach the input of a flip-flop at a rising edge of

the clock is increased as the number of rising edges per second is higher. Figure 45 shows an example of SET generated in an LUT and broadened during its propagation.

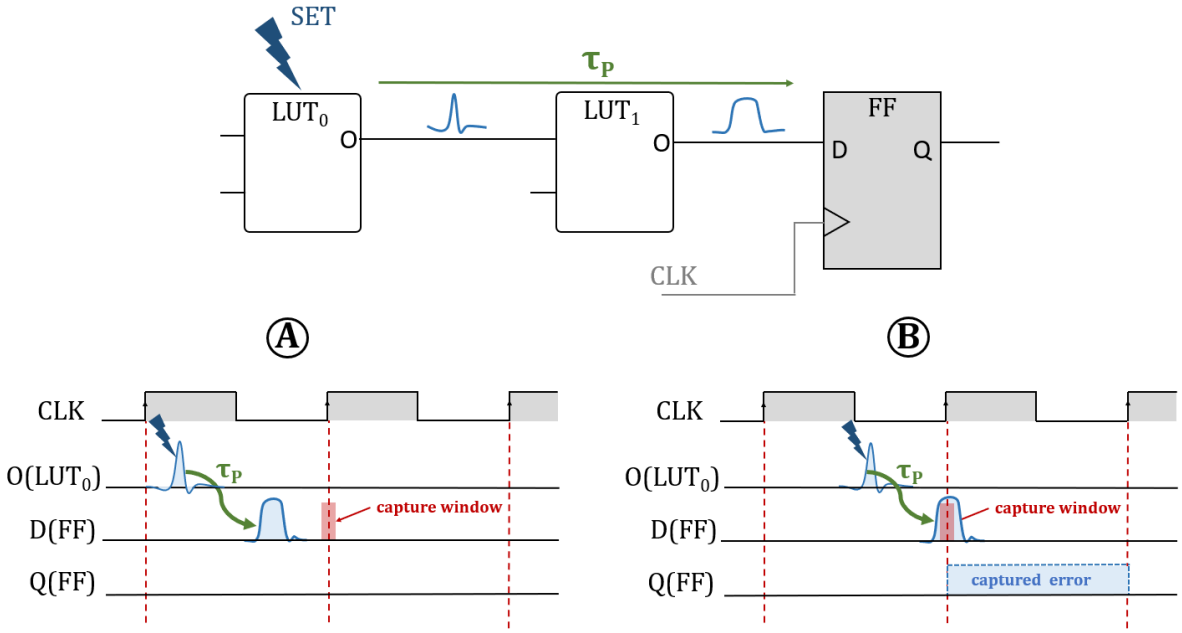


Figure 45: Propagation and capture of SETs generated in a LUT. The SET must reach the input of the flip-flop with sufficient amplitude during the capture window to be captured (B).

Once captured, flip-flops SEUs and SETs generate a synchronous error that can no longer be temporally masked, only logical masking can prevent their propagation.

Regarding Single Event Functional Interrupt (SEFI) for FPGAs, there is a wide diversity of symptoms generally associated with the configuration functionalities of the component. These symptoms depend on configuration logic architecture of the FPGA considered, but SEFI are commonly induced by SEU or SET affecting the configuration logic or global signals. For instance, in [59], a list of potential SEFI symptoms for *Xilinx* FPGAs are described. These SEFIs can affect either the capacity to read or write the configuration memory through one of the configuration interfaces or the reset of all internal storage cells. These types of events generally require a complete reconfiguration or power-cycle of the device to return normal operation.

In the other hand, Single Event Latchup (SEL) are due to a particle triggering the parasitic thyristor formed by the arrangement of an NMOS and PMOS transistor (see section 1.2.3.3). SEL behaves in a similar way on FPGAs as on other digital devices: a sudden increase in the supply current is observed, potentially accompanied by loss of functionality, which must be corrected by power-cycling the device to avoid the risk of damaging the component permanently. Given the severity of such events, the SEL sensitivity of an FPGA may be a prohibitive feature for its use in radiative environments. SEL tolerance is generally characterized by the LET threshold at which SELs are observed.

2.3.4. Summary of radiation effect on FPGAs

Given the complexity of FPGA, an exhaustive list of all radiation-induced failure mechanisms cannot be established. The predominant effects can nevertheless be retained. Regarding TID, the main concern are a progressive degradation of the propagation delay and the power consumption. Flash based FPGA are also prone to the configuration memory corruption and a loss of the

configuration capacity. As for SEE, the main effects are SEUs affecting the user memory blocks and flip-flops, and SET generated and propagated in the combinatorial logic. SRAM-based FPGA are also subject to configuration memory (CRAM) upset that may modify the design topology until reconfiguration. As explained in section 1.2.3.4, the SEE sensitivity of a device is usually characterized using the cross section (σ) concept, corresponding to the probability that an interaction occurs between a particle and the investigated device.

To illustrate the sensitivity of FPGAs to radiation, TABLE II, provide some examples of COTS and radiation hardened FPGAs from different vendors with qualitative radiation test results for TID, SEL or SEU.

TABLE II:
RADIATION TEST RESULTS SUMMARY OF COTS AND RADIATION HARDENED FPGAS FROM DIFFERENT VENDORS*

Xilinx (AMD)

FPGA	Tech.	Configuration memory cross section (cm^2/bit)		TID krad(Si)	SEL LET_{TH} Mev.cm^2/mg	Refs.
Virtex4 (SRAM)	COTS 90nm	Neutrons (atm) Heavy Ions (LET=20MeV.cm^2/mg)	1.5e-14 6.5e-09			[60], [61]
Virtex-4QV (SRAM)	Rad-hard 90nm	Heavy Ions (LET=100MeV.cm^2/mg) Heavy Ions (LET=20MeV.cm^2/mg) Protons (Energy=100MeV) Protons (Energy=20MeV)	5.0e-08 1.0e-08 3.0e-14 1.5e-14	>300	100	[62]–[64]
Virtex5 (SRAM)	COTS 65nm	Neutrons (atm) Heavy Ions (LET=60MeV.cm^2/mg) Heavy Ions (LET=20MeV.cm^2/mg) Protons (Energy=200MeV) Protons (Energy=65MeV)	6.7e-14 9.0e-08 4.0e-08 8.6e-14 6.4e-14			[61], [65]
Virtex-5QV (SRAM)	Rad-hard 65nm	Heavy Ions (@LET=60MeV.cm^2/mg) Heavy Ions (@LET=20MeV.cm^2/mg)	5.5e-10 2.5e-12	>1000	100	[62], [66]
Virtex6 (SRAM)	COTS 40nm	Neutrons (atm) Protons (Energy=120MeV)	1.3e-14 9.7e-15			[61], [67]
Artix7 Spartan7 (SRAM)	COTS 28nm	Neutrons (atm) Protons (Energy=200MeV)	7.0e-15 9.4e-15	>356		[61], [68], Chap. 4
Kintex7 Virtex7 (SRAM)	COTS 28nm	Neutrons (atm) Heavy Ions (LET=20MeV.cm^2/mg) Heavy Ions (LET=100MeV.cm^2/mg) Protons (Energy=180Mev)	5.7e-15 2.8e-09 8.0e-09 4.1e-15		<15 (micro SEL)	[61], [69]–[71] Chap.4
Virtex Kintex Ultrascale (SRAM)	COTS 20nm	Neutrons (atm) Heavy Ions (LET=80MeV.cm^2/mg) Heavy Ions (LET=20MeV.cm^2/mg) Protons (Energy=20mEV) Protons (Energy=100MeV)	2.5e-15 3.0e-09 1.0e-09 5.0e-16	>620	80	[39], [61], [72]
RT Kintex Ultrascale (SRAM)	Rad-Tol 20nm	Heavy Ion (saturation, LET_{TH}=0.5)	8.0e-10	>100	>80	[73]
Virtex Kintex Ultrascale+ (SRAM)	COTS 16nm FinFET	Neutrons (atm) Protons (Energy=64Mev) Heavy Ions (LET=20MeV.cm^2/mg)	2.7e-16 2.8e-16 3.0e-10	>340	Sensitive -neutrons -proton -HI (LET_{TH}= 5.7)	[61], [74]–[76]

Versal (SRAM)	COTS 7nm FinFET	Neutrons (atm) Protons (Energy=64MeV)	2.6e-17 2.8e-17		Insensitive to -neutrons -64MeV protons	[61], [77]

Intel / Altera

FPGA	Tech.	CRAM cross section (cm²/bit)		TID krad(Si)	SEL LET_{TH} Mev.cm²/mg	Refs.
Cyclone (SRAM)	COTS 130nm	Protons (Energy=60MeV)	1.6e-12	>1000	<35	[78], [79]
Stratix (SRAM)	COTS 130nm				2.8	[80]
Stratix II (SRAM)	COTS 90nm				0.87	[81]
Aria GX (SRAM)	COTS 90nm	Protons (Energy=20MeV)	1.6e-16			[82]
Stratix IV (SRAM)	COTS 40nm				112	[83]
Stratix V (SRAM)	COTS 28nm	Neutron (atm)	4.8e-15			[71]

Microsemi (Microchip) / Actel

FPGA	Tech.	Flip-flop SEU cross section (cm²/bit)		TID krad(Si)	SEL LET_{TH} Mev.cm²/mg	Refs.
ProASIC3 (Flash)	COTS 130nm	Heavy Ions (LET=80MeV.cm²/mg) Heavy Ions (LET=20MeV.cm²/mg)	3.0e-07 6.0e-08	59	55	[84]
RT-ProASIC3 (Flash)	RadTol 130nm	Heavy Ions (LET_{Th}=6MeV.cm²/mg) σ_{SAT} Protons (Energy=63.5MeV)	 2.0e-07 5.0e-14	>25	>70	[85]
SmartFusion2 (Flash)	COTS 65nm	Protons (Energy=200MeV)	1.1e-14	40		[68]
RTG4 (Flash)	RadTol 65nm	Protons (Energy=200MeV)	2.0e-15	>160	>103	[86], [87]
RT PolarFire (Flash)	RadTol 28nm	Heavy Ions (LET=30MeV.cm²/mg)	8.0e-08	>300	>63	[88]

NanoXplore

FPGA	Tech.	CRAM cross section (cm²/bit)		TID krad(Si)	SEL LET_{TH} Mev.cm²/mg	Refs.
NG-Medium (SRAM)	Rad-hard 65nm	Heavy Ions (LET=62MeV.cm²/mg) Heavy Ions (LET=20MeV.cm²/mg) Protons (Energy=230MeV) Protons (Energy=30MeV)	5.2e-09 5.9e-10 5.0e-16 4.2e-17	100	60	[89]

** To be noted that the test results provided above are quantitative and affected by test condition. For more accurate data and for further comparison, please consult the provided reference.*

2.4. RADIATION HARDENING

To mitigate the radiation effects, there are different radiation hardening techniques that can be applied at different levels. During the design and manufacturing of the FPGA, techniques can be applied at the process and layout level to decrease the cross section and/or the sensitivity to dose effects of the different FPGA elements. On the user side, hardening techniques can be applied to the implemented design to prevent the manifestations of errors and reduce the system failure probability.

2.4.1. PROCESS BASED HARDENING

Many process modifications have been explored in the last decade to reduce the radiation sensitivity of electronic components. They usually relate to the application of different materials, variation of doping profiles and substrate technology. For instance, boron purification process can be introduced into the manufacturing process to reduce the abundance of ^{10}B and thus reduce the susceptibility to thermal neutrons. Another example is the use of the Silicon-On-Insulator (SOI) technology (not limited to rad hard components). This technology uses an additional buried layer of insulator such as sapphire or SiO_2 as shown in Figure 46.

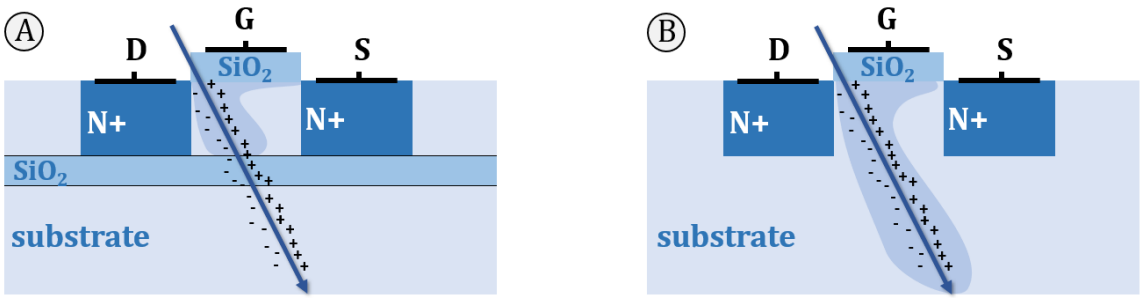


Figure 46: Energy deposition of a charged particle in SOI (A) and bulk (B) technologies.

Besides the reduction of parasitic capacitance allowing performance improvements, the use of SOI technology tends to increase the radiation robustness versus bulk counterparts as the charge collection volume is limited by the buried silicon oxide. In addition, this layer prevents the charge sharing effect between adjacent nodes by cancelling the charge diffusion process. The parasitic thyristor present between bulk CMOS transistors is also eliminated by the buried oxide layer thus immunizing the component from SELs [90].

Due to the complexity and cost of additional manufacturing steps required to achieve radiation hardening and due to the usually low-volume production, radiation hardened technologies usually experience a technological time-lag with regard to the state-of-the-art transistor scaling trend while providing lower performance than COTS counterparts.

2.4.2. LAYOUT BASED HARDENING

Part of layout-based techniques rely on geometrical modifications of transistors and gates layout. For instance, to minimize the extent of radiation-induced leakage currents (RILC), a well-established technique is to use Enclosed Layout Transistors (ELT) as shown in Figure 47. As described in section 1.2.2.1, holes trapped in STI oxides due to TID can create a parasitic conduction path on the transistor edges resulting in an increased source-drain current in the transistor off state. The use of ELT removes the connection between the transistor junction and the trench oxides thus reducing the RILC.

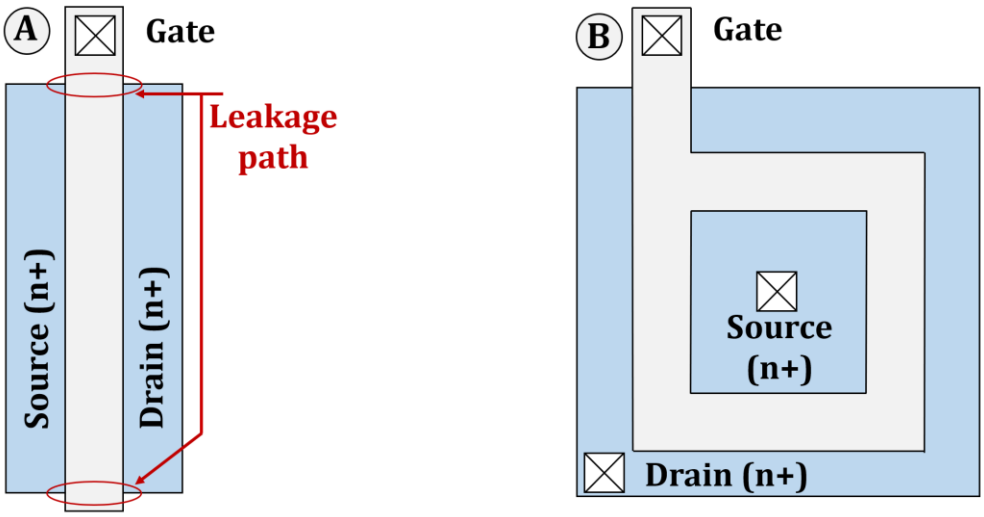


Figure 47: Comparison between classical finger layout (A) and ELT (B), from [91].

On another note, the TID induced propagation delay degradation of programmable interconnexions point held by Flash cells can be overcome by using a conventional pass-transistor held by a pair of Flash transistors in push-pull configuration. This technique has been applied in *Microsemi*'s RTG4 radiation-tolerant FPGA [92] extending its TID tolerance above 100 krad (SiO_2) [93].

As for SEE, considering the example of SRAM cells, a first mitigation approach is to increase the LET threshold by increasing the width of the transistors to increase their capacity and conductance. Another widely used technique is to insert resistors between the cross-coupled inverters of the SRAM cell [94] as shown in Figure 48.

The inserted resistors increase the propagation delay between the two inverters allowing enough time to recover from the collapsed node voltage. Similarly, radiation tolerant flip-flops can be built by adding redundant transistors to the cell. A well-established example used in NanoXplore FPGAs is the Dual Inter-locked CEll (DICE) [95] shown in Figure 49. The DICE cell uses four CMOS inverters (N1-P1, N2-P2, N3-P3, N4-P4) where each inverter has its N-transistor and P-transistor controlled separately by two adjacent nodes storing the same state (A and C or B and D). The four nodes thus store the data as two pairs of complementary values. As each node is driven by separate transistors, two nodes must be simultaneously upset to change the stored value thus greatly improving the overall SEU tolerance.

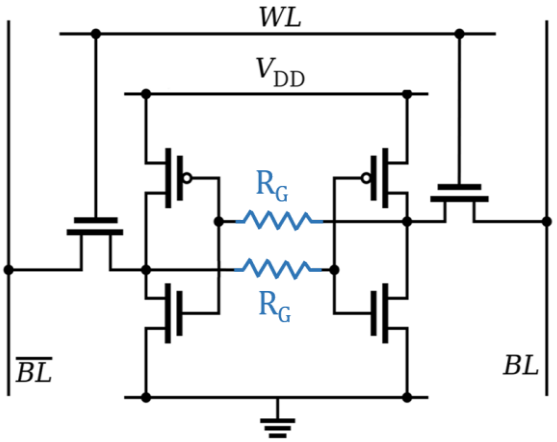


Figure 48: SRAM cell radiation hardened by inserting resistors between the cross-coupled inverters.

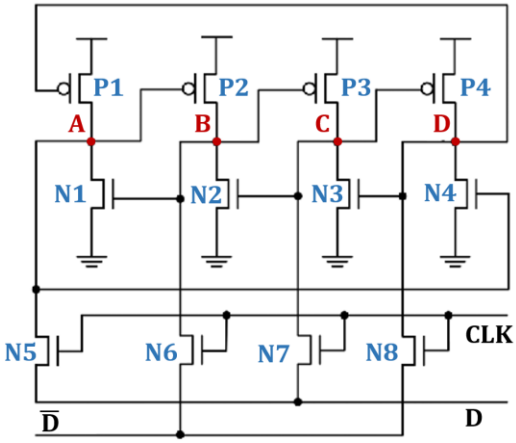


Figure 49: DICE cell formed by four cross-coupled inverters where each pair of transistors is driven by separated nodes.

These layout mitigation techniques inevitably translate into an increase of area and power consumption as well as a decrease in timing performances thus preventing their integration in COTS components. Designers who, for financial or performance constraints, are considering COTS FPGAs must therefore rely on mitigation techniques based on design modifications.

2.4.3. Circuit based hardening

Mitigation techniques based on circuit modification are intended to prevent errors from manifesting and cause system failures. They are generally based on spatial or data redundancy. The most straightforward and widely used mitigation technique is the Triple Module Redundancy (TMR). This technique may be found in different flavors, but the principle remains the same: instantiation of three identical copies of a circuit and linking of their outputs to a majority voter. During normal operation, the three circuits output the same value but when one of the three replicas is affected by a SEU, its output value differs from the other two. The majority voter then logically masks the erroneous value by transmitting only the majority value. To propagate an error across the majority voter, two circuit instances must be erroneous at the same time. This technique thus leads to a strong decrease in overall system susceptibility. The TMR scheme can be applied at different levels of granularity as shown in Figure 50.

The simplest and most economical implementation of the TMR scheme is the Local TMR (LTMR), which consists in triplicating only the flip-flops of the design. Combinatorial Logic and voters are not protected by LTMR; SETs generated in the combinatorial logic can propagate to the different flip-flop replicas and generate soft errors. As SEUs in the configuration memory can alter the behavior of combinatorial paths, the implementation of this scheme in SRAM FPGAs is ineffective. Distributed TMR (DTMR) offers a stronger protection by triplicating all combinatorial paths, flip-flops and voters. However, global signals that are shared between the three replicated data paths (reset and clock signals) remain single points of failure. This issued can be tackled by using Global TMR (GTMR) where global signals are also triplicated. This requires the use of three independent and synchronized clock signals.

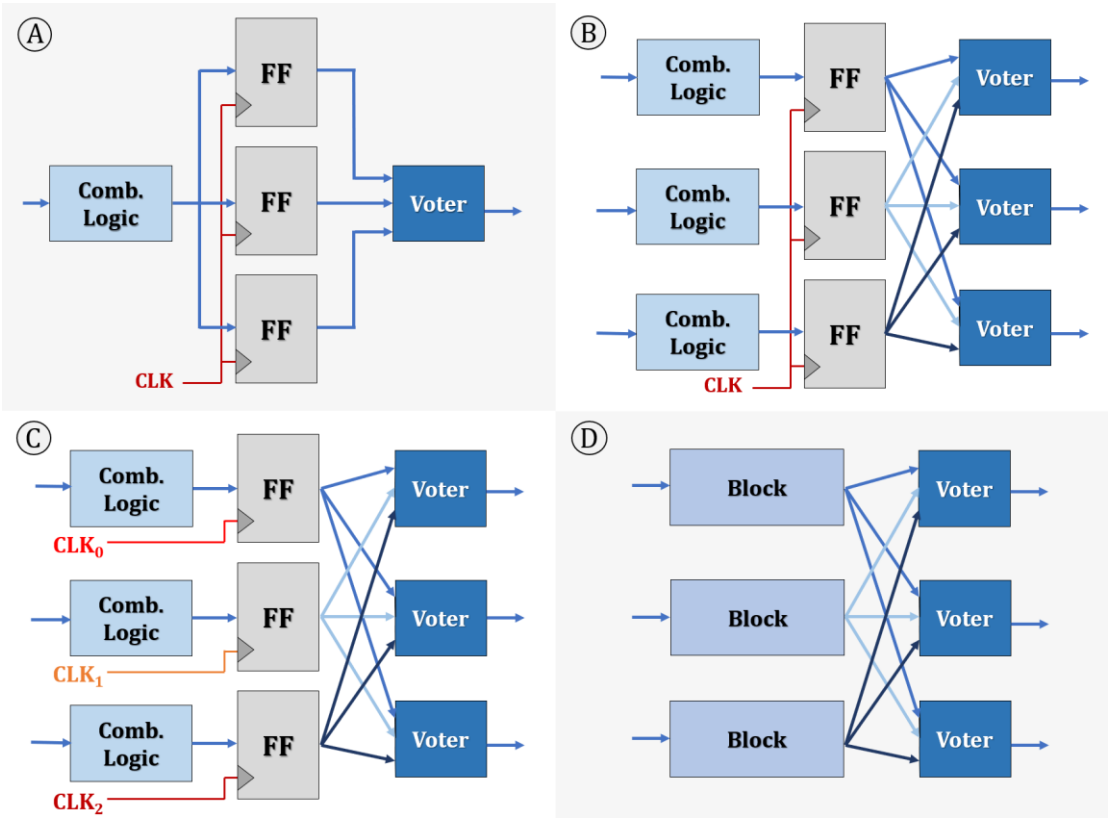


Figure 50: TMR scheme applied at different level. Local TMR (A) only triplicated flip-flops; Distributed TMR (B) triplicated combinatorial logic, voters and flip-flops; Global TMR (C)add triplicated global signals; Block TMR (D) triplicates entire blocks, from [96].

To be noted that the area overhead induced by DTMR and GTMR schemes can easily exceed 200% due to the insertion of majority voters after each flip-flop. In addition, timing issues arise as the results of the redundant subsystems need to arrive at the same time at the voters, leading to a great reduction of the maximum speed the design may reach. A great effort has been made in the last decades to find efficient mitigation solutions with reduced area overhead. In [97], a methodology to reduce the number of voters called partial TMR is proposed. The main concept is to identify the most sensitive nodes of the design using fault injections tools (described in section 4.6) and to insert voters in specific locations (such as feedback loops) to prevent errors to persist after the configuration memory upset is repaired. Selective TMR (STMR) [98] is another approach that propose to identify sensitive subcircuits based on their input signal probabilities. TMR is then selectively inserted in the most sensitive parts of the design to reduce the area overhead with a small loss of SEU immunity. Similarly, Approximate TMR (ATMR) [99] use approximate versions of the circuit to be triplicated as redundant copies. These approximate copies are functions that match the original function only to a fraction of its input space. Two different approximate versions are used as redundant replicas so that at least two out of the three replicas respect the intended functionality for all input possibilities. By efficiently exploiting the FPGA architecture, the use of ATMR have shown to be effective in reducing the area overhead with respect to DTMR while being able to provide a similar or even greater SEU immunity in some cases.

As these fine-grain implementations of the TMR scheme can be challenging to implement manually, development tools have been developed to automatically generate TMR scheme from RTL design. For example, XTMR, an automated tool by *Xilinx* provide features to insert TMR in

Virtex-5QV and Virtex-4QV based designs. Synopsys and Mentor also provide TMR insertion capabilities in their FPGA synthesizer (Synopsys Synplify [100] and Mentor Graphics Precision Hi-Rel [101]). HDL templates can also be used to facilitate the manual instantiation of triplicated structures directly in the RTL description [102].

On another note, Block TMR applies the TMR scheme at a higher level of granularity, by triplicating entire blocks and using a majority voter only at the output of these blocks. This approach benefits from a reduction of instantiated voters and can be applied with encrypted IP cores.

Some system can handle errors as long as they are detected and reported. In that case, there are more area-efficient methods to simply detect errors such as Duplication With Comparison (DWC). Two replicas of the same module are instantiated and compared. The output error detector can be used to rise error flags that must be handled by the system to ensure proper decision making and recovery procedures. The DWC can be supplemented with error recovery structures such as adding AND/OR voters to mask part of the errors with reduced overhead as proposed in [103] or by combining DWC with Current Error Detection (CED) as proposed in [104]. In the latter case, each module replica is supplemented with an encoding/decoding structure as a self-checking feature based on time redundancy. When a mismatch is detected between the two redundant modules, the CED block detects which of the two modules is faulty, the module outputs can then be multiplexed accordingly to mask the faulty circuit.

2.4.4. MEMORY HARDENING

To prevent the manifestation of error generated in user memory blocks, methods based on Error Detection and Correction Codes (EDAC), can be implemented. These methods, extensively used in telecommunications protocols, rely on data redundancy brought by Error Correction Codes (ECC). The idea is to encode the data to be stored with redundant information. This redundancy is used to detect and correct a limited number of errors that may have affected the data. For example, *Hamming(7,4)* code adds three parity bits to four bits of data as shown in Figure 51.

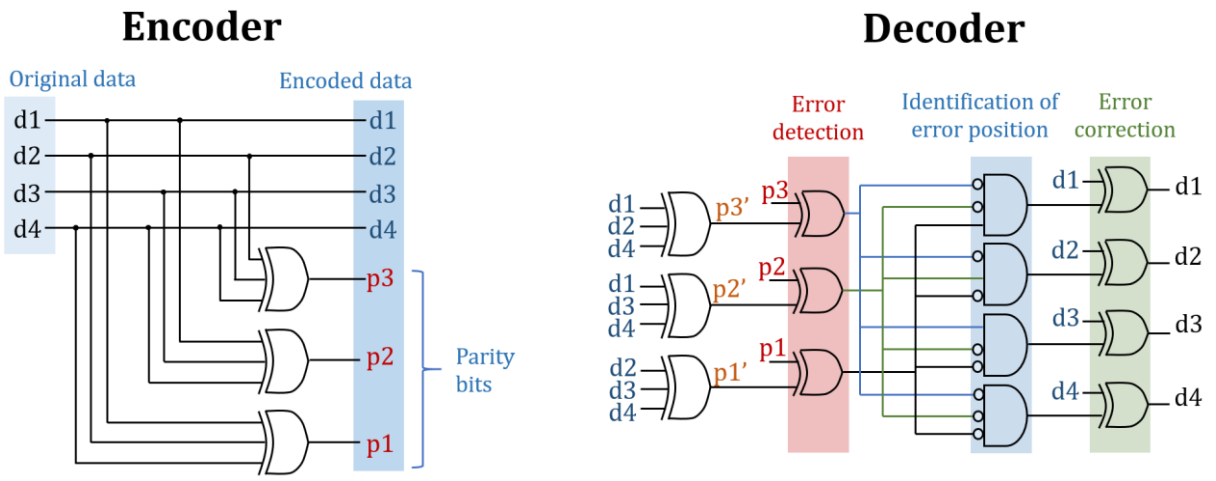


Figure 51:Hamming encoder and decoder.

To encode the data, each redundant bit computes the parity of three out of the four data bits. On the decoding side, the parity bits are recomputed and compared with the redundant bits. When a mismatch is detected, the error position is identified and the corresponding data bit is flipped. However, if two bits have been corrupted between encoding and decoding, the decoder is no longer able to restore the original data. There is a wide diversity of ECCs with different error correction capabilities. These ECCs can be used to protect the memory block content by encoding the data at writing and decoding at reading. To be noted that some recent FPGAs include hard-wired encoder/decoder in the BRAM interfaces. Interleaving schemes can also be applied to these memory blocks to reduce the probability that an MBU affects several bits on the same data word. The principle of EDAC can also be applied to the Finite State Machine (FSM) to increase their reliability. Fault tolerant FSMs can be built by encoding the state vector with ECC, invalid state transitions can then be detected and recovered.

Identifying the effectiveness of the different mitigation methods presented is not a simple task. Indeed, the methods based on hardware redundancy can be cancelled by the presence of single points of failure. Especially for SRAM-based FPGAs, some memory points of the configuration RAM control the behavior of several resources and thus potentially affect two replicas of a triplicated system from a single bit upset. The presence of MBUs can also compromise the efficiency of TMR by simultaneously affecting multiple replicas of the TMR scheme. For example, it has been shown that the efficiency of TMR on SRAM technologies such as *Xilinx*, can be greatly reduced by these single points of failure [105], [106]. One method to overcome this issue is to impose a physical separation between the different replicated blocks to avoid sharing resources and logical blocks, thus decreasing the number of single points of failure and the probability of MBUs affecting several blocks simultaneously.

On the other hand, the effectiveness of the TMR scheme in SRAM FPGAs relies on the fact that errors do not accumulate in the configuration memory. The probability of defeating the protection provided by TMR increases greatly with the number of bitflips accumulated in the configuration memory. To avoid this accumulation, a common practice is to use memory scrubbing [107]. This scrubbing technique can be applied by using an external controller that will periodically readback the content of the configuration memory and compare it to the original bitstream. When a mismatch is detected, the bitstream is reloaded on the FPGA to correct the upsets. Comparing the two bitstreams and writing the whole bitstream again introduces a significant delay. To reduce the duration of the SEU induced errors, Cyclic Redundancy Check (CRC) can be used to localize the errors and reload only the affected frames through partial reconfiguration. To reduce the complexity of the off-chip scrubbing system, the bitstream can be reloaded periodically without readback check (blind scrubbing). The scrubbing period for both blind and readback scrubbing must be adapted to the estimated error rate to reduce the probability of several SEUs occurring between two scrubbing period. To overcome the need for an external scrubber, FPGA manufacturers have developed internal memory scrubbers: *Xilinx* (Soft Error Mitigation Controller -SEM IP) [108]), NanoXplore (Configuration Memory Integrity Check -CMIC [109]) and *Intel* [110]. These internal scrubbers use the internal configuration access port to continuously check for error using CRC and localize and correct the potential bitflips using ECC. The effectiveness of such solutions will be evaluated and discussed in chapter 4.

2.5. CONCLUSION

In this chapter, the operating principle and architecture of FPGAs have been presented along with a description of the main radiation effects and the associated failure mechanisms. As described, the radiation behavior of FPGAs is complex and difficult to characterize. Depending on the technology they are based on, different types of radiation effects must be considered.

On the one hand, TID induce a degradation of the propagation delay in the combinatorial logic and an increase of the power consumption. Moreover, they represent a major risk of premature end-of-life for flash-based FPGAs due to the high sensitivity of their configuration cells and associated circuitry.

On the other hand, a wide range of Single Event Effect can disrupt the functioning of the system. SEL, the most critical, can cause a complete shutdown of the component, requiring a power cycle to avoid permanent damages. SET and SEU in the user flip-flops can cause system failures by momentarily altering the state of the logic signals. The effects of SEU are even more important for SRAM components for which the configuration memory can be corrupted, potentially leading to a modification of the implemented circuit topology.

Several techniques have been developed to reduce the sensitivity of these components, notably by adjusting the manufacturing process parameters or by using specific logic cell geometries. These hardening techniques have contributed to the emergence of a small range of radiation-hardened FPGAs. However, in a context of decreasing costs of spacecraft payload and the demand of increasing performances, the technological time-lag imposed by the application of these hardening techniques and the high cost of these rad-hard components pushes the space industry among others to consider the use of COTS FPGAs.

Due to their flexible nature and the particularity of their behavior under radiation, FPGAs require unique methodologies to test them, to estimate the reliability and protect the implemented systems. Two key points in the development of radiation testing methodologies for FPGAs is to improve the tests standardization for comparison purposes and to improve the genericity of the test results so that they can be reuse to estimate the reliability of any implemented design. The next chapters will describe the proposed test methodology for both TID (chapter 3) and single event effects (chapter 4 and 5).

3. FPGA TESTING METHODOLOGIES FOR TID EFFECTS ASSESSMENT

To ensure the reliability of FPGA-based systems in high radiation environments, a first crucial step is to evaluate the tolerance of the component to dose effects. From the mission profile and the radiative environments to which the system is exposed, the dose level and dose rate received by the system over its entire mission can be calculated. The next step is to select FPGA components that can withstand the required ionizing dose level and to evaluate the parametric degradation suffered as a function of the absorbed dose. These steps rely on radiation testing for TID effects assessment: the components under test are exposed to high levels of artificially generated ionizing radiation like X-ray or γ-ray. The objective is not only about measuring the total dose they can absorb before loss of functionality, but also to evaluate the degradation of the component's performance versus the absorbed TID, to ensure that the component will meet its specifications and respect the system constraints during the entire mission. The observed performance degradation, namely, a degradation of the propagation delay of logic gates and an increase in power consumption, can lead to reconsider the choice of components or to implement mitigation strategies. The heterogeneity and reprogrammability of FPGAs makes them a particularly complex component to test since the performance degradation is strongly dependent on the resources mobilized and the architecture used by the implemented circuit. FPGAs thus requires dedicated testing methodologies and dedicated benchmarking structures to precisely evaluate timing and power consumption degradation.

This chapter focuses on the evaluation of parametric degradations: **propagation delay drift and power consumption increase**. The state of the art of test methodologies for the evaluation of TID effects will be firstly described, then a new test methodology will be proposed, based on the development of dedicated benchmarking structures allowing the extension of the timing degradation evaluation to all the logic and routing resources of the device. A new cost-effective technique of in-situ delay measurement is also proposed, based on the reprogrammable feature of embedded PLLs. The effectiveness and benefits of this methodology are demonstrated through X-ray radiation tests. Test results on three FPGA families, *Xilinx* Spartan7, Artix7 and *Intel* Cyclone10LP are presented, compared and discussed.

3.1. FROM TESTING METHODOLOGIES TO BENCHMARKING

3.1.1. STATE OF THE ART OF TESTING METHODOLOGIES

Several methodologies have been developed in the past to address the challenge of TID testing of FPGAs. Some of those, particularly adapted to the most sensitive FPGAs, focus on the identification of the failure mechanisms and on the measurement of the maximum dose before functional failure [111]. Focused synchrotron X-ray irradiation can be used to identify the most vulnerable inner circuit in the component by locally irradiating different sub circuits with a fine beam resolution. Other studies focus on the evaluation of parametric degradation. To measure the propagation delay drift, several approaches are proposed in the literature. In [112]–[114], methods based on external instrumentation are described. An external signal generator and/or a signal analyzer are used to measure the time taken for the signal to pass through a certain number of logic elements of the FPGA as shown in Figure 52.However, this approach can be challenging to

deploy, especially when using commercial development boards where signal integrity issues arise and PCB tracks may exhibit frequency limitations.

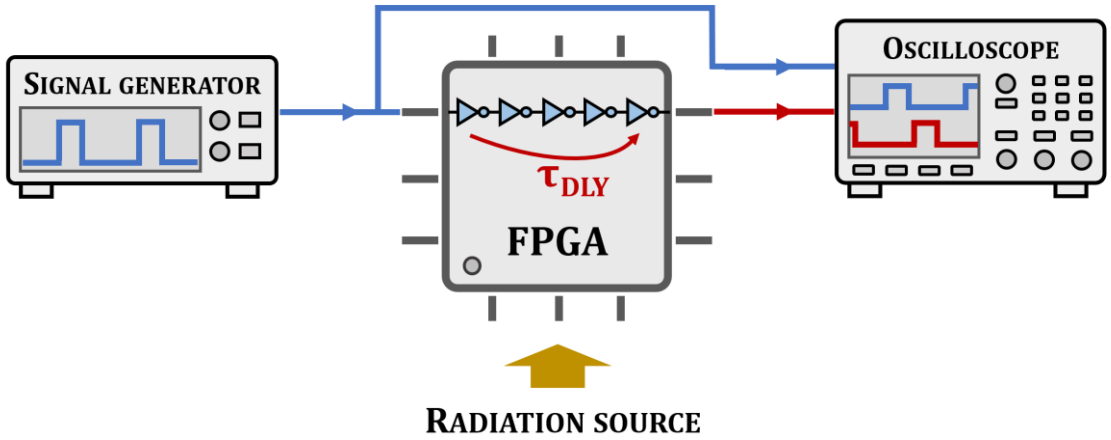


Figure 52: Propagation delay measure with external signal generator and oscilloscope.

In [115]–[118], ring oscillator structures are used to internally measure the propagation delay drift of Look Up Tables (LUT) with a good time resolution. Ring oscillators are formed by looping an odd number of LUTs configured as inverters as shown in Figure 53. The resonance frequency of the structure is thus inversely proportional to the signal propagation delay along the structure. However, these test structures are limited to the LUT's propagation delay evaluation while the propagation delay evolution of the other resources is not examined.

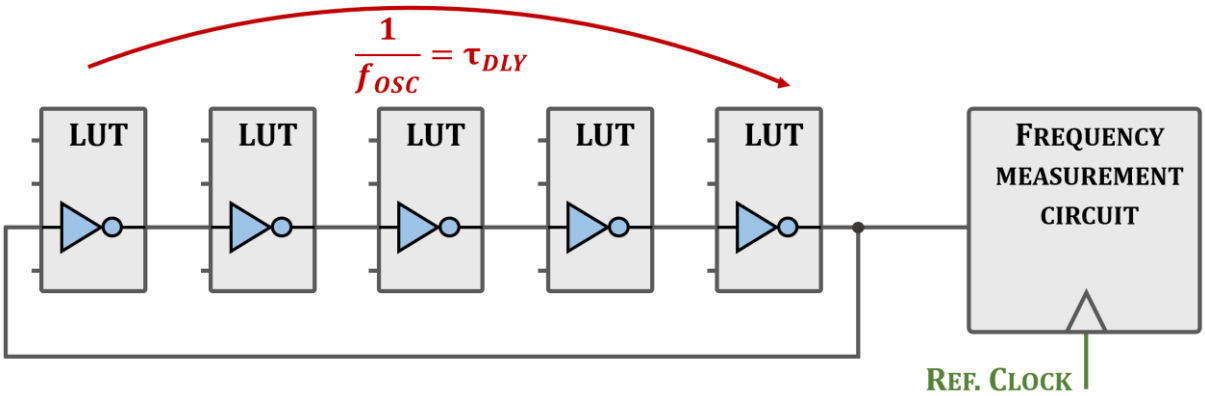


Figure 53: Ring oscillator structure use to measure the LUT propagation delay by monitoring its oscillating frequency.

3.1.2. EXTENSION OF DEGRADATION EVALUATION TO ALL RESOURCES

The main contribution of the developed methodology proposed in this work is to extend the propagation delay degradation evaluation to all the logical and routing resources inside the FPGA. Indeed, most state-of-the-art methodologies focus only on propagation delay degradation of LUTs while the degradation of the other resources might be different. In a real design, logic paths are made up of different types of resources: LUT, carry logic, DSP, routing segments, programmable interconnection points (PIP) etc. The assumption that the timing degradation observed on the LUTs is the same for all the FPGA resources can lead to a bad estimation of the timing degradation and thus to a misguided application of the timing constraints.

3.1.3. BENCHMARKING STRUCTURES

In this study, the focus is on the most abundant resources that can possibly suffer TID-induced degradation, namely: LUT, DSP, Carry propagation circuits (CARRY) and programmable interconnexion points (PIP). For each type of resource, dedicated benchmarking structures similar to the ones propose in [119] have been developed. The term "benchmarking" implies that these structures can be implemented on any FPGA family in order to compare their performance degradation on a common basis. These monomorphic structures are built using long chains of cascaded elements as shown in Figure 54.

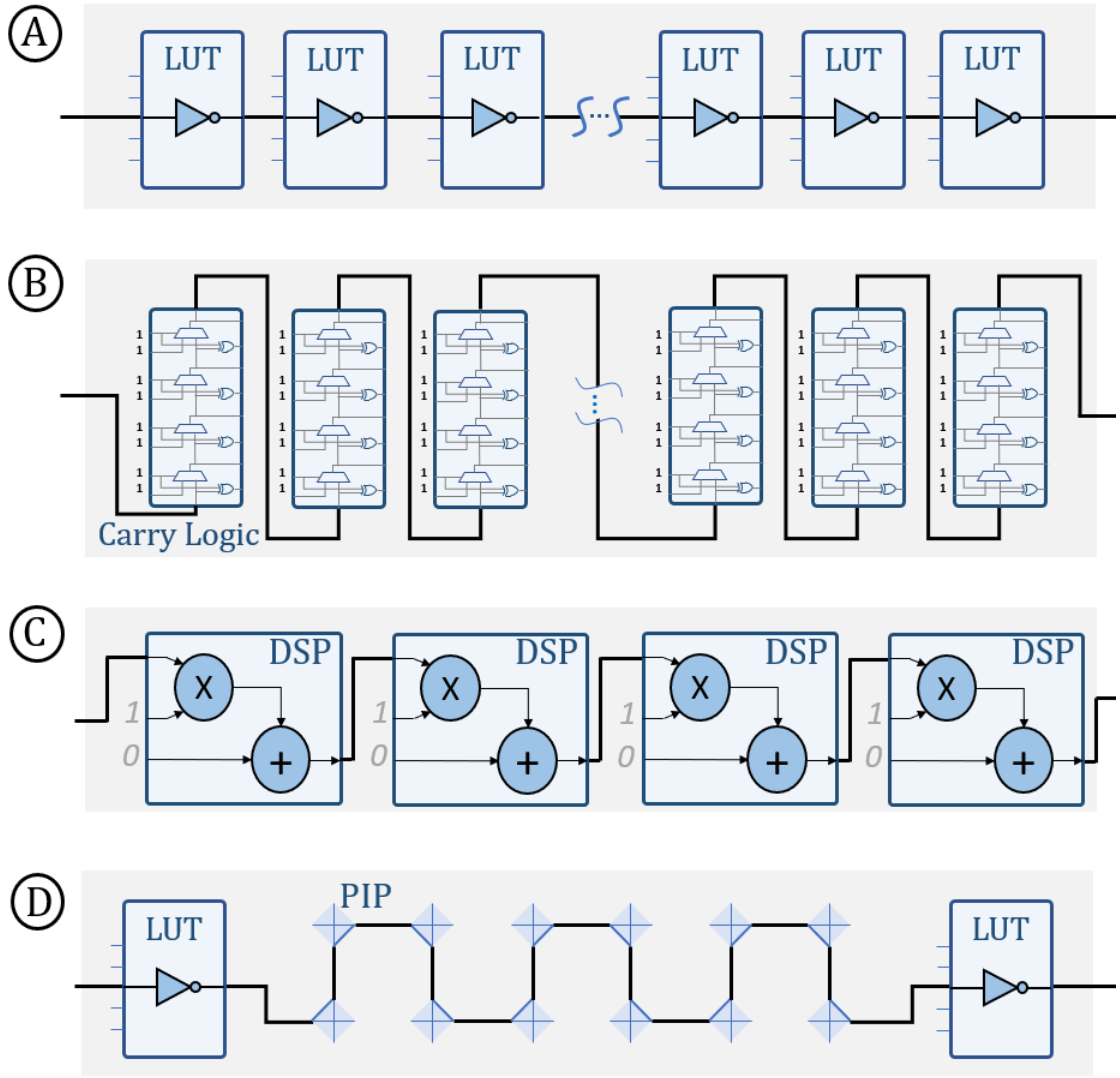


Figure 54: (A) LUT are configured as inverter; (B) Carry chain uses an adder like structure; (C) DSP, configured as multiply-adder, are cascaded by connecting the DSP's output to one of the two input multiplicands; (D) an increased number of PIPs are inserted between each LUT.

The propagation delay of these long chains of cascaded elements is the sum of the individual contribution of each element. The propagation delay measure of such structures thus provides the average propagation delay over all the instantiated elements. By adjusting the length of the chains and the number of chain replicas, the statistics can be improved and the timing dispersion between the elements can be smoothed. For each structure and for each FPGA component, a manual placement of the primitives has been performed as shown in Figure 55.

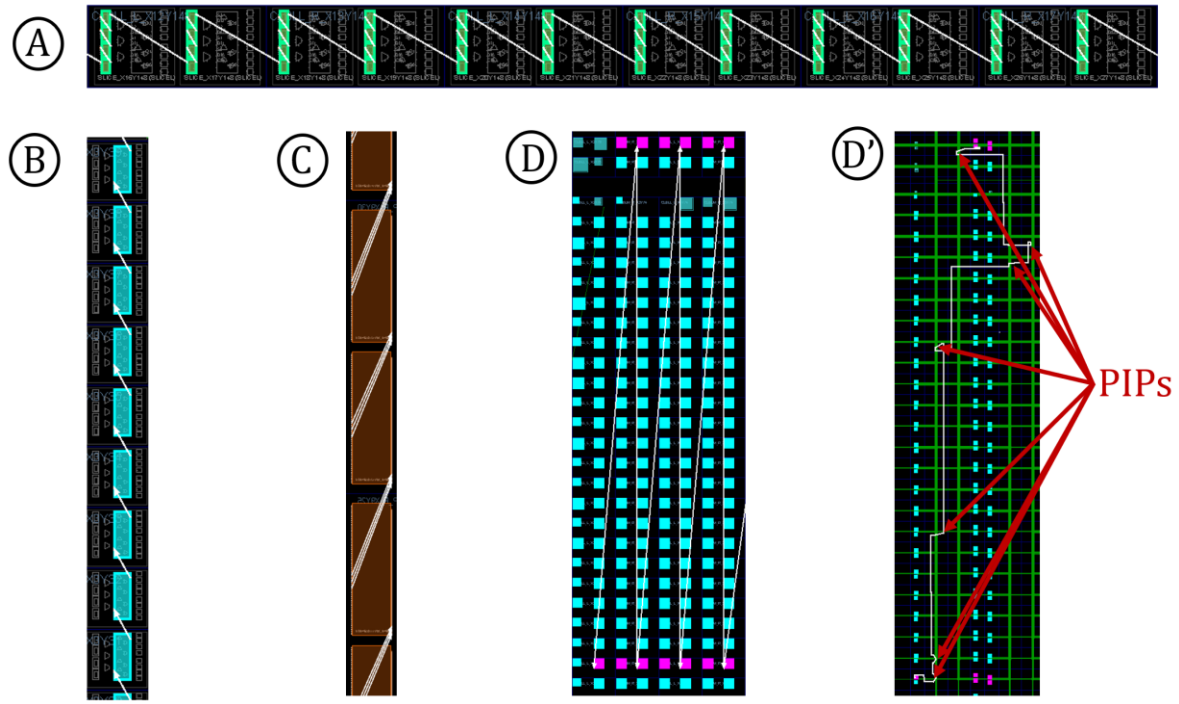


Figure 55: Device view of the manually placed benchmarking structures implemented on *Xilinx* Artix7. In (A), LUTs are placed with the minimum physical separation to reduce the number of PIP between each gate. In (B) and (C) Carry logic and DSP respectively are placed in adjacent logic blocks to avoid any PIP usage. In (D), LUT are placed with a large physical separation to instantiate a large number of PIPs between each gate.

For the LUT chain (A), the cells were placed with the minimum physical separation between each consecutive LUT to reduce the number of instantiated PIP. On the contrary, the PIP chain (D) was built using LUTs manually placed with a large physical separation to increase the number of PIPs instantiated by the routing tools. Hereafter, the benchmarks A and D will be referred to as "LUT close" and "LUT spread" respectively (referring to their placement). CARRY and DSP chains were also carefully placed contiguously to force the use of dedicated hard-wired paths thus preventing any PIP instantiation in these chains. In addition, this manual placement provides a perfect symmetry between the different chain replicas.

3.2. Test setup

3.2.1. Propagation delay measurement

As previously mentioned, ring oscillators provide an efficient way to measure the propagation delay of inverting logic gates. However, they are not suited for logic gates that have a non-inversing behavior such as multiplexer and arithmetic operators. Moreover, their loop shape necessarily involves the instantiation of a large number of PIP to link the last and first primitives of the chain.

The propagation delay measurement technique proposed in this work can be used internally without any constraints on the structure shape. It is based on the evaluation of the maximum clock frequency that can be used by the chain before violating the setup time of the end point flip-flop. In practice, a square signal that toggles at each rising edge of the clock signal is injected at the input of the chains presented in Figure 54. A flip-flop captures the propagated signal on the output

of the chain. The captured signal is compared to the input signal delayed by one clock cycle as shown in Figure 56.

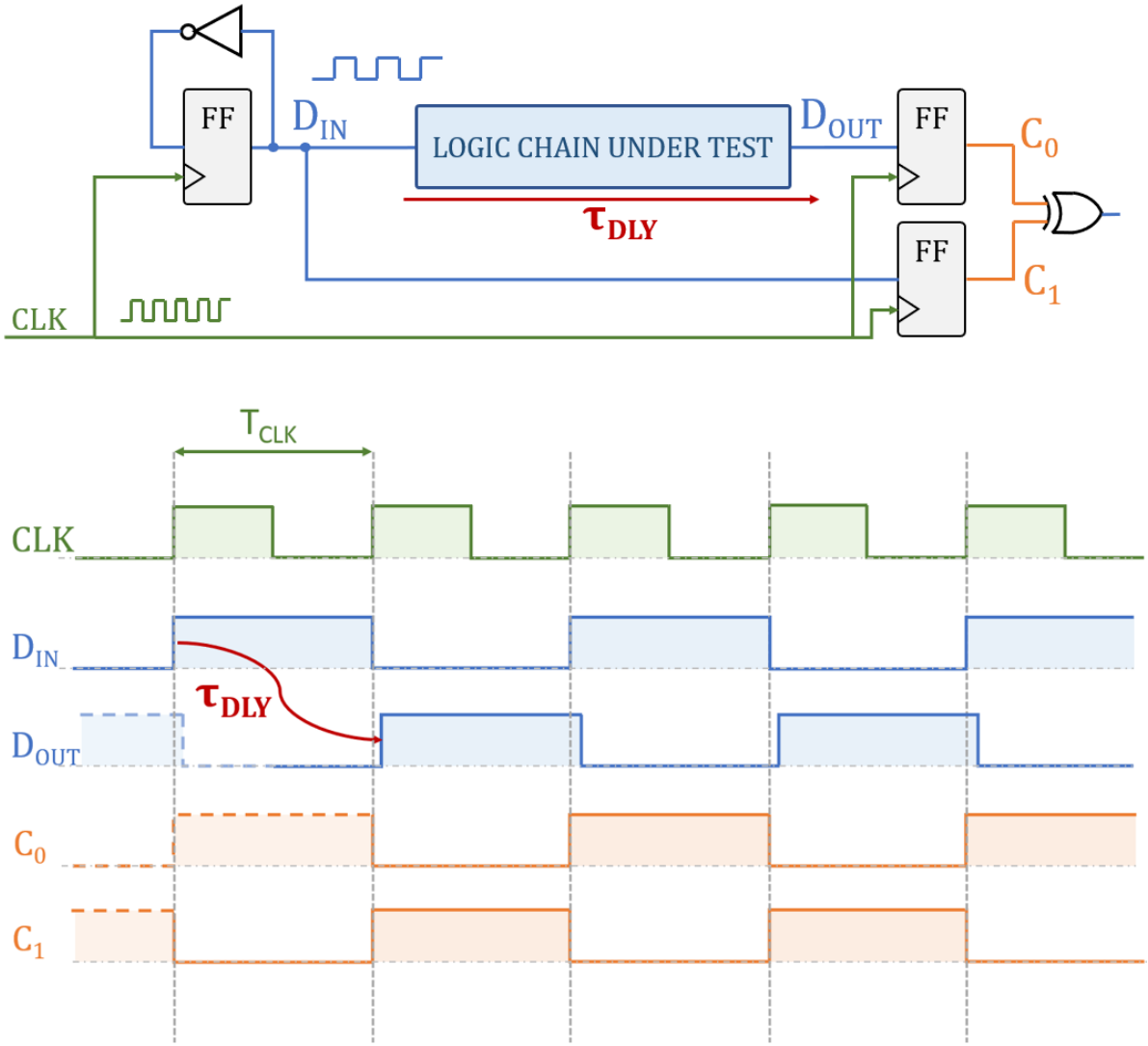


Figure 56: Propagation delay measure principle: a varying frequency square signal (D_{IN}) propagates through the chain under test and the output signal (D_{OUT}) is registered (C_0) and compared to the input signal delayed by one clock cycle (C_1). The highest clock period (T_{CLK}) for which a mismatch is detected corresponds to the propagation delay of the chain (τ_{DLY}).

Whenever the clock period (T_{CLK}) gets lower than the propagation delay of the chain (τ_{DLY}), the output flip-flop captures the previous signal value or enters a metastable state. The lowest frequency for which a mismatch is detected on the output comparator has, therefore, a period equal to the propagation delay of the chain (neglecting the delays on the clock path). The measurement is carried out by performing an increasing frequency sweep of the clock signal until a mismatch is detected at the comparator. The comparator output is accumulated, formatted and transmitted to a host computer to report setup time violations. To achieve the frequency sweep with reduced instrumentation, the reprogrammable PLL feature is used. Most modern FPGAs now integrate PLLs whose parameters of the synthesized clock signal (clock multiplier and divider, phase delay, etc.) are handled by reprogrammable registers. These registers can be dynamically reprogrammed through user-accessible configuration interfaces. For each FPGA type (*Xilinx*

Spartan7, Artix7 and *Intel* Cyclone10LP), a reconfiguration system linking the PLLs configuration port to a serial interface has been developed. A python script is used to periodically perform a frequency sweep of the clock signal from a remote computer while continuously checking for errors in the output comparator. When an error is detected, the affected chain ID and the corresponding frequency are recorded.

As the measurement circuit is integrated in the DUTs, it can suffer from the same TID induced degradation during the experiment. A particular attention must be paid to ensure that the degradation induced to the measurement system does not impact the propagation delay measurements. Large margins on the setup and hold time must be been taken for the path used to delay the input signal in order to avoid any false mismatch detection on the output comparator. On the other hand, the propagation delay of the clock signal between the input flip-flop and the output flip-flops (clock skew) directly impacts the measurement as this clock delay is directly subtracted from the one of the chains under test. The composition of the clock path must be carefully considered. Here, the section connecting the input and output registers is composed by only two programmable interconnexion points. The delay drift of these two PIPs is thus directly subtracted to the one of the chains under test. As experimental results will demonstrate, the drift of two PIPs can be fairly neglected with respect to the drift suffered by the chains under test.

The timing measurement precision is limited by the frequency resolution of the reprogrammable PLLs. For *Xilinx* FPGAs, the Mixed Mode Clock Manager (MMCM) have been used that allow to synthesize clock signals with a frequency resolution of 125kHz. The content of the configuration registers corresponding to each frequency in the swept frequency range where extracted and stored in advance. The configuration bytes sent by the host PC are received in the FPGA part by a softcore processor that manages the MMCM reconfiguration through an AXI bus. The softcore also allows to record the FPGA junction temperature and the fabric supply voltage through the internal ADC. For the *Intel* FPGAs, the frequency resolution of PLL being 500kHz, a second PLL with a division factor of 4 was cascaded with a first one to reach the same frequency resolution as *Xilinx* MMCMs. The softcore microprocessor was replaced by a dedicated hardware design to receive the configuration bytes from the serial link and reconfigure the PLLs.

To ensure that the delay measurement is not affected by possible deviation of the synthesized clock signal frequency, the clock signal has been continuously monitored by an external oscilloscope. During the whole experiment, no deviation has been observed neither on the clock frequency nor its duty cycle.

3.2.2. DEVICE SELECTION

As TID experiments are destructive tests, one of the objectives of the experiment was to show that the proposed methodology can be applied with low cost development boards without specific connectors for high frequency analog signals. For this experiment, four types of SRAM FPGAs have been initially selected as shown in TABLE III and Figure 57.

During the first tests, the *TE0722* board (Zynq7) showed premature failures around 50krad. This system failure was caused by the presence of a programmable oscillator on the backside of the PCB, at the FPGA chip level. This programmable oscillator seems to have a high sensitivity to dose effects, preventing its use for this experiment. This board was therefore removed from the set of components under test.

TABLE III
LIST OF SELECTED COMPONENTS WITH TECHNOLOGICAL INFORMATION

Manufacturer	Family	Node size	Process	Foundry	Board
Xilinx	Spartan7 (XC7S25)	28nm	high-κ metal gate, HPL	TSMC	CMOD A7 [120]
Xilinx	Artix7 (A7-35T)	28nm	high-κ metal gate, HPL	TSMC	CMOD S7 [121]
Xilinx	Zynq7 (Z-7010)	28nm	high-κ metal gate, HPL	TSMC	TE0722 [122]
Intel	Cyclone10LP (10CL025)	60nm	low-κ dielectrics	TSMC	CYC1000 [123]

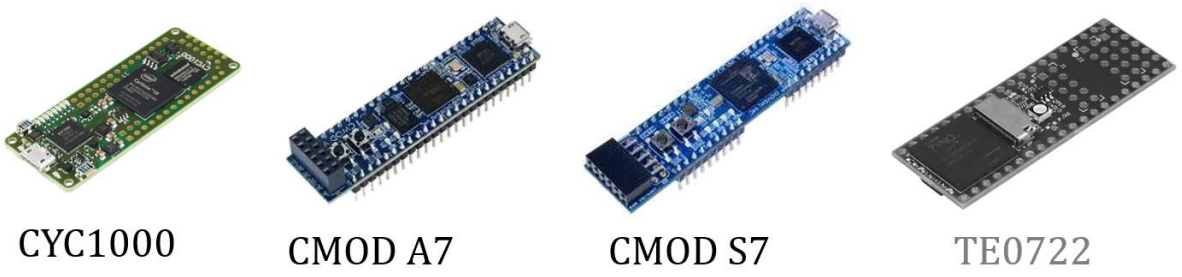


Figure 57: Selected boards for X-ray irradiation

The four test structures presented in Figure 54 were divided in two components (two benchmarks on each component) and implemented on the three FPGA families for a total of 6 tested components. For each FPGA type, the length of the chains has been adapted either to obtain logical chains that cross the entire chip or to fit into frequency ranges where the PLL can provide the best frequency resolution. Each chain was replicated 16 times on the chip to maximize the statistics. The chain length for each DUT is summarized in TABLE IV.

TABLE IV
BENCHMARK PARAMETERS: CHAIN LENGTH AND COMPOSITION

	LUT chain (A)	PIP chain (D)	Carry chain (B)	DSP chain (C)
Artix7	160 LUTs 180 PIPs	80 LUTs 640 PIPs	100 Carry4	10 DSP
Spartan7	160 LUTs 180 PIPs	80 LUTs 640 PIPs	100 Carry4	10 DSP
Cyclone10LP	100 LUTs	25 LUTs	500 Carry1	8 Multipliers

To be noted that the Cyclone10LP has a different logic block and routing architecture compared to *Xilinx* FPGAs, the carry propagation circuit are 1-bit wide instead of 4-bits wide and the DSP blocks are replaced by simple multipliers. The benchmarking structures have been adapted

accordingly as described in TABLE IV. In addition, the number of programmable interconnexion points used by a specific net cannot be clearly identified through the *Intel* FPGA development tool (Quartus), and as a result, their propagation delay cannot be properly identified.

3.2.3. X-RAY GENERATOR AND PARAMETERS

The six DUTs were irradiated with the X-ray generator from PRESERVE Platform [17] at the Montpellier University at a dose rate of 11.63 rad(air)/s. A correction factor of 2.5 can be applied to obtain an equivalent dose in krad(SiO_2). The photons beam was collimated to the chip dimension and the rest of the onboard components were shielded with lead layers to reduce the exposure of components other than FPGAs. Power consumption at the board level was also recorded during the entire experiment.

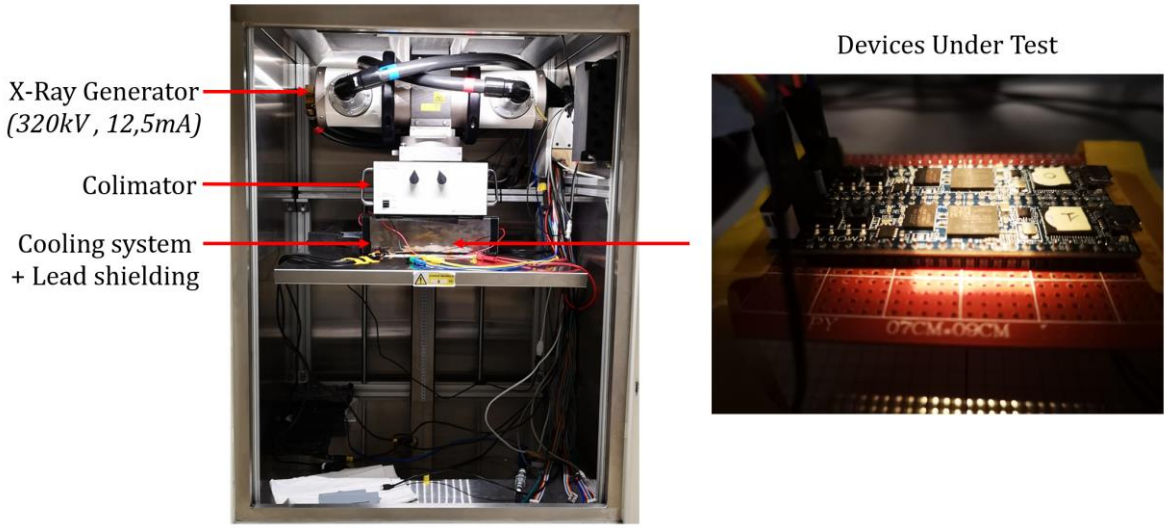


Figure 58: X-ray generator from PRESERVE Platform at the Montpellier University.

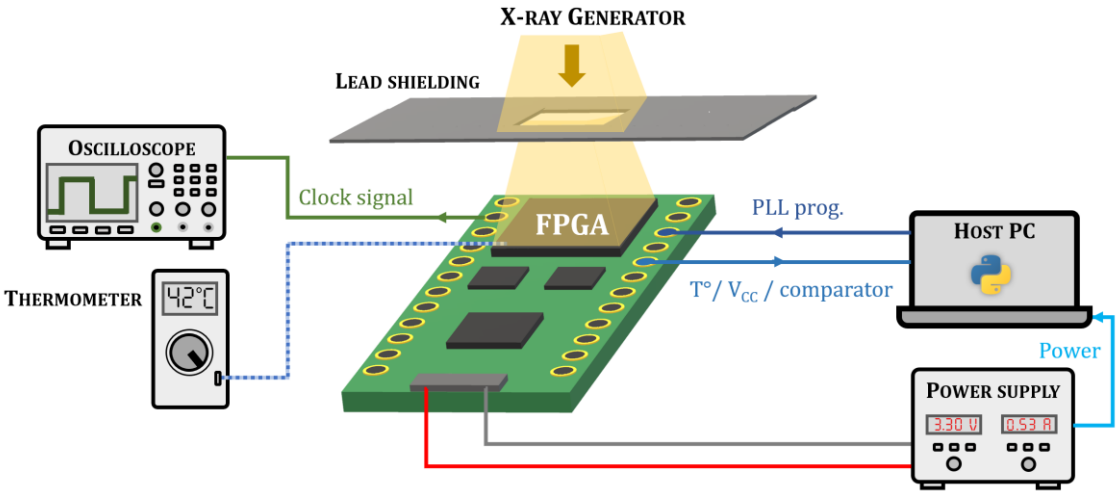


Figure 59: Experimental setup description. An external oscilloscope is used to monitor the clock signal. An external thermometer is used to ensure that the junction temperature recorded by the internal A DC is not affected by TID. A host PC with a python script running is used to send the command to the internal softcore processor (clock frequency selection) and record all the data sent by the FPGA (junction temperature, fabric supply voltage and output comparator results). The board supply voltage and current measured by the power supply are also sent to the host PC to record the power consumption.

A cooling system, made with a lead tunnel (drilled on the top to let the beam pass) with two fans on both ends, has been added inside the X-Ray radiation chamber to reduce the impact of temperature variation. Indeed, the increased power consumption caused by TID induced degradation, leads to an increased chip temperature. This temperature increase generates parametric deviation competing with the TID effect itself. By increasing the heat dissipation power with a cooling system, the impact of this side effect is reduced. The entire experiment was conducted with a junction temperature between 40°C and 55°C. As the transistor voltage also has a major influence on the propagation delay, the fabric voltage was monitored through the experiments. No voltage variation was observed. Figure 58 and Figure 59 show the experimental setup.

3.3. RADIATION TEST RESULTS

All components have been irradiated during 8 hours and 30 minutes at 11.63 rad(air)/s for a total dose of 356krad(air) without any functional failure. After 48 hours of annealing at around 40°C (close to the irradiation temperature), the components were tested again to analyze the annealing and time dependent effects.

3.3.1. PROPAGATION DELAY DEGRADATION RESULTS

Propagation delay degradation evolution of the three FPGA types is shown in Figure 60, Figure 61 and Figure 62 for Artix7, Spartan7 and Cyclone10LP respectively.

The first result that can be observed is a large difference in the evolution of the propagation delay between the different tested resources. For *Xilinx* FPGAs, the LUT chain with close placement (A) is the most affected, followed by the chain of DSPs (C), the LUT chain with Spread Placement (D) and the Carry Logic chain (B). On the contrary, for the Cyclone10LP, the DSPs are the ones with the highest deviation.

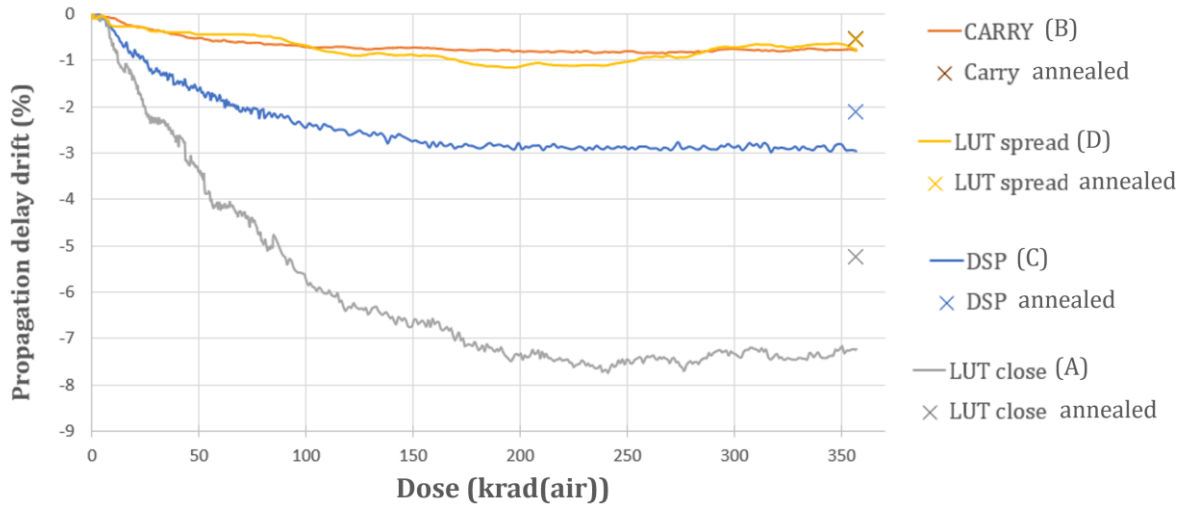


Figure 60: TID induced relative propagation delay drift on **Xilinx Artix7.**

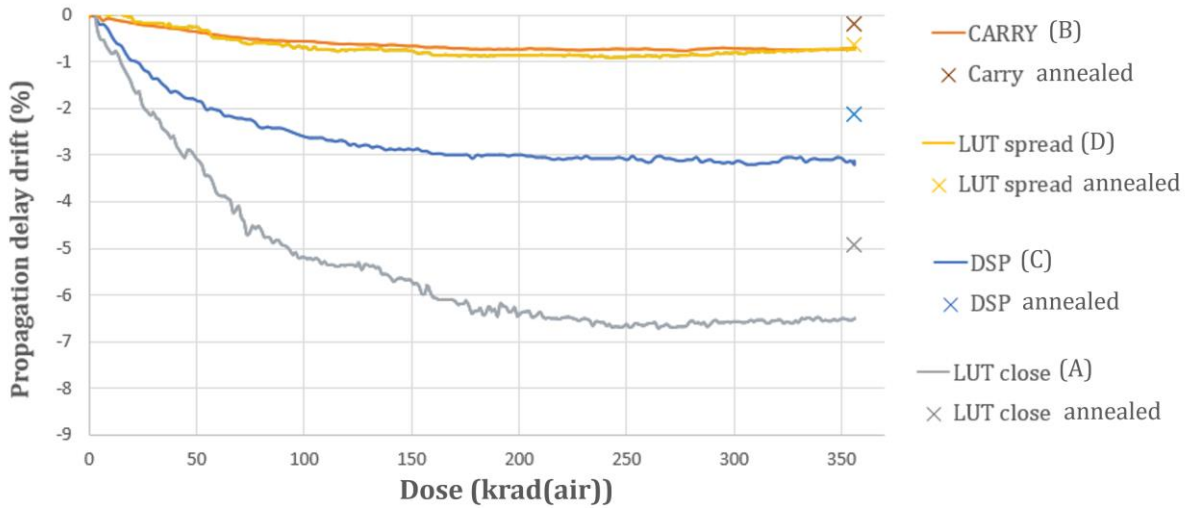


Figure 61: TID induced relative propagation delay drift on **Xilinx Spartan7.**

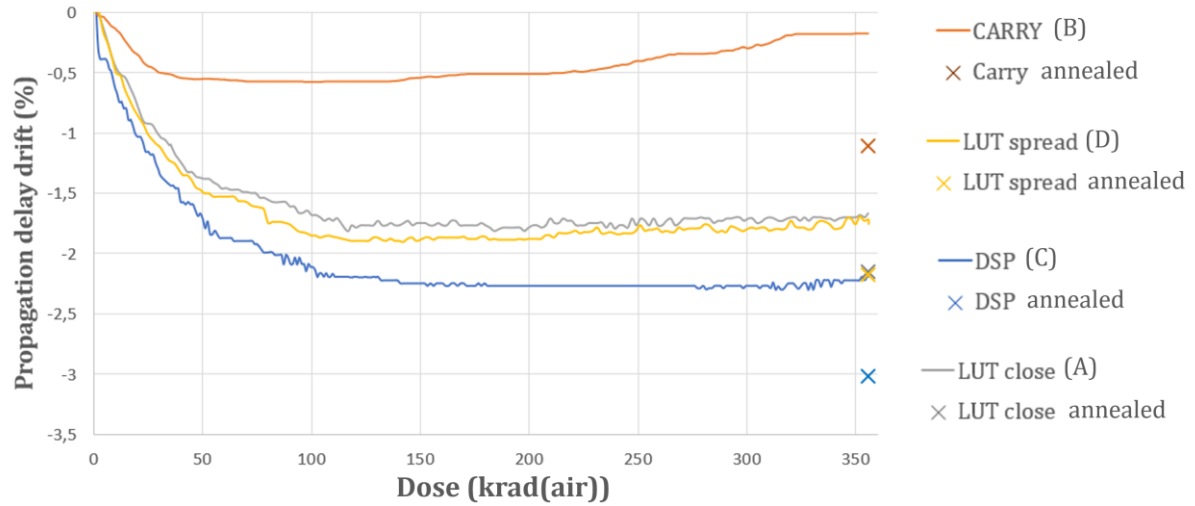


Figure 62: TID induced relative propagation delay drift on **Intel Cyclone10LP.**

The second noticeable result for the three FPGAs is a decrease of the propagation delay with the absorbed TID instead of increasing as it is mainly observed on most FPGAs [39], [78], [112], [113], [117], [124].

The lack of information on the parametric degradation induced by dose effects at the transistor level does not allow to explicitly justify this behavior. This reduction in propagation delay can nevertheless be explained by different phenomena. Indeed, when the threshold voltage decreases due to charge trapping in the oxides, the drain-source resistance of the NMOS transistors in the saturated state is reduced but the drain-source resistance in the blocked state also. The phenomenon is the opposite for the PMOS transistors since the threshold voltage shift is always negative. Decreasing the drain-source resistance of the transistor in the saturated state tends to reduce the switching time while decreasing the drain-source resistance of the transistor in the blocked state tends to increase it. The switching time of a CMOS gate can therefore be affected positively or negatively depending on the parametric drift that dominates. The initial trend can then be reversed when the threshold voltage exceeds a certain point. As for the reduction of the

transconductance induced by interface states, it always induces an increase of the switching time of the CMOS cell, which comes in addition to the effects of the threshold voltage shift. The relative predominance of these different phenomena (shown in Figure 63) could explain the trends observed on the deviation of the propagation time.

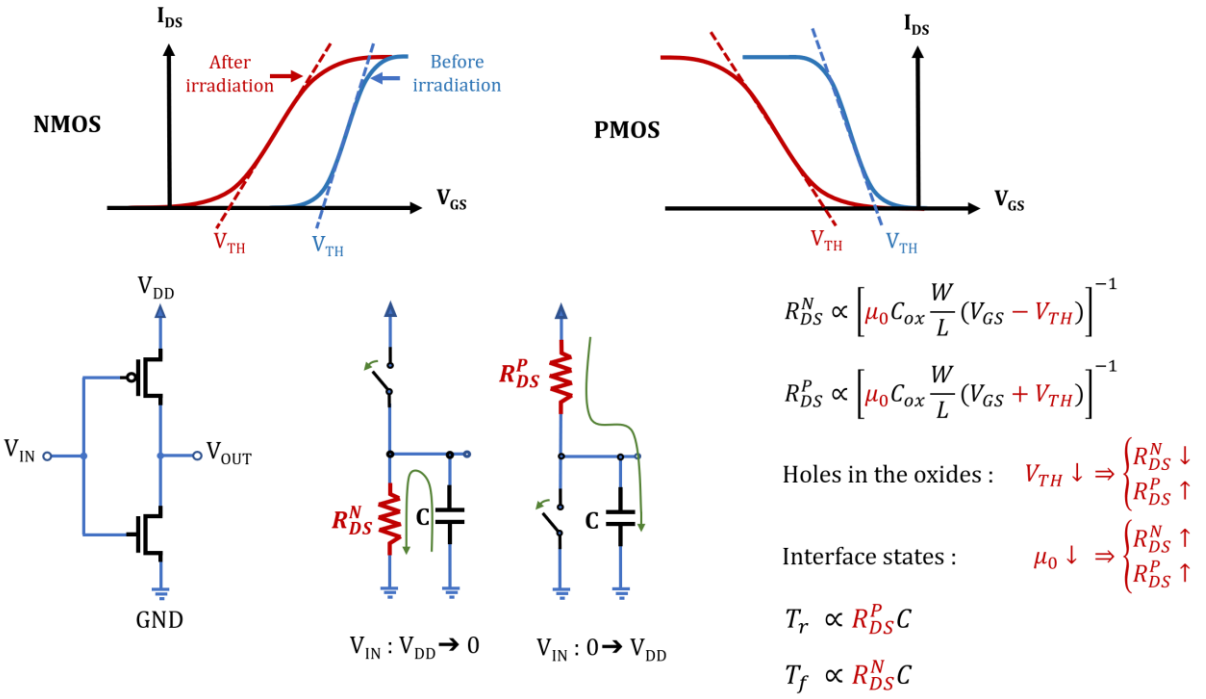


Figure 63: Simplified TID effects on rising and falling time of a CMOS inverter. The drain source resistance on the ON state for the NMOS (R_{DS}^{N}) and PMOS (R_{DS}^{P}) are directly affected by the threshold voltage (V_{TH}) shift and the carrier mobility (μ_0) degradation. The rising and falling time are conditioned by the drain source current of the activated transistor and are thus affected positively or negatively depending on the predominant influence between V_{TH} shift and carrier mobility degradation. To be noted that this model does not consider the RILC that may influence the drain source impedance in the OFF state.

This type of behavior had already been observed on FPGAs from the *Xilinx 7 Series* family in [115], [117], [118]. In [115], LUT-based ring oscillators implemented on a Zynq7 were irradiated with a $^{90}Sr/^{90}Y$ electron source and a decrease of 1.7%of the propagation delay was observed after 350 krad(SiO_2). In [117], LUT-based ring oscillators also implemented on a Zynq7 were irradiated with the same X-ray generator than this study. A decrease of 2% of the propagation delay was observed after 350 krad(air). In the current study, after 350 krad(air), a decrease of 7,2% is obtained on the LUT chains (close placement) for the Artix7 (5.2% after annealing) and 6.2% (4.9% after annealing) for the Spartan7. In [118], 4 LUTs-long ring oscillators densely implemented on Zynq7 exhibit an average decrease of propagation delay around 5.3% after 40krad(SiO_2) while a decrease of around 1.2% on both Artix7 and Spartan7 for the same absorbed TID (16krad(air)) has been observed in this experiment. The differences in results with [115], [117], [118] can be explained by different experimental conditions (power supply, voltage, biasing conditions, temperature, annealing time), lot to lot variations and by different LUT chain composition. Indeed, the number of PIP instantiated between each LUT can greatly vary depending on the placement constraints imposed at implementation stages. In this study, the side by side manually placed LUTs, allowed to greatly reduce the number of inserted PIPs with respect to a chain implemented

with automatic placement. This hypothesis is confirmed by our results on the two LUT chains. Indeed, for both *Xilinx* FPGAs, the LUT chain with close placement (A) uses twice as many LUTs as the chain with spread placement (D), yet the observed drift of the propagation delay of chain **A** is more than six times higher than for chain **D**, which suggests that the number of instantiated PIPs clearly influences the observed drift.

From these experimental results, the individual deviations of each resource type can be extracted. For Carry Logic and DSP benchmarks, as the chains are monomorphic, the individual deviations are obtained by dividing the total deviation observed on the chain by the number of elements it is composed of. Results for CARRY and DSP are shown in Figure 64 and Figure 65 respectively.

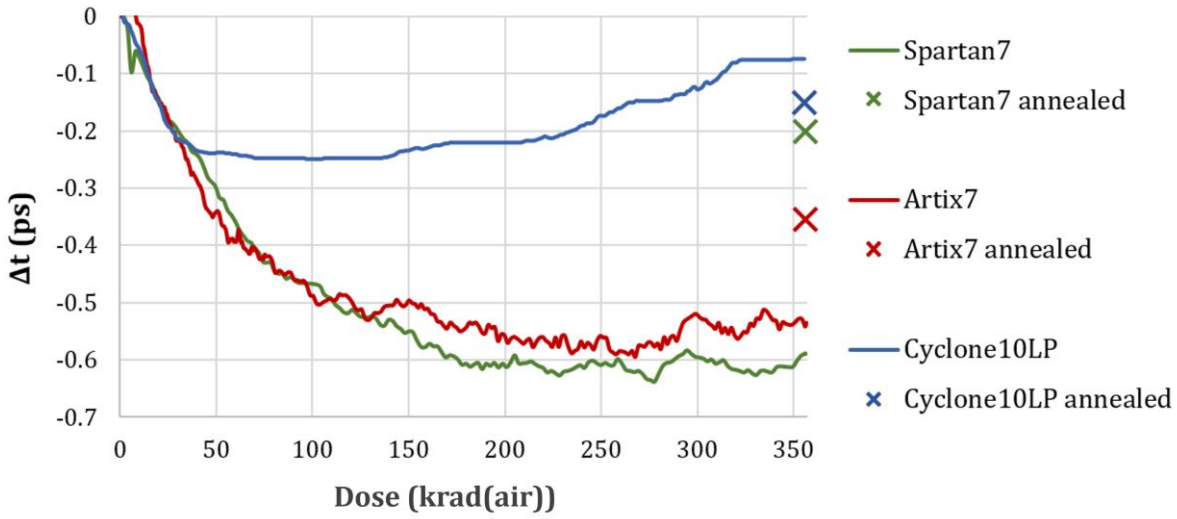


Figure 64: **4bits CARRY** propagation delay degradation for the three tested FPGAs. These values are obtained by dividing the total degradation of the chain by the number of 4bits-CARRY logic resources.

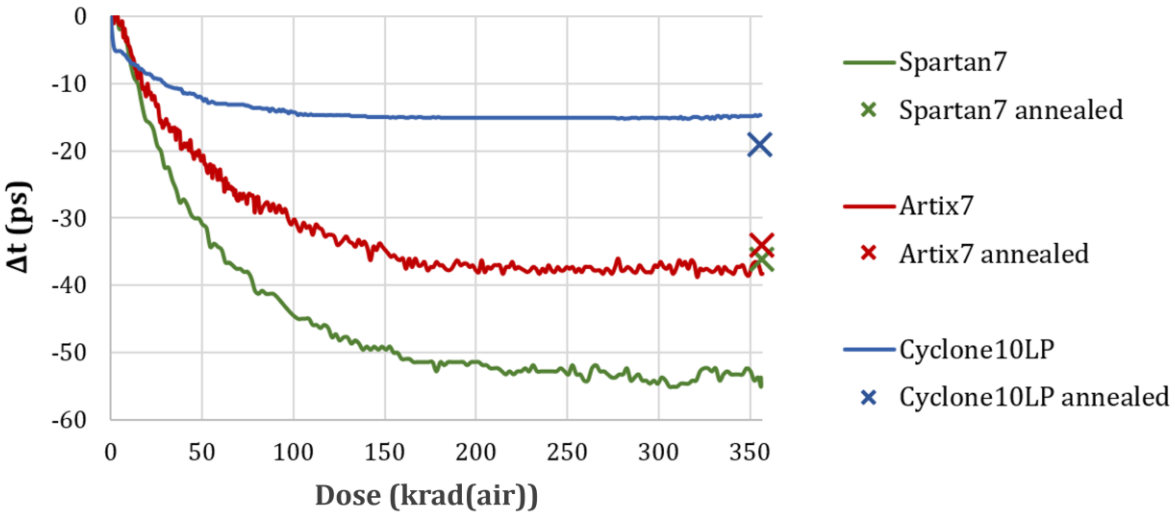


Figure 65: **DSP** propagation delay degradation for the three tested FPGAs. These values are obtained by dividing the total delay degradation of the chain by the number of DSP(Spartan7/Artix7)/Multiplier (Cyclone).

Regarding the propagation delay of LUTs and PIPs, for the *Xilinx* FPGAs, the two LUT chains have been used: the one with close placement (Δt_A) and the one with spread placement (Δt_D). As the two chains have a different ratio of PIPs and LUTs, as described in TABLE V (data extracted from TABLE IV), and considering the global chain delay drift can be fairly approximated by the sum of the delay drift of all individual elements that compose it, the delay drift of one LUT (Δt_{LUT}) and one PIP (Δt_{PIP}) can be computed separately using a system of two equations with two unknowns (9).

TABLE V
LUT CHAINS COMPOSITION

	LUT close placement (A)	LUT spread placement (D)
#LUT/chain	$N_{LUT}^{A} = 160$	$N_{LUT}^{D} = 80$
#PIP/chain	$N_{PIP}^{A} = 180$	$N_{PIP}^{D} = 640$

$$\begin{cases} \Delta t_A = N_{LUT}^{A} \cdot \Delta t_{LUT} + N_{PIP}^{A} \cdot \Delta t_{PIP} \\ \Delta t_D = N_{LUT}^{D} \cdot \Delta t_{LUT} + N_{PIP}^{D} \cdot \Delta t_{PIP} \end{cases} \quad (9)$$

The average individual propagation delay of LUTs and PIPs have been extracted for both *Xilinx* FPGAs as shown in Figure 66.

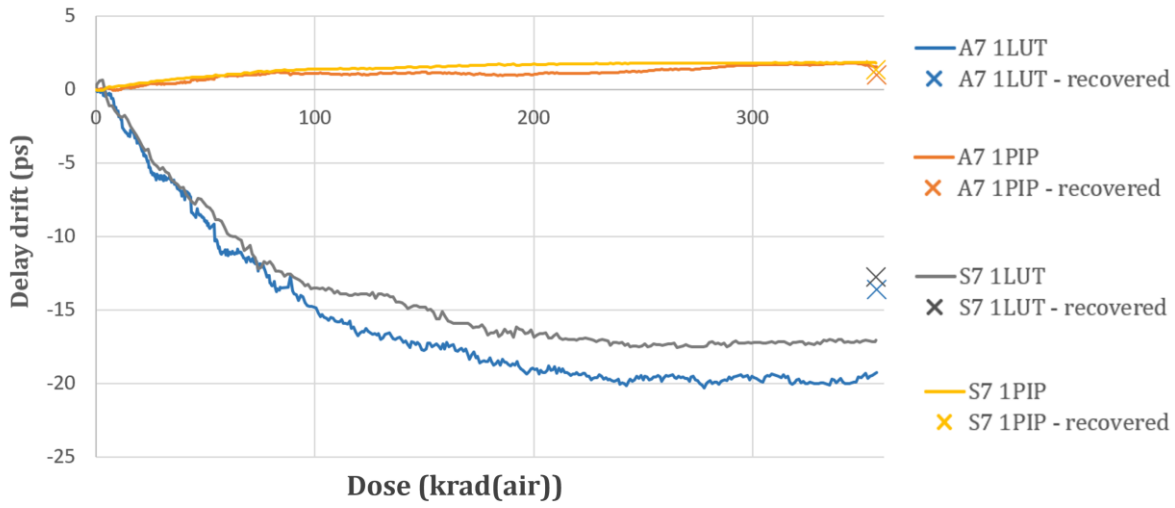


Figure 66: Extraction of TID induced propagation delay drift of PIPs and LUTs on **Xilinx Spartan7** and **Artix7.**

This result shows that the propagation delay drift of PIPs is positive while the one of LUTs is negative. This means that the degradation of PIPs partially compensates the one of LUTs. The more PIPs are used, the lower the observed degradation. This result is consistent with the hypothesis made to (partially) explain the drift difference observed with [115], [117], [118]. Indeed, as the PIPs have an obvious influence on the total propagation delay of the LUT chains, the number of PIP instantiated influence the test results and can partially explain the result differences. This outcome reinforces the benefits of the manual primitive placement and of the extraction of the individual primitive degradation in order to standardize TID test results.

For the Cyclone10LP, the number of PIPs cannot be clearly identified. The methodology presented above could not be strictly applied to this component to extract the individual propagation delay drift of PIPs and LUTs. However, the difference between the two LUT chains lies in the use of an additional 26 logic blocks-wide extra routing segment between each LUT for the chain with spread placement. Thus, using a similar approach (system of two equations with two unknowns) the delay drift of one LUT and associated internal routing segment (routing points internally contained inside a logic block) and the one of the extra routing segments (26 logic blocks-wide) can be extracted separately as shown in Figure 67.

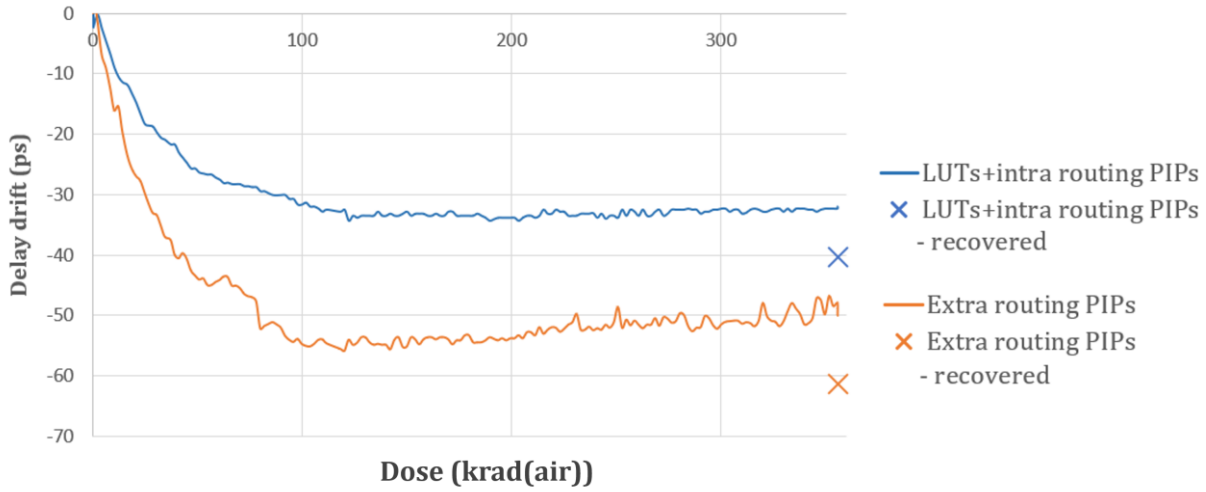


Figure 67: Extraction of TID induced propagation delay drift of one extra routing segment and combined intra routing segment and LUT (**Intel Cyclone10LP**).

In contrast to what was observed for *Xilinx* FPGAs, extra-slice routing segment show a negative delay degradation almost twice as important as the one of a LUT and its associated internal routing segment.

TABLE VI: SUMMARY OF THE PROPAGATION DELAY DRIFT FOR EACH TYPE OF LOGIC AND ROUTING RESOURCE

	50 krad(air)			100 krad(air)			356 krad(air)			annealed		
	min	**mean**	max	min	**mean**	max	min	**mean**	max	min	**mean**	max
Spartan7												
Δt_{LUT}(ps)		**-7.93**			**-13.5**			**-17.1**			**-12.8**	
Δt_{PIP}(ps)		**+0.89**			**+1.40**			**+1.82**			**+1.33**	
Δt_{CARRY4}(ps)	0.00	**-0.25**	-0.52	-0.17	**-0.39**	-0.70	-0.26	**-0.52**	-0.88	0.17	**-0.11**	-0.35
Δt_{DSP}(ps)	-29.7	**-31.0**	-34.9	-40.6	**-45.0**	-49.9	-51.4	**-55.0**	-60.2	-33.4	**-36.2**	-38.6
Artix7												
Δt_{LUT}(ps)		**-8.92**			**-14.8**			**-19.3**			**-13.9**	
Δt_{PIP}(ps)		**+0.67**			**+1.09**			**+1.57**			**+1.12**	
Δt_{CARRY4}(ps)	0.00	**-0.30**	-0.51	-0.36	**-0.51**	-0.66	-0.29	**-0.52**	-0.81	-0.07	**-0.30**	-0.51
Δt_{DSP}(ps)	-18.6	**-21.4**	-25.0	-27.5	**-30.9**	-35.4	-36.6	**-38.7**	-41.9	-25.0	**-27.4**	-29.7
Cyclone10LP												
$\Delta t_{LUT+INTRA}$(ps)		**-1.06**			**-1.27**			**-1.27**			**-1.59**	
Δt_{EXTRA}(ps)		**-1.75**			**-2.20**			**-2.04**			**-2.47**	
Δt_{CARRY4}(ps)	-0.12	**-0.25**	-0.35	-0.12	**-0.25**	-0.35	0.00	**-0.08**	-0.14	-0.34	**-0.48**	-0.58
Δt_{DSP}(ps)	-10.6	**-12.2**	-13.6	-12.7	**-14.3**	-15.5	-12.7	**-14.7**	-15.8	-14.8	**-19.0**	-25.8

The propagation delay drifts for each primitive type and for the three tested FPGA families are summarized in TABLE VI. To illustrate the dispersion of the results among the different chain replicas, the data are provided with the minimum and maximum drifts observed on an individual chain for the DSPs and CARRY. For LUTs and PIPs, as the calculation depends on a combination of results from several chain, these minimum and maximum cannot be provided.

All the TID results presented above reveal that the TID induced degradation can greatly vary between the different primitive types. These differences can be explained by several factors. On the one hand, each primitive type probably has variations in transistor layout and architectural differences in CMOS logic; for example, the propagation delay between drain and source of a pass-transistor (multiplexers) is probably affected differently than for a logic gate where the signal propagates through the transistor gate. On the other hand, the relative degradation is affected by the ratio between the routing track (metal tracks being immune to TID effects) and CMOS logic (number of transistors in the logic path) in terms of propagation delay. Indeed, the propagation delay along the metal track contributes to the total delay but not to its TID induce degradation. When dividing the delay drift by the initial delay, the bigger the contribution of metal tracks to the total delay, the lower the relative degradation is.

Comparing the responses of the different devices, it can be noticed that the Artix7 has a higher maximum delay drift (7,7%), followed by the Spartan7 (6,7%) and finally the Cyclone10LP (2,2%). In addition, after 48h of annealing, the drift decreases for the two *Xilinx* FPGAs while it increases for the *Intel* component. This opposite annealing effect can be explained by the same assumption made earlier. The propagation delay is affected by a competition of the threshold voltage drift and the decrease of transconductance between NMOS and PMOS transistors. During annealing, the rate at which charges are trapped and released in the isolation oxides as well as their impact on the propagation delay can differ between the two types of transistors. The annealing can thus have a stacking or enhancing impact on the propagation delay depending on the technological and architectural parameters of the device.

These time dependent effects suggest that in real conditions, with lower dose rates (and thus more time to anneal), a higher drift for the Cyclone10LP and a lower one for the two *Xilinx* FPGAs could be expected.

3.3.2. THERMAL EFFECT CONSIDERATION

To go further in the standardization of the test results, the FPGA temperature influence must be minimized. As explained previously, the increase in consumption induced by TID tends to raise the junction temperature. This temperature increase in turn impacts the propagation delay in competition with the TID effect itself. Even if the cooling system integrated in the test setup allowed to maintain the temperature of the tested chips between 40°C and 55°C, the temperature variation influence must be evaluated. To quantify this influence, a new experiment was conducted consisting in measuring the propagation delay as a function of temperature. New non-irradiated DUTs were configured with the same benchmarks and run with the same voltage condition in an oven by varying the temperature from 40°C to 100°C. As performed during irradiation, the propagation delay of each chain was continuously monitored. The results of these tests for Spartan7 and Cyclone10LP are presented in Figure 69 and Figure 68 respectively.

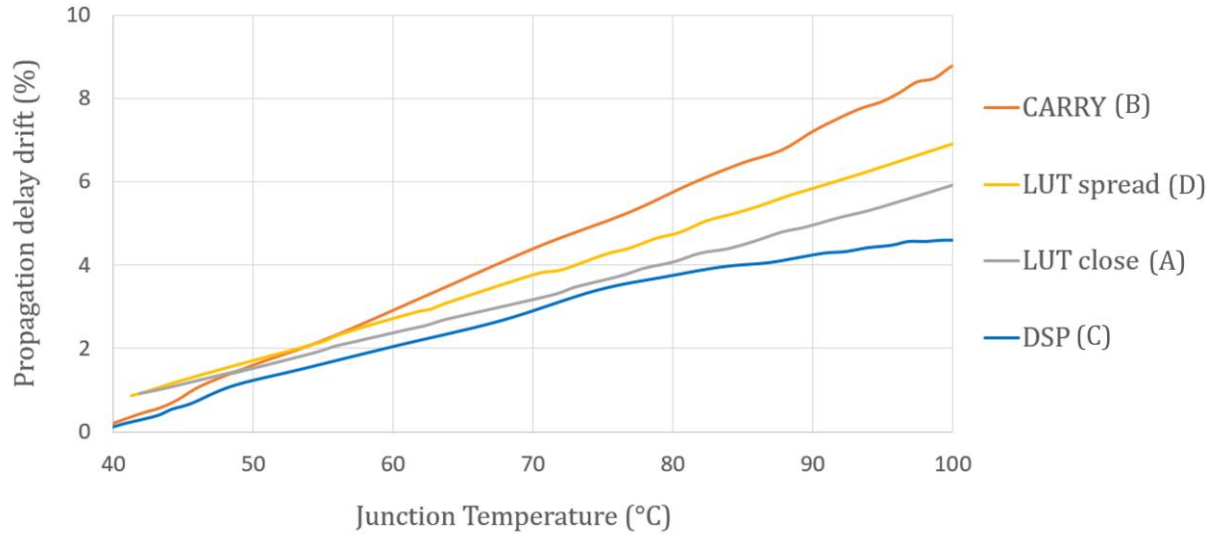


Figure 69: Thermal effects on timing performance on **Xilinx Spartan7.**

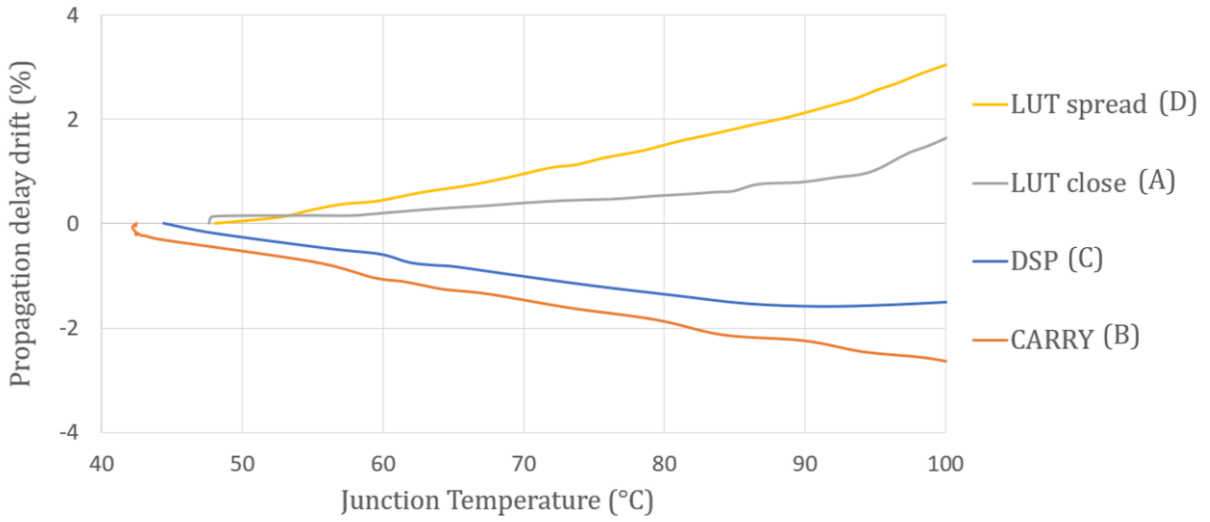


Figure 68: Thermal effects on timing performance on **Intel Cyclone10LP.**

On both tested components, the relative propagation delay behavior is very different from what was observed with TID. To be noted that the temperature not only influences the propagation delay of CMOS logic but also the one of routing tracks by altering the electrical conductivity of the routing materials.

In a first approach, it can be considered that the degradation observed during the X-ray experiment can be approximated by the linear combination of the temperature induced delay drift and the TID induced delay drift at constant temperature. Under this assumption, the data from this experiment can be used to subtract the effect of the temperature variation undergone during the test to isolate the TID-induced delay drift. For each measurement point, the delay drift corresponding to the temperature variation from the initial value is deduced.

As an example, this compensation has been applied to the previously presented results on Spartan7. Figure 70 shows the DUTs' temperature evolution during X-ray irradiation used to apply for the compensation.

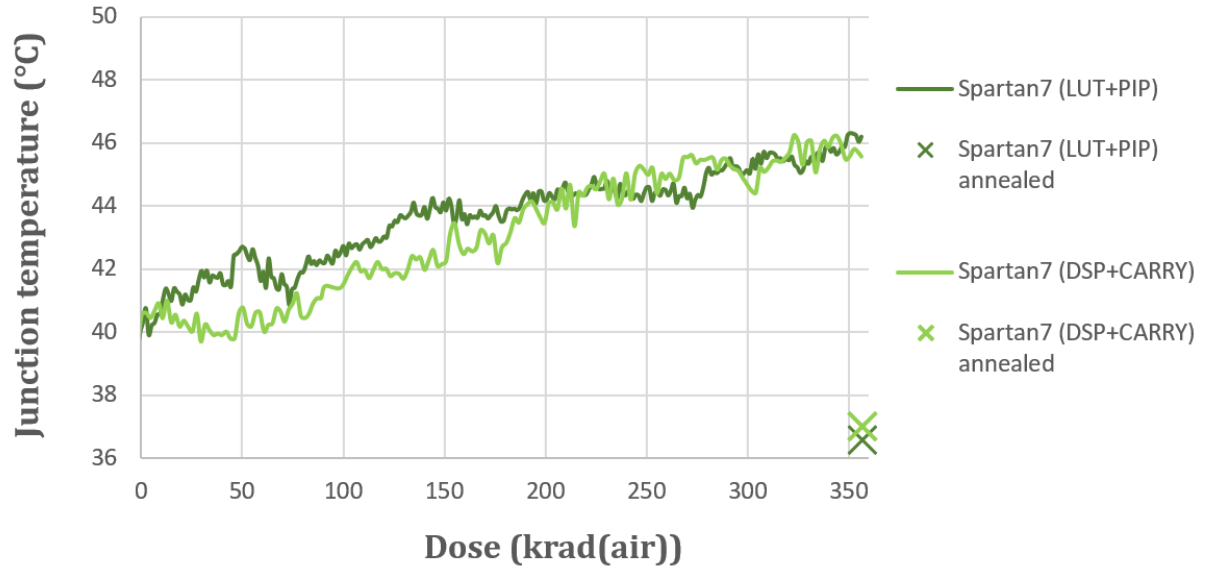


Figure 70: Temperature evolution of Spartan7 FPGAs during X-ray irradiation recorded by the internal ADC.

To be noted that this temperature evolution is not only impacted by the TID induced power consumption increase but also by the propagation delay reduction: as the propagation delay decrease, the operating frequency of the circuit increases, thus, increasing the dynamic power consumption and, consequently, the chip temperature. It should also be noted that the temperature after annealing is lower than the initial temperature due to the fact that the X-ray generator was powered off during the annealing process; the heat generated by the X-ray generator being canceled, the ambient temperature was therefore lower for these measurement points.

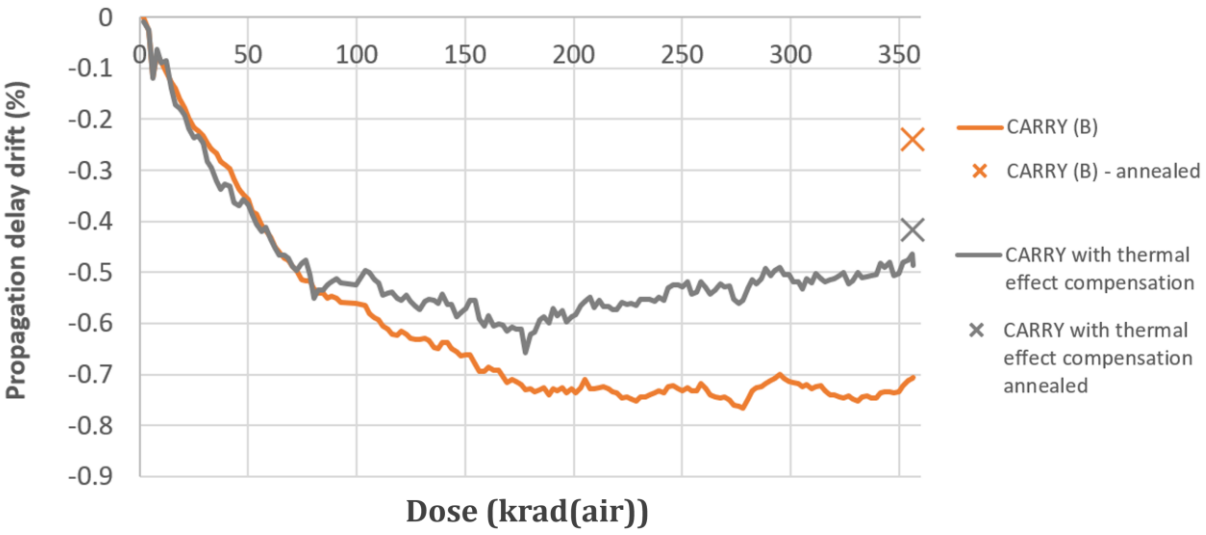


Figure 71: Thermal effects compensation on the Carry chain on **Xilinx Spartan7.**

The compensation has shown a negligible influence on the "DSP" (C) and "LUT close" (A) results as the temperature dependent deviation is much lower than the TID induced deviation. However, for the "Carry"(B) and "LUT spread"(D), the thermal effect compensation (applied to results presented in Figure 61) significantly changes the results (up to 30%) as shown in Figure 71 and Figure 72 respectively.

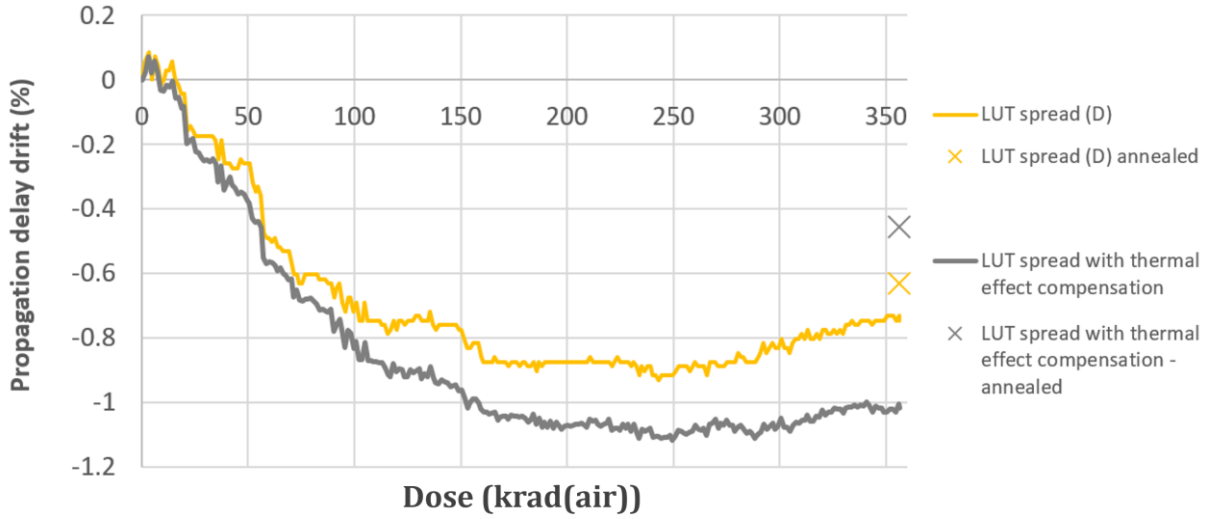


Figure 72: Thermal effects compensation on the LUT chain (spread) on **Xilinx Spartan7**.

These results clearly show the importance of taking thermal effects into account and to limit the temperature variation during the experiments. Indeed, without using any cooling system, the temperature could have reached over 70°C and the temperature influence could have been even higher. Nevertheless, it should be noted that TID and thermal effects have a much more complex mutual influence. Indeed, the irradiation temperature has a major impact on the holes trapping phenomenon in the insulating oxides and thus on the induced threshold voltage drift [125]. Further studies using parametric drift measurements at different irradiation temperatures would be necessary to more accurately relate the intricacy of these two parameters.

3.3.3. POWER CONSUMPTION

After studying the propagation delay drift, this section focuses on the deviation of the power consumption. With the setup used for the experiment, only the global consumption of the board could be analyzed based on the voltage and current delivered by the power supply. The evolution of the power consumption as a function of the dose for all the tested DUTs is presented on Figure 73. From these results, the relative power consumption increase has been computed as shown on Figure 74.

These results show that the relative power consumption increase is very different among the three tested components. Artix7 showed a strong power increase (up to 71%), while the power increase of Spartan7 and Cyclone10LP is apparently much lower (25% and 10% respectively).

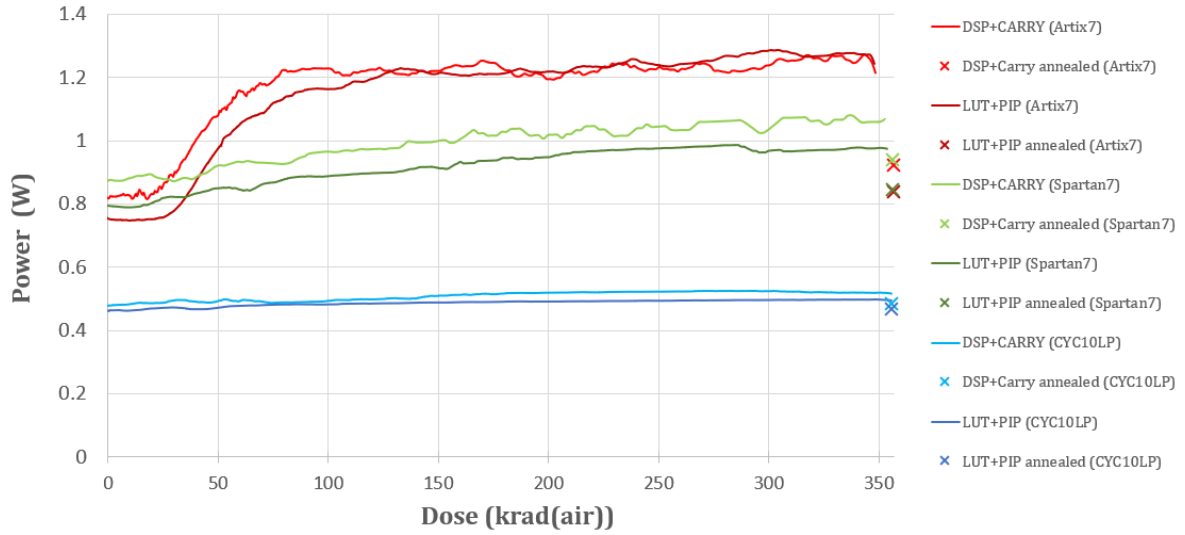


Figure 73: Power consumption (U*I) evolution during TID experiment for the 6 tested DUTs. For each FPGA type, the four benchmarks are split on two DUTs (A and D on one side and B and C on the other).

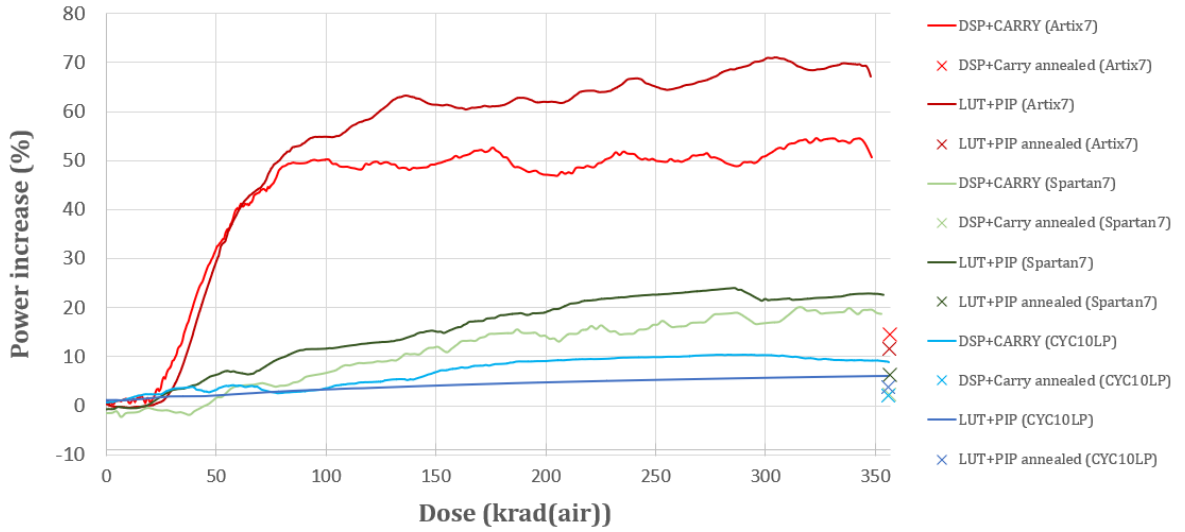


Figure 74: Relative TID induced power increase in the 3 FPGA types. For each FPGA type, the four benchmarks are split on two DUTs (A and D on one side and B and C on the other).

Likewise, for both *Xilinx* FPGAs, the power increase is higher for the DUTs embedding the LUT chains benchmarks (dark red and dark green on Figure 74). This result suggests that the nature of the instantiated primitives influences the power consumption increase. However, as mentioned in the previous section, the power consumption evolution can be directly influenced by the propagation delay reduction. Indeed, during the testing procedure, the range of the frequency sweep performed on the clock signal to measure the propagation delay is continuously adapted to the actual propagation delays of the chains. When the propagation delay is reduced, the mean operating frequency increase thus increasing the global power consumption. As the LUT chain (A) is the one that undergoes the strongest deviation of the propagation delay (Figure 60 and Figure 61), this could explain (at least partially) these current increase differences. This side effect could be canceled by modifying the testing procedure to impose a fixed operating frequency during the power consumption measurement.

This increased consumption must also be attributed to the parametric drift of the transistors in the user logic and in particular to the increase of the leakage current in the off state. The possible presence of internal voltage regulation circuits could also contribute to this effect as they could consume an increased current when exposed to radiation.

3.3.4. DESIGN MARGINS CONSIDERATION

By providing TID induced timing degradation data for each FPGA resource individually (TABLE VI), the test results can be reused to estimate the degradation that will be experienced for a specific design. For each design path, based on its composition (number and type of resource used), the timing degradation suffered after a certain dose level can be estimated by linearly combining the individual degradation extracted previously. This estimation can therefore be used to define timing margins tailored to each critical path of the design and to provide designers with tools to improve their design to meet their timing specification. As an example, most of the designs integrating long arithmetic operators will be limited in frequency by the critical path of the carry propagation. Knowing that the Carry logic or DSP resources do not have the same sensitivity than the LUTs, prevents from over or under constraining the design and may help to reconsider the implementation.

In the specific case of the components tested in this study, a decrease of the propagation delay has been observed. This means that precautions must be taken, not on the setup time of the flip-flops (the signal must arrive before the rising edge of the clock) but on the hold time (the signal must be maintained for a certain time after the rising edge to ensure that it is properly captured). For these components, the observed drift remains nevertheless relatively low. The focus should be mainly on the power consumption drift, specifically on the *Xilinx* FPGAs, by ensuring that the entire system (converters, regulators, battery and other power sources) can handle such an increase in power demand.

3.4. CONCLUSION

In this chapter the state of the art of test methodologies for the evaluation of TID effects on FPGA has been described. To overcome their limitation and to improve the TID testing results standardization, a new test methodology based on the development of dedicated benchmarking structures allowing the extension of the timing degradation evaluation to all the logic and routing resources of the device have been presented. In addition, a new cost-effective technique for in-situ delay measurement has been proposed.

The presented test methodology, referred to as benchmarking can be implemented on commercial boards and tested with reduced external instrumentation. The test structures, evaluating different types of resources have been irradiated with X-Rays over three types of FPGAs. The results presented here show the ability of the methodology to highlight the most sensitive resources by providing detailed measurements of their degradation and to provide quantitative comparison of the FPGA families between them. The results thus provide essential information for the consideration of TID effects in circuit design. Depending on the component used and the logic resources that dominate the critical paths of the design, timing and power margins can be carefully chosen to ensure the proper operation of the system over the entire duration of the mission with maximum performance. The use of a cooling system and the temperature experiments performed allowed the proper decoupling of TID effects from thermal effects. This

thermal effect decoupling along with the individual timing degradation evaluation for each primitive type, enabled by manual placement of the logic gates, provide a step forward in the standardization of TID test results.

In a more pragmatic way, all three components showed a strong resistance to dose effects up to 356krad(air). As the propagation time decreased with the dose, additional consideration should be taken into account for the hold times in the static timing analysis.

4. FPGA TESTING METHODOLOGIES FOR SEE ASSESSMENT

Due to their heterogeneity, flexibility and increasing complexity, the behavior of FPGAs to SEEs is difficult to analyze. In chapter 2, the different SEE types that may disturb the operation of FPGA based systems have been discussed. Namely:

- Single Event Latchup, a sudden increase in power consumption until the component is power cycled.
- Single Event Upset on the flip-flops: corruption of flip-flop state until next clock cycle.
- Single Event Upset on the user memory blocks: the data contained in a memory block is corrupted.
- Single Event Upset on the configuration memory: alteration of the topology of the implemented circuit until next configuration (main concern for SRAM based FPGAs).
- Single Event Transient generated in the combinatorial logic.

To address these reliability constraints, the main point is to understand how to analyze the sensitivity of an FPGA to these different SEE types and the associated failure mechanisms. This type of investigation may need to be applied to several components, coming from different families or different vendors, in order to compare their sensitivity and to choose the most suitable component for the mission profile, considering the other constraints of the system (cost, capacity, consumption, performance). Once the component has been selected, the reliability of the circuit to be implemented must be estimated. Namely, to determine the expected error rate on the different parts of the circuit for a given radiative environment. If the reliability of the system does not meet the constraints of the mission profile, the implementation of mitigation techniques must be considered and their effectiveness evaluated. Radiation tests are commonly performed for these last two steps.

In this chapter, state-of-the-art testing methodologies are firstly discussed. Then, a new testing methodology will be presented, based on the development of specific benchmarks for SEE sensitivity testing. The efficiency and the advantages brought by this methodology are finally evaluated through different test campaigns under accelerated particles and through fault injection.

4.1. STATE OF THE ART METHODOLOGIES

In this section, the main challenges of SEE testing are firstly discussed. Different methodologies used to evaluate the sensitivity of the configuration memory and the different primitive resources are then presented. Finally, the test methodologies used to evaluate the sensitivity of a design through application specific testing or through the use of benchmarks are introduced.

4.1.1. SEE TESTING CHALLENGES

The main difficulty with SEE testing comes from the very nature of FPGAs. Given, the complexity of these components, many questions arise. How to test a component whose radiation sensitivity entirely depends on the way it is configured? Should one perform new tests for each implemented circuit? Or is it better to find dedicated testing structures to evaluate the intrinsic sensitivity of each primitive? How to reuse these results to estimate the reliability of the final application in that case?

In addition to this main challenge, come many technical limitations related to the tests under beam of accelerated particles. On the one hand, the experimentation time is limited by the high cost of the beam time in radiation test facilities (500-900€ per hour). On the other hand, even when using a focused beam, the radiation level in the rest of the room can impact the other components of the PCB and limits the proximity between the DUT and external instrumentation. The acquisition systems can usually be installed in a nearby radiation-free room, implying a significant cable length to reach the DUTs and therefore some limitation in the external interfaces. These technical difficulties limit the test complexity, the insight view on the internal state of the system and the error statistics generated during the campaign.

Any SEE testing methodologies must consider these technical limitations.

4.1.2. CONFIGURATION MEMORY SENSITIVITY EVALUATION

For SRAM-based FPGAs, the main SEE induced failure generally comes from SEU in the configuration memory. Its sensitivity can be evaluated through static test [36], [77], [126]–[128]. These tests are similar to those performed on regular SRAM or DRAM memories: the FPGA is programmed and irradiated in static mode (clock signal off, no read nor write operations) with a certain fluence. Once the beam is off, the content of the configuration memory is read back and compared to the initial content. This test procedure is described in Figure 75.

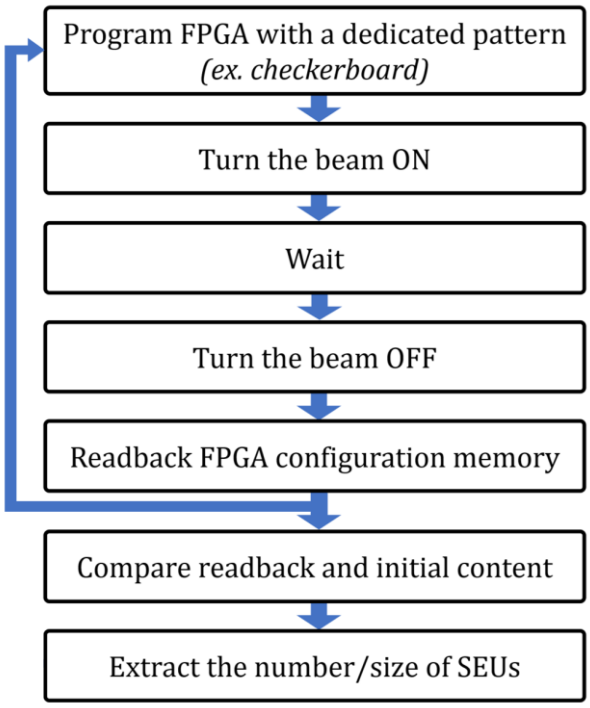


Figure 75: Static test of the FPGA configuration memory.

The device cross section can be obtained by dividing the number of SEUs by the fluence (10).

$$\sigma_{device} = \frac{N_{SEU}}{\Phi} \tag{10}$$

The cross section per bit can then be obtained by dividing the device cross section by the total number of bits in the configuration memory (11).

$$\sigma_{bit} = \frac{\sigma_{device}}{N_{bit}} = \frac{N_{SEU}}{\Phi \times N_{bit}} \quad (11)$$

Static tests allow to clearly identify the cross section of the configuration memory as well as the proportion and the size of the multiple bit upsets. For example, in [127], the percentage of Multiple Bit Upset have been measured as a function of the heavy ion LET (Figure 76) and as a function of the angle of incidence (Figure 77) for *Xilinx* Virtex-5 FPGA.

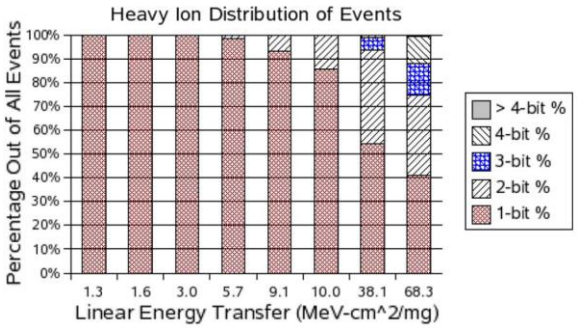


Figure 76: Distribution of MBU in the configuration memory of Virtex-5 as a function of LET, from [127].

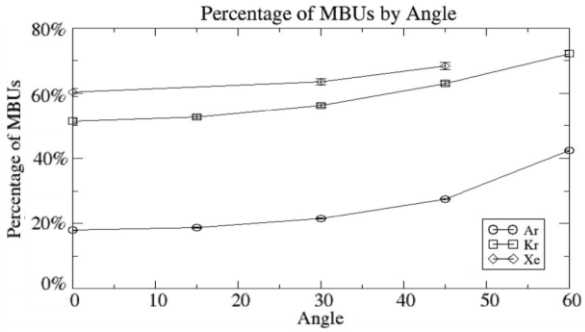


Figure 77: Distribution of MBU in the configuration memory of Virtex-5 as a function of the angle of incidence, from [127].

The fluence used in static tests must be carefully selected based on a pre-test estimate of the bit cross section to avoid an excessive accumulation of errors. Indeed, the probability of two SEUs affecting two neighbor bits (that can result in a false MBU detection) must be minimized.

With *Xilinx* FPGAs, the configuration memory cross section can also be measured dynamically by taking advantage of the error detection reports provided by the integrated soft error mitigation (SEM) controllers mentioned in section 2.4.4. Indeed, modern *Xilinx* FPGAs now integrate a dedicated hardwired circuitry originally designed to prevent the accumulation of errors in the configuration memory [129]. In the background of the user design, the dedicated logic continuously reads back the configuration and computes a CRC (Cyclic Redundancy Check) code for each frame. The computed CRC is compared to the one computed at device initialization. When a CRC mismatch is detected, the frame error is read back again and the error is localized and corrected by using the pre-computed syndrome (at startup or at bitstream generation) from another layer of Error Detection And Correction codes (EDAC). This integrated mitigation engine can be supplemented by a soft controller, the SEM IP [130], implemented on the user fabric and connected to the internal configuration ports. This controller can be used with an external interface (UART) to report the status of the mitigation engine, the detection of events and the corresponding addresses in the configuration memory and the potential failure to correct the memory corruption (multiple bit upsets in the same frame). By exploiting this SEM controller feature during dynamic SEE testing, the configuration memory sensitivity can be evaluated by counting the number of reported events. This technique has been used in [72], [131] and the resulting configuration memory cross section showed good agreement with the one obtained by static tests. To be noted that this approach has one main limitation: the configuration frames associated to user memory elements (BRAM, LUTRAM, SRL) are masked to the mitigation controller to avoid any interference with the user design. Therefore, the errors occurring in these frames are not detected. Even though the effects on the configuration memory (CRAM) are generally predominant on SRAM FPGAs, the evaluation of its cross section does not allow to

estimate the reliability of the implemented system. For this purpose, it is necessary to estimate the susceptibility of the design to CRAM corruptions, by evaluating the number of configuration bits that can cause failures when corrupted (defined as “critical bits”). It is also essential to evaluate the sensitivity to other types of phenomena, especially for Flash and antifuse FPGAs.

4.1.3. PRIMITIVE LEVEL SENSITIVITY EVALUATION

A first approach, widely used for FPGA SEE testing, consists in measuring the intrinsic sensitivity to SEU for each type of primitives. For each primitive type, dedicated test methods or specific testing structures must be developed. They are generally made up of long monomorphic chains of cascaded elements. For example, a widely used testing structure to measure the SEU susceptibility of flip-flops is the shift-register [25] as shown in Figure 78.

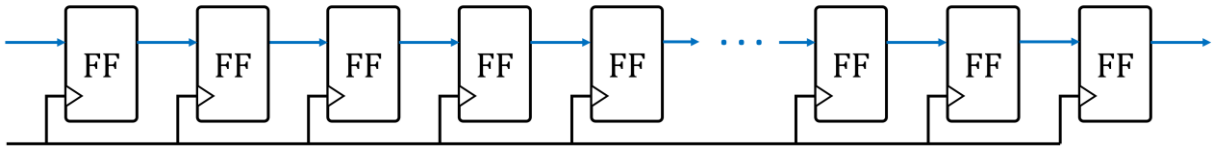


Figure 78: Shift-register: flip-flops are cascaded to form a long chain.

By injecting a pattern in the first flip-flop, the values propagate from flip-flop to flip-flop at each clock cycle. The output signal is monitored and compared to the input signal to check for potential errors. The flip-flop SEU cross section can then be extracted based on the number of events observed for a given fluence. This widely used structure can be supplemented by inserting one or more LUTs between each flip-flop, as shown in Figure 79.

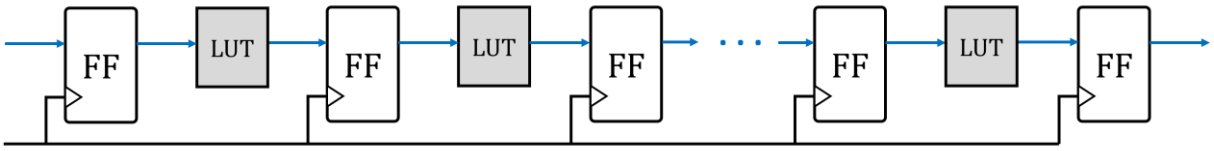


Figure 79: Shift register supplemented by inserting an LUT between each register.

By repeating dynamic tests with these structures at different clock frequencies, the SEU and SET rates can be extracted. Indeed, as explained in section 2.3.3, the SET capture rate is proportional to the clock frequency while the SEU generated on a flip-flop has a lower probability to propagate to the next flip-flop when the clock frequency increases. This technique can be further improved by implementing LTMR on the flip-flops as shown in Figure 80.

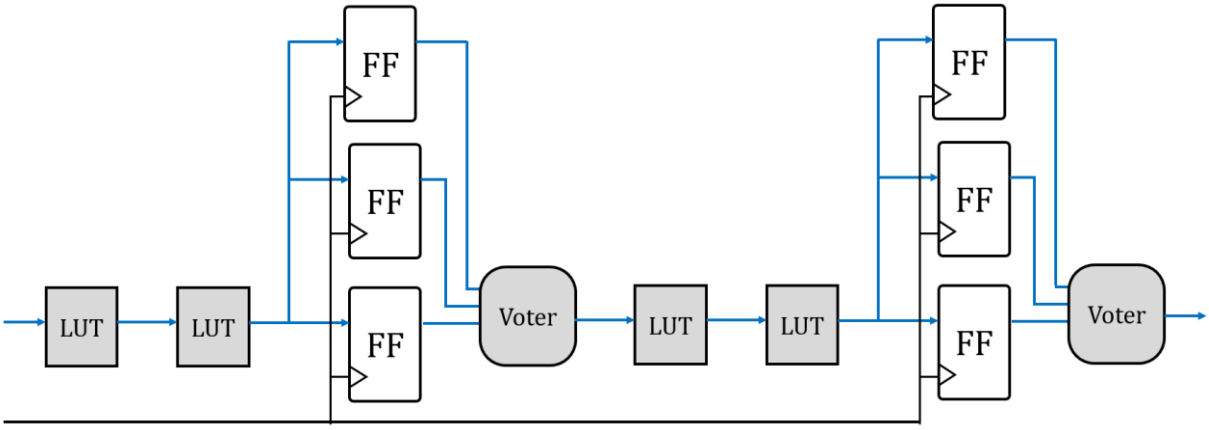


Figure 80: Shift register with LUT inserted between each triplicated flip-flop.

By triplicating the flip-flops, the SEUs on flip-flops are totally masked and their contribution to the global shift-register cross section is theoretically removed. The event recorded with such structure are then the result only of the capture of SETs as shown in Figure 81. In is worth mentioning that, in some technologies, the insertion of majority voters can increase the SET sensitivity.

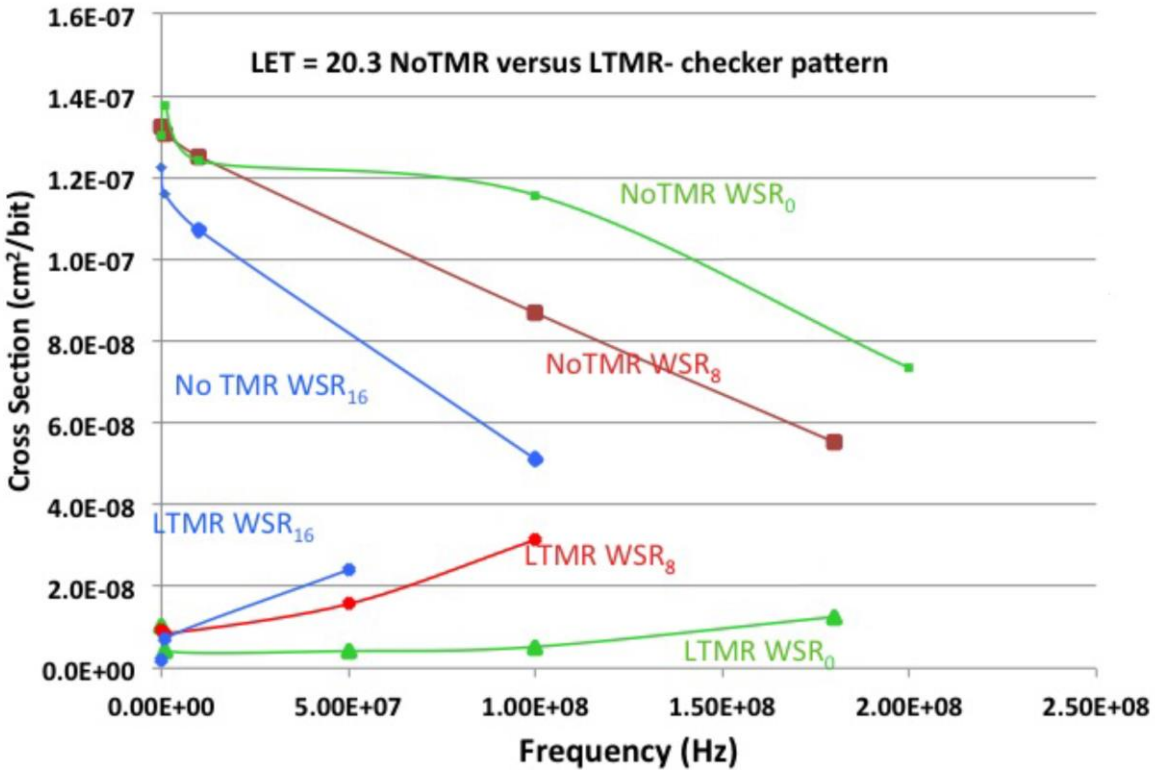


Figure 81: Heavy ion test results on proASIC3 FPGA performed with 6 different shift-register test structures. WSR_0 refer to shift register with no LUT inserted. WSR_8 refer to shift-register with 8 LUTs inserted between each flip-flop. WSR_{16} refer to shift-register with 16 LUTs between each flip-flop, from [132].

As shown in this figure, the test structures with flip-flop protected by LTRM have an increase SEE sensitivity when the operating frequency increases. This corresponds to the SET capture

phenomenon. On the contrary, the cross section of the test structures without LTMR decreases when the frequency increases. This mainly corresponds to SEUs in the flip-flops.

Other test structures have been developed to extend the evaluation of SEU sensitivity to the other FPGA primitives:

- **BRAM**: Their sensitivity evaluation can be performed through static tests described earlier as their content is part of the configuration memory. Other approaches have also been proposed to dynamically test these memory blocks. In [133], an ECC encoder is used to write data into the memory through the writing port, and an ECC decoder is used in the reading port to detect the presence of memory corruption as shown in Figure 82.
- **DSP:** Their sensitivity can be evaluated by different test structures. In [67], DSPs configured as multipliers are simply used in tandem by comparing their output signals at each clock cycle to detect the presence of errors. More complex testing structures may also be used such as matrix multiplication circuits and digital filters [132].
- **Transceivers:** In [134], a test structure is proposed to evaluate the sensitivity of GTX transceivers. It uses two FPGAs, one under test and one used as mirrored responder. Data frames are internally generated and send through the TX line. The mirrored FPGA received the transmitted data frames through the RX line and checked for data consistency. Both TX and RX lines are concurrently tested as shown in Figure 83.

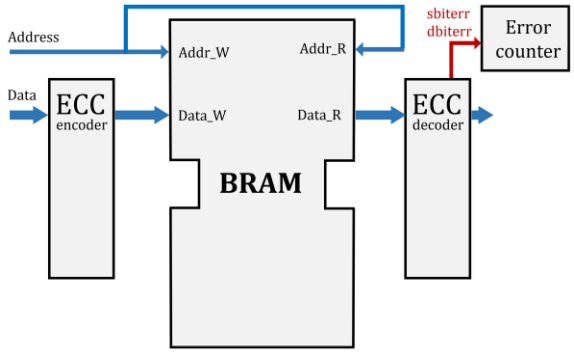


Figure 82: BRAM test structure. Error monitoring is performed by writing ECC encoded data and decoding the read output, from [133] .

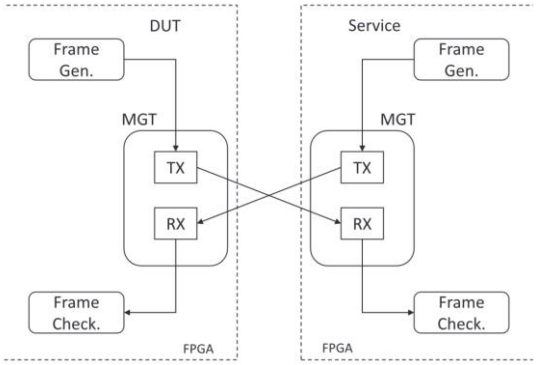


Figure 83: Transceivers test structure. Two mirrored FPGA are used to transmit and received data frame between themselves, from [134].

To test the functionality and detect errors in these different structures, different approaches exist. The first one consists in injecting the signals and monitoring the output signals externally, either using specific measurement instruments (digital signal generator/analyzer) or using another FPGA, outside the particle beam, usually integrating error detection and processing as shown in Figure 84. However, this test setup can be challenging to deploy due to the signal frequency limitations of the board-to-board connectors.

The second approach, called Built-In Self-Test (BIST) consists in managing the generation of input data, the detection and processing of errors as well as the transmission of error logs to the external computer directly inside the FPGA under test. The obvious limitation of this approach is the

sensitivity of the testing structure itself. Errors generated in this circuit part can be erroneously identified as events happening in the test structure. It is then necessary to ensure that the sensitivity of the BIST circuit can be neglected with respect to the sensitivity of the test structure itself. The BIST circuitry is ideally hardened by integrating mitigation techniques and the way errors are handled should be done carefully so that errors appearing in the BIST circuitry can be differentiated from errors generated in the test structure.

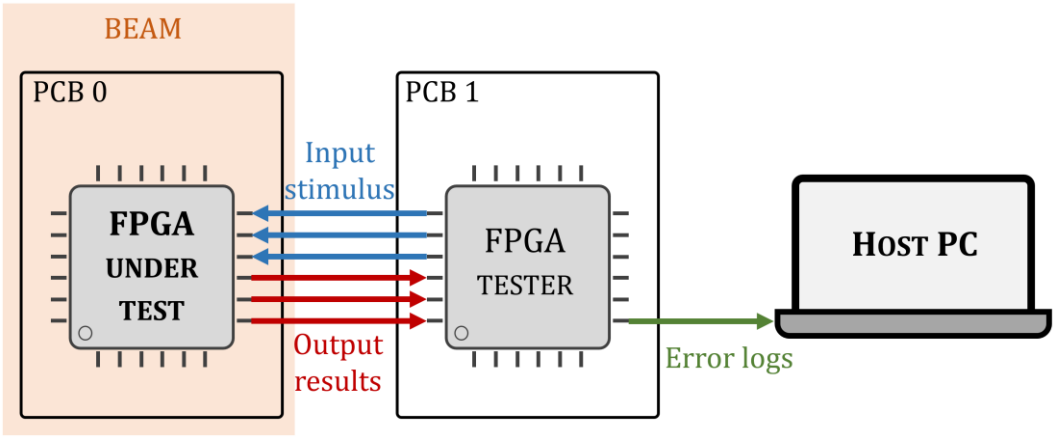


Figure 84: Test setup with stimulus generation and output monitoring performed external by another FPGA. Error detection and processing can be performed by the FPGA tester and reported to an external computer.

The general approach described in this section provides useful information about the sensitivity of the different FPGA resources. Nevertheless, this approach faces many difficulties regarding the reliability estimation of real designs. These test structures are often monomorphic and offer very little diversity in the arrangement of resources. For the example, with structures like shift-registers, all the cells have a fan-in (number of input connections) and a fan-out (number of output connections) of one. The number of logic levels between each flip-flop is also uniform. This lack of diversity in the circuit parameters prevents the extrapolation of the radiation test results to estimate the sensitivity of real application designs since the types of interactions between resources are highly dependent on the topology of the implemented circuit.

4.1.4. FINAL APPLICATION TESTING

Due to the limitations of the previously presented test methodologies to estimate the sensitivity of real application designs, designers might consider another approach which consists in directly testing the FPGA with their final design implemented as done in [135], [136]. This approach can be valuable for validation purposes, to ensure, during the final design stages, that the implemented system complies with the reliability constraints imposed by the mission profile. However, one must be aware of the various limitations of this approach. Real designs implemented on FPGAs are generally very complex. Logical error masking, especially for circuits where mitigation strategies have already been applied, implies a low error rate and therefore a low statistical significance for reliability assessment. Also, since the state space is generally very large, not all circuit states can be sufficiently exposed during a single radiation test to be representative enough of the failures that may occur during real operation conditions. Finally, the low visibility provided by these tests does not allow neither to identify properly the most sensitive areas of the design nor to identify the predominant failure mechanisms.

4.1.5. RADIATION TEST BENCHMARKING

The other approach to SEE testing that will be developed in this study is based on the use of hardware benchmark as proposed in [137]. In the referenced study, the use of the I99T portion of the ITC'99 benchmark [138], [139] (originally developed to improve automatic test pattern generation tools) is proposed as a universal reference for radiation testing to allow researchers and designers to compare on a common basis their test results on different components and to compare the effectiveness of their mitigation solutions. These benchmarks, implemented in a common HDL, are composed of different functions from finite state machines to soft-core microprocessors.

However, the ITC'99 benchmark may not be perfectly adapted to radiation testing as it is limited in the diversity of used resources and circuit topologies with a major shortcoming with regards to arithmetic operators, a structure widely used in all computationally intensive applications.

In this thesis, a new test methodology is proposed based on the development of a dedicated benchmark adapted to radiation testing. This new benchmark brings different improvements regarding the diversity of the circuit architectures in order to increase the visibility on the sensitivity of the FPGA resources and on the predominant failure mechanisms.

4.2. BENCHMARKING REQUIREMENTS

The benchmark approach lies in the middle of the two approaches mentioned above: sensitivity testing at the primitive level and testing of the final application. It is intended to reproduce the diversity and complexity of interactions found in a real circuit while providing good visibility on the predominant failure mechanisms and potential vulnerabilities. In an analogous way to the testing methodologies used at system level (e.g. an electronic board), where one would seek to test each of the sub-assemblies that compose it (e.g. each component) to estimate the reliability of the complete system, the benchmark approach must propose test structures corresponding to subsets of circuits used in real applications. That way, the test results can be used in a generic way by designers to estimate the sensitivity of the different modules that compose their designs. Another key point of the benchmarking approach lies in its ability to be used to compare the radiation sensitivity of different components and identify for each of them the main vulnerabilities and to provide recommendations regarding the mitigation techniques to be applied.

To build an effective benchmark for SEE testing, it is necessary to identify the criteria that define its quality and its ability to meet the expectations mentioned above:

- **Portability:** For comparison purposes, the benchmark should be provided in generic HDL so that it can be implemented on FPGAs from different families and different vendors. In the case where code structures specific to the FPGA architecture (e.g. primitive level instantiation) are used, the benchmark must be derived in different versions by proposing a solution for each architecture type.
- **Scalability:** In order to compare test results on FPGAs of different sizes or to analyze the effect of data vector size on the sensitivity of a specific function, it is required that the benchmarking structures can be repeated and implemented at different scales. For this purpose, emphasis should be placed on the use of generics (an HDL object used to

customize a module) to define the dimensions of the circuit (size of data vectors, number of replicas, etc.).

- **Heterogeneity:** To ensure representativity of all the designs implemented on FPGAs, a wide diversity of circuits parameters must be proposed, regarding the number and type of instantiated resources, the fan-in and fan-out of each cell, the number of logic stages and the propagation delay between flip-flops.
- **Testability:** A low error masking rate is recommended to increase the number of observed errors and consequently the statistical significance. A reduced state space should also be considered so that each system state is sufficiently exposed. The state space refers to the set of possible states that the circuit can have, a state being defined by the values carried by all the signals of the circuit. This criterion ensures that test results are representative enough of the failures that may occur during real operation conditions.

4.3. BENCHMARK SELECTION

In order to respect the criteria mentioned above, it was decided to develop several test structures around arithmetic operators and particularly around parallel multipliers.

Achieving an efficient and performant multiplier implementation is a critical challenge to hardware designers. As described in [140], this operator can be implemented in various ways, leading to different compromises in terms of resource utilization, power consumption and performance, and as a consequence, impact the radiation performance of the system. Moreover, the same compromises are met in larger systems and thus, lessons learned from the benchmarking structures could be scaled to larger designs. In addition to the above-mentioned criteria, the multiplier has been chosen as a test structure for the following reasons:

- Multipliers are widely used functions in all computationally intensive applications (digital signal and image processing, cryptography, artificial intelligence, etc.). The utilization of logic functions close to those actually implemented improves the representativity and facilitates the interpretation and reuse of the results.
- Multiplication functions can be implemented using different resources (DSP Blocks, Carry Logic, LUT, Flip-flop) and different arrangements between those basic elements. This allows, through the same logic function, to mobilize the most abundant resources of the FPGA and to use a great diversity in the circuit parameters (number of logic stages between the flip-flops, fan-in and fan-out of each element, LUT utilization, etc.).
- Multiplication is an operator that offers a great visibility of errors. With a careful choice of input test vectors, the very nature of the operation allows to reveal most of the errors generated in the circuitry by monitoring only the output of the operator. By using different implementations of the multiplier, a dual objective can thus be achieved: the comparison of the susceptibility of the operator on its different implementations can serve directly as a guideline for the designer while efficiently evaluates the sensitivity of the most abundant logic resources.

In this section, a brief reminder is given about binary arithmetic and its implementation. The architecture of the different implementations of multipliers used in the benchmark are then detailed and the structure of the finite impulse response filter, used to group and efficiently test the multipliers will be presented.

4.3.1. HARDWARE IMPLEMENTATION OF ARITHMETIC OPERATORS

4.3.1.1. BINARY ADDITION

The binary adder is the backbone of all other arithmetic operators. The addition of two bits can be implemented by an XOR gate for the sum bit and an AND gate for the carry bit. This structure, depicted in Figure 85, is called "half-adder".

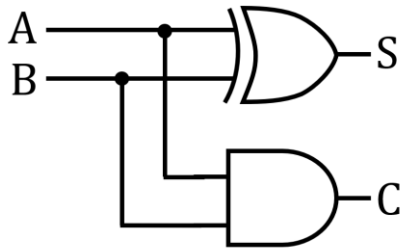


Figure 85: **Half-adder** circuit. Sum (S) is computed by a XOR gate between the two bits (A & B) and the Carry (C) is computer by an AND gate.

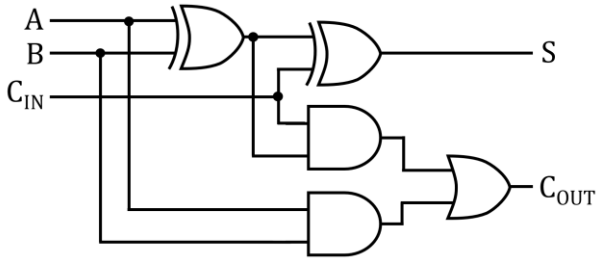


Figure 86: **Full-adder** circuit. Supplement the half-adder by integrating an input carry bit (C_{IN}) to compute the Sum (S) and the output Carry (C_{OUT}).

To add two binary numbers, this operation must be performed bit by bit between the bits of the same weight. For each weight, except the Lowest Significant Bit (LSB), the operator has to consider the carry bit resulting from the addition of the two lower weight bits. The carry is integrated in the calculation using a full-adder, the structure of which is presented in Figure 86. A binary adder is thus composed by cascading a half-adder for the LSB with a series of full-adders for the other bit weights. This structure called ripple-carry adder is shown in Figure 87.

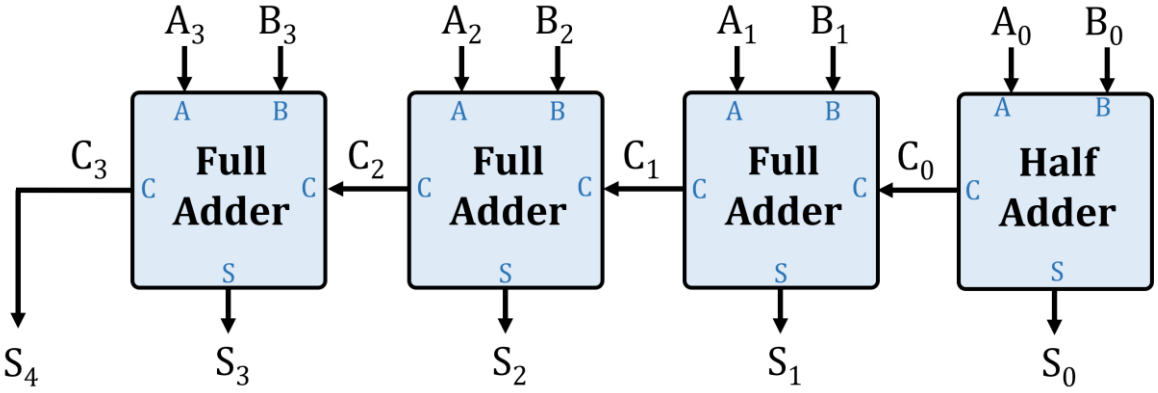


Figure 87: **Ripple-carry adder** structure composed of full-adders and one half-adder for LSBs.

The critical path of such structure is imposed by the propagation of the carry from the LSB to MSB. The carry propagating through one full adder passes through two logic gates (one AND gate and one OR gate as depicted in Figure 86). The addition of long binary vectors can thus suffer from heavy timing penalties. Pipelining can be realized by inserting registers in the carry propagation path to increase the maximum operating frequency. In that case, registers must also be added in the input bits and sum bits to synchronize the data flow.

On FPGA, a ripple-carry adder could be implemented using a 2-output LUT for each full adder. However, the structure of the logical blocks would make this implementation very inefficient because the propagation of carries between each full-adder would have to go through the extra-slice routing network, inducing long propagation delays.

To overcome this limitation, modern FPGAs integrate a Carry propagation circuit in each slice (definition in section 2.2.2). This solution is based on the Carry Look-Ahead (CLA) arithmetic described in Figure 88.

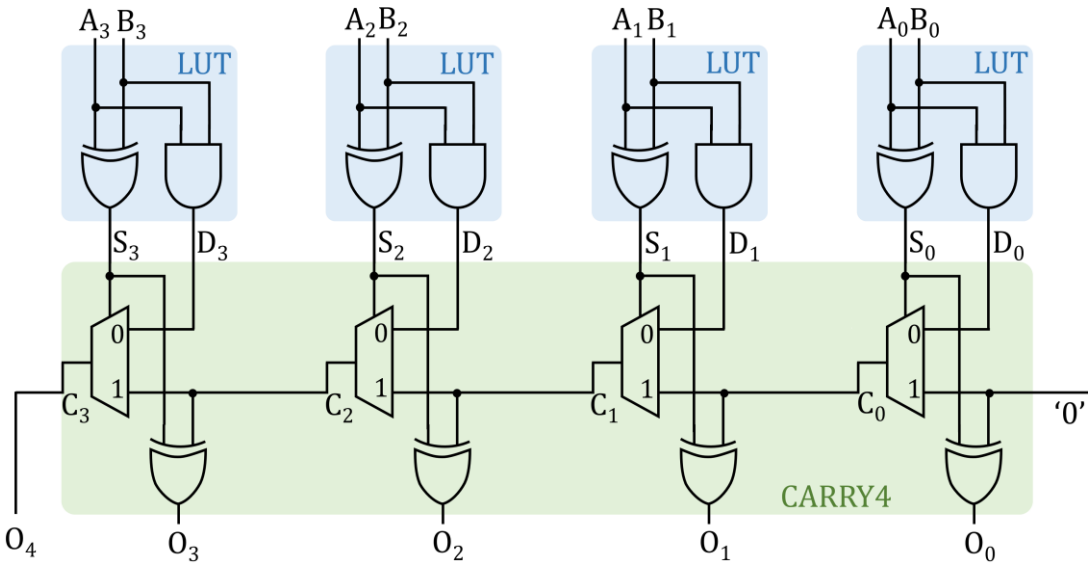


Figure 88: Carry Look-Ahead adder implemented on *Xilinx* FPGA logic bloc. Each LUT implements a half-adder while the CARRY4 gate computes the adder output through a XOR gate (integrating the carry bit from the lower weight bit) and compute the output carry bit through a multiplexer. This multiplexer propagates the value of the previous carry when the two inputs are of different value (S=0) or the results of A&B (D) when the two input have the same value (S=1).

This adder structure computes the addition in two stages. In the first stage, the two bits of the same weight are fed to a half adder implemented on a two-outputs LUT. In the second stage, the CARRY logic cell integrates the carry from the lower weight bits to compute the output sum and the output carry. Formally, given $\boldsymbol{S}$ the output of the XOR between the two input bits ($\boldsymbol{A}$ and $\boldsymbol{B}$), the output results ($\boldsymbol{O}$) can be computed with a XOR gate between $\boldsymbol{S}$ and the input carry ($\boldsymbol{C_{IN}}$). If $\boldsymbol{S}$ is equal to '0', this means that the two input bits are of the same value. In this case, the output carry is equal to $\boldsymbol{A}$ AND $\boldsymbol{B}$ whatever the input carry state is. Otherwise, the output carry is equal to the input carry. This structure, implemented on FPGA, provides better timing performances as the carry propagate through only one multiplexer per bit and using delay-optimized intra-slice routing segment.

4.3.1.2. TERNARY ADDER

FPGAs with 6 inputs, 2 outputs LUT architecture can integrate in the same logic block the addition of 3 binary vectors using an architecture similar to the binary adder as described on Figure 89. The result of an addition of 4 bits (three data bits plus carry bit from lower weight addition) must be stored on 3 bits. The lowest significant bit is directly propagated to the output result, the second bit is propagated to the next addition using the dedicated carry chain logic while the Most Significant Bit (MSB) must be returned to the input of the higher order LUT. To do this, the MSB must exit the logic block and re-enter it using the extra-slice routing matrix, which implies delay penalties.

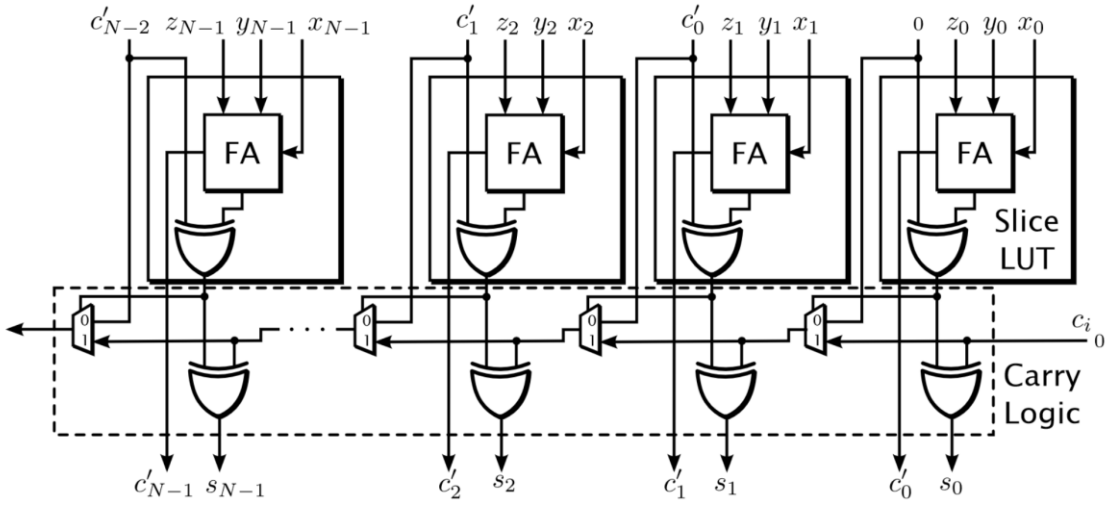


Figure 89: Ternary adder implemented on *Xilinx* FPGAs. Each LUT computes the 3 bits of the same weight and integrates the carry from the previous sum. This carry bit must propagate outside the slice to meet the LUT to LUT connection, from [141].

4.3.1.3. BINARY MULTIPLIER PRINCIPLE

A binary multiplier computes a set of partial products and then sums the bit-shifted partial products together. The partial product computation is done by AND-gating the first operand with one bit of the second operand as shown in Figure 90. The multiplier result is obtained by adding the partial product set using a process called "partial product reduction". The main differences in the implementation of multipliers lie in this reduction of partial products that can be implemented in many different ways. This reduction process is the multiplier part that contributes the most to the to its delay, power and area.

$$A * B = A * B_3B_2B_1B_0 = A * B_0*2^0 + A * B_1*2^1 + A * B_2*2^2 + A * B_3*2^3$$
$$= (A \text{ AND } B_0)<<0 + (A \text{ AND } B_1)<<1 + (A \text{ AND } B_2)<<2 + (A \text{ AND } B_3)<<3$$

A * B =				$A_3.B_0$	$A_2.B_0$	$A_1.B_0$	$A_0.B_0$
+			$A_3.B_1$	$A_2.B_1$	$A_1.B_1$	$A_0.B_1$	
+		$A_3.B_2$	$A_2.B_2$	$A_1.B_2$	$A_0.B_2$		
+	$A_3.B_3$	$A_2.B_3$	$A_1.B_3$	$A_0.B_3$			

Figure 90: Binary multiplication principle. For each bit in the second operand (in red), a partial product is computed by AND-gating the first operand (in blue) with the bit from the second operand and shifting the result according to the bit weight. All partial product generated must be added together to obtain the multiplication result.

4.3.1.4. CARRY-SAVE MULTIPLIER

A first implementation of this partial product reduction tree uses the Carry-Save arithmetic as shown in Figure 91. With this structure, the carry bits resulting from each partial product are propagated vertically along with the sum bits to the next partial product stage. Each partial product stage thus receives the results of the sum of all previous partial product stages and adds its own partial product result to it, before propagating it to the next stage. The final partial product

is then added to the resulting sum of the previous stages, by using a ripple carry adder. This structure offers many advantages in terms of timing by reducing the length of the critical paths. It can also be efficiently pipelined to increase the maximum operating frequency. However, while this structure can be efficiently implemented on ASIC, its implementation on FPGA will results in poor timing performances as it cannot benefit from the dedicated carry propagation circuitry embedded in the logic blocks. To improve timing performances, pipeline stages can be inserted. Pipelining consists in inserting flip-flops on the signal propagation path to decrease the propagation delays and thus increase the maximum operating frequency at the expense of an additional clock cycle latency. The insertion of pipeline stages imposes a delay of one clock cycle on the signals, the flip-flops must then be inserted on all the signals used in the calculation to synchronize the data coming from the same input vector. Adding a pipeline stage induce a clock latency between the input and output of the circuit.

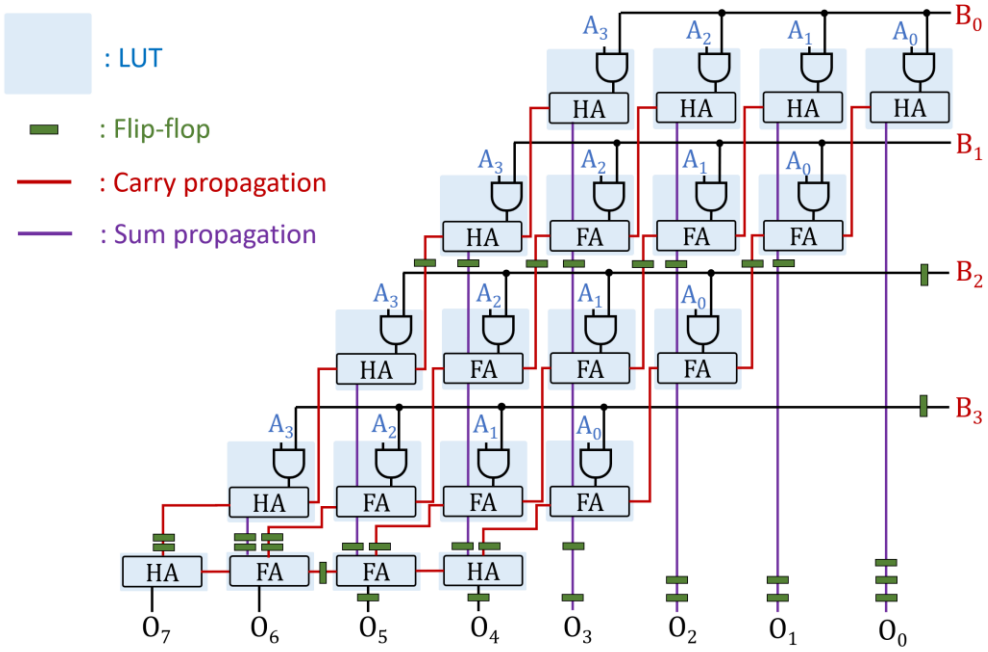


Figure 91: **Carry-Save Multiplier (CSM)** architecture. The carry and sum results of each partial product are propagated vertically to the next partial product and integrated in the computation of their own sum and carry bits. This structure can benefit from efficient pipelining by inserting register in the middle of the critical path.

4.3.1.5. SPEED OPTIMIZED MULTIPLIER

To efficiently use of the resources dedicated to the carry propagation, the most direct implementation of the multiplier consists in reducing the partial products by a series of binary carry look-ahead adders as shown in Figure 92. The adjacent partial products are first combined two by two, and then the resulting sums are combined again two by two. This process is repeated until the final result is obtained. The first adder stage can take advantage of the LUTs structure by integrating the calculation of the two partial products (AND gates) to be added and thus mobilizing 4 inputs. This partial product reduction tree is the most efficient in terms of timing, considering that pipeline stages can also be inserted between the different addition stages to further increase the maximum operating frequency. However, this implementation consumes a large number of logical resources.

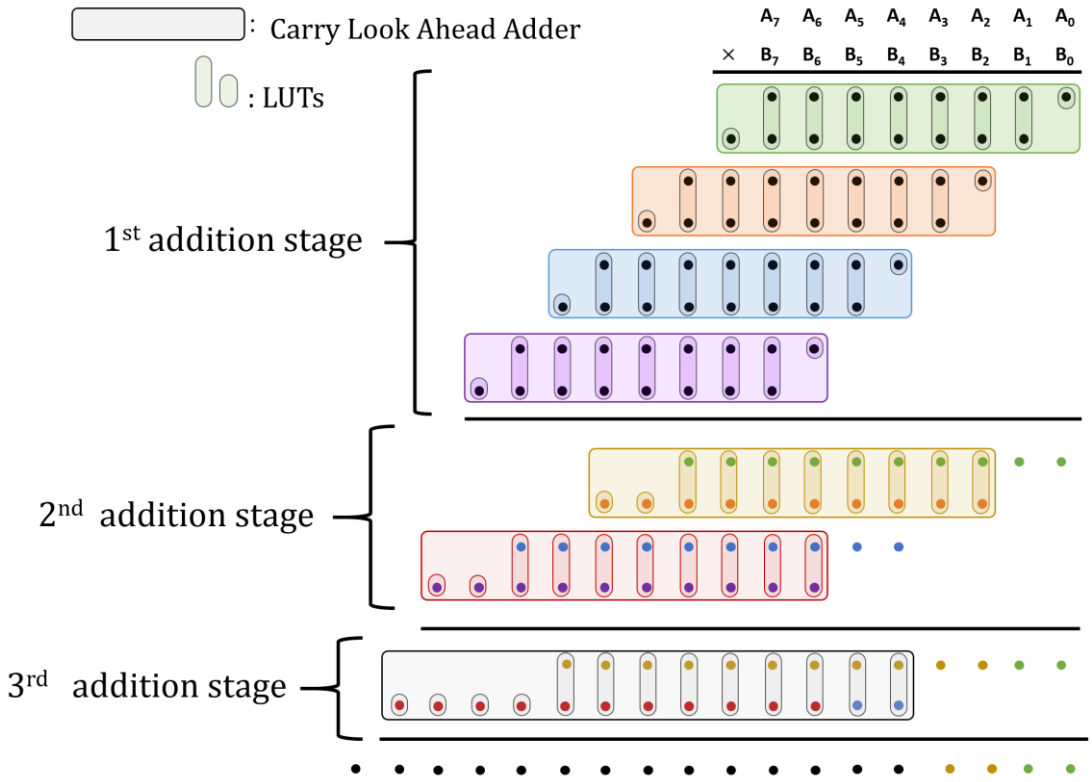


Figure 92:Dot diagram of a **speed optimized** 8x8 **Carry Look Ahead Multiplier**. The 1st addition stage integrates the computation of the partial products and their addition two by two. The results of these additions are then added two by two until the final result is obtained.

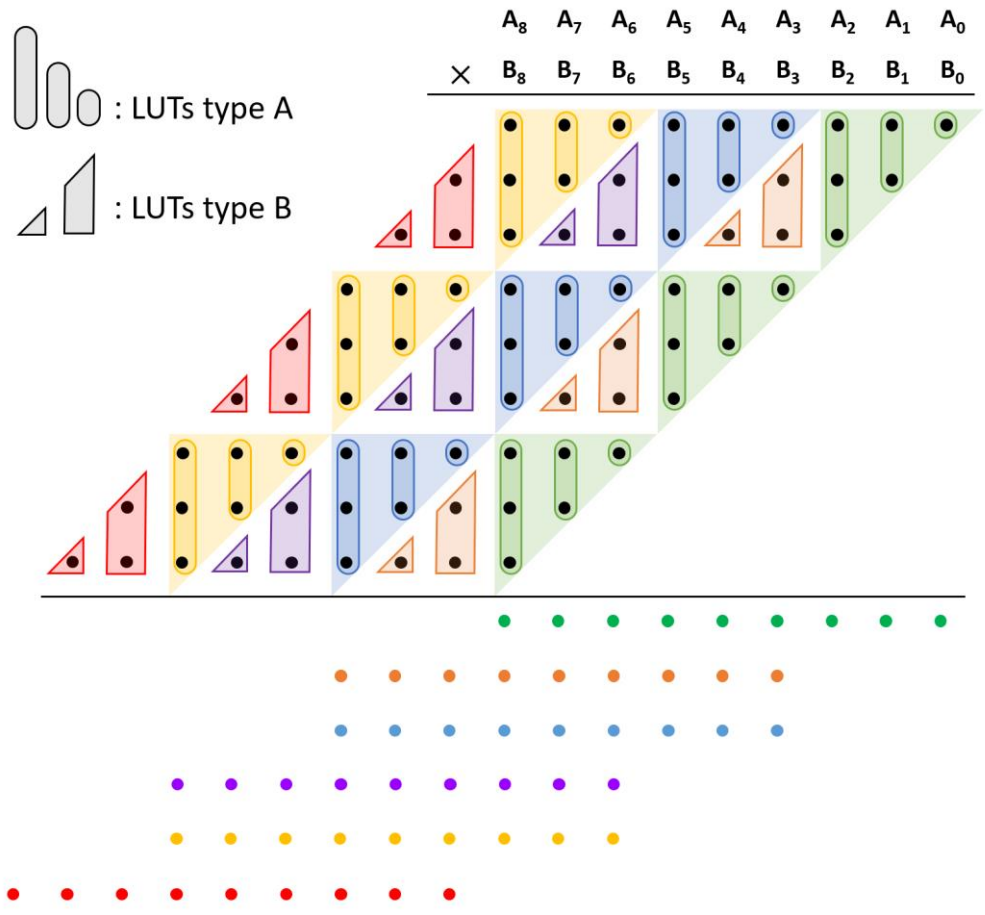


Figure 93: Dot diagram example of *area optimized* 9×9 multiplier partial product reduction tree.

4.3.1.6. AREA OPTIMIZED MULTIPLIER

For FPGA architectures with six input LUTs, the number of resources used by the multiplier can be reduced by using an implementation that takes advantage of the six inputs. With this architecture, the bits of the partial products are first reduced with a stage of optimally arranged LUTs as depicted in Figure 93. One part of LUTs (Type A) only computes the SUM bits of partial product over 1, 2 or 3 bits of the same weight ignoring the carry bit. The other part of LUTs (type B) compute the SUM and CARRY bits of 1 or 2 bits of the same weight while integrating the carry bit from the sum of the lower weight bits sharing the same input bits. The vectors extracted diagonally from this compression stage are then added 3 by 3 using ternary adders taking advantage of the carry chain architecture until the final result is obtained. To be noted that this architecture cannot be implemented on FPGAs that only integrate 4-inputs LUT such as the Igloo2 FPGA (*Microsemi*) tested in this study.

4.3.1.7. BOOTH ENCODING OPTIMIZATION

A well-established algorithm to reduce of the number of partial products called "Booth encoding" is presented in [142]. In a first step, the multiplier LSB is padded with one zero and its MSB with 2 zeros. The multiplier is then divided into overlapping groups of 3-bits as shown in Figure 94. For each group, a partial product scale factor is applied based on the booth encoding table (TABLE VII).

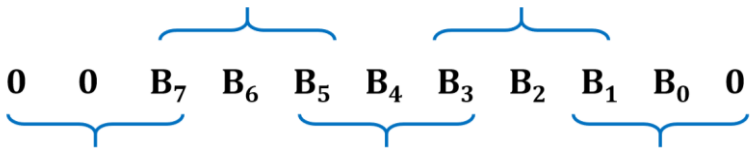


Figure 94: Multiplier padded with one 0 at LSB and two 0 at MSB and divided into overlapping groups of 3bits.

TABLE VII: BOOTH ENCODING TABLE

B_{i+1}	B_i	B_{i-1}	Scale Factor
0	0	0	+0
0	0	1	+A
0	1	0	+A
0	1	1	+2A (A<<1)
1	0	0	-2A (-A<<1)
1	0	1	-A
1	1	0	-A
1	1	1	-0

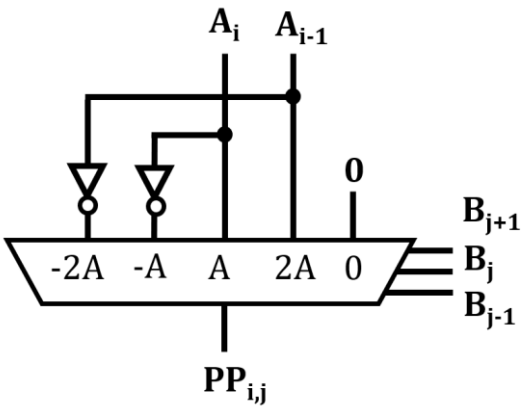


Figure 95: Modified Booth multiplier.

Only one product table is generated for each group with the corresponding scale factor (in this example, five partial products are generated instead of eight). Each partial product bit $PP_{i,j}$ can be computed using a five-input gate as shown in Figure 95.

Base on this approach, a multiplier merging the booth recoding scheme of a booth multiplier and the summation of the partial products in a single stage of LUTs and Carry chains association was

proposed in [140]. This implementation can be used on FPGA architectures with 6 inputs LUT, showing a significant reduction in the number of instantiated resources and an improvement in performance.

4.3.1.8. CONSTANT MULTIPLIER OPTIMIZATION

Some applications use multiplications with a constant number that does not need to be updated during execution. The hardware implementation of such multiplier can then benefit from optimization to reduce resource usage and increase performances. Indeed, considering a multiplication of a multiplicand A by a constant multiplier B, the result is simply obtained by adding, for each nonzero digit of B, a copy of A bit-shifted according to the index of the digit. All partial products corresponding to zero digits in B are simply omitted. When B contains only a few nonzero digits, the number of additions is thus greatly reduced as shown in Figure 96.

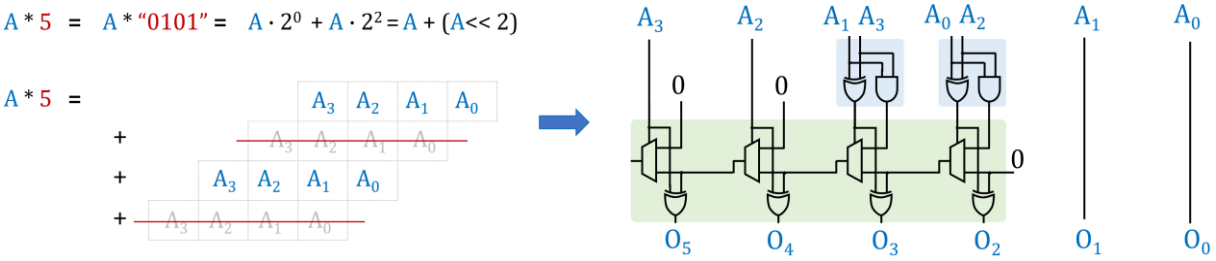


Figure 96: Example of constant multiplier optimization. A is multiplied by 5, which correspond to the addition between A and 2bits-shifted A. This multiplication can be efficiently implemented using only two LUTs and a CARRY4 gate.

Depending on the multiplier value, the implementation of constant multiplication can be further reduced by combining additions and subtractions. For example, if a 4-bit coded multiplier B is equal to 7 (=b"0111"), instead of using 2 additions of A bit-shifted by 0,1 and 2 bits, the result can be computed with a single subtraction between A bit-shifted by 4 bits and A. This optimization is illustrated in Figure 97.

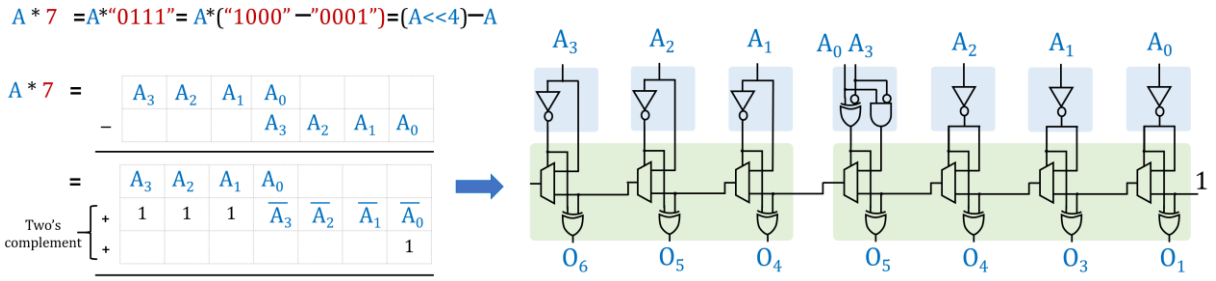


Figure 97: Example of constant multiplier optimization replacing 2 additions by one subtraction. This subtraction is implemented using two's complement representation and efficiently implemented using 7 LUTs and 2 CARRY4 gates.

4.3.1.9. SELECTED MULTIPLIER IMPLEMENTATIONS

Many other approaches have been developed to optimize the implementation of arithmetic operators on FPGA [143]–[146]. However, the performance criteria used to compare these implementations are essentially focused on the maximum operating frequency and the number of instantiated logical resources while the susceptibility to SEEs is not investigated. By integrating several implementations of the same arithmetic function in the benchmark and comparing their

sensitivity, the test methodology fulfills an additional objective providing designers with quantitative information to choose the type of implementation adapted to their needs.

In this study, a total of seven multiplier structures is selected to be included in the benchmark:

- **CSM: C**arry **s**ave **m**ultiplier (section 4.3.1.4)
- **SOM: S**peed **o**ptimized **m**ultiplier (section 0)
- **AOM: A**rea **o**ptimized **m**ultiplier (section 4.3.1.6)
- **KUM:** implementation proposed by M. **Kum**m in [140]
- **PAA:** implementation proposed by H. **Pa**randeh-**A**fshar in [143]
- **DSP:** instantiation in hardcoded **d**igital **s**ignal **p**rocessing blocks (section 2.2.4)
- **CST:** Multiplier taking advantage of **c**on**st**ant multiplication optimization

This group of multipliers was chosen to maximize the diversity in the circuit's architecture regarding the type of instantiated resources (LUTs of different sizes, DSP, CARRY, flip-flops) as well as the structure (fan-in and fan-out of the logic gates, number of logic stages, propagation time between the flip-flops, routing scheme). For instance, **CSM** use mainly 4-inputs LUTs with a vertical propagation of carry and no use of the dedicated carry logic; **SOM** and **CST** uses a sequence of binary carry look-ahead adders (horizontal propagation of carries) benefiting from dedicated carry logic; **AOM** use a first stage of LUT-based compressors and a second stage of ternary adders; **KUM** and **PAA** tries to exploit the resources of each logic block to the maximum by compressing the number of partial products calculated in the same stage while **DSP** use specific logic resource dedicated to arithmetic operations.

So that the differences between these implementations are clearly revealed while still providing high maximum operating frequencies, 16x16bits multipliers are selected in this study.

4.3.1.10. FINITE IMPULSE RESPONSE FILTER

The difficulty in monitoring the operation of a multiplier and detect the presence of errors lies in the size of the signals to be analyzed. The multiplication of two *N* bits long numbers provides a result coded on *2N* bits (multiplication without truncation). The size of the circuitry required to monitor each multiplier independently would be prohibitive as it could compromise the results of the experiment by an excessive false error detection rate. To overcome this issue, a Finite Impulse Response (FIR) filter structure has been adopted.

A FIR filter is a digital filter characterized by a response based only on a finite number of values of the input signal. As shown in equation (12), the output $y[n]$ is calculated by performing a discrete convolution between the input signal $x[n]$ and a function represented by the coefficients b_k corresponding to the discrete impulse response of the filter.

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k] \tag{12}$$

This convolution can be implemented mainly in two different ways, the direct form (Figure 98) and the transposed (Figure 99).

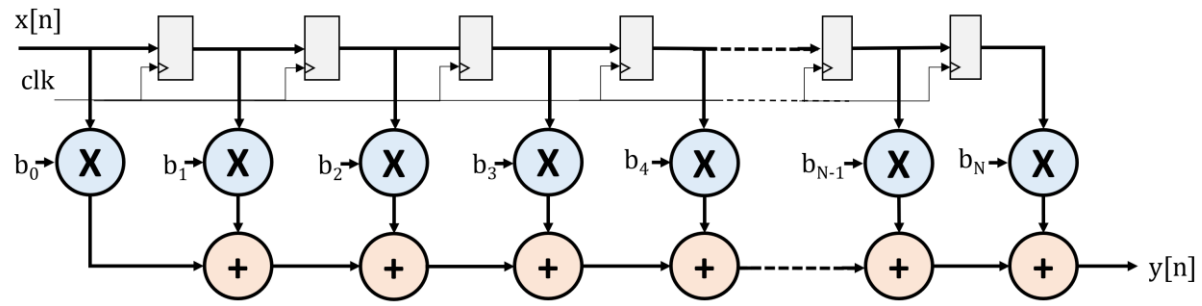


Figure 98: Direct form of an FIR filter. The input signal $x[n]$ pass through a sequence of register. Each delayed input signals ($x[n-k]$) is multiplied to the corresponding coefficient b_k. All the multiplication results are then added simultaneously to produce the output result.

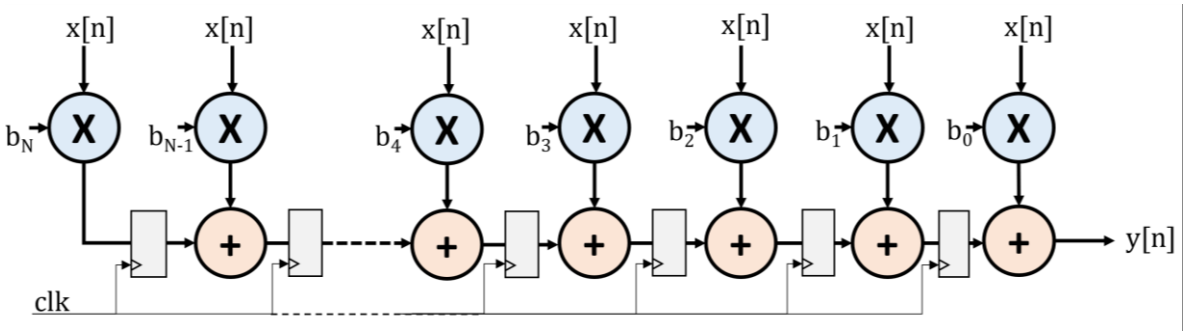


Figure 99: Transposed form of an FIR filter. The input signal $x[n]$ is simultaneously multiplied to each coefficient b_k. The output of each multiplier is then summed with the higher order multiplication result delayed by one clock cycle.

Each implementation brings its own pros and cons. The direct implementation allows to limit the fan-out of the input signals because the signal is only distributed to one multiplier and to the flip-flop stage. However, this implementation may suffer from timing penalties in the adder sequence because all multiplier outputs are added simultaneously. The addition of multiple vectors in parallel can be optimized with one of the reduction tree techniques described in previous sections, and pipelining can be inserted to further reduce the delay of the critical paths at the expense of an increased latency. On the other hand, the transposed form does not suffer from timing penalties on the adder sequence because each addition is separated by a flip-flop stage. However, the input signal is distributed simultaneously to each of the multipliers which introduces a very large fan-out. Indeed, at the multiplier level, the input signal is distributed to a large number of cells to perform the calculation of each partial product. If the input of the multipliers is not registered, the fan-out on this signal can be penalizing in terms of timing and routing capabilities.

The FIR filter structure offers many advantages for SEE testing. First, the forward propagation of signals without feedback loops avoids the presence of persistent errors. Once the source of the error is corrected, the output of the filter returns to its theoretical value after a maximum number of clock cycles defined by the number of register stages separating the input and output of the filter. This characteristic is very valuable because it allows to determine the error duration and thus to identify the involved failure mechanism. Second, the FIR filter structure allows multiple multipliers to be combined together with the low error masking rate provided by adders. The vast majority of errors generated within the structure can thus be detected by monitoring only the output of the filter. Finally, this circuit is perfectly in line with the benchmark concept as this type of architecture (convolution) is commonly used in a wide range of applications (digital signal processing, neural networks, etc.).

The benchmark proposed in this study not only deal with testing the different multiplier implementations using a single filter structure but is also supplemented with more FIR filter variants to evaluate the influence of structural parameters on the circuit sensitivity. In particular, a filter version using distributed arithmetic is considered.

4.3.1.11. DISTRIBUTED ARITHMETIC

Distributed arithmetic (DA) is a computation algorithm that performs multiplication using precomputed lookup tables instead of logic [147]. DA targets the sum-of-products operation such as filter implementation and matrix multiplication. DA can be particularly well suited to implement FIR filters when constant (non-updatable) coefficients are used.

When applied to FIR filter and considering a fully parallel implementation, the main idea behind distributed arithmetic is to decompose the computation of the sum-of-product into different computation units carrying the computation for a single bit of the input signal.

Formally, considering a fixed-point positive integer, the input signal $x[n]$, a binary vector of Wbits can be represented as shown in equation (13).

$$x[n] = \sum_{i=0}^{W-1} x_i[n] \cdot 2^i \tag{13}$$

The equation of an FIR filter with Ncoefficients can be thus be decomposed in a sum of bit-shifted sums of 1 by N products as described in (14).

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k] = \sum_{i=0}^{W-1} 2^i \cdot \left(\sum_{k=0}^{N-1} b_k \cdot x_i[n-k] \right) \tag{14}$$

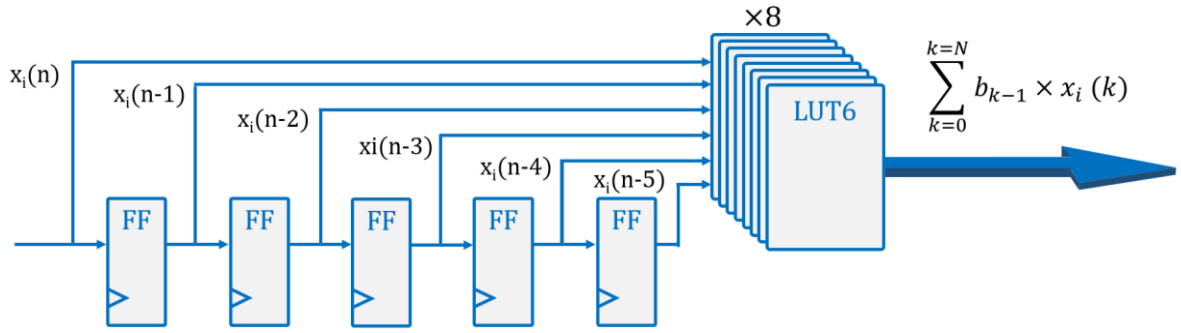


Figure 100: Example of one input bit computation. The input signal bit is delayed in a shift-register and transmitted to a set of LUTs carrying the precomputed addition of the input coefficients.

The calculations inside the brackets of equation (14) can be computed independently using look-up tables. Indeed, the coefficients being constant, this result is computed by adding the coefficients for which the input signal bit is at '1'. Considering the example of an FIR filter with 6 coefficients and an 8bits input signal and coefficients, a total of 64 possible combinations of coefficients must be precomputed. The precomputed results are then stored on 8 LUTs with 6 inputs as shown in Figure 100.

This structure is replicated for each input signal bit and the results are finally bit shifted according to the input bit position and summed together as shown in Figure 101.

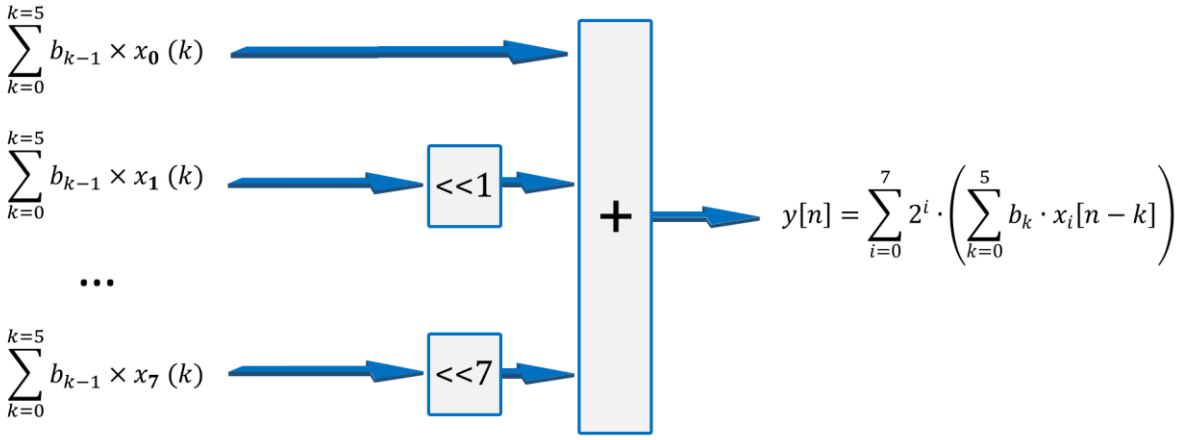


Figure 101: Distributed arithmetic implementation. The computation results for each input bit are bit shifted and added.

4.4. Built-in self-test

As mentioned earlier, particle beam testing imposes many technical constraints related to the radiation exposure of all the equipment installed in the irradiation room and to the length of the cables that usually connect the DUTs and the equipment installed in a radiation-free area. Test setups based on an auxiliary system for injecting stimuli and monitoring FPGA output signals [148] require additional engineering time (especially when using a custom PCB) while their reliability is not guaranteed when placed close to the DUTs. For these reasons, a Built-In Self-Test (BIST) approach was adopted in this study. However, its design imposes other types of constraints. In one hand, the circuitry used to inject the test signals and to monitor the output signals of the circuit under test must be sufficiently reliable with respect to the sensitivity of the circuit under test itself to limit the false error rate (error detection induced by a SEE on the test circuitry). This testing circuitry must therefore be as SEE immune as possible by limiting the number of instantiated resources and preferably hardened with mitigation schemes. On the other hand, the data rate sent by the component is limited by the type of interface and the length of the cables connecting the irradiation room and the radiation-free area. Therefore, the error detection should be carried out internally and the error reports sent to the control computer must be judiciously formatted.

As mentioned previously, in order to group several instances of multipliers in a single structure and limit the number of signals to be monitored, a FIR filter structure is used. In this section, the choice of the test patterns injected in the structure are first discussed. Then, the architecture used around the filter to inject the input signals, monitor the output signals, detect, format and transmit the errors are detailed. Finally, the test setup and procedures used during the experiments are presented.

4.4.1. Test pattern selection

To detect the presence of errors in a circuit, the standard approach is to inject a set of test patterns into the input and check that the outputs respect the expected theoretical values. The

choice of test patterns to be injected into the circuit has a major impact on fault coverage, namely the percentage of faults that can be detected during the test. Many tools have been developed for Automatic Test Pattern Generation (ATPG) [149], [150]. These tools aim at automatically generate the shortest possible test patterns for a given fault coverage. Based on circuit analysis, the fault model used by these tools is generally focused on hardware structural defects such as stuck-at-fault (one net is stuck either at logical '0' or '1') or bridge faults (a short circuit between two signal lines). Thus, these test techniques, devised for ASICs, are not effective in detecting SEEs in FPGA based system, especially regarding SEUs in the configuration memory [151], [152]. A test pattern generator dedicated to SEU's in the configuration memory of FPGA have been proposed in [153] based on a fault model developed by the same authors [154] targeting *Xilinx* Virtex II FPGA. The detailed fault model (discussed in chapter 5) is specific to each FPGA architecture and has not been established for the FPGAs tested in this study. Therefore, in this study, ATPG tools are not used for test pattern selection. Instead, the test patterns are chosen using a simpler and more intuitive approach applicable to FIR filters. The test vectors for the input signal must propose a large diversity of values over the entire signal amplitude while trying to maximize the alternation and balance of the number of bits in the '0' and '1' logic states. According to this concept, a test pattern composed of 128 test vectors representing a complete period of a sinusoidal signal with the maximal amplitude has been chosen as illustrated in Figure 102. The filter coefficients have been selected to implement the behavior of a low-pass filter. Improvements have been made through the different test campaigns by removing for example the coefficients equal to 0 (which are usually removed on a real application) or by normalizing the sum of the coefficients to 1 (sum of the coefficients $= 2^N$-1, N being the number of bits of the input vector) to avoid the bit growth on the addition stages.

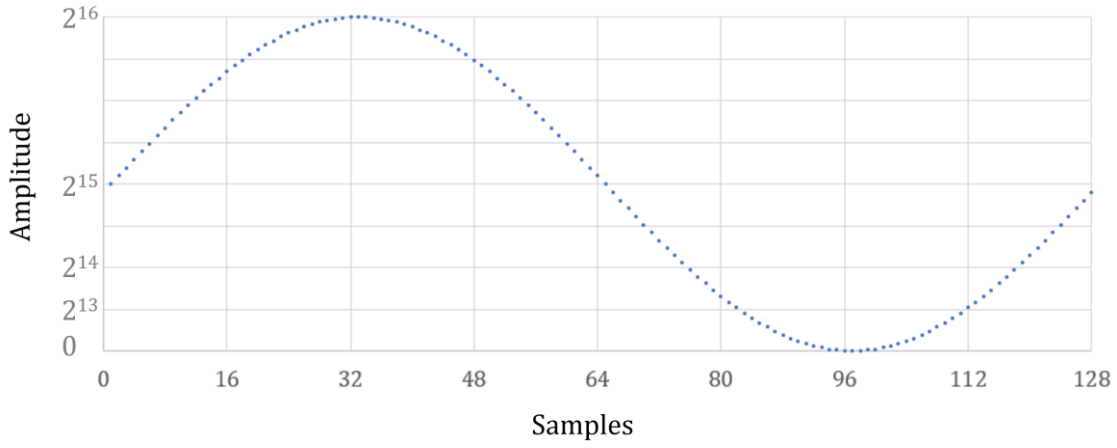


Figure 102: Input test pattern. 128 samples of 16bits representing one period of a sinusoidal signal.

4.4.2. BIST ARCHITECTURE

To detect the presence of errors in the FIR filter, the output signal is continuously compared to a golden reference previously extracted from a behavioral simulation. The architecture used for the built-in self-test is presented in Figure 103.

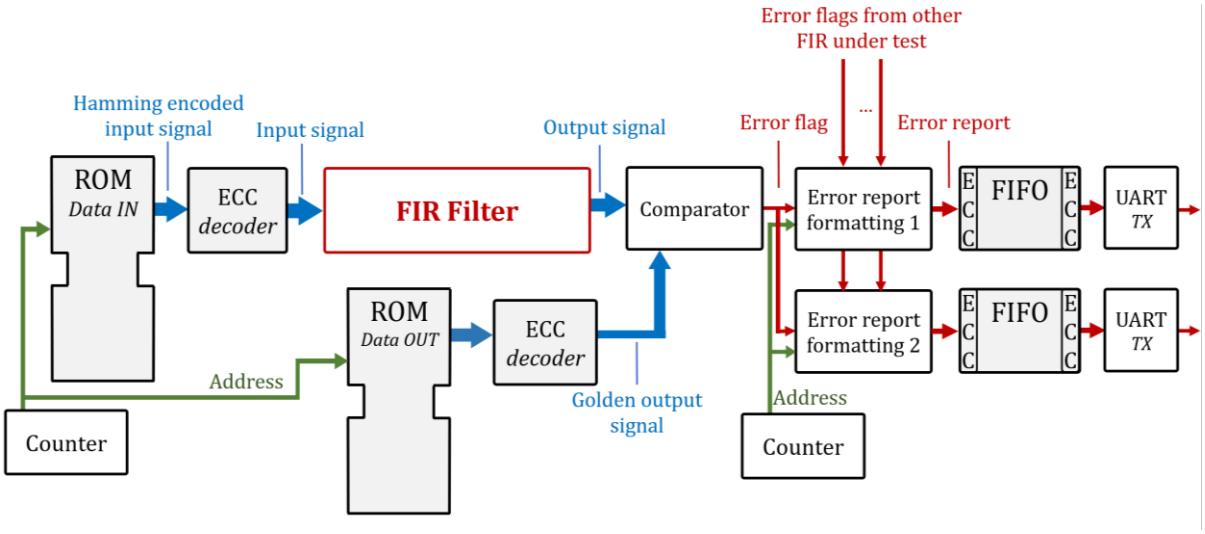


Figure 103: Built-in self-test structure implementation. Memory blocks protected by ECC are used to generated input stimulus and output golden signals. The FIR output is compared with the golden signal to detect the presence of errors. The error flags raised by the comparator are transmitted to an error report formatter which generates and transmit the error reports to an external computer through a serial interface (these last modules are duplicated with two different types of formatting).

The input test pattern and the output golden reference are stored in BRAMs configured as Read Only Memory (ROM). A counter ranging from 0 to 127 is used as an address generator for the two memory blocks. To avoid false error detection induced by bitflips in these memory blocks, the stored data have been previously encoded with ECC (Hamming) and the output of the memory blocks are protected by an ECC decoder with Single Error Correction and Double Error Detection (SEC-DED) capability. The filter output is compared to the golden signal to detect the presence of errors. When an error is detected, an error flag is raised which triggers the sending of an error report to a control computer through a serial link (UART protocol). To avoid false error detections induced by SEE in the report formatter and UART transmission link, these modules are duplicated while using two different types of formatting. Hence, error induced in one of the duplicated modules can be detected and eliminated on the control computer side.

4.4.3. Error formatting

The bandwidth of a the UART links is limited by the length of the cables and by the data acquisition and processing capacity of the control computer, the error reports must then be formatted carefully to limit the number of bytes to transmit while providing enough information to identify the affected filter and the type of error mechanism involved. Two distinct error formatting mechanisms have been developed. The first consists of concatenating a byte containing the indexes of the affected filters and a byte representing the temporal position of the error (value of the counter). During the first experiment, this formatting showed some limitations. Indeed, when the error duration was too important, the number of bytes sent per second could not be managed and an important loss of data was observed. This loss apparently originates from an overflow of the buffer at the USB controller level. This error formatting thus prevents the appropriate measuring of the error duration. To overcome this limitation, a second formatting system was added. This formatting system accumulates the number of errors detected for each filter during one second and then transmits for each one of them the content of the accumulator along with the index of the affected filter.

In the absence of errors, a dedicated frame is sent periodically to ensure that the circuitry dedicated to error formatting and data transmission by UART is still operating and has not been interrupted by a SEE.

4.4.4. TEST SETUP AND PROCEDURE

In this chapter, three *Xilinx* FPGAs (Spartan7, Artix7 and Kintex7) and one Flash FPGA from *Microsemi* (Igloo2) are tested. When performing SEE tests on *Xilinx* FPGAs, the SEM controller (Soft Error Mitigation presented in section 4.1.2) is systematically integrated into the implemented design. This system enables the correction of the majority of the SEUs in the configuration memory and thus prevents the accumulation of errors. If the event is corrected, the test can continue without having to reconfigure the component. On the other hand, from the SEU detection messages reported through the SEM controller serial interface (UART), the cross section of the configuration memory can be calculated. By cross-checking the information reported by the SEM controller with the error reports from the BIST structure, the type of failure mechanism can also be identified.

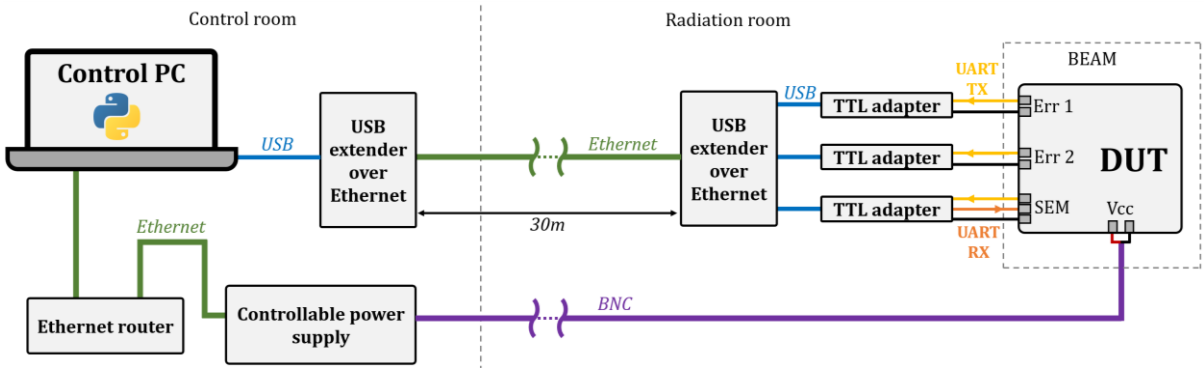


Figure 104: Test setup. Three serial links are reported to the control PC through TTL adapters and USB expanders over Ethernet. The supply voltage is provided through a BNC cable by a power supply remotely controlled by the host computer.

A total of three serial links per device are reported to the control computer (two from the Built-in self-test structure and one from the SEM controller). The UART signals from the FPGA are converted to USB using one TTL adapter for each serial link. Due to the length of the cables connecting the irradiation room and the control room (~30m), the USB bus cannot be used directly. A USB expander over Ethernet is used to transmit the data through an RJ45 cable. This expander is composed of two modules, one installed in the irradiation room receives the data from three USB links provided by the TTL converters and transmits them on the ethernet link. The other one, installed in the control room, transfers the data sent on the Ethernet back to a USB which can be directly connected to the control PC as shown in Figure 104. This system acts transparently to the user and everything behaves as if the three USB cables were directly connected to the computer.

The USB expander module and TTL adapters located in the irradiation room are placed as far as possible from the DUTs to limit their exposure to radiation. These elements are also placed behind thick paraffin bricks (for neutron tests) or lead bricks (for proton tests) to further limit their exposure.

The supply voltage is provided through a BNC cable and is generated by a power supply remotely controllable by the control PC through Ethernet. The remote control allows the PC to automatically perform power cycles to force the FPGA reconfiguration. The power supply also has a current limiter to avoid potential damage caused by SEL.

The experiment is fully controlled and automated through a python script. Data acquisition from each serial link is managed in an independent process using a different processor thread. These processes take care of the recording of the data sent in a text file for further processing of the data. These processes also check that the communication is still established (via specific frames sent periodically and with a 5s stopwatch) and detect the presence of permanent errors (when the error report flow is continuous during more than 5 seconds). When one of these two events occurs, a power cycle is triggered to force the FPGA reconfiguration as shown in Figure 105.

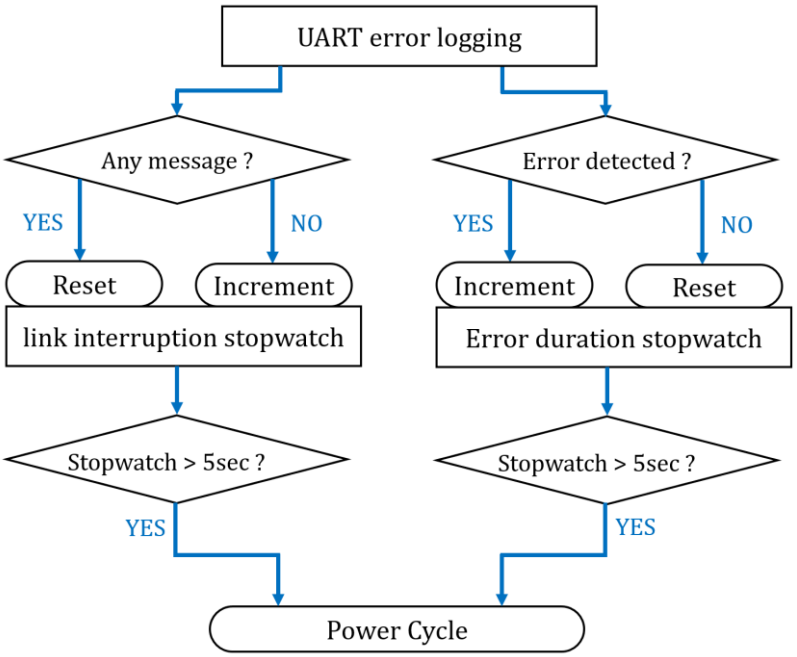


Figure 105: Process of UART frame acquisition and detection of permanent errors and link interruptions. When the error flaw is continuous or no message is received during more than 5 seconds, a power cycle is triggered to force the FPGA reconfiguration.

4.5. EXPERIMENTAL RESULTS

A total of three SEE test campaigns has been performed in this study. Each new campaign completes the benchmark with new structures, whether by proposing new multiplier structures, new placement and routing strategies, or by experimenting different design parameters in the FIR filter architecture. In addition, each campaign has benefited from the lessons learned in previous campaigns by making different improvements to the BIST architecture and test setup as described earlier.

In this chapter, the results of the three SEE test campaigns are presented successively. The two first campaigns are performed in the ChipIR facility (UK) presented in section 1.1.3.3. As a reminder, this facility provides a neutron beam with an atmospheric like spectrum and with a neutron flux of $5 \cdot 10^6\ cm^{-2}s^{-1}$ (for neutron energy higher than 10MeV). The third campaign is

performed at PARTREC facility (Netherlands) providing a proton beam with energy up to 180MeV at DUT position and a controllable flux up to $1 \cdot 10^8 cm^{-2} s^{-1}$.

4.5.1. First campaign: influence of timing constraints

This first campaign aims to evaluate the predominance of the different failure mechanisms as well as to evaluate the influence of the operating frequency imposed during the design phase on the radiation sensitivity of different multiplier implementations.

Indeed, the synthesis, placement and routing algorithms are driven by the user defined timing constraints. To respect these constraints, these algorithms use different optimization techniques: for example, when the user imposes a high operating frequency, the placement algorithm tries more aggressively to physically place the connected cells closer together and the routing algorithm further explore several PIP associations to find the one that minimizes the delay of each net. Conversely, by imposing a low operating frequency, the algorithm tries to minimize the number of logical blocks to be used which may result in a more compact implementation but with longer propagation delays. One of the objectives is thus to analyze whether these differences in implementation have an impact on the radiation sensitivity.

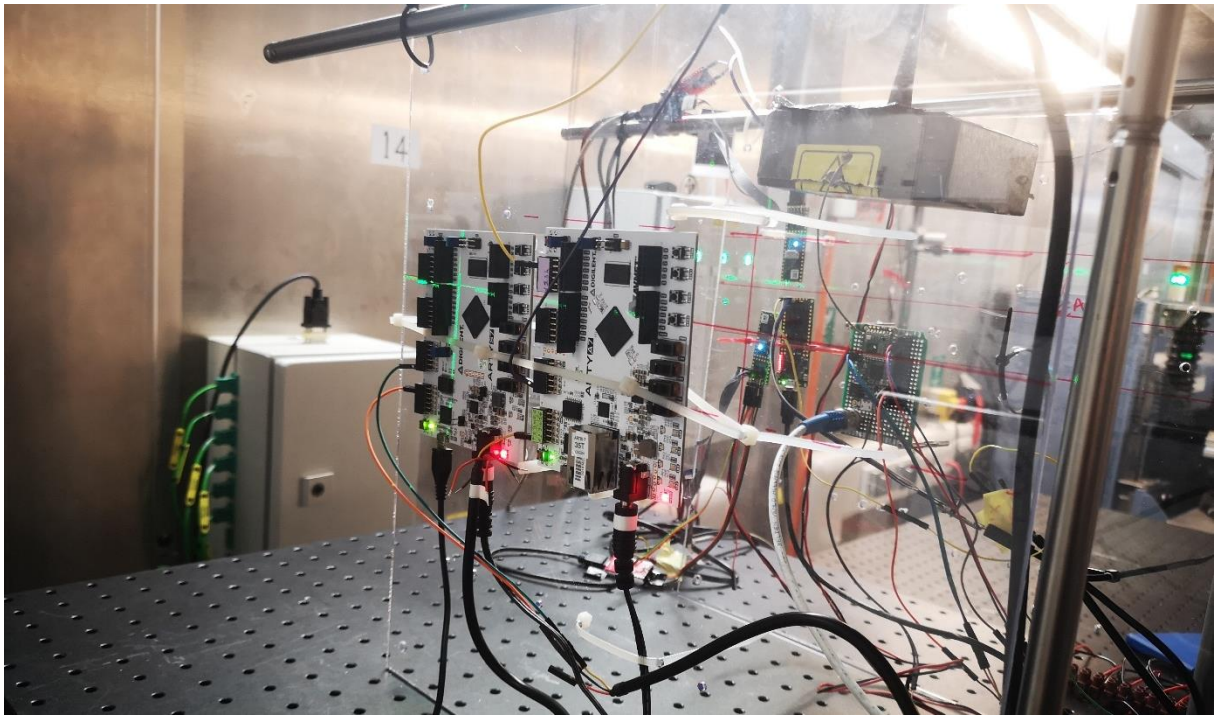


Figure 106: Test setup at ChipIR facility. The two FPGA boards are mounted on a plexiglass board and positioned vertically in the neutron beam spot.

4.5.1.1. Setup details

Tests are performed on two **Spartan7** (SRAM - 28nm) from *Xilinx* embedded on commercial development boards Arty S7-50 from *Digilent*. The test setup in the irradiation room is depicted in Figure 106.

Three multiplier implementations presented in section 4.3 are evaluated:

- **CSM:** Carry Save Multiplier
- **SOM:** Speed optimized Multiplier

- **DSP:** instantiation in hardcoded **D**igital **S**ignal **P**rocessing blocks

Each multiplier is fully pipelined and integrated in a FIR filter in transposed form with 9 coefficients, 16 bits width vector for data input and coefficients. As the coefficients are not normalized in this setup, the full precision along the filter calculations was maintained resulting to a 40-bits width output vector. These filter dimensions were chosen to offer a good compromise between the number of resources instantiated to verify correct operation of the filter and the number of replicas of the circuit that can be integrated in the chip (3 replicas for CSM and SOM, 8 replicas for DSP). Each structure is tested with four different operating frequencies: **10MHz, 50MHz, 100MHz and 200MHz**. This results in a total of 8 designs to be tested as depicted in TABLE VIII.

TABLE VIII
SUMMARY OF THE DIFFERENT TESTED DESIGNS

Run	Composition		Operating frequency
n°1	3 **CSM** filters	8 DSP filters	**10** MHz
n°2	3 **CSM** filters	8 DSP filters	**50** MHz
n°3	3 **CSM** filters	8 DSP filters	**100** MHz
n°4	3 **CSM** filters	8 DSP filters	**200** MHz
n°5	3 **SOM** filters	8 DSP filters	**10** MHz
n°6	3 **SOM** filters	8 DSP filters	**50** MHz
n°7	3 **SOM** filters	8 DSP filters	**100** MHz
n°8	3 **SOM** filters	8 DSP filters	**200** MHz

By carrying test at different frequencies, the SET capture phenomena and SEU propagation between registers can also be analyzed. In this first BIST implementation, the BRAM storing the input signal and output signal where common to all FIR filters. As the input signals are distributed to all the multipliers of every filter, these nets had to be duplicated and registered to reach the highest operating frequency.

To be noted that due to technical issues, the SEM controller reports could not be recorded during this campaign, so the bit cross section of the configuration memory could not be extracted. This cross section is extracted through the next irradiation campaigns.

4.5.1.2. RESULTS

During this campaign, the FPGA are irradiated for 27 hours with an average neutron flux of $5.6 \cdot 10^{6} cm^{-2}.s^{-1}$obtaining a total fluence of $5.5 \cdot 10^{11} cm^{-2}.s^{-1}$. Over the entire irradiation period, a total of 1105 errors is observed. Based on error reports send by the BIST structure, the exact timing of the errors in the data stream passing through the filter can be determined. The resulting errors are classified into three categories based on their footprint as demonstrated in Figure 107.

Non-persistent upsets are due to the corruption of one or more configuration bits related to the configuration of the implemented circuits (affecting either the routing resources, the configuration of LUTs, or DSPs). These bit flips are corrected by the SEM IP after a certain amount of time directly linked to the detection and correction latency of the scrubbing system. These configuration memory related failure modes are further discussed in chapter 5.

Persistent upsets show a similar behavior to the non-persistent ones, with the difference of not recovering until the FPGA is fully reconfigured. Since the test structure does not contain any feedback loops, the structure should resume normal operation after the configuration bit responsible for the error has been corrected. The persistence of these errors is caused by different mechanisms related to the proper functioning of the SEM controller.

During the experiment, the SEM IP was configured in "*enhanced repair*" mode allowing the correction of a single bitflip per frame or double adjacent bitflips. However, non-consecutive multiple errors per frame cannot be corrected. Likewise, some configuration bits can generate multiple errors that cannot be corrected by the SEM controller. When the SEM controller is unable to correct an error, it goes back to an IDLE state, thus preventing the correction of subsequently generated errors. As the SEM IP is also sensitive to SEUs, its operation can be compromised thus preventing the correction of subsequent SEUs impacting the design. All these mechanisms leading to persistent errors are further studied through fault injection in section 4.6.

Single errors, although very few, are the result of flip-flop SEUs or SET capturing. This assumption is supported by the frequency dependency of the cross section for all filter types. As the number of single errors observed is rather low and overwhelmed by the event induced in the configuration memory (98%), it is difficult to take advantage of the benchmark properties to clearly identify the predominant phenomena between the capture of SETs and the generation of flip-flop SEUs.

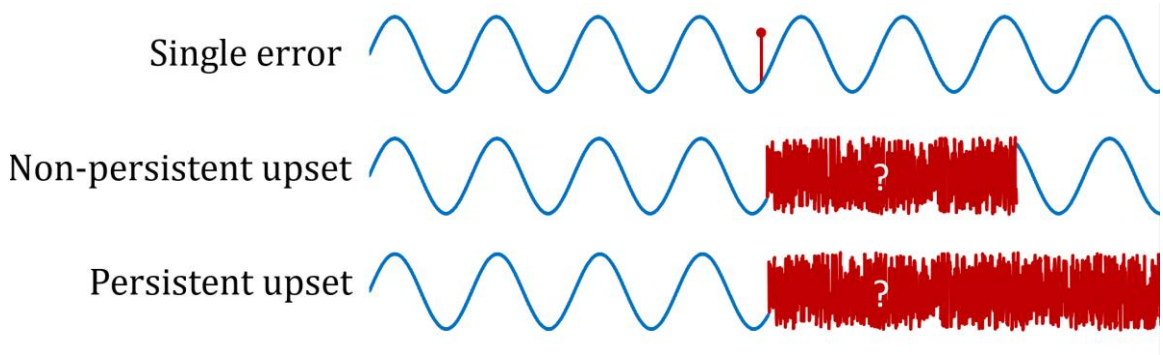


Figure 107: Classification of the detected errors. Single error: only one error is reported; Non-persistent upset: long frames of errors that recover automatically; Persistent upsets: long frames of errors that stop only after complete reconfiguration.

For each error type, the cross section for a single FIR filter is calculated as shown in TABLE IX. The cross sections for each multiplier implementation type as a function of the operating frequency are displayed in Figure 108, Figure 109, Figure 110 for each type of error (Non-persistent, Persistent and Single errors respectively).

TABLE IX
FIR FILTER CROSS SECTION RESULTS

	Frequency (MHz)	Cross section							Fluence (cm^{-2})
		Non-persistent		**Persistent**		**Single error**		Total	
		cm^2	%	cm^2	%	cm^2	%	cm^2	
SOM	**10**	$1.5 \cdot 10^{-10}$	*39.2*	$2.3 \cdot 10^{-10}$	*58.8*	$7.6 \cdot 10^{-12}$	*2.0*	$3.9 \cdot 10^{-10}$	$4.4 \cdot 10^{-10}$
	50	$2.5 \cdot 10^{-10}$	*60.3*	$1.6 \cdot 10^{-10}$	*38.1*	$6.5 \cdot 10^{-12}$	*1.6*	$4.1 \cdot 10^{-10}$	$5.1 \cdot 10^{-10}$
	100	$2.0 \cdot 10^{-10}$	*50.0*	$2.0 \cdot 10^{-10}$	*50.0*	0	*0.0*	$4.0 \cdot 10^{-10}$	$6.6 \cdot 10^{-10}$
	200	$2.6 \cdot 10^{-10}$	*56.1*	$1.7 \cdot 10^{-10}$	*36.4*	$3.5 \cdot 10^{-11}$	*7.6*	$4.6 \cdot 10^{-10}$	$4.8 \cdot 10^{-10}$
CSM	**10**	$5.4 \cdot 10^{-10}$	*65.3*	$2.9 \cdot 10^{-10}$	*34.7*	0	*0.0*	$8.3 \cdot 10^{-10}$	$4.9 \cdot 10^{-10}$
	50	$5.2 \cdot 10^{-10}$	*74.4*	$1.7 \cdot 10^{-10}$	*24.8*	$5.3 \cdot 10^{-12}$	*0.8*	$7.0 \cdot 10^{-10}$	$6.3 \cdot 10^{-10}$
	100	$4.0 \cdot 10^{-10}$	*60.3*	$2.2 \cdot 10^{-10}$	*33.3*	$4.2 \cdot 10^{-11}$	*6.3*	$6.7 \cdot 10^{-10}$	$8.7 \cdot 10^{-10}$
	200	$6.5 \cdot 10^{-10}$	*79.5*	$1.6 \cdot 10^{-10}$	*19.3*	$9.7 \cdot 10^{-12}$	*1.2*	$8.2 \cdot 10^{-10}$	$1.4 \cdot 10^{-11}$
DSP	**10**	$1.4 \cdot 10^{-12}$	*50.0*	$1.4 \cdot 10^{-12}$	*50.0*	0	*0.0*	$2.7 \cdot 10^{-12}$	$9.3 \cdot 10^{-10}$
	50	$2.1 \cdot 10^{-12}$	*66.7*	$1.1 \cdot 10^{-12}$	*33.3*	0	*0.0*	$3.2 \cdot 10^{-12}$	$1.2 \cdot 10^{-11}$
	100	$3.3 \cdot 10^{-12}$	*36.4*	$4.1 \cdot 10^{-12}$	*45.5*	$1.6 \cdot 10^{-12}$	*18.2*	$9.0 \cdot 10^{-12}$	$1.5 \cdot 10^{-11}$
	200	$5.4 \cdot 10^{-12}$	*47.1*	$3.4 \cdot 10^{-12}$	*29.4*	$2.7 \cdot 10^{-12}$	*23.5*	$1.2 \cdot 10^{-11}$	$1.8 \cdot 10^{-11}$
TOTAL		**64%**		**34%**		**2%**			

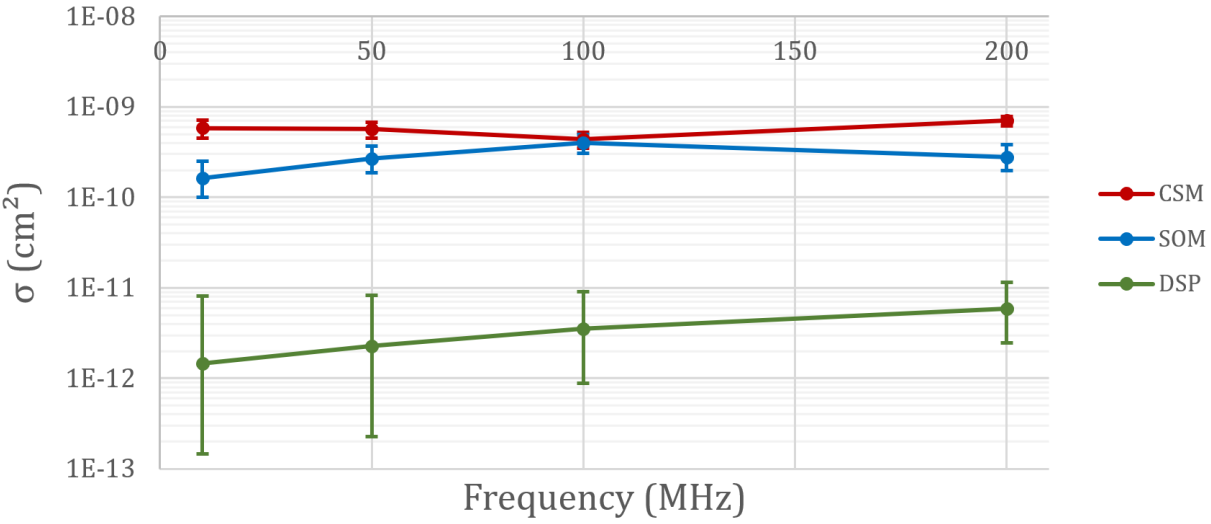


Figure 108: Cross section for **non-persistent errors** of FIR filters tested at different frequencies, displayed with the 95% confidence interval.

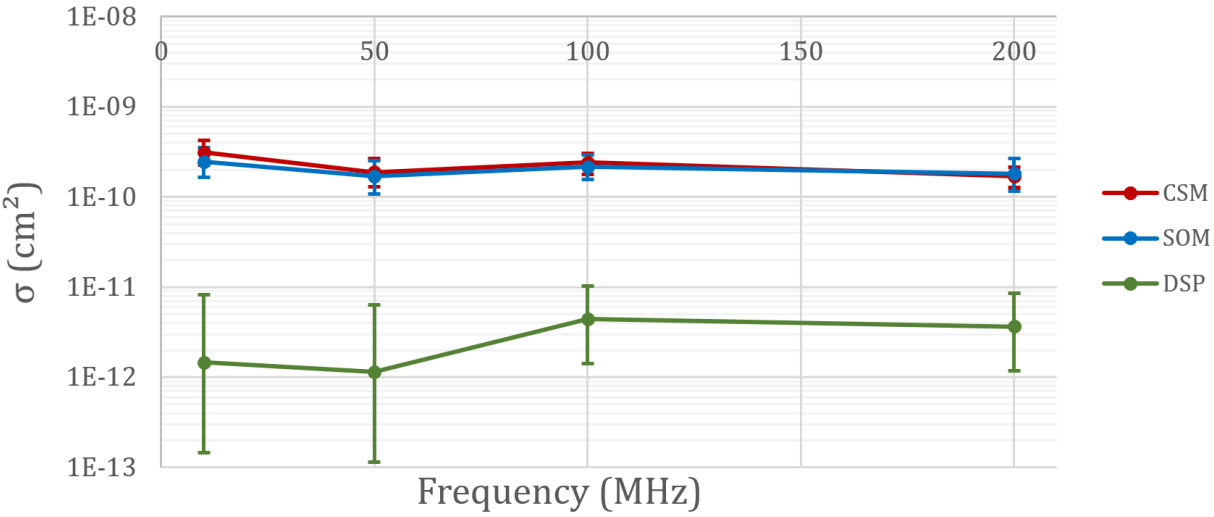


Figure 109: Cross section for **persistent errors** of FIR filter implementations tested at different frequencies, displayed with the 95% confidence interval.

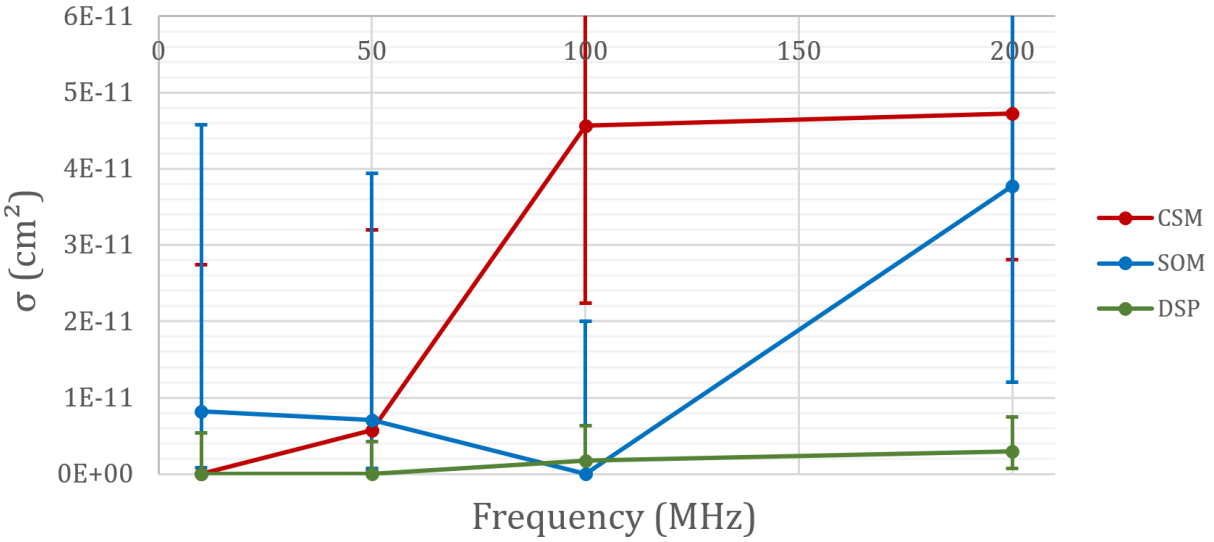


Figure 110: Cross section for **single errors** of FIR filter implementations tested at different frequencies, displayed with the 95% confidence interval.

The first result that stands out when comparing the cross section for the different types of errors (TABLE IX) is that the errors related to the SEU in the configuration memory (persistent and non-persistent) are largely predominant (98%) over errors related to flip-flop SEUs and SETs (single errors). This observation is however less valid for DSP-based filters for which the sensitivity to the different types of errors are of the same order of magnitude. This can be attributed to the fact that DSP blocks are resources with low flexibility that require a reduced number of configuration bits to define their functionality. This effectively translates into a large reduction in overall sensitivity (considering all error types) compared to fabric-based filters (CSM and SOM). Indeed, as shown on Figure 108 and Figure 109, the cross section of DSP-based filters is two orders of magnitude lower than the other filters.

Regarding the two distributed multipliers (CSM and SOM), considering all types of events, the CSM filter turned out to be twice more sensitive than the SOM based filter. This result is particularly evident in the non-persistent (Figure 108) and persistent events (Figure 109). This increased sensitivity can be correlated to the number and type of used resources shown in TABLE X. It can be noticed that CSM filter use half as much flip-flops than SOM filter while it requires twice more pipeline stages to reach equivalent timing performances. This flip-flop reduction is enabled by the instantiation of LUT-based shift registers (SRL) to add clock cycles delay on the data paths. Indeed, the CSM multiplier has been instantiated with 16 pipeline stages to reach a maximum frequency of 200MHz. The clock cycle delays required to synchronize all the signal propagated in the multiplier can be automatically implemented into one shift-register LUT (instead of multiple flip-flops) when the number of clock cycles to be delayed reach a certain threshold. However, the reduction of the number of flip-flops is highly compensated by a higher use of LUTs (CSM use 73% more LUT than SOM). The fact that the sensitivity of the CSM is twice as high as the one of SOM shows that the flip-flops utilization has a very small impact on the sensitivity while the one of LUTs has a strong impact. The increased sensitivity of the CSM structure also seems to be impacted by an increased number of instantiated programmable interconnexion points (PIPs). Indeed, the vertical propagation of carry bits (architecture detailed in section 4.3.1.4) prevents the usage of CARRY4 resource and the LUT-to-LUT connections force the activation of extra-slice routing programmable interconnexion points (PIPs). CARRY4 are hardwired primitives that do not require any configuration bit contrary to PIPs. This means that the usage of CARRY4 not only improves the multiplier performances but also tends to reduce their susceptibility to SEU in the configuration memory.

TABLE X
FILTER RESOURCE UTILIZATION, POWER AND PERFORMANCE COMPARISON BETWEEN THE THREE MULTIPLIER IMPLEMENTATION TYPES

	LUT	Flip-flop	CARRY4	DSP	Power (mW)	Pipelining (multiplier)	Max. frequency (MHz)
CSM	4468 (694 as SRL)	1847	0	0	326	16 stages	250
SOM	2577	3688	829	0	181	8 stages	300
DSP	0	0	0	9	93	1 stage	454

From Figure 108 and Figure 109, no clear trend can be identified on the influence of the operating frequency regarding the filter's sensitivity to SEUs in the configuration memory. However, for single errors (Figure 110), despite the high uncertainty of the results due to a low number of events (43 events across all designs), an increased sensitivity with operating frequency can be identified, which can be attributed to the increased probability of SET capture. This phenomenon is more pronounced on the DSP filter which has a fixed implementation, while the implementation of the other two filters is influenced by the user defined frequency.

Some rare events were observed during the test campaign. For example, the failure of several filters simultaneously was observed (56 events). This type of event could be caused by SEUs affecting input signal PIPs (common to all filters). Indeed, in the current BIST version, all filters of the same type share the same BRAM to store the input signal. A large routing tree is then instantiated to propagate the input signal to every filter replica thus requiring a large number of

PIPs to build this routing tree. Some of this PIPs are shared by routing segments feeding different filter replicas and therefore affect several filters simultaneously when affected by SEUs. Some failures or interruptions of the UART communication were also observed (124 events), probably due to SEL, SEFI or persistent SEUs affecting either the PLL and routing network, the UART transmission block or the error report formatter and the associated FIFO.

In summary, different trends can be identified from these test results:

- Failures related to configuration memory corruption are highly predominant (98%).
- DSPs are much less sensitive than distributed operators (by two orders of magnitude).
- The number of instantiated flip-flops has little influence on the sensitivity.
- The number of instantiated LUTs has a major impact on the sensitivity.
- The use of CARRY4s reduces sensitivity by limiting the number of PIPs used.
- The operating frequency has no significant influence on the configuration related failures.

4.5.2. SECOND CAMPAIGN: IMPROVEMENTS AND EXTENSIONS

4.5.2.1. TEST SETUP MODIFICATIONS

This second experiment is performed in the same facility using the same neutron beam. The test setup, however, receives different types of modifications:

- The DUT selection is supplemented with two additional FPGA families:
  -**Artix7** (SRAM – 28nm) from *Xilinx*
  -**IGLOO2** (Flash – 65nm) from *Microsemi*.

The test setup with the three FPGA boards simultaneously positioned on the neutron beam spots is shown in Figure 111.



Figure 111: Neutron test setup. The three FPGA boards are mounted on a plexiglass board and aligned with the neutron beam spot. IGLOO2 board on the left, Spartan7 board in the middle and Artix7 on the right.

- For both *Xilinx* FPGAs, the benchmark is extended with an additional benchmarking structure: an FIR filter with the same parameters (16 bits width inputs and coefficients, 9 coefficients in transposed form and full precision) using the **Area Optimized Multiplier** (**AOM**) presented in section 4.3.1.6. The AOM could not be implemented on the Flash based device as its architecture use 4-inputs LUTs (AOM requires 6-inputs LUTs).
- Based on the results obtained in the first test campaign, the benchmark was simplified by using only the implementation with a working frequency of 200MHz for all structures of the benchmark.
- DSP based FIR filter is implemented in direct form instead of transposed form.
- The BRAM storing the input signal and output signal where fully replicated along with their ECC decoders for each replica of the FIR filters under test. Indeed, the analysis of the physical implementation of the previously used benchmarks revealed that by sharing the BRAM for all filters, a great amount of routing resources was instantiated to propagate the input signal across the chip that can lead to multiple filters affected by the same event.
- SEU detection reports sent by the SEM controller are correctly recorded.

4.5.2.2. RESULTS

During this campaign, the three FPGAs are irradiated for 67 hours with an average neutron flux of $5.3 \cdot 10^{6} cm^{-2}.s^{-1}$obtaining a total fluence of $1.3 \cdot 10^{12} cm^{-2}.s^{-1}$. Over the entire irradiation period, a total of 4254 errors was observed, distributed between the different filter types and the different FPGAs as shown in TABLE XI.

TABLE XI
NUMBER OF DETECTED EVENTS AMONG THE THREE TESTED FPGAS

	Filter Type	Number of events			
		Non-persistent	Persistent	Single	Fluence (cm^{-2})
Spartan7	**CSM**	522	187	51	2.59E+11
	SOM	410	295	44	5.44E+11
	AOM	314	184	44	2.90E+11
	DSP	86	55	64	1.28E+12
Artix7	**CSM**	478	196	50	2.70E+11
	SOM	184	121	20	3.20E+11
	AOM	339	160	34	2.79E+11
	DSP	48	31	52	9.25E+11
Igloo2	**CSM**	0	0	35	3.47E+11
	SOM	0	0	19	2.65E+11
	DSP	0	0	231	8.94E+11

Based on the number of events recorded for each type of failure mechanism, the cross section of the different benchmarking structures is computed for each FPGA family as shown in Figure 112.

The first result that stands out from Figure 112 is the sensitivity contrast between the different FPGAs. The global sensitivities (considering all error types) of both SRAM devices (Artix7 and Spartan7) are almost equivalent (Spartan7 ~9% higher than Artix7) while the sensitivity of the flash FPGA (Igloo2) is one order of magnitude lower. This difference in sensitivity between the

two types of FPGA is explained by the absence of persistent and non-persistent errors on the Flash technology, which also confirms the immunity of its configuration memory to atmospheric neutrons.

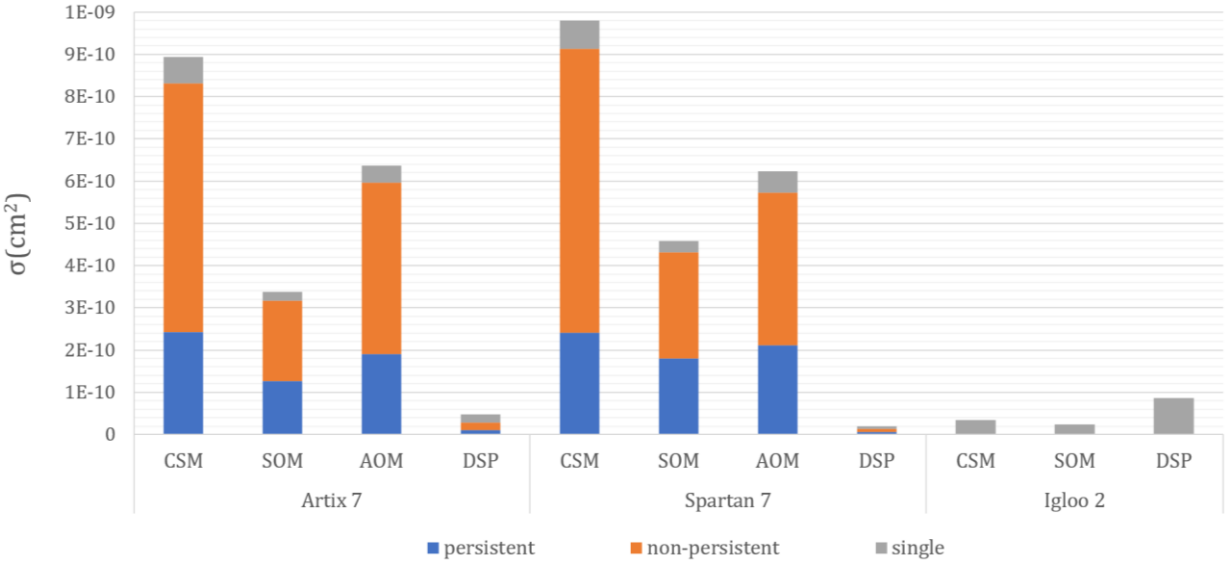


Figure 112: Neutron cross section of FIR filter for each multiplier implementation type and for the three tested FPGA families.

The comparison of sensitivity between the different filters must be put in perspective with their resource utilization detailed in TABLE XII.

TABLE XII
RESOURCE UTILIZATION PER FIR FILTER

	Spartan7 and **Artix7**			
	LUT	**Flip-Flop**	**CARRY4**	**Shift-register LUT**
CSM	4734	1982	4	694
SOM	2607	3919	842	66
AOM	2355	4009	469	279

	Igloo2		
	LUT	**Flip-Flop**	**1bit-CARRY**
CSM	6569	6297	0
SOM	3183	4246	2669

First of all, as shown in Figure 112, the relative sensitivity between the different filter types is not the same between SRAM-based FPGAs and the Flash-based FPGA. Indeed, the DSP filters are much less sensitive than the fabric-based filters (CSM, SOM, AOM) on both SRAM devices while on the Flash FPGA, the DSP filter is the most sensitive filter which even exceeds the sensitivity of the DSP filters implemented on the two SRAM devices. This last result demonstrates that the reliability oriented designing guidelines must be adapted to each device, reinforcing the importance of a tailored benchmark for radiation qualification.

Secondly, regarding the sensitivity of the three fabric-based filters implemented on the SRAM-based FPGAs, the CSM turn out to be the most sensitive filter. This increased sensitivity has been discussed in the previous campaign and has been attributed to an increase number of LUTs and

extra-slice routing connections. However, the AOM filter appears to be more sensitive than the SOM filter (+88%) while using fewer logical resources (LUT, flip-flops, CARRY4, SRL). This show the complexity of modelling the susceptibility of circuits implemented on SRAM-based FPGAs and that the number of used resources is not a sufficient parameter to estimate it. Indeed, the functionality of the design itself can be modified (routing, LUT configuration) by the configuration memory corruptions, the architecture of the circuit, the type of LUT configuration, and the placement and routing parameters have then a major impact on radiation sensitivity. The increase sensitivity of AOM over SOM while using fewer resources can be explained by further analyzing their respective topologies (section 4.3.1). A first factor that explain this increased sensitivity is the replication of input data nets: with the AOM multiplier, some of the sum bits and carry bits of the partial products are computed by separated LUTs, the input signal bits are therefore distributed to an increased number of LUTs thus increasing the number of used programmable interconnection points (PIP). When analyzing the resource utilization, especially for LUTs, the size of the LUTs that are mobilized (the number of inputs used) should be considered. Using an additional input in a LUT actually double the number of configuration bits required to define its functionality thus increasing its susceptibility to configuration memory corruptions. The detailed LUT utilization analysis for SOM and AOM multiplier is detailed in Figure 113.

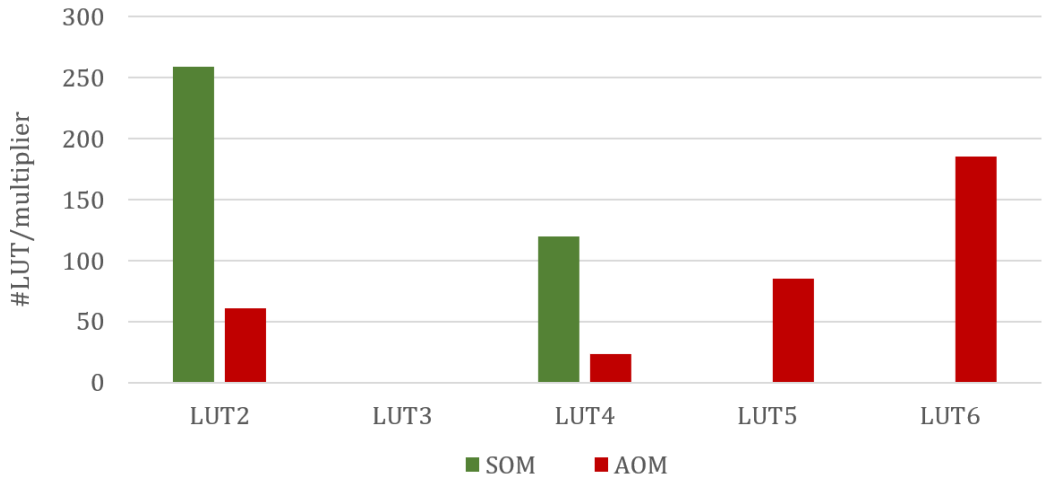


Figure 113: Comparison of LUT utilization per multiplier between SOM and AOM for different input sizes.

Figure 113 reveals that the partial product reduction tree used in AOM use fewer LUTs than SOM but with an increased number of used inputs, thus contributing to an increased susceptibility. Furthermore, the implementation of ternary adders on *Xilinx* FPGAs cannot simply use the LUT/CARRY4 combination contained in a logic block. Indeed, the output from the LUTs must be reused for the calculation of the higher order bit, forcing the use of extra-slice routing (see section 4.3.1.2). Since extra-slice routing resources are enabled by several PIPs, they bring an extra sensitivity that could counteract the reduction of used resources over binary adders.

Additional observations can be made by comparing the resource utilization (TABLE XII) and the filter cross section (Figure 113). The use of CARRY4 cells to propagate the carry bits appears to be an efficient way to reduce the configuration memory related error as no configuration bits are required to define their functionality (as opposed to extra-slice routing). In addition, the number of single errors tends to show that the use of carry logic blocks does not significantly increase the number of captured SETs.

From these results, some guidelines for the implementation of arithmetic operators can be extracted. For SRAM FPGAs, the flexibility of the instantiated resources comes at the cost of an increase in the susceptibility to SEUs in the configuration memory which are predominant in this technology (91% of all events). The use of specialized blocks with a reduced flexibility is therefore recommended, DSP blocks are thus advised when available and the use of structures employing Carry Logic blocks should be favored over structures employing only LUTs.

On the contrary, for FPGAs with an immune configuration memory, irradiation tests are required to determine the relative sensitivity of the different resources. In the specific case of the Igloo2 FPGA, fabric-based operators are less sensitive to SEUs than DSP blocks but the penalty in power consumption and area may counterbalance this advantage.

On another note, the strong presence of persistent errors (~one-third of all failures as shown in TABLE XI) questions the correction capacity of the SEM IP. The analysis of the reports sent by the SEM controller during the irradiation provides an insight into these failure mechanisms and their prevalence. Based on these reports, the number of events detected in the configuration memory can be counted. For each event, the report indicates the position of the affected memory bit and whether the bitflip was corrected or not. Based on the number of detected bitflips, the cross section of the configuration memory for Spartan7 and Artix7 FPGAs can be calculated ($5.0 \cdot 10^{-15} cm^2$/bit and $4.9 \cdot 10^{-15} cm^2$/bit respectively) which is lower than the one measured at LANSCE ($7.0 \cdot 10^{-15} cm^2$/bit) according to [61]. This discrepancy can be explained by the difference between the neutron spectrum of both facilities as shown in [155].

By crossing the data extracted from the SEM controller reports with the data from the comparator at the output of the filters, the assumption made during the first test campaign could be confirmed. It clearly appears that each non-persistent error coincides with the detection and correction of a bitflip in the configuration memory. The delay between the occurrence of the bitflip and its correction can be calculated based on the number of clock cycles where the filter is reported as being faulty. An average detection time of 2.7ms has been measured which matches the one provided by the manufacturer (2.9ms) in [130].

Regarding the presence of persistent errors, the cross-referencing of data also reveals that each persistent error recorded is preceded by the failure of the SEM controller itself or by the detection of an uncorrectable error, forcing the mitigation system to leave its detection mode. These observations confirm the assumption made on the failure mechanisms involved in this experiment. According to the SEM IP documentation [130], when using the "*enhanced mode*" of the controller as done in this experiment, the system is unable to correct the multiple non-adjacent errors in the same memory frame. The probability of two particles interacting with two bits of the same frame between two readbacks (4.6ms) being extremely low, the detected uncorrectable errors must originate from Multiple Bit Upsets (MBU) or from particular configuration bits whose corruption would be considered as uncorrectable. This aspect is further studied through the fault injection campaign. Nevertheless, this tool remains a very effective solution to avoid the accumulation of errors in the configuration memory as 98.4% of the events could be corrected.

The main results from this campaign can be summarized by the following statements:

- Flash-based FPGAs are globally less sensitive that SRAM-based FPGAs thanks to the SEU immunity of their configuration memory.
- The reliability-oriented design guidelines must be adapted to each device type.
- DSP blocks are more sensitive than fabric-based multipliers for *Igloo2* FPGA.

For *Xilinx* FPGA:

- DSP blocks are far less sensitive than fabric-based multipliers.
- 91% of errors are due to configuration memory corruptions.
- The flexibility of a resource comes at the cost of an increased sensitivity.
- CARRY4 does not significantly impact the SET generation rate.
- The number of instantiated cells is not a reliable parameter to estimate the circuit sensitivity.
- The size of used LUTs and the number of extra-slice connections have to be considered.
- Uncorrectable errors are caused by prior failures of the SEM controller or deactivation of the associated scrubbing engine (probably due to MBU detection).

4.5.3. THIRD CAMPAIGN: FILTER'S STRUCTURAL PARAMETERS

This campaign is performed with a 180MeV proton beam at the PARTREC facility. The benchmark is supplemented with new structures allowing to analyze the influence of different structural parameters of the FIR filter over its radiation sensitivity. Namely, the use of flip-flop control signals (reset and clock enable), of the number of pipeline stages inserted and of the shape of the FIR filter. A Triple Module Redundancy (TMR) version of the filter is also added to the benchmark to evaluate the effectiveness of this mitigation scheme. On the other hand, two types of constant coefficient filters (non-reloadable) implementations are also evaluated to analyze how the enabled optimizations can improve the radiation tolerance. Finally, a new implementation of multiplier is integrated to the benchmark.

4.5.3.1. TEST SETUP

To evaluate the influence of different parameters on the sensitivity of the filter, the benchmark is derived in 6 structures, each structure varying a single parameter.

- **DIR**: FIR filter in **dir**ect form with 9 SOM 16x16 fully pipelined multipliers: already tested in previous experiments and used as a reference.
- **TRA**: FIR filter in **tra**nsposed form with 9 SOM 16x16 fully pipelined multipliers. used to compare direct and transposed FIR filter form.
- **COM**: FIR filter in direct form with 9 SOM 16x16 **com**binatorial multiplier (no pipeline stages): use to analyze the effect of pipeline stages.
- **RCE**: FIR filter in direct form with 9 SOM 16x16 fully pipelined multipliers. Each flip-flop is supplemented with asynchronous **r**eset and **c**lock **e**nable signals: used to quantify the influence on the filter sensitivity induced by the use of control signals.
- **TMR**: FIR filter in direct form with 9 SOM 16x16 fully pipelined multipliers. The filter is implemented three times with physical separation, and majority voters are inserted on the triplicated filter output: evaluate the hardening capacity of this block-TMR scheme.
- **PAA**: FIR filter in direct form with 9 **Pa**randeh-**A**fshar ([143]) 16x16 fully pipelined multipliers: another type of multiplier with different architectural parameters.

Two additional structures are used to compare different implementation of constant coefficients FIR filters:

- **CST**: use a transposed form FIR filter by replacing SOM parallel multipliers by **c**on**st**ant multipliers benefiting from further optimizations as explained in 4.3.1.8.
- **DAF: d**istributed **a**rithmetic FIR **f**ilter (presented in 4.3.1.11) with 16bits data width and 6 coefficients.

The test campaign is performed on a much larger component, the **Kintex7-325T** (*Xilinx* 28nm) embedding more than six times as many logic cells than previously tested FPGAs. Radiation testing on components of this scale enables new test plan possibilities but also imposes additional constraints. Instead of testing each test structure successively by reconfiguring the component between each run, all test structures are implemented simultaneously on the same chip. The entire duration of the campaign can thus be appreciated by executing a single continuous run. The test setup can also be lightened by removing the connections required to reprogram the component. However, implementing and testing multiple circuits on the same chip require particular attention to the placement of these circuits. Ideally, each benchmarking structure should be placed in an exclusive and physically delimited area of the chip (called PBlock for *Xilinx* components). This constrained placement ensures that the sensitivity of each part is not influenced by the other part of the circuit. In this experiment, each structure is replicated up to four times on the chip. To ensure a good homogeneity of the physical implementation between the replicas, the same PBlock size and structure is used for each of them. The physical layout of the different resourced is shown in Figure 114.

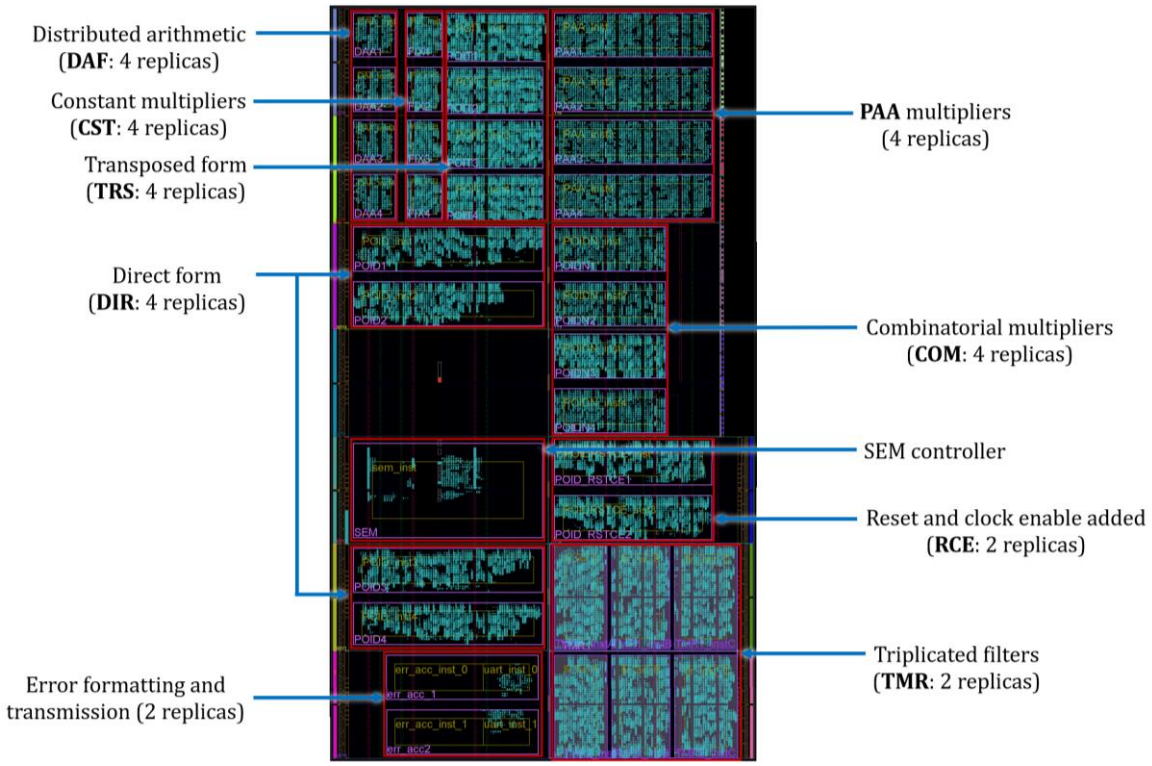


Figure 114: Physical layout of the benchmarking replicas on the Kintex7-325T. Each replica is implemented and self-contained in a dedicated PBlock.

To be noted that for the TMR structure, each block of the TMR scheme is implemented in a separated sub-PBlock to minimize the number of single of points of failures.

The BIST structure and testing procedures used in previous experiment are reused identically. A total of three DUTs programmed with the same design are simultaneously irradiated during the experiment. When performing radiation experiments with protons, one must ensure that the protons can reach the active zone of the DUT with sufficient energy after crossing the eventual package and silicon layers. SRIM simulator [19] can be used to evaluate the penetration of proton inside the different layers. The components used in this campaign are delided flip-chip components. The active zone lies behind a 700μm layer of silicon, requiring protons of at least 10MeV to reach the active zone. In the PARTREC facility, the maximum proton energy (180MeV) is used. In the first instance, the flux of proton is tuned to maximize the event rate while limiting the time spent reconfiguring the component when an uncorrectable error is detected. The best compromise is reached with a flux of $5 \cdot 10^6$ $s^{-1}cm^{-2}$ corresponding to almost one event in the configuration memory every second and a reconfiguration (uncorrectable error) every 15 seconds. A total fluence of $5.9 \cdot 10^{11}$ cm^{-2} is reached considering all DUTs.

4.5.3.2. RESULTS

Now that the origin of persistent and non-persistent error has been identified, the detected events are grouped considering events related to configuration memory upsets (persistent and non-persistent errors) on one side and events related to flip-flops upset and SET capture on the other side (single errors) as shown in TABLE XIII. As performed in previous experiment, the cross section for each type of benchmarking structure is computed. These results are displayed on Figure 115 and Figure 116 respectively.

TABLE XIII
NUMBER OF DETECTED EVENTS ACROSS THE DIFFERENT FILTER TYPES

	Number of events							
	DIR	**TRA**	**COM**	**RCE**	**TMR**	**PAA**	**DAF**	**CST**
Persistent & non-persistent errors	1072	1572	846	762	77	2141	732	702
Single errors	65	105	32	29	1	71	30	27
Number of replicas	3	4	3	2	2	4	4	4

The first observation that can be made on Figure 115 and Figure 116 is that the proportion of transient errors is as low as the one observed with neutron tests (between 1% and 7% depending on the type of filter).

To investigate the influence of the different filter parameters under test, the cross section of each filter is compared to the reference filter (DIR). By putting these results in perspective with the resource utilization (detailed in TABLE XIV), an explanation will be proposed to justify one by one the differences in sensitivity induced by each type of architectural modification. In a second step, the analysis of filters with constant coefficients (DAF and CST) will be performed by comparing them with each other.

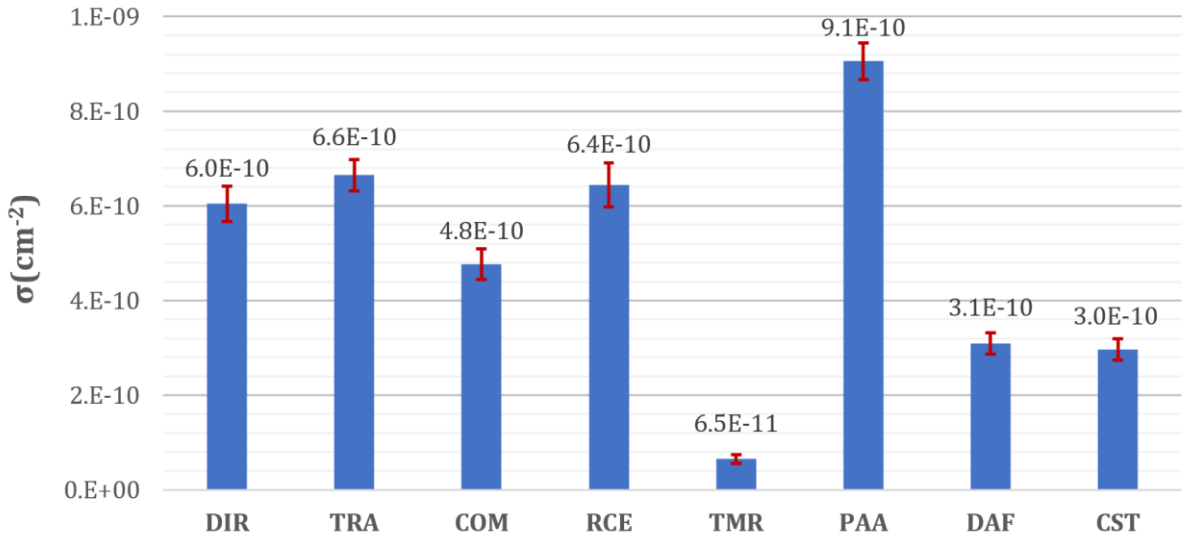


Figure 115: Proton cross section of different FIR filter structures for **persistent and non-persistent** errors (configuration memory upset related failures).

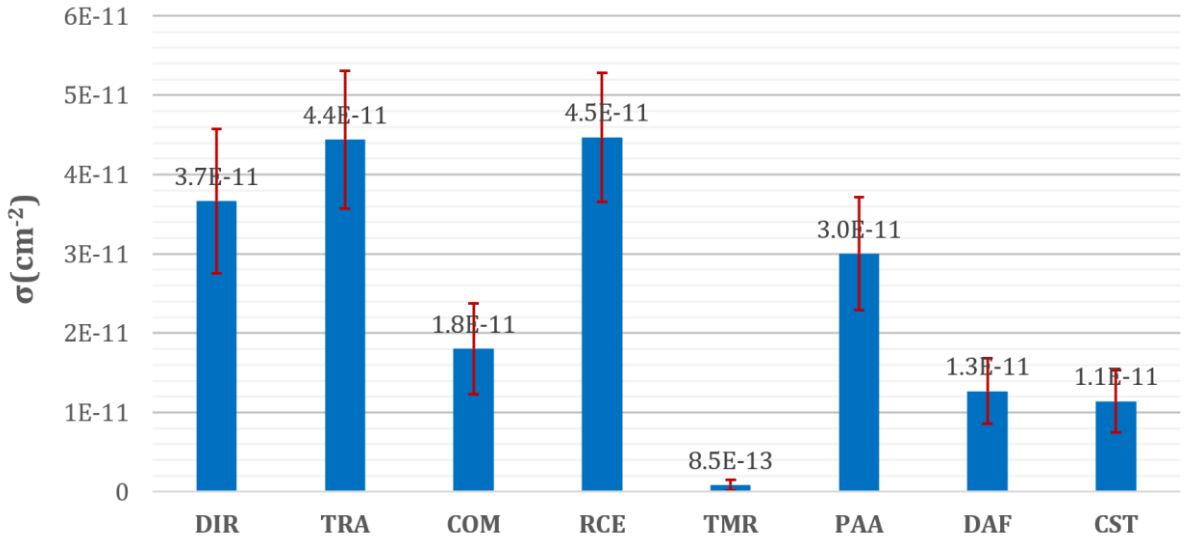


Figure 116: Proton cross section of different FIR filter structures for **single** errors (flip-flop upsets and SET capture related failures).

TABLE XIV
RESOURCE UTILIZATION OF THE DIFFERENT FIR FILTERS

	DIR	TRA	COM	RCE	TMR	PAA	DAF	CST
Flip-flop	3149	4175	522	3149	9447	2413	615	1155
LUT	2683	2648	2652	2676	8151	3813	759	739
CARRY4	705	760	705	705	2115	57	10	155

Filter form: the transposed form (TRA) is significantly more sensitive than the direct form (DIR) (+10%). The main assumption is that this increased sensitivity can be attributed to the large fanout nets used to propagate the input signal to each multiplier. High fanout nets require a lot of PIPs while a large proportion of these PIPs, those located on the first stages of the distribution

tree, can alter the propagated values of several multipliers simultaneously, thus increasing the actual probability of creating failures.

Number of pipeline stages: by using combinatorial multipliers (COM) a 20% reduction of the sensitivity is granted over the fully pipelined version (DIR). This reduction is attributed to the large reduction of flip-flop usage (-83%) and the configuration bits used to define their functionality and those used by intra-slice routing multiplexers driving the LUT's output to the flip-flop inputs. It is worth mentioning that the operating frequency used in this experiment is 10MHz for all designs. The fully pipelined versions are obviously over-designed for this operating frequency. However, the implementation of combinatorial 16 bits multipliers might be more challenging for the routing tool, forcing a reduction of the net lengths, thus reducing the number of instantiated PIPs. The sensitivity reduction is even higher when considering single errors (50%) that can be directly correlated to the number of used flip-flops.

Usage of control signals: the usage of reset and clock enable signals (RCE) has a non-negligible impact on the filter cross section (+7%). This sensitivity increase can be attributed to the PIPs used to drive the control signal to each flip-flop. The usage of these signals also influences the number of single errors (+20%) that can be attributed to the generation of SETs in the propagation tree of these control signals.

Triplication: the block TMR scheme effectively grant a good reduction of the filter sensitivity (by one order of magnitude). The sensitivity remains nonetheless significant, considering that the number of resources is more than tripled to provide this protection. The physical separation constrained between the structure replicas theoretically removes the presence of single points of failure (configuration bits that can impact several versions of the TMR scheme). However, the clock signals are driven by the same buffer and propagate to the flip-flops with routing trees sharing a few PIPs. The resulting sensitivity of the triplicated structure can be partially attributed to the clock routing network and to the majority voter. However, this majority voter handles three 32 bits output vectors and is implemented with no more than 12 LUTs and one CARRY4 cell, suggesting that other sources of failure are involved. The output signal of the majority voter is compared to the golden reference stored in a BRAM protected by ECC in compliance with the BIST structure presented in section 4.4.2. This ECC decoder can be another source of reported failures. Finally, the observed failures can also be attributed the presence of Multiple Bit Upset (MBU) affecting simultaneously several configuration bits related to different filter replicas.

Multiplier architecture: the multiplier proposed by Parandeh-Afshar in [144] offers better timing performances while reducing the number of instantiated flip-flops. However, for 16bits multiplier, the number of instantiated LUTs is much higher than the SOM multiplier. This increased LUT usage is particularly noticeable in the filter sensitivity (+50%). This increase sensitivity can also be attributed to the usage of LUT with higher input counts as discussed in section 4.5.2.2.

CST and DAF: Regarding the two constant coefficients filters (CST and DAF), the optimization features enabled by constant multiplications on one side and the use of distributed arithmetic on the other side provides a clear reduction of the radiation sensitivity (~50%). Both implementations have a similar sensitivity while having very different structures. The sensitivity of distributed arithmetic could be mainly attributed to a wide use of 6-inputs LUT to store the

precomputed combinations of coefficients while the one of constant multiplications can be attributed to higher number of PIPs to route an increased number of connections.

The assumptions made on the predominant source of failures are further explored through fault injection in section 4.6.3 and through detailed netlist analysis in section 5.4.2.

On the other hand, as performed in previous campaign, the bit cross section of the configuration memory is extracted based on the bitflip detections reported by the SEM controller. Using the timing and physical address of the error reports, the presence of MBUs can be identified. For each MBU size, the proportion among all events are computed as shown in Figure 117.

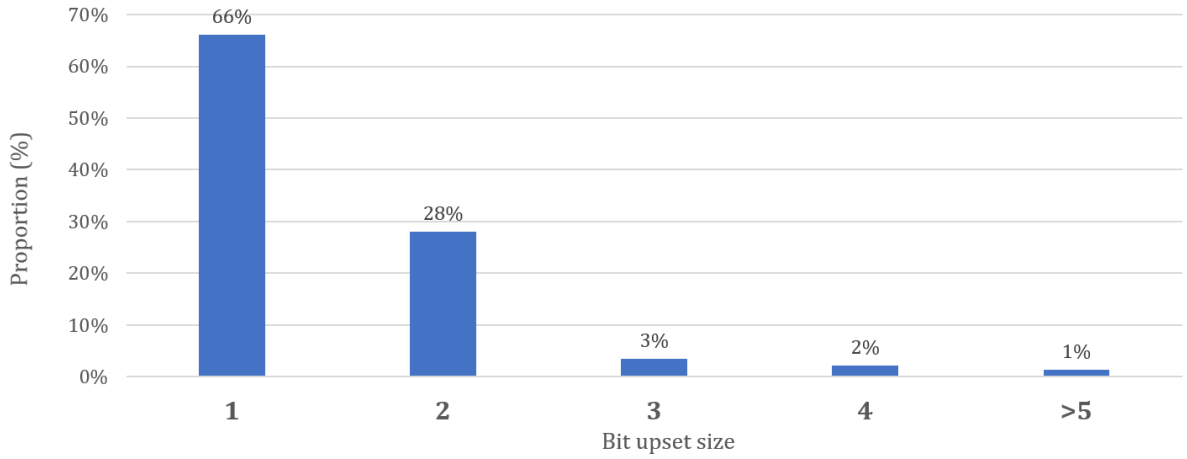


Figure 117: Proportion of SBUs, DBUs and MBUs in the configuration memory of Kintex7 FPGA irradiated with 180MeV protons.

This result reveals that the number of Double Bit Upsets is significant, representing 28% of the total number of events. Only 6% of the total number of events are affecting more than two bits simultaneously. Two types of bit cross section are then defined for the configuration memory. The first one is computed by considering the total number of events, all error sizes included, divided by the fluence and the number of configuration bit (BRAM bits removed) as shown in (15) . The second one, weights each type of event by their respective size to account for the real number of corrupted bits as shown in (16).

$$\sigma_{bit}^{event} = \frac{N_{1BU} + N_{2BU} + N_{3BU} + \dots}{\Phi \times N_{CRAM\ Bit}} = 4.1 \cdot 10^{-15}\ cm^{-2}/bit \tag{15}$$

$$\sigma_{bit}^{weighted} = \frac{\mathbf{1} \times N_{1BU} + \mathbf{2} \times N_{2BU} + \mathbf{3} \times N_{3BU} + \dots}{\Phi \times N_{CRAM\ Bit}} = 5.7 \cdot 10^{-15}\ cm^{-2}/bit \tag{16}$$

Both calculations are not formally correct, the first one assumes that each event is an SBU (while only representing 66% of all events) and the second one considers MBUs as being equal to several SBUs while they are not equivalent. Indeed, as opposed to multiple SBUs, MBUs affect several bits that are necessarily around the same physical location. On the other hand, when considering several SBUs, each SBU can be corrected before the next one appears, unlike MBUs that affect several bits simultaneously and can create failures that would not occur if the bits were affected independently. In addition, the MBU detection capability of the SEM controller has not yet been

validated by comparison with static testing. For these reasons, these two cross sections will be used as lower and upper bounds respectively for the sensitivity calculations used by the fault injection and detailed netlist analysis methods (chapter 5).

4.6. FAULT INJECTION

Fault injection is a well-known complementary methodology to characterize the reliability of a circuit against soft errors. The concept of fault injection is to artificially reproduce the effect of a soft error to analyze the behavior of the system in response to the perturbation without using particle beam testing. In this section, different fault injection techniques are presented. A fault injection procedure for *Xilinx* FPGAs using the SEM controller is then presented. The technique is finally applied to the benchmark implemented during the second and third test campaigns and compared to the experimental test results.

4.6.1. STATE OF THE ART METHODOLOGIES

One of the main objectives of fault injection methodologies is to identify and enumerate the number of specific circuit locations where soft error can induce system failures, in other words, which modifies the circuit outputs. When applied to FPGAs, different techniques can be used, targeting different error mechanisms.

Laser testing [156] reproduces the charge deposition mechanism induced by high energy particles by applying a focused laser beam directly on the FPGA chip where the circuit is implemented. In this way, the different types of radiation induced failure mechanisms can be reproduced (SEU on flip-flop, SET, SEU on configuration memory).

Simulation based fault injection [157] uses RTL simulation tools. The purpose is to force the value of one of the logical nets of the design during the simulation to observe the response of the system to the upset. This approach can be implemented with different techniques. In [158], a technique base on HDL code modification introduces multiplexer structures called *saboteurs* on the data paths selected for fault injection. These multiplexers can be activated to alter the net state with different types of error model (bitflip, stuck-at-0, stuck-at-1). In [159], the fault injection relies on the use of built-in simulator TCL commands to force the value of a given net. This technique can be applied during simulation without HDL code modification. However, the simulation time might be prohibitive when a large number of faults has to be injected. As these techniques rely on behavioral simulation, the failure model of the specific FPGA architecture must be properly assessed and artificially recreated. The lack of low-level schematics available to the end user makes it difficult to reproduce the behavior of errors in the configuration memory. It may also fail to identify complex faulty behaviors, especially when simulations are done at high abstraction levels.

Emulation-based fault injection is achieved by artificially reproducing different types of errors directly on the FPGA where the design is implemented. SEUs on the user flip-flops can be reproduced with different techniques. Instrumentation-based approaches use extra logic (added on the HDL code) in the flip-flop input path. This extra logic can then be activated during run-time for bitflip insertion. Reconfiguration-based approach [148], [160], applicable to SRAM-based FPGAs, use the reconfiguration capabilities of the component to modify the content of the flip-flop while the clock is stopped. The injection of errors on specific registers is enabled thanks to a map

provided by the EDA tools [161] linking the different flip-flops of the design to their configuration memory address. This injection process can be accelerated by using the partial reconfiguration capabilities of the component as demonstrated in [162]. This reconfiguration approach can be extended to every bit of the configuration memory. However, without a deep knowledge of the bitstream composition, the function of the bits and their position on the FPGA fabric cannot be determined. The faults are therefore “blindly” injected. The baseline concept is to inject a fault on each of the configuration bits and count the number of bits that alter the operation of the system. The size of the configuration memory of the largest FPGAs can reach several hundreds of million bits. The duration of the campaign can therefore be excessively long. To reduce this duration, the list of faults to be injected can be reduced. For *Xilinx* FPGAs, the EDA tool can provide a file indicating the position of all configuration bits called "essential". This terminology refers to all the configuration bits that can potentially affect the architecture of the implemented circuit. Among these bits, the set of bits that actually create a failure by impacting the output signals are defined as “critical”. To reduce the duration of the campaign, the fault list can be restricted to the essential bit list. It is then about identifying the proportion of critical bits in this set. An exhaustive injection is not always required to determine this proportion, by injecting only a statistically significant number of faults, the campaign length can be further reduced. On the other hand, the injection time per fault is a crucial point for the duration of the campaign. The type of configuration interface used to inject errors (JTAG, SelectMAP, ICAP) and the error detection technique have a major impact on the fault injection time. A fast error detection approach, proposed in [148], [160], relies on the use of a dedicated hardware platform using two FPGAs with the same configuration, one where the fault is injected, the other one is used as a golden reference to detect faults by comparing the outputs of the two FPGAs. However, its use relies on a custom PCB which limits its extension to commercial development boards.

In this study, an emulation-based fault injection campaign focusing on the effects on configuration memory is presented and applied to *Xilinx* FPGAs. Using an approach similar to the one proposed in [163], faults are injected via the Internal Configuration Access Port (ICAP) using the SEM controllers. The BIST structure used during the neutron test campaigns is reused to detect the errors. This approach, fast and easy to implement, can use exactly the same configurations and test setups as the one used for the SEE tests.

4.6.2. FAULT INJECTION PROCEDURE

The SEM controller provides a simple way to create and correct bitflip in the configuration memory through its serial interface. For error injection a specific command must be sent providing the memory location of the bit to be flipped as shown in Figure 118.

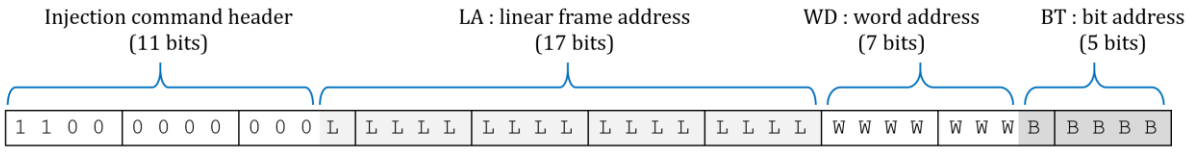


Figure 118: SEM controller error injection command. The memory location of the bit to be flipped is defines by its frame address (LA), its word address (WD) and its bit address (BT).

The *Xilinx* EDA tools (Vivado) can provide the position of all the essential bits of the implemented design by using the following commands in the constraints file.

```
set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_design]
```

This property will induce the creation of the EBD file at bitstream generation step. This EBD file is an ASCII text file that has an informational header, followed by a number of lines, where each line has 32 characters that are either 0 (non-essential) or 1 (essential). Each line represents a word of the configuration memory with LSB on the right. In the *7 series* FPGA family, each frame consists of 101 words. The first 101 line in the EBD description correspond to a dummy frame that should not be accounted to compute the frame address. For each essential bit in this EBD file, based on the position in the file (*line* and character *position* in the line), the corresponding SEM injection command can be determined through equations (17)(18)(19).

$$WD = line \bmod 101 \tag{17}$$

$$LA = \frac{line - \mathrm{WD}}{101} - 1 \tag{18}$$

$$BT = 31 - position \tag{19}$$

A python script is used to read the EBD file and generate the list of injection commands in hexadecimal format for each essential bit. This list is randomly shuffled to ensure the homogeneity of the bit positions when the fault injection is not exhaustive. When performing statistical injections, only a subset of the essential bits is injected. Given N_{INJ} the number of bits injected, N_{ESS} the total number of essential bits and N_{CRIT} the number of observed critical bits among N_{INJ}, the total number of critical bits N_{CRIT}^{TOTAL} can be extrapolated with equation (20).

$$N_{CRIT}^{TOTAL} = \frac{N_{CRIT}}{N_{INJ}} \times N_{ESS} \tag{20}$$

According to [164], to evaluate the proportion of critical bits among essential bit through statistical injections, the total number of bits N_{INJ} to be injected is defined by equation (21).

$$N_{INJ} = \frac{N_{ESS}}{1 + e^2 \times \frac{N_{ESS} - 1}{t^2 \times p \times (1 - p)}} \tag{21}$$

e: error margin on the extrapolated proportion

t: confidence level factor with respect to normal distribution (*t=1.96* and *t=2.58* for a confidence level of *95%* and *99%* respectively). This level is the probability that the exact value is actually within the error interval.

p: the estimated proportion of critical bits

The fault injection campaign is performed with the same test setup as for the neutron tests. A specific python script is developed to automate the fault injection, detection, correction and classification of errors according to the procedure described in Figure 119.

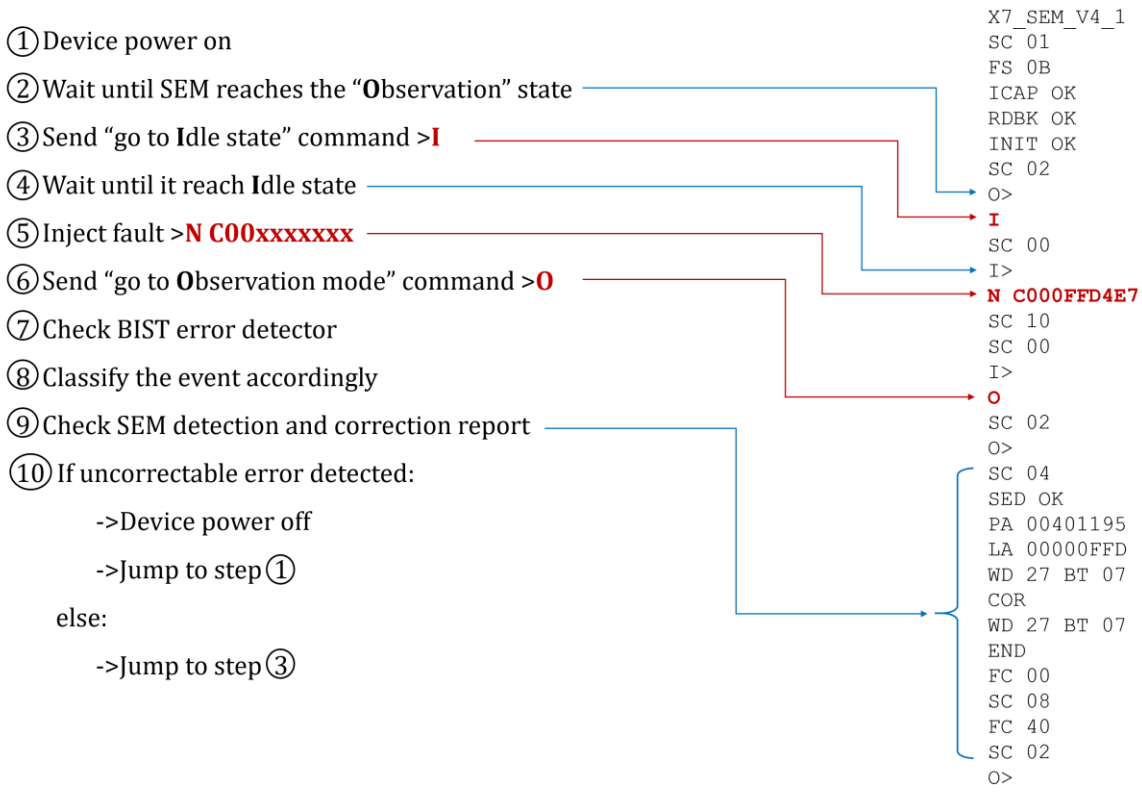


Figure 119: Automated fault injection procedure. The different steps of the procedure are described on the left part and the corresponding SEM logs are detailed on the right side.

Following this procedure, the accumulation of faults is prevented in order to test the criticality of each bit independently. For each injected fault, the error reports from the BIST structure must be acquired and one must ensure that no error is reported any longer after the correction of the fault before proceeding to the next injection. The latency between injection and error correction can be up to 10ms for the FPGA used. The error reporter being based on an error accumulation for half a second, each fault can potentially induce an error accumulation on two successive counts, thus limiting the minimum latency for each injection at approximately 1 sec. This latency could have been greatly reduced by changing the formatting of the error reports to send only one short report immediately after error detection.

4.6.3. Experimental results

4.6.3.1. Statistical analysis

To assess the statistical influence of the number of injected faults on the critical bit percentage, an exhaustive fault injection is performed on a simple design containing a single FIR filter and an improved error formatting to speed up the fault injection process. The number of essential bits for this design is 467,526. The ratio of critical bits is computed while using different batch sizes of randomly selected bits among the total number of injected faults. For each batch, the critical bit percentage is compared to the one obtained with exhaustive fault injection (24%). This operation is repeated one hundred times. For each batch size, the mean relative error on critical bit proportion estimation and associated standard deviation are computed. The errors on critical bit percentage estimation are compared to the theoretical error value computed by reversing equation (21) as shown in equation (22) with p=0.24; t=2.58 (99% confidence level) and N_{ESS}=467,526. Results are shown in Figure 120.

$$e = t \times \sqrt{\frac{p \times (1-p)}{n} \times \frac{N_{ESS} - N_{INJ}}{N_{ESS} - 1}} \quad (22)$$

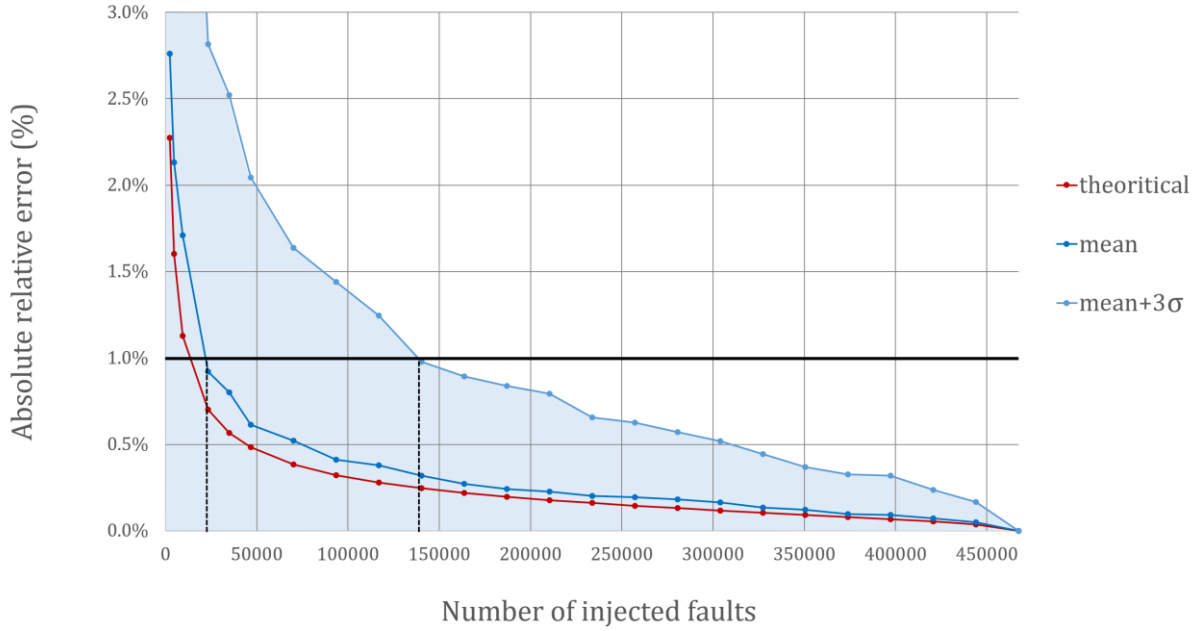


Figure 120: Absolute relative error of critical bit proportion estimation. Comparing theoretical error value with the statistical error obtained with 100 batches of randomly selected bits for each batch size. Values in dark blue represent the mean error between the 100 batches and the blue area represents the mean plus three standard deviation (equivalent to 99,73% of values for normal distribution) to represent the worst-case scenario.

This statistical analysis shows that the error made on critical bit proportion estimation quickly decreases under 1% after 25,000 faults injected (5% of essential bits) in average but the worst-case scenario requires up to 140,000 faults injected (30%) to reach the same precision.

4.6.3.2. APPLICATION TO THE SECOND TEST CAMPAIGN BENCHMARK

The three designs used during the second SEE test campaign contain a maximum of 3 million essential bits. Each design contains 3 fabric-based filters (CSM, SOM or AOM) and 8 DSP-based filters. According to the equation (21), the minimum number of faults to inject so that the true critical bits proportion is between $\pm 1\%$ around the estimated proportion with a confidence interval of 99% is 16,500 considering the worst-case proportion (50%). To ensure that the estimation error can be neglected, this value was multiplied by 10, for a total of 165,000 faults injected for each design, equivalent to 45 hours of fault injection per design.

Using the *Spartan7* cross section per bit measured in section 4.5.2.2 ($\sigma_{bit} = 4.97 \cdot 10^{-15}$ cm^2/bit), the cross section of each filter is calculated by multiplying the extrapolated number of critical bits to the bit cross section. In Figure 121, these results are compared to those from the irradiation campaign considering only the configuration memory corruption event (persistent and non-persistent).

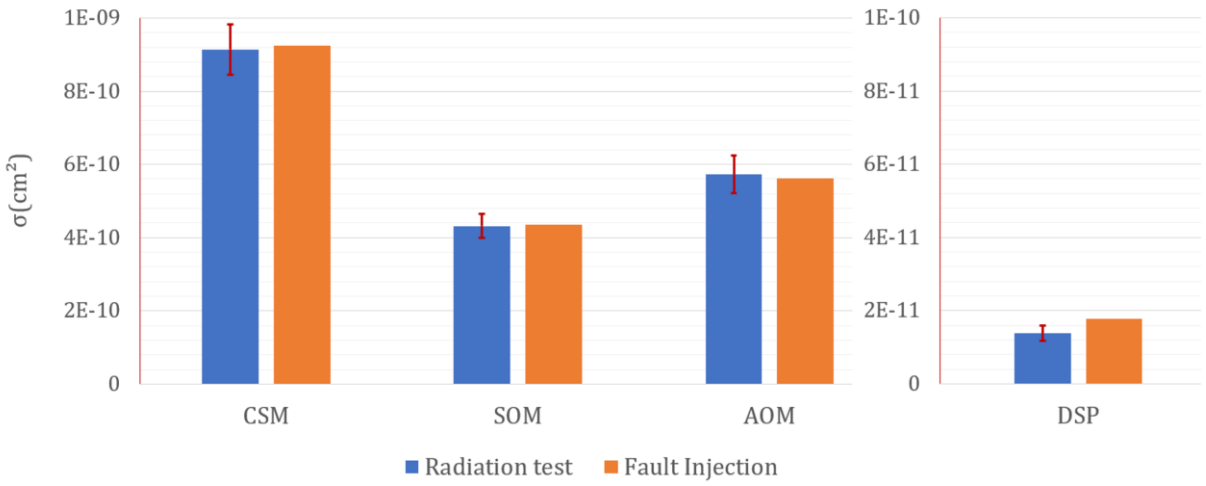


Figure 121: Comparison of filters cross sections between the radiation tests (single error excluded) and the fault injection campaign (Spartan7).

As shown in this figure, the results from the two approaches are in very good agreement: for each multiplier implementation, the cross section computed from fault injection results fall inside the error margins of the radiation test results. This observation confirms the origin of the persistent and non-persistent errors observed during the campaign. It also shows that the extraction of the bit cross section through the SEM controller reports can be reused along with a fault injection campaign to evaluate the susceptibility of another design without the need for further irradiation experiments. However, the fault injection approach does not assess the single errors due to SEUs in flip-flop or SETs in the combinatorial logic (<10% of errors) nor certain categories of SEFIs.

Concerning the uncorrectable errors, a number of bits were detected as uncorrectable after their injection or caused the failure of the SEM IP. As realized in section 4.5.2.2, the cross section corresponding to all these bits has been calculated: $\sigma_{SEM,FI} = 5 \cdot 10^{-11} cm^2$. This cross section represents only 17% of the cross section evaluated during the irradiation campaign. This result suggests that a large part of the uncorrectable errors observed during the irradiation campaign are due to MBUs or configuration bits that are not accessible for error injection by the SEM IP (internal device control registers and state elements). In the other hand, a fault injection was performed on the non-essential bits to verify that none of these bits could generate uncorrectable errors, this was confirmed: over 100 thousand bits randomly injected in non-essential bits, none had any effect.

4.6.3.3. APPLICATION TO THE THIRD TEST CAMPAIGN

The fault injection procedure is applied to design used during the third SEE test campaign. This design contains more than 14.5 billion of essential bits among which 2.4 billion of faults have been injected over a period of 2 weeks. For each type of filter, the number of detected critical bits is multiplied to the 180MeV proton bit cross section considering the lower and upper bounds defined in section 4.5.3.2 ($\sigma_{MIN} = 4.1 \cdot 10^{-15} cm^{-2}/bit$ and $\sigma_{MAX} = 5.7 \cdot 10^{-15} cm^{-2}/bit$). The resulting cross sections are compared to the one obtained during the SEE test campaign considering only configuration memory related failure as shown in Figure 122.

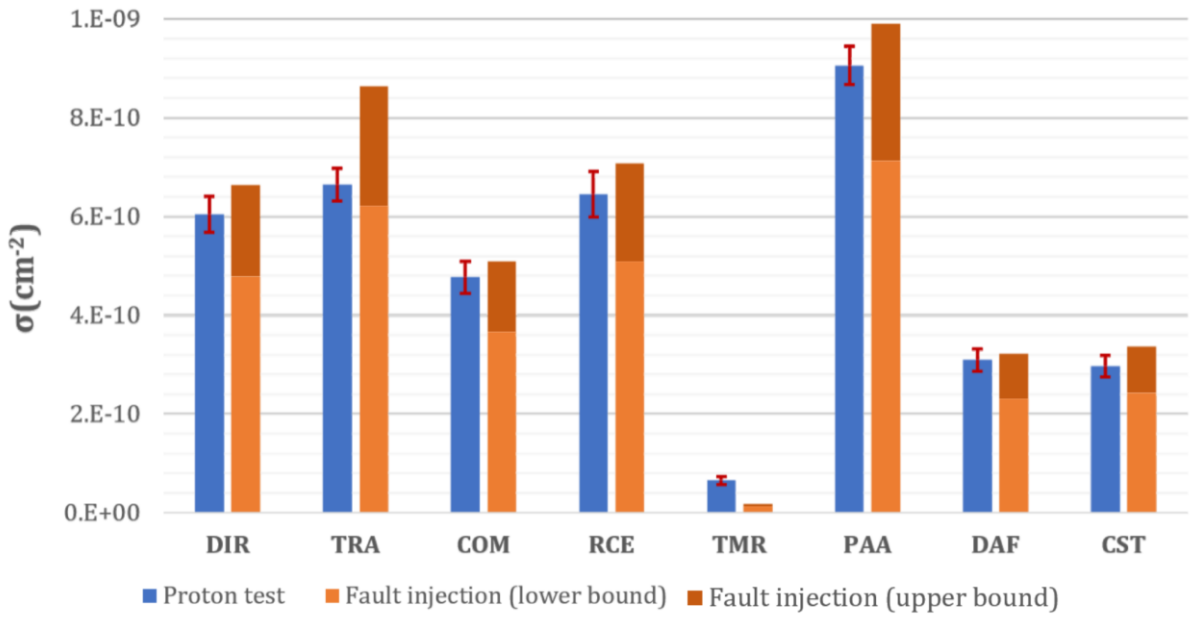


Figure 122: Comparison of the filter cross sections between the proton experiment and the fault injection considering lower and upper bound of the bit CRAM bit cross section (Kintex7).

As shown in this figure, the results of both approaches are in good agreement, for all filter implementations, the radiation test result are contained between the lower and upper bound of the fault injection estimation, excepted for the TMR version. This result suggests that the TMR related failures observed during the proton experiment are mainly related to multiple errors in the configuration memory. This assumption is further investigated by checking the SEM report logs in instants when errors on the TMR circuit are reported. This analysis revealed that the vast majority of TMR circuit failures occurred because of MBUs (mainly double bit upsets) or when the SEM controller is out of service. This can happen either because of a previous detection of an uncorrectable error or because the controller is being initialized. Indeed, when the component is power cycled, the bitstream is loaded in less than 2 secs but the SEM controller is operational only 8sec later. During these times when the scrubbing system is not operational, errors can accumulate in the configuration memory and alter different replicas of the TMR scheme. This phenomenon is then due to the high fluxes used during accelerated radiations experiment. It should not be considered for the real operation of the component (when the radiation flux is much lower) if a scrubbing system preventing the accumulation of faults is correctly implemented. However, TMR failures related to MBU are a real concern for the reliability of systems protected by this type of hardening technique. Indeed, the physical separation of replicas is not entirely sufficient to prevent MBU from defeating the TMR protection. A study of the spatial distribution of the configuration memory bits could allow to define implementation rules to avoid that configuration bits linked to two replicas of the same circuit are not positioned too close to each other.

4.7. Conclusion

In this chapter, the state-of-the-art of SEE test methodologies for FPGAs has been established. Given the limitations of primitive level testing techniques and application-level testing techniques, the benchmark approach appears to be an efficient technique to reproduce the diversity and complexity of interactions found in a real circuit while providing good visibility on the resource sensitivity and the predominant failure mechanisms. Another key point of the benchmarking

approach lies in its ability to be used to compare the radiation sensitivity of different components, to identify for their main vulnerabilities and to provide recommendation regarding the mitigation techniques to be applied. In this chapter, a new benchmark specific to radiation testing has been developed based on parallel multipliers. By proposing several types of implementation of the same arithmetic function, these test structures allow to test circuits with a large architectural diversity and to evaluate the influence of different parameters on the SEE sensitivity: the number and type of instantiated resources, the number and type of connections, etc. To effectively test these arithmetic operators while limiting the number of false error detections, these multipliers have been grouped together in the form of a FIR filter, limiting the number of signals to be monitored. The detection is realized thanks to the development of a dedicated BIST architecture which allows to inject test patterns into the structure and to monitor the output signals internally before transmitting the detected errors to an external system. This BIST architecture can be used without constraints on the external interfaces available on the PCB used during the tests and on the data exchanged with external monitoring systems. This approach has been tested on three experimental campaigns under neutron beam and proton beam and has been confronted with fault injection experiments.

Results have shown the ability of the proposed approach to address the different requirements of radiation testing of FPGAs. The diversity of benchmarks regarding resource utilization and circuit topology allows to assess the sensitivity of the basic elements composing the FPGA fabric while providing useful guidelines for the reliability of computationally intensive designs. The test results provided a clear characterization of failure and recovery mechanisms on SRAM-based FPGAs equipped with an internal scrubbing system. The use of this scrubbing system also offered an efficient way to estimate the cross section of the configuration memory bits that can be used without interruption of the circuit operation, unlike other methods based on external readback of the configuration memory. This scrubbing system can also be used as a fault injection system based on error emulation in the configuration memory. This fault injection process allows to determine the number of critical bits for a given circuit. The cross section from radiation tests can be then be reused jointly with a fault injection campaign to estimate the radiation sensitivity of other designs without further radiation tests. The fault injection campaign based on the use of the SEM controller also allowed to confirm the experimental results and to explain more precisely the origin of observed failure mechanisms. Beyond this feature, experimental results have confirmed the SEM IP controller as a convenient and a very efficient way to avoid the accumulation of errors in the configuration memory as 98.4% of the errors are corrected. Nevertheless, uncorrectable errors can compromise this mitigation system. Specific actions will have to be taken to manage this type of events (reconfiguration, external scrubbing, etc.). The comparative results of the different benchmarking structures and the different components have highlighted some FPGA specific design rules encouraging the use of the least flexible logic resources (DSP and Carry logic) for configuration sensitive FPGAs and to avoid the use of ternary adders in partial product reduction trees. Finally, the benchmark has shown that for Flash-based FPGAs, even if the sensitivity to single errors is of the same order of magnitude as SRAM based FPGAs, the immunity of the configuration memory to SEUs makes them much more tolerant to SEEs except for DSP based circuits.

The different test campaigns performed in this study confirmed the predominance of the effects of configuration memory corruption on the reliability of systems based on *Xilinx* SRAM FPGAs.

Due to the complexity of the error model induced by the architectural modifications of the circuit when the configuration memory is impacted, fault injection appears to be an essential and complementary tool to extrapolate the results of SEE tests for the reliability assessment of other circuits. However, emulation-based fault injection has some limitations. Due to the lack of information on the composition of the bitstream and due to the limited internal visibility of the circuit, the detailed processes linking the CRAM corruptions to the failure cannot be precisely evaluated. To understand these processes more precisely, it may be interesting to identify for each configuration bit, its position in the FPGA fabric, the type of resources impacted, the conditions of the fault activation and their propagation in the rest of the circuit. In the next chapter, a new analytical approach is developed to model and analyze these processes. By analyzing the physical netlist of the implemented circuit, this approach can identify the different configuration bits that can affect the behavior of the circuit and for each of these bits, to evaluate its criticality by identifying the conditions of fault activation and propagation.

5. SEE SUSCEPTIBILITY EVALUATION TOOL

In this chapter, a new software, based on an analytical approach is proposed to estimate the impact of SEUs in the configuration memory on the reliability of circuits implemented on *Xilinx 7 series* FPGAs. This approach is based on the identification of the configuration bits likely to modify the topology of the circuit (the sensitive bits). The criticality of each sensitive bit is then evaluated by determining the conditions of the fault activation and propagation. Activation conditions refer to the conditions applied to the input signals of the affected resource that reveal the fault through an alteration of its output value. As for propagation conditions, they refer to the conditions, applied to the input signals of the different logic gates crossed to reach the system's outputs, enabling the propagation of the error over the logic gate (not logically masked). These conditions are confronted with the actual values of the design signals, extracted from behavioral simulation, to determine if the fault is critical (propagated to the outputs) or transparent (logically masked). As a result, this approach allows both to identify the number of critical bits for any circuit but also to identify the predominant sources of failures and the sensitive points of the design. This tool can then be used to assess the reliability of a design (its cross section) by multiplying the number of critical bits to the bit cross section (as performed with fault injection) and to evaluate the effectiveness of hardening techniques based on error masking.

One of the contributions of this study is to establish a complete CRAM related failure model of the *7 series* FPGA architecture. This failure model has been established and detailed thanks to reverse engineering results of the bitstream composition realized within the framework of the Project X-Ray [165] as well as to the use of localized fault injection. Localized fault injection refers to emulation-based fault injection targeting specific configuration bits of known functionality (thanks to decoded bitstream information). The other major contribution of this this work is the integration of the circuit's workload in the evaluation of the criticality of the bits. The circuit's workload refers to the set of input signals injected into the circuit in real conditions. Based on behavioral simulation and user-defined testbenches, the state of each net of the circuit and their evolution are extracted and integrated in the analysis to determine if the activation and propagation conditions are respected. Given the increasing complexity and size of circuits implemented on FPGAs, reliability analysis must rely on software tools to automate the netlist parsing, the analysis of the circuit and the subsequent reliability calculations. The approach presented in this study is based on the RAPIDWRIGHT Application Programming Interface (API) [166], an open-source platform developed by the *Xilinx* Research Labs. The workflow of this software is described in Figure 123.

Firstly, the state-of-the-art bitstream reverse engineering techniques and SEE susceptibility analysis tools are presented as well as Computed Aided Design (CAD) tools and APIs. Then, the methodology used to determine the failure model based on the results of bitstream reverse engineering and those from localized fault injection are presented while detailing the established failure model for each resource type. Then the structure of the developed software and the algorithms used to browse the circuit netlist and extract the critical bits are presented. Finally, the efficiency of the methodology is evaluated by comparing its results with those of SEEs testing and fault injection campaigns presented in chapter 4.

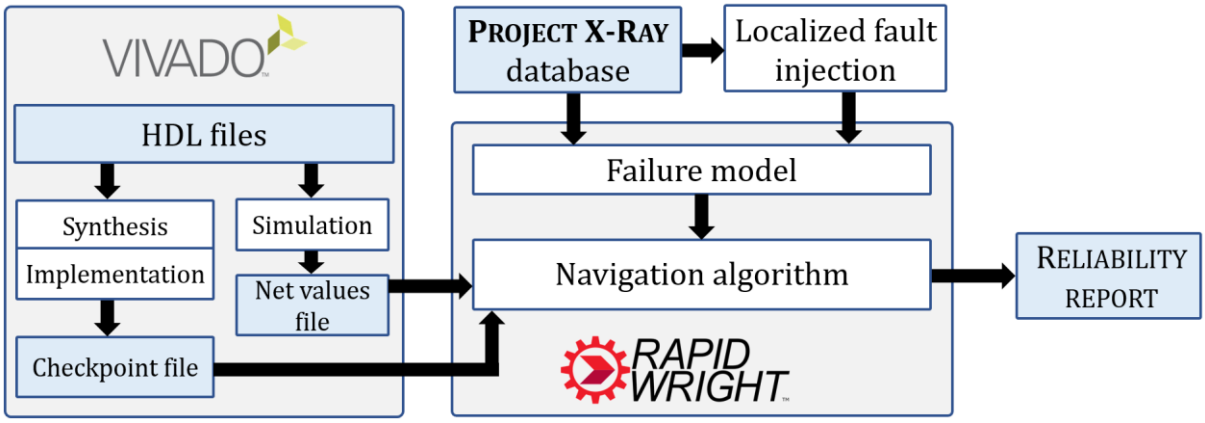


Figure 123: Workflow of the susceptibility analysis software. The circuit description files and its workload are extracted from the Xilinx EDA tool (Vivado) and imported in the software. The susceptibility analysis is based on a failure model defined thanks to the decoded bitstream database from [165] and thanks to localized fault injection. Based on this failure model, the software (that use the RAPIDWRIGHT API [166]), navigate through the circuit netlist and extract the set of critical bits of the design.

5.1. STATE OF THE ART SEE SUSCEPTIBILITY PREDICTION TOOLS

5.1.1. BITSTREAM REVERSE ENGINEERING

The knowledge of the bitstream composition is a key point to understand how the corruption of the configuration memory can alter the topology of any implemented circuit. For confidentiality and security reasons, the composition of the bitstream is not shared with the end user, nor is the detailed composition of the various primitives (transistor-level schematics, layout, etc.). The in-depth study of CRAM related failures must therefore rely on reverse engineering techniques to decode the bitstream. This involves determining the role of each of the configuration bits: the type of resource concerned, their position in the FPGA fabric and their contribution to the definition of the resource's functionality. Each FPGA having its own configuration memory architecture, this reverse engineering process must be renewed for each FPGA. However, FPGAs of the same family share a common logic block structure. Even if the number of logic blocks and their arrangement within the fabric differ, reverse engineering results for one device of the family can provide insight into how the different resources are configured. The establishment of a valid failure model is thus enabled, and it can be applied to all other devices of the family. However, for non-decoded devices, the exact memory address of a particular configuration point cannot be identified.

All bitstream decoding techniques [165], [167]–[173] are based on the same principle. A set of simple designs are generated by varying different parameters such as the type of resources instantiated, the way they are configured, their position in the FPGA fabric, the routing points utilized, etc. The bitstream of each design is generated and the resulting binary files are compared to make the links between the resources present or absent and the bits activated in the bitstream. Assuming that the configuration data layout is highly regular, in accordance with the regular structure of the fabric (the relationship between the configuration bits and corresponding hardware resource is coherent across the different logic blocks.) the information gathered from different sites can be cross-correlated and the results obtained can be extrapolated to rest of the fabric.

The detailed algorithms used to identify the relationship between the configuration bits and the corresponding hardware functionality are out of the scope of this study. Nevertheless, their basic working principle can be briefly illustrated by taking the example of LUT content identification [172]. A set of simple design integrating one LUT is generated while using a different LUT truth table for each design. The bitstreams are generated, and the bits that differs across the different design are identified. The relationship between LUT content and bitstream be established by comparing the LUT content patterns and the values of these configuration bits.

Different bitstream decoding algorithms have been proposed in the literature, targeting different FPGA families. In this study, the failure model has been established thanks to the results from **PROJECT X-RAY** [165] focused on *Xilinx 7 Series* FPGAs.

At a higher level of granularity, different software tools have been developed to facilitate the manipulation of the bitstream content such as JBITS [174], BITMAN[175], COMET [176] and PYXEL [177]. These tools rely on a partial knowledge of the bitstream but allow to establish the relationship between the physical blocks of the FPGA fabric and the associated memory areas. These tools not only contribute to the bitstream decoding progress but also allow to inject faults by targeting some physical areas of the component or even to modify the content of the LUTs or the memory blocks without repeating the whole synthesis and routing process.

5.1.2. VULNERABILITY ANALYSIS

To build an efficient SEE susceptibility prediction tool, the vulnerability model of the studied FPGA must be clearly identified. This involves analyzing the type of architectural modification that can be caused by an error on each type of configuration bit. Some configuration bits, for example the LUTs contents, have a relatively obvious functionality whose effect on the circuit can be deduced simply from the bitstream decoding and the manufacturer's documentation. Others, however, may exhibit behavior when corrupted which is much more difficult to predict, especially those related to routing resources.

Different approaches can be used to study the architectural modification induced by CRAM corruption. In [178], a combination of radiation testing and simulation-based fault injection are proposed. During the radiation test, the implemented structure is continuously monitored. When a permanent error is detected, the configuration memory is readback to identify the position of the corrupted bits. For each recorded SEU, the bitflip is reproduced by injecting a fault in the Native Circuit Description file (NCD). With the component used in this study (*Virtex XCV300*), a tool was provided to convert the NCD file into a behavioral description of the circuit. The erroneous circuit could then be simulated to evaluate the effect of the bitflip on the circuit behavior. Modern FPGAs can no longer rely on this type of bitstream to HDL converters and other approaches must be considered. In [154], the E^2STAR tool, an extended version of STAR [179], is used to identify the electrical effects of PIP's configuration bit corruption. Emulation-based Fault Injection (FI) is performed targeting specific PIPs instantiated in the design (localized FI) and the corresponding outputs are monitored. Local faults are then simulated in a behavioral model of the same application and the simulated outputs are compared to the one extracted from emulation to identify the logical behavior of the routing faults. Similarly, in [177], the PyXEL tool is used to identify the electrical effect of PIP open faults (the PIP is disabled) and conflict/bridge faults (a PIP is enabled creating a new connection between two distinct nodes). Conflict faults effects were

identified by using a design with two cascading 8-bit registers driven by a UART receiver and feeding a UART transmitter. The first two bits signal between the cascaded registers are routed through a target switch matrix using two specific PIPs. A conflict fault is emulated through localized fault injection by enabling a PIP that connects the two nets. By injecting different stimulus and monitoring the output signal through UART, the electrical behavior of the conflict fault could be identified.

An approach similar to the one proposed in [177] will be used in this study with the objective of establishing a more comprehensive model of the PIPs.

5.1.3. SEE SUSCEPTIBILITY PREDICTION TOOLS

Once the failure model is clearly established, the challenge of these prediction tools is to identify, for a given circuit, all the faults that can alter its functionality. Several approaches are proposed in the literature [167], [179]–[184].

The most straightforward approach, as proposed in [167] and [184], is to simply consider as critical, any configuration bit that either enables a PIP used by one of the circuit's net, or enables a PIP creating a bridge between two existing nets or defines the content of LUTs instantiated in the circuit. However, this approach does not take into account the fault activation and propagation which can mask a large proportion of errors. It can therefore suffer from a strong overestimation of the number of critical bits, especially for circuits hardened by TMR-type logic masking techniques.

STAR [179] and VERI-PLACE [181] are tools aimed to assess the impact of SEU faults in configuration memory using a static analysis with an emphasis on TMR protected circuits. The algorithm uses the circuit and layout description files of the mapped design and translates it into a graph-based representation. The vertices of this graph representation correspond to logic blocks and input/output ports of switchboxes while the edges correspond to wires and PIPs. The graph is then colored with a different color for each TMR partition and for majority voters. All possible SEUs affecting the graph's vertices (logic resources) are propagated to the circuit's outputs vertices. If the propagation tree affects more than one graph coloring, the fault is considered as a potential violation of the TMR scheme. By exploiting the relationship between configuration-memory's bits controlling the routing resources and the routing-graph's edges, a new routing graph is computed for each possible SEU affecting routing resources. A propagation tree is computed for each vertex of the new graph to check if two partitions of the TMR schema are somehow included in a common propagation tree. These tools have been validated experimentally as an efficient way to identify single points of failures in a TMR hardened circuit. The VERI-PLACE tool, developed by the same authors, is also supplemented with additional features by analyzing the effect of SEU accumulation in the CRAM. The tool performs a similar topological analysis considering all the possible configuration memory modification with a given number of bit flips accumulated. This procedure can be repeated with a large number of different bitflips combination. The error rate for a particular number of accumulated bitflips can thus be computed by dividing the number of errors (according to the topological analysis) by the number of accumulations performed. By repeating this error rate estimation with different number of bitflips accumulation, an optimal scrubbing rate can be defined. However, to the best of the writer's understanding, beside the TMR related considerations, no error masking mechanism is taken into

account to ensure that the SEU-induced topological modification actually results in alteration of the output signals. Without these considerations, the number of critical bits can commonly be overestimated. Furthermore, the effectiveness of mitigation techniques that were not specifically addressed in the analysis cannot be evaluated.

To take into account the logical masking in the evaluation of the criticality of bits, a statistical approach is proposed in [183], [185], [186] based on error propagation probabilities. For each node in the design, the signal probability *SP* is computed using the synthesized netlist. The signal probability is defined as the probability that this particular net is at '1' state. By considering that all primary inputs have an equal probability of being '1' or '0' *(SP=0.5)*, the signal probabilities can be propagated to internal nets using the propagation probabilities of the crossed logic gates. For example, the output of an AND gate is '1' only when both inputs are '1' simultaneously. Therefore, the output of an AND gate with two inputs with a signal probability of 0.5 have a signal probability of 0.25 (0.5×0.5). The signal probability is thus computed for every node in the design. Each node in the circuit is then weighted by a failure probability based on the number of configuration bits controlling that node (e.g. number of PIPs likely to create an opening or short fault). Similarly, the failure probability of LUT outputs is computed as the sum of each LUT content configuration bits error rate weighed by its activation probability (based on input signal probabilities). The structural paths from each error site to all reachable outputs are then extracted and the associated propagation probabilities are computed. These propagation probabilities used the signal probability of the inputs of every crossed logic gate in the structural path to statistically asses the probability that the error generated in this particular node reach one of the primary outputs. The global system failure rate is finally calculated by integrating the failure probabilities of each node weighted by the associated error propagation probabilities. This technique could be effective in estimating an average error rate without considering a specific workload of the circuit (set of input vectors used). However, beyond the fact that this technique has not yet been extended to more recent FPGAs, a more accurate estimation of the reliability of the circuit under real operating conditions could be obtained by considering user-defined usage scenarios. This would allow to check if a failure on a particular node in the circuit actually (not statistically) propagates to one of the outputs for this particular usage scenario. Indeed, the proportion of topological modifications of the circuit that actually cause an alteration of the output signals is very dependent on the circuit's workload.

5.1.4. CAD TOOLS AND APIS FOR FINE-GRAINED CIRCUIT MANIPULATION AND ANALYSIS

The analytical approach developed in this study would never have been possible without CAD tools to parse and manipulate the netlist files generated by the design software (Vivado for *Xilinx* FPGAs). Different tools have been developed in this sense [166], [187]–[190], targeting different FPGA families and different functionalities. The most comprehensive for *Xilinx* new generation FPGAs (the one used in this study) is called RAPIDWRIGHT [166]. It is a Java API developed by *Xilinx* Research Labs that enables logical and physical netlist manipulation. The checkpoints files generated by Vivado can be directly parsed using built in functions of the API. Java being an object-oriented programming language, all the elements that compose the FPGA fabric and the different instances of elements used by the imported design, both at the logic and physical level, are represented by objects using a terminology and a hierarchical representation similar to the one used by Vivado [191]. This API offers many features for modifying imported designs such as

automatic routing tools, possibilities to modify user data as well as features for relocating partial bitstreams to other locations. The terminology and structure of this API is be briefly presented here, focusing on *7 series* FPGA architecture and the classes used to develop the proposed analytical approach.

The RAPIDWRIGHT API is based on three main packages of cross-connected objects. The `Device` package contains all the classes to represent the hardware construct of the FPGA fabric at different level of hierarchy as shown in Figure 124. Among the class used, the `Device` class itself is the higher-level representation that contains all the hardware information of a particular FPGA chip. Each device is made of a grid of different `tiles` which can be composed either of two BRAM, two DSP blocks, two regular `sites` or a switchbox. A `site` is a collection of Basic Element of Logic (`BEL`). A regular `site` directly corresponds to a slice (4 LUTs, 1 CARRY4, 8 flip-flops, 3 logic multiplexers). The `BEL` is the lowest level description of the resources (primitives) of the FPGA. This class is used to represent logic elements (LUT, CARRY, multiplexers, flip-flops, buffers, IO, DSP, BRAM etc.), routing elements (static intra-`site` multiplexers) and `sitePorts`. The routing structure is described with `Wire`, `SiteWire`, `Node`, and `PIP` objects as well as `SitePin` and `BelPin` objects as shown in Figure 125.

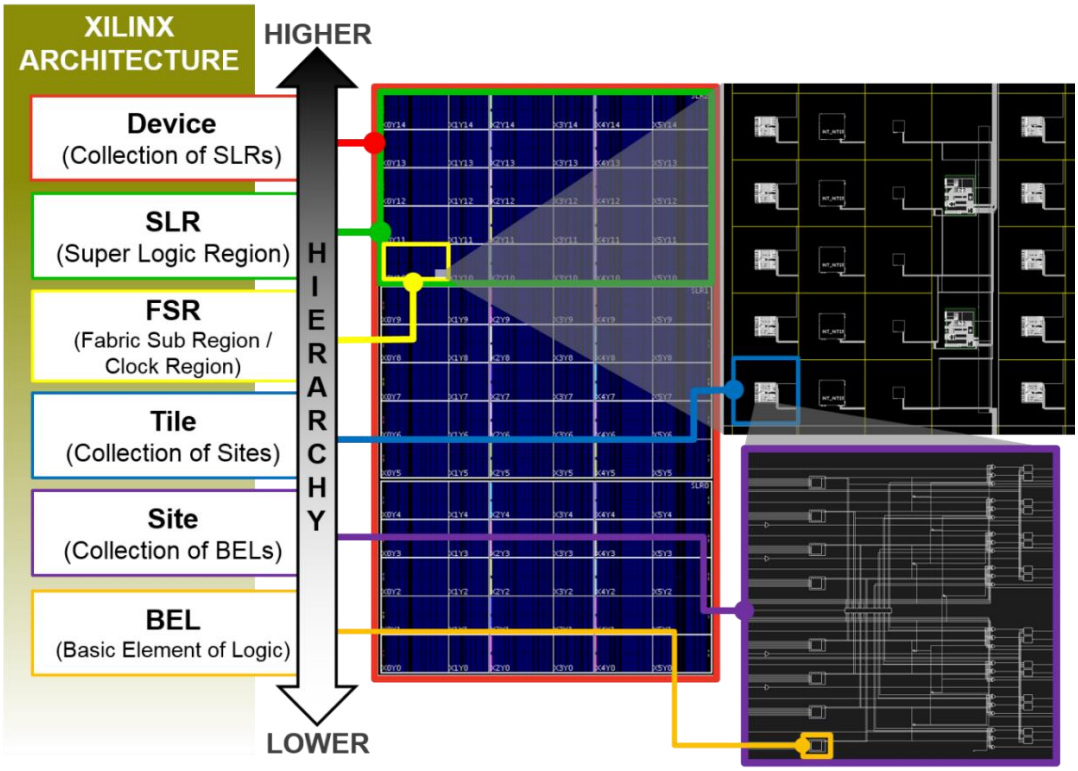


Figure 124: Hierarchical representation of the *Xilinx* FPGA Fabric, from [192]

The `EDIF` package contains all the information related to the logical netlist of the circuit. Every logic element and hierarchical modules are represented by cells through the `EDIFCell` class. The cell input and output ports are represented by the `EDIFPort` class while the net used to connect these ports together are represented by `EDIFNet`. The `Design` package is used to describe how the logical netlist maps to the physical device netlist. The `Net` class is the physical representation of `EDIFNet` from which the list of `Wires`, `Nodes`, `Pips` or `SitePin` can be extracted. The `Cell` class links the logical `EDIFCell` to their physical representation (`BEL`). Additional information on the *Xilinx* FPGA terminology and the RAPIDWRIGHT hierarchical organization can be found at [191], [192].

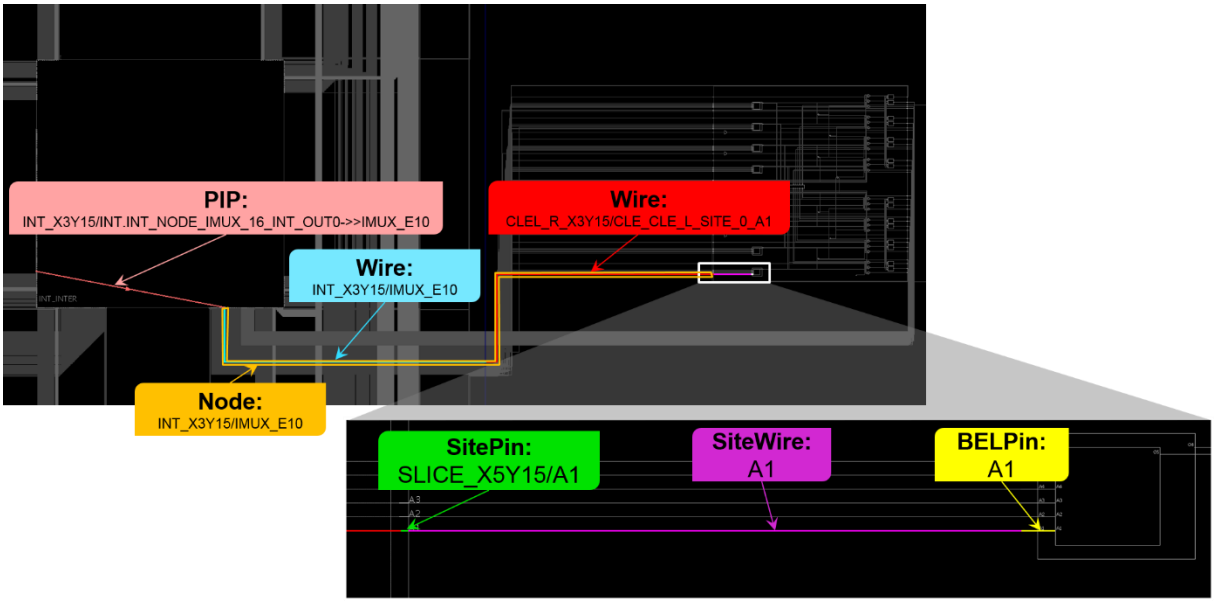


Figure 125: Routing architecture and terminology. `Wires` are portion of metallic tracks. `Nodes` are collections of `wires` that are electrically connected together. `PIPs` are programmable routing segments contained in switchboxes, `SitePin` represent the interface pin between a `siteWire` and an extra-slice `Wire` and `BELPin` represent the physical pin of a `BEL`. From [192].

A summary of all the previously mentioned tools for bitstream decoding, bitstream manipulation, vulnerability analysis and SEE susceptibility estimation as well as the CAD tools and APIs for low-level manipulation of netlist is shown in Figure 126.

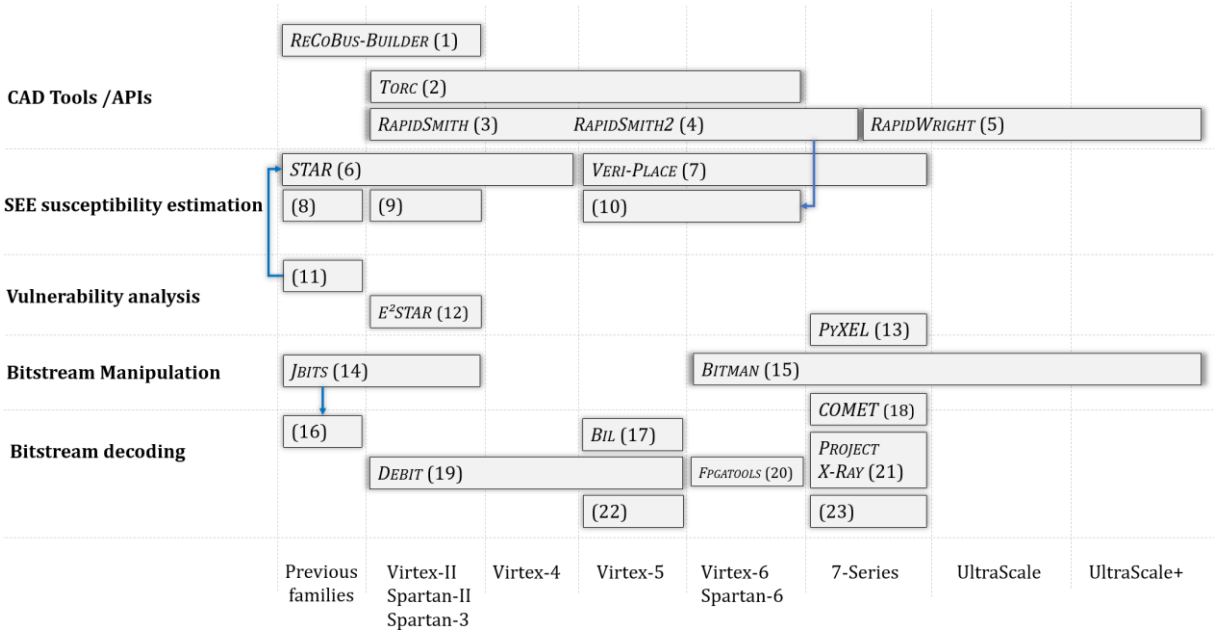


Figure 126: Review of the various third-party tools for bitstream decoding or manipulation, vulnerability analysis and software error susceptibility estimation as well as CAD tools and APIs for fine-grained circuit manipulation and analysis. These tools are categorized according to the FPGA architecture on which they have been evaluated (here only on *Xilinx* FPGAs) but most of these methods can be extended to other FPGA architectures. References: (1)→[187]; (2)→[188]; (3)→[189]; (4)→[190]; (5)→[166]; (6)→[179], [193]; (7)→[181]; (8)→[167]; (9)→[182], [185], [186]; (10)→[184]; (11)→[178]; (12)→[154]; (13)→[177]; (14)→[174]; (15)→[175]; (16)→[173]; (17)→[169]; (18)→[176]; (19)→[168]; (20)→[171]; (21)→[165]; (22)→ [170]; (23)→[172].

5.2. FAILURE MODEL ESTABLISHMENT

Failure models related to configuration memory corruptions have been widely documented for older FPGA architectures. However, the innovations introduced with the new generations of FPGAs have significantly modified the structure of the configuration memory. No failure model is exhaustively described in the literature for the *Xilinx 7 Series* FPGAs to the best of the author's knowledge. To build an SEE susceptibility analysis tool, the failure model must be completed. In this study, the *Xilinx* documentation crossed with the results of the bitstream decoding of the *Artix7* FPGA performed in the framework of the PROJECT X-RAY is used to identify the role of each of the configuration bits and to understand the type of faults that can be induced when they are corrupted. For some configuration bits, especially those related to PIPs, the effect of the SEU on the functionality of the circuit is difficult to predict. Thanks to bitstream decoding results, the memory address of each configuration bit can be extracted. Localized fault injections can then be performed to identify these specific effects as well as to validate the overall failure model.

In this section, a description of the configuration memory of the *7 series* FPGAs is made at first, outlining how to extract the address of the different memory points. Then, the procedure used to inject faults targeting specific memory location to identify the electrical behavior of the errors related to the routing resources is described. All the elements of the FPGA fabric (switchbox, intra-slice routing multiplexers, logic multiplexer, CARRY4, LUTs, flip-flops, SRL, BRAM) are finally reviewed to identify their respective failure model and their error propagation conditions.

5.2.1. DECODED BITSTREAM DATABASE

The configuration memory of *7 Series* FPGAs is composed of frames of 101 words of 32 bits. The configuration bits defining the functionality of a specific tile are contained in words distributed over consecutive frames (with one or two words per frame). Two main file types from the PROJECT X-RAY database are used to extract the address of a specific configuration bit.

A device specific file named *tilegrid.json* provides addressing information of every tile in the fabric:

- **Base address**: correspond to the physical address (in hexadecimal) of the first frame that contains configuration bits related to the tile.
- **Offset**: correspond to the number of words to be skipped before reaching tile-related configuration bits.
- **Words**: the number of words used to define the tile functionality.

A set of generic files for every Artix7 FPGAs, named *segbits_*tile_type*.db* that provide the detailed content of each tile type. Each bit is provided with two address information:

- **Frame index**: correspond to the number of frames to skip, starting from the base address**,** to reach the configuration bit.
- **Bit Position**: correspond to the index of the bit in the word. For some tiles, the bit position is contained between 0 and 63. As each word contains only 32 bits, for bit positions higher than 31, the frame index should be incremented and 32 should be subtracted to obtain the real bit index.

Based on this information the Physical Frame Address (PFA) of each bit can be determined as shown in Figure 127. This PFA can be fed to the SEM controller for fault injection purposes.

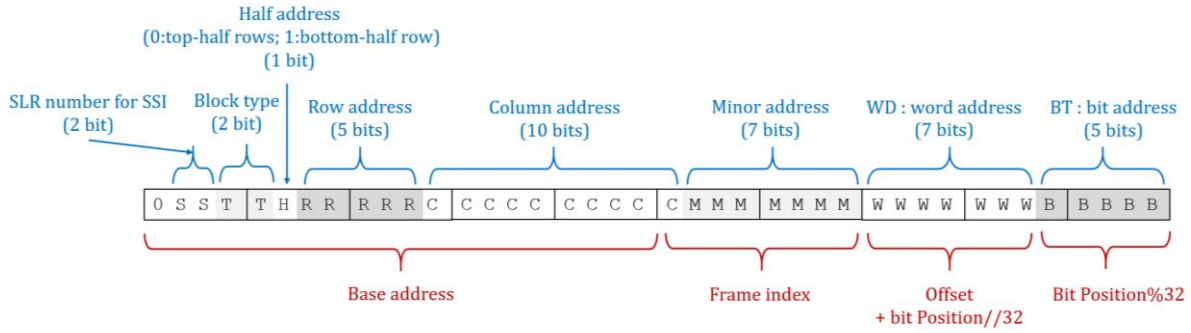


Figure 127: Physical Frame Address (PFA) composition. The base address, frame index, offset and bit position from the decoded bitstream database can be used to determine the PFA of each configuration bit.

5.2.2. LOCALIZED FAULT INJECTION

To analyze the effects of bitflips in the PIPs configuration bits, a dedicated design based on the same principle as the one proposed in [177] is used. A set of LUTs, hosting two buffers per LUT, are manually placed on the same tile (a pair of slices, as defined in section 2.2.2). With 4 LUTs per slice, a total of 16 buffers is instantiated in the same tile. The input and output signals are all routed through the same switchbox thus maximizing the switchbox occupation and the bridge possibilities as shown in Figure 128.

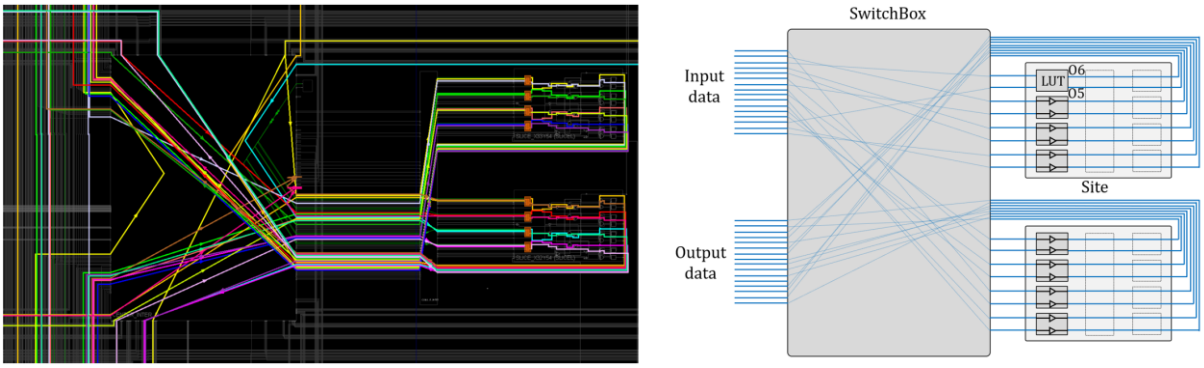


Figure 128: PIP open and bridge fault testing structure. The two sites associated with a switchbox are fully populated with LUTs configured as buffers, with two buffers per LUT. All 16 input and output signals are routed through the same switchbox.

To test the bitflip effects, all possible combinations with two bits activated simultaneously (120 combinations) are sent sequentially to the input of the structure. The injection flag of the SEM controller is used to trigger the beginning of the test. The output signals of the structure are continuously compared to the input signals. When a mismatch is observed, the corresponding input signal is sent to the control computer via UART as shown in Figure 129. Faults are injected independently one after the other on all the configuration bits of the switchbox. For each injected fault, the input vectors for which a mismatch is detected are recorded and analyzed to determine the logical effect of the bitflips.

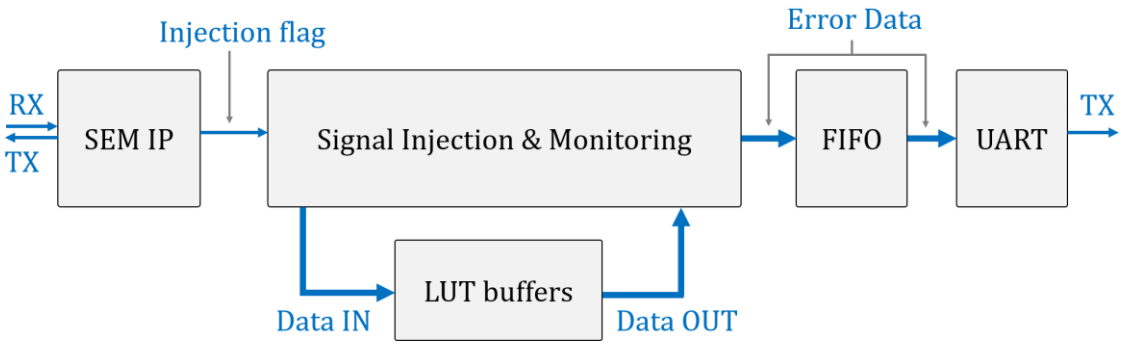


Figure 129: BIST test structure for PIP open and bridge fault analysis.

5.2.3. SWITCHBOXES

The switchboxes are the core of the FPGA signal routing network. These routing boxes link 588 different nodes including 172 sink nodes.

5.2.3.1. FAILURE MODEL

By crossing the results from the bitstream decoding and from the localized fault injection, an equivalent model of the switchbox is built. It appears that every sink node can be connected to 16, 20 or 24 input nodes using static multiplexers (driven by configuration memory bits). These multiplexers are totally independent in the sense that all multiplexers are driven by distinct sets of memory bits and in the sense that the state of a multiplexer cannot impact the state of other multiplexers even if they share common nodes. For example, when two input nodes of a multiplexer are connected to each other by a bridge fault, the conflict does not propagate back to the input nodes but only appears on the output of the multiplexer. This means that a single bit upset (SBU) cannot corrupt two distinct nets simultaneously (no domain crossing possibilities on TMR circuits).

Through this fault injection and the bitstream analysis, two types of multiplexers have been identified based on the way they are controlled by CRAM bits. An equivalent model of these multiplexers is then built as shown in Figure 130 and Figure 131.

Multiplexers are driven by either 8 bits (16 input nodes), 9 bits (20 input nodes) or 10 bits (24 input nodes). This set of configuration bits is divided in two stages, and the set of input nodes are divided in 4 or 5 groups of equal size. In the first stage, each configuration bit drives a pass-transistor for one of the input nodes of each group. All the nodes of the same group are driven by different configuration bits and are then electrically connected together resulting in one junction per group. The resulting junctions are then multiplexed to the output through a second stage of configuration bits. In the first multiplexer type, each junction goes through another layer of pass-transistor directly driven by a different configuration bit. As for the second multiplexer type, the way the pass transistors are driven by configuration bits is slightly different. This behavior is identified by extending the localized fault injection campaign by accumulating several bitflips on this set of configuration bits to test all possibilities.

PIPs driving INT_L.EE2BEG0

	Source Node	1st stage B_0	B_1	B_2	B_3	B_4	2nd stage B_5	B_6	B_7	B_8
0	NL1END0	1	–	–	–	–	1	–	–	–
1	EL1END0	–	1	–	–	–	1	–	–	–
2	ER1END0	–	–	1	–	–	1	–	–	–
3	NR1END0	–	–	–	1	–	1	–	–	–
4	LGC_OUT12	–	–	–	–	1	1	–	–	–
5	NE6END0	1	–	–	–	–	–	1	–	–
6	SE6END0	–	1	–	–	–	–	1	–	–
7	SS6END0	–	–	1	–	–	–	1	–	–
8	NN6END0	–	–	–	1	–	–	1	–	–
9	EE4END0	–	–	–	–	1	–	1	–	–
10	NE2END0	1	–	–	–	–	–	–	1	–
11	SE2END0	–	1	–	–	–	–	–	1	–
12	SS2END0	–	–	1	–	–	–	–	1	–
13	NN2END0	–	–	–	1	–	–	–	1	–
14	EE2END0	–	–	–	–	1	–	–	1	–
15	LGC_OUT22	1	–	–	–	–	–	–	–	1
16	LGC_OUT4	–	1	–	–	–	–	–	–	1
17	LGC_OUT0	–	–	1	–	–	–	–	–	1
18	LGC_OUT18	–	–	–	1	–	–	–	–	1
19	LGC_OUT8	–	–	–	–	1	–	–	–	1

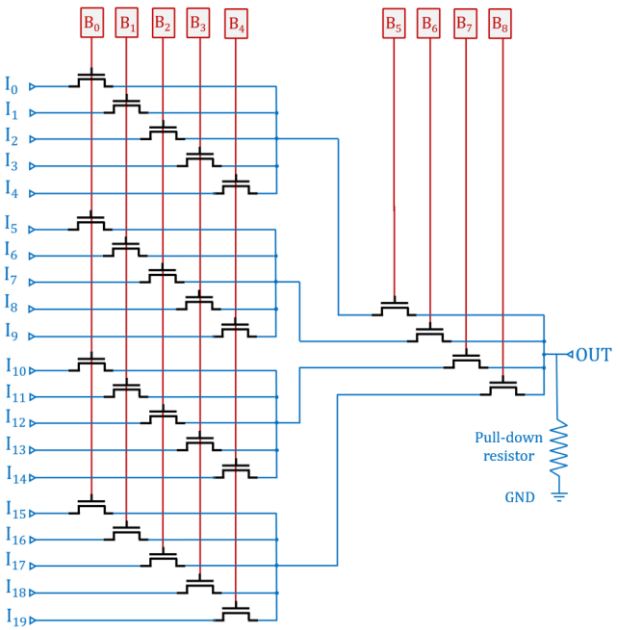


Figure 130: Truth table (left) and equivalent model (right) of configuration memory control over switchbox's multiplexers (first type).

PIPs driving INT_L.CTRL_L1

	Source Node	1st stage B_0	B_1	B_2	B_3	2nd stage B_4	B_5	B_6	B_7
0	ER1END2	1	–	–	–	0	1	0	0
1	SE6END2	–	1	–	–	0	1	0	0
2	BYP_BNC4	–	–	1	–	0	1	0	0
3	SW6END1	–	–	–	1	0	1	0	0
4	WW4END2	1	–	–	–	0	0	0	1
5	EE4END2	–	1	–	–	0	0	0	1
6	FAN_BNC1	–	–	1	–	0	0	0	1
7	WR1END2	–	–	–	1	0	0	0	1
8	NE6END2	1	–	–	–	1	1	0	1
9	SR1END2	–	1	–	–	1	1	0	1
10	GFAN1	–	–	1	–	1	1	0	1
11	NR1END2	–	–	–	1	1	1	0	1
12	NN6END2	1	–	–	–	0	1	1	1
13	SS6END2	–	1	–	–	0	1	1	1
14	GFAN0	–	–	1	–	0	1	1	1
15	NW6END2	–	–	–	1	0	1	1	1

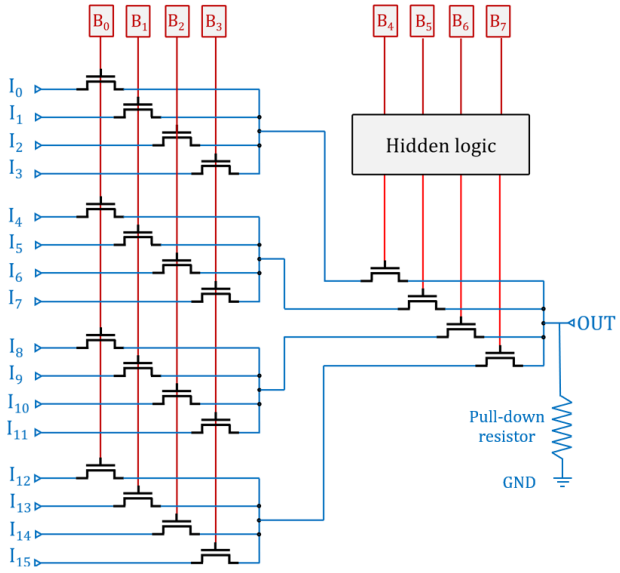


Figure 131: Truth table (left) and equivalent model (right) of configuration memory control over switchbox's multiplexers (second type).

The equivalent model that provides the best representation of the observed behavior for most combinations is given in Figure 132. Apparently, the configuration bits are simply reversed while two of them are inverted. This behavior is equivalent to a similar pass-transistor structure as the one used by first multiplexer type with the only difference of using two PMOS pass-transistors instead of NMOS. However, for some rare PIP junctions, the observed behavior for certain combinations does not match the proposed model. Further investigation should be conducted to define a more representative model for all cases.

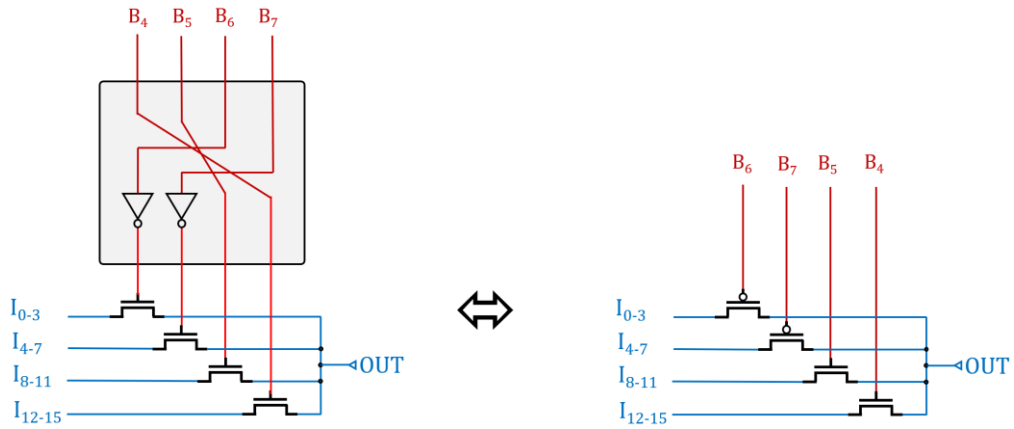


Figure 132: Configuration memory logic driving the pass transistors of the second multiplexing stage.

These multiplexer equivalent models reveal the different failure modes induced by SBU. Firstly, for each connection, the two configuration bits controlling the activated pass transistors are prone to open faults. Secondly, the bridge possibilities are limited to the nodes within the same groups (by activating one of the other configuration bits in the first stage) and to one node of each other group (the one that shares the same configuration bits in the first stage) by activating one of the other configuration bits in the second stage. For example, in Figure 130, if the first node is the one selected (I_0), configuration bits B_0 and B_5 are the only bits activated. These bits are prone to open faults. In addition, B_1, B_2, B_3, B_4, B_6, B_7 and B_8 can create bridge faults with nodes I_1, I_2, I_3, I_4, I_5, I_{10} and I_{15} respectively.

By injecting faults, one by one, on all the configuration bits on 50 different PIP junctions, the electrical effects are identified by analyzing the reported erroneous input vector as described in section 5.2.2. The observed electrical effect of open faults is a stuck-at-0 (~75% of cases) and stuck-at-1 (~25% of cases). Regarding bridge faults, when the bridged node is not occupied by any net of the design, no fault is observed. When the node is occupied by another net, the main electrical effect observed is a wired-AND between the two nets (>95% of cases). On some rare cases, a forced-by behavior is also observed (the output value is forced by the bridged node value). These observed behaviors are different from the one observed in [177] on the same component where only stuck-at-1 effects are reported for open faults and 57% of wired-AND, 40% of wired-OR, and 3% of forced-by effects are reported for bridge faults. This discrepancy is probably due to the fact that the technique used in [177] consists in modifying simultaneously several configuration bits to activate or deactivate the different connections while in this study, the faults are generated by the single error injection.

By default (all configuration bits set to 0), the second type of multiplexers are connected to VCC. When this is the intended configuration, no "open" faults are possible (VCC is equivalent to stuck-at-1), however, bridge faults are possible with the nodes of the first and second group by activating one of the pass-transistors of the first multiplexing stage.

Localized fault injection could be extended by using different designs to cover all connections and bridge possibilities, and by repeating the operation on switchboxes located at different places or even on components from different batches. This analysis would define if the fault behavior is deterministic for each type of connection or if the electrical effect depends on the variability of the manufacturing process, the supply voltage, the temperature, etc.

5.2.3.2. ACTIVATION CONDITIONS

At the level of the SEE susceptibility analysis software, for each PIP, a list of sensitive bits is created. Two sensitive bits are added for open faults (except for VCC nets) and the set of input nodes likely to create a bridge fault (the neighboring nodes) is browsed to check whether they belong to a net of the design or not. For each neighboring node actually involved in one of the design nets, a bridge type fault is added to the list of sensitive bits. From the database of the decoded bitstream, a text file is created describing for each possible connection in the switchboxes, all the neighboring nodes. At program initialization, this text file is parsed to create a map linking the name of each PIP to a list of neighboring nodes. This map is used during the analysis of the PIPs to extract the number of bridge faults.

At creation, every sensitive bit is associated with an activation condition. These are defined by the conditions on the input signal and eventually on the bridged net for which the output value is different from the one expected with the fault-free version of the PIP. The activation conditions for each failure mode are described in Figure 133.

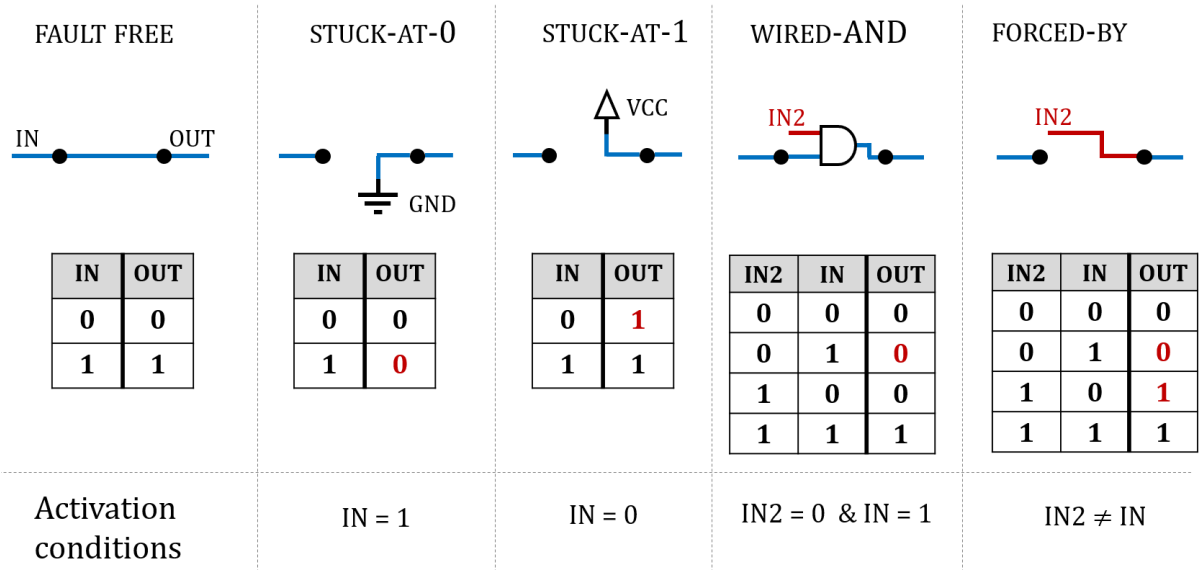


Figure 133: Activation conditions of PIP faults. Stuck-at-0 faults are activated when the input state is "1"; Stuck-at-1 faults are activated when the input state is "0"; wired-AND faults are activated when the input state is "1" and the bridged input state is "0"; forced-by faults are activated when the input net and the bridged net are in a different state.

As the fault behavior is not identical for all PIPs and not all fault possibilities could be tested, a conservative approach is currently implemented: open faults are activated unconditionally (regardless of the input state) while the bridge faults are activated when the input net and bridged net are in a different state. The real activation conditions can be implemented later if the fault behavior is exhaustively determined for all possibilities and if it turns out to be deterministic.

5.2.4. INTRA-SLICE ROUTING MULTIPLEXERS

Each slice of the FPGA fabric contains 25 routing multiplexers. These routing multiplexers are statically controlled by configuration bits to drive the different intra-slice signals to the different logic gates or slice outputs. These routing (static) multiplexers are to be distinguished from the logic multiplexers which are controlled by user signals as shown in Figure 134.

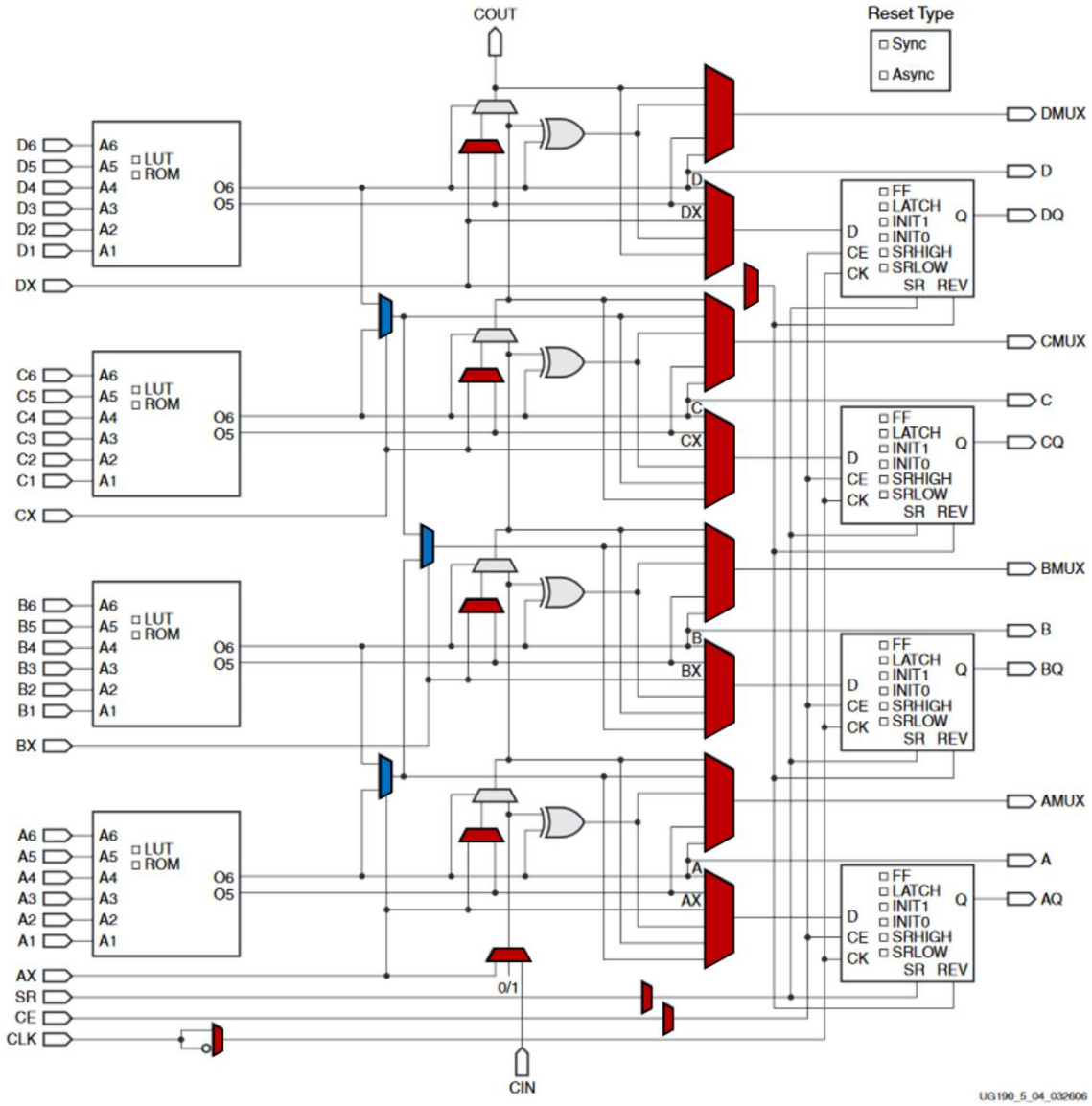


Figure 134: Routing multiplexers (red) and logic multiplexers (blue) in a slice of *Xilinx 7 Series* FPGA.

Among these routing multiplexers, some are used to configure the flip-flop control signals (clock, reset and clock enable). These signals are global for the whole slice, the corruption of these multiplexers induces a failure on all the flip-flops of the slice. This highlights the importance of not placing elements of distinct TMR partitions within the same slice to avoid the presence of single points of failure. The other multiplexers are used to drive the input signals of the CARRY4 logic gate from the output signals of the LUTs or to drive the flip-flop inputs and the slice outputs.

5.2.4.1. FAILURE MODEL

Different behaviors can be observed depending on the type of signals the multiplexer is driving as shown in Figure 135. Multiplexers associated with control signals and CARRY4 inputs use only one configuration bit while 2:1 multiplexers used to drive the input of part of the flip-flops use two configuration bits. The fault behavior depends on the type of multiplexer: for muxes driven by one configuration bit, the bitflip simply results in the exchange of the selected input, which can then be characterized as an inversion fault, a stuck-at-0, a stuck-at-1, or a forced-by bridge. Muxes driven by two configuration bits have two failure modes, open faults and bridge faults with the same behavior as switchbox pips.

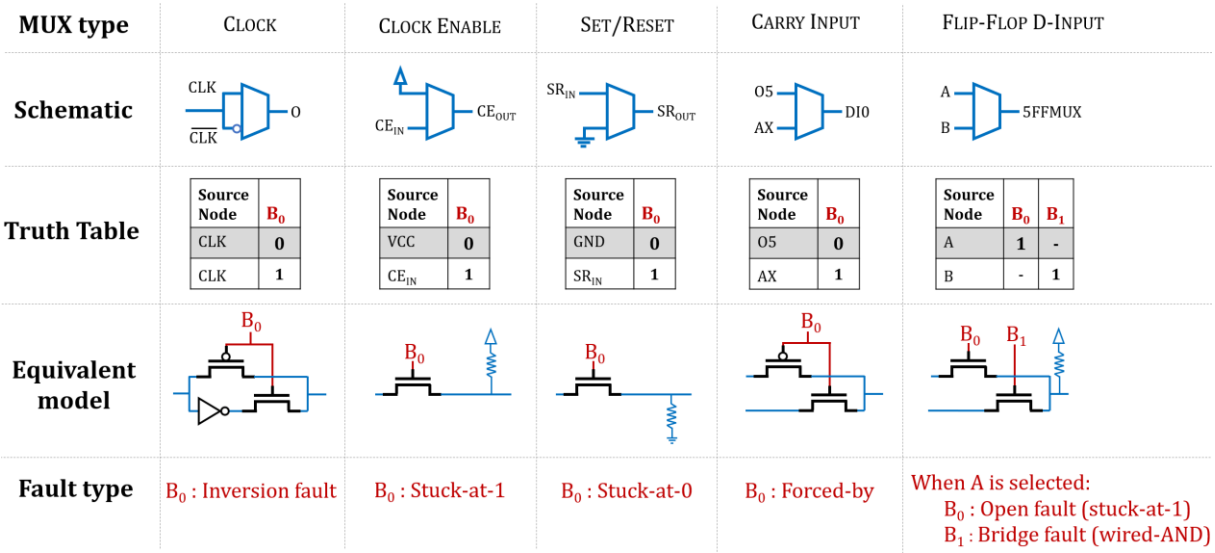


Figure 135: Failure model of intra-slice routing multiplexers. Multiplexers driven by one configuration bit are used to either inverse the clock polarity, drive a constant state (0 or 1) or multiplex two input signals. Multiplexers driven by two configuration bits are also used to multiplex two input signals.

Multiplexers 6:1 driving the combinatorial outputs and the input of flip-flops use a similar structure as the one used in switchboxes as shown in Figure 136.

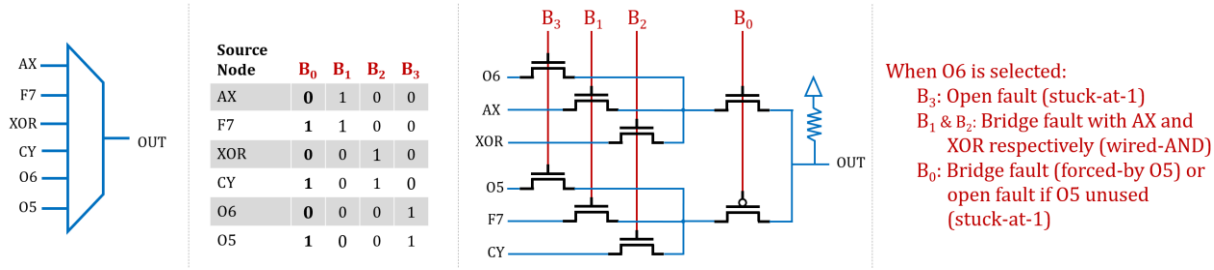


Figure 136: 6:1 multiplexers use 4 configuration bits, three of them (B_3, B_2, B_1) are used to select one of the 3 nodes in each group and the last configuration bit (B_0) is used to select one of the group junctions.

A main difference lies in the second multiplexing stage that uses a single configuration bit with a NMOS and PMOS pass-transistors to select one of the group junctions. This results in a slightly different failure model: whatever the selected input node, the activated pass transistor in the first stage can cause open faults and the other two configuration bits potentially induce bridge faults (wired-and) with the nodes from the same group.

The configuration bit of the second stage is always sensitive: it either results in a bridge "forced-by" fault with a node of the other group driven by the same configuration bit or an open fault (stuck-at-1) when this node is unused.

5.2.4.2. Activation conditions

The activation conditions depend on the type of the driven signal:

- Clock: the inversion of the polarity of the clock signal corresponds to a 180° phase shift for all the flip-flops in the slice, this will result to corruption of the latched data because the

flip-flops are not synchronized with the rest of the circuit. This fault is thus considered activated as soon as the clock signal is toggling and the clock enable is high.

- Clock Enable: a stuck-at-1 on the clock enable signal is activated when the input clock enable is at "0" state.
- Reset: similarly, stuck-at-0 on the reset signal is activated when the input reset is at "1" state.
- Data: like for switchbox pips, open faults are activated unconditionally while bridge faults (forced-by and wired-AND) are considered active when the input net state and the bridged net state are different.

5.2.5. LOGIC MULTIPLEXER

Each slice contains three logic multiplexers driving the LUT's output and controlled by extra-slice signal to build larger logic functions as shown in Figure 134. These multiplexers are not configurable, they do not add any configuration bit to the list of sensitivities but they can potentially mask the error propagated to their inputs.

Propagation conditions are defined as the conditions on the input signals for which an error on a particular input node modifies the output state. Considering A and B, the two data signals and S the control signal, the error on A can only propagate when A is driven to the output (S=0) and error on B when S=1. The error on S input induces a swap of the driven signal that can only propagate to the output when the two input signals are different.

The propagation conditions in the presence of error on multiple inputs must also be considered as they do not result from the combination of the single error propagation conditions. These propagation conditions are defined using the truth table of the logic gate. For each combination of inputs (A.B, S.A, S.B, S.A.B), every line of the truth table is paired with the line in which the values of the considered inputs are inverted. For example, if an error on A and B are considered, the line 0**00** is paired with 0**11**; 0**01** with 0**10**; 1**00** with 1**11** and 1**01** with 1**10**. If the output values of the paired lines are different, both input values are added to the propagation condition. The propagation conditions are then simplified using Boolean minimization techniques. Propagation conditions for both single input error and multiple input error are presented in Figure 137.

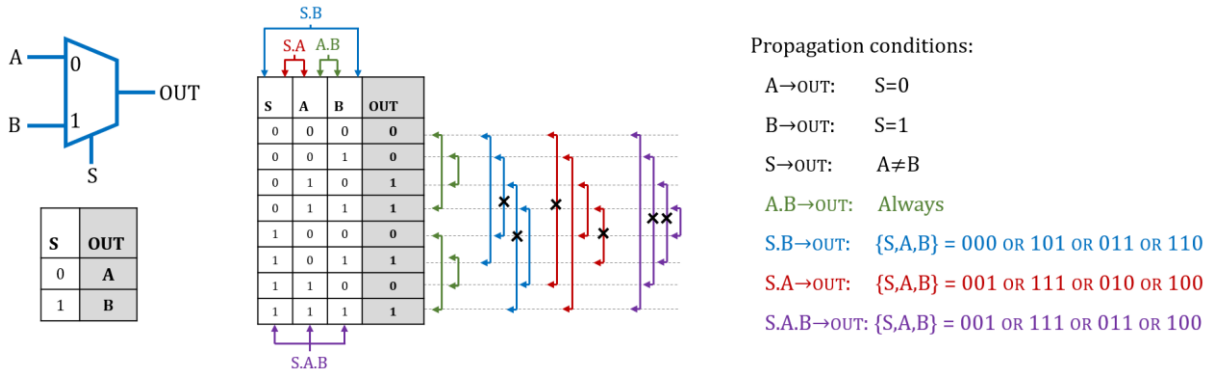


Figure 137: Propagation conditions of logic multiplexers. Double arrows represent the paired lines for a given erroneous input combination. Green stand for A and B affected simultaneously, blue for S and B, red for S and A and purple for all three inputs affected simultaneously. When the output values of the paired lines are different, the two corresponding input values are added to the propagation condition.

5.2.6. CARRY LOGIC

Just like logic multiplexer, the CARRY4 logic gate is non-configurable. No configuration bits are added to the sensitivity list but its logical masking capabilities must be considered in the propagation conditions.

CARRY4 cells use 9 inputs and 8 outputs. The number of possible combinations of errors on multiple inputs is too large to manually determine the propagation conditions for all cases. The regularity of the structure is then used to decompose the resource into subparts corresponding to the propagation of a single carry bit as shown in Figure 138.

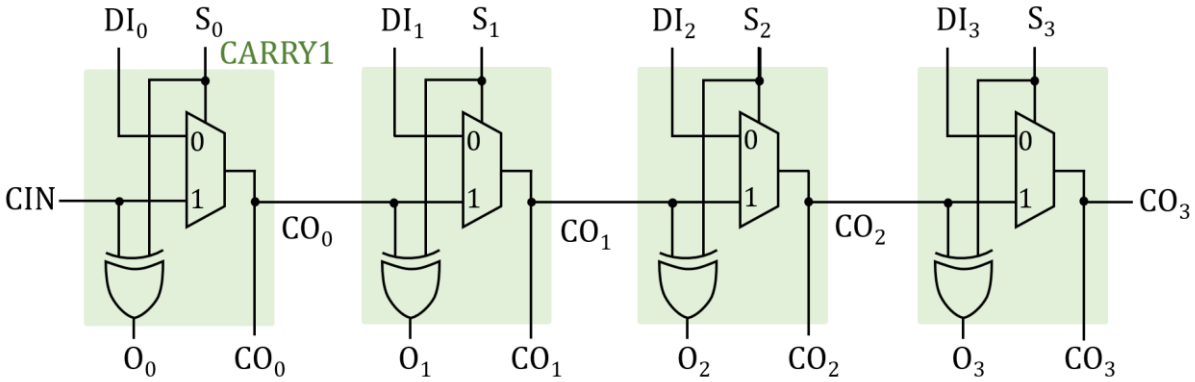
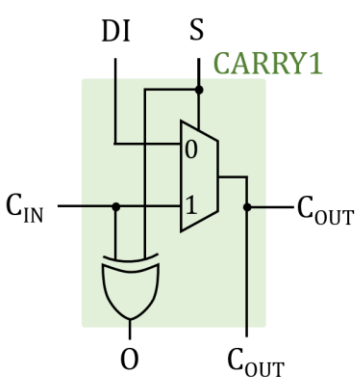


Figure 138: CARRY4 decomposition in four identical CARRY1 cells composed of one XOR gate and one multiplexer.

The propagation conditions of these sub circuits are much simpler to define. Those of the multiplexer have been previously defined, and those of the XOR gate are straightforward: errors present in only one of the inputs propagate unconditionally while errors present in both inputs never propagate. The propagation conditions of the CARRY1 cell are detailed in Figure 139.



S	DI	C_{IN}	C_{OUT}
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

S	C_{IN}	O
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Propagation conditions:

C_{IN}	$\rightarrow O$:	Always
S	$\rightarrow O$:	Always
$S.C_{IN}$	$\rightarrow O$:	Never
C_{IN}	$\rightarrow C_{OUT}$:	S=1
DI	$\rightarrow C_{OUT}$:	S=0
S	$\rightarrow C_{OUT}$:	$C_{IN} \neq DI$
$C_{IN}.DI$	$\rightarrow C_{OUT}$:	Always
$S.C_{IN}$	$\rightarrow C_{OUT}$:	$\{S,DI,C_{IN}\}$ = 000 OR 101 OR 011 OR 110
S.DI	$\rightarrow C_{OUT}$:	$\{S,DI,C_{IN}\}$ = 001 OR 111 OR 010 OR 100
$S.DI.C_{IN}$	$\rightarrow C_{OUT}$:	$\{S,DI,C_{IN}\}$ = 001 OR 111 OR 011 OR 100

Figure 139: CARRY1 propagation conditions.

Instead of computing the propagation conditions for the full CARRY4 chain, these conditions are computed bit by bit, starting from LSB and transmitting those resulting from the C_{OUT} port to the C_{IN} port of the higher order CARRY1.

5.2.7. LOOK UP TABLES

Slices in *7 Series Xilinx* FPGAs are equipped with 4 LUTs with 6 inputs and 2 outputs. These LUTs are configured by 64 configuration bits, allowing the generation of any arbitrary 6-inputs

logic functions when using only one output (O6). These LUTs can also be configured as two 5-inputs LUTs (or smaller) sharing the same inputs as shown in Figure 140 and Figure 141. LUTs are the core of the logic operations performed within an FPGA. They must be considered both for their intrinsic sensitivity and for the logical masking of errors.

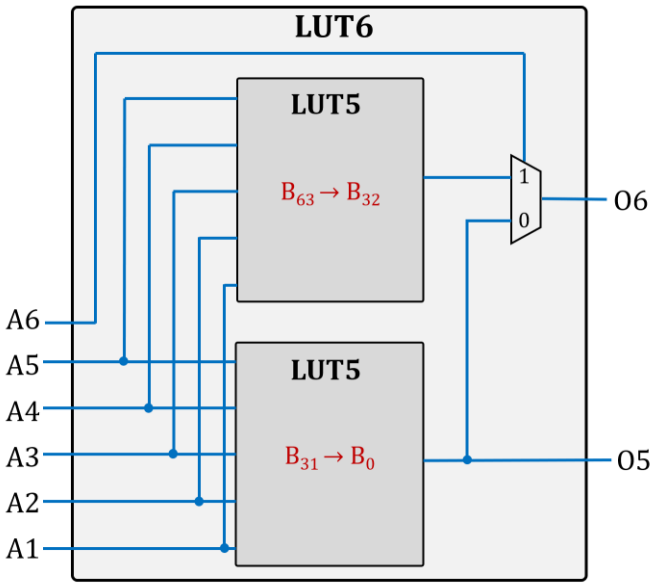


Figure 140: LUT6 structure inside *7 series Xilinx* FPGA. These LUT are actually composed of two 5-inputs LUTs configured by 32 distinct configuration bits. These LUT can be used to generate a 6-input LUT using only O6 output or as two 5-inputs LUT sharing the same inputs.

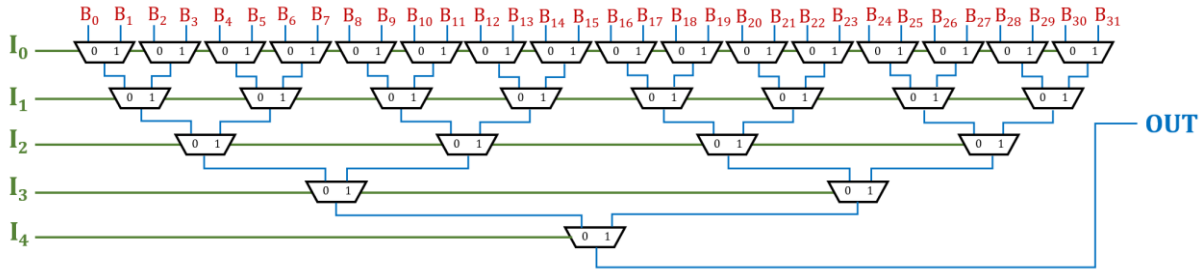


Figure 141: LUT5 structure: a multiplexer tree driven by the input signal is used to drive the content of one of the configuration bits to the output.

5.2.7.1. Activation conditions

Depending on the way LUTs are configured, the fault activation conditions must be handled differently. The following configuration types must be distinguished:

- A single LUT is implemented using up to 6 inputs (using O6)
- Two LUTs are implemented (1 to 5 inputs) sharing some, all or none of their inputs (A6 is tied to VCC).
- One of the outputs is configured as “route-through” while the remaining output is unused. This configuration allows external signals to be routed inside the slice by passing through the LUT (configured as driver).
- One LUT is used (1 to 5 inputs) while the remaining output is used as route-through.

When used as single LUT, the number of sensitive bits to be accounted is directly related to the number of inputs ($N_{BIT}=2^{Ninputs}$). The activation conditions of these bits are straightforward. Each configuration bit is driven to the output only when the corresponding input value in the truth table are presented on the input nets. Thus, the activation conditions of the configuration bit in a given position are defined by the input values in the truth table in this position as described in Figure 142.

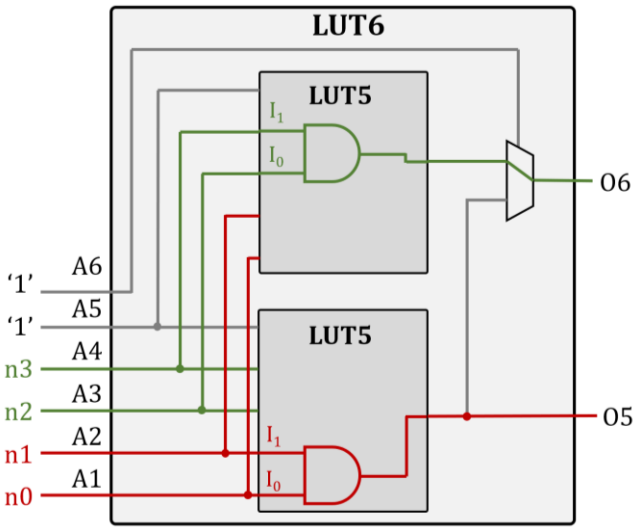
A6	A5	A4	A3	A2	A1	OUT
0	0	0	0	0	0	B_0
0	0	0	0	0	1	B_1
0	0	0	0	1	0	B_2
...						
1	1	1	1	1	0	B_{62}
1	1	1	1	1	1	B_{63}

Activation conditions:
B0 : {A6,A5,A4,A3,A2,A1} = 00000
B_1 : {A6,A5,A4,A3,A2,A1} = 00001
B_2 : {A6,A5,A4,A3,A2,A1} = 00010
...
B_{62} : {A6,A5,A4,A3,A2,A1} = 11110
B_{63} : {A6,A5,A4,A3,A2,A1} = 11111

Figure 142: single LUT activation conditions of configuration bit defining its functionality.

From the software perspective, the logical cell (in the EDIF package) instantiated in LUTs use I0, I1 … I5 to name the input ports. The associated INIT property (defining the LUT content) is provided referring to this logical input port ordering. However, the actual content of the LUTs refers to the physical routing of nets to the input pins of the BEL (A1, A2 … A6) given that all unused ports are tied to VCC. In the algorithms developed in this study, the logical INIT property is systematically converted into the physical content of the LUT. For example, considering a LUT using two inputs I0 and I1 mapped to A5 and A4 respectively and configured as an AND gate; the logical INIT property is `b"1000"` (LSB represent the LUT's output for {I1,I0}="00") but these four configuration bits used to define the functionality are actually mapped to B_{31}, B_{23}, B_{15} and B_7 respectively if using the O5 output or mapped to B_{63}, B_{55}, B_{47} and B_{39} respectively if using the O6 output. As it will be presented in section 5.3.1, the sensitive configuration bits are distinguished by a name (a string variable) which includes the BEL type from which the failure is generated, its position in the fabric as well as a property to differentiate the configuration bits associated with the same BEL. The names of the LUT sensitive bits are defined by their index in the LUT truth table. The conversion of the "logical" content of the LUT into its actual content is thus necessary to avoid that two cells instantiated in the same physical LUT generate two sensitive bits with the same name while being physically stored in different locations.

When two cells are instantiated in the same LUT BEL, the two sensitive bit lists cannot be built independently. If one or more inputs are not shared between the two cells, the input of both cells should be considered to extract the number of sensitive bits. Indeed, even if one input is not logically used by one of the LUT5, this LUT is still affected by the state of this input. The logical function handled by this LUT must remain valid whatever the unused port state is. This means that the configuration bits of the LUT must reflect the intended functionality for all possible combinations of states on unused ports as shown in Figure 143. As a result, the number of sensitive bits to be accounted can be greater than $2^{\#inputs}$, the inputs used by both cells are considered to determine the number of sensitive bits and their associated propagation conditions.



A5	A4	A3	A2	A1	O5	O6
1	0	0	0	0	$B_{16}=0$	$B_{48}=0$
1	0	0	0	1	$B_{17}=0$	$B_{49}=0$
1	0	0	1	0	$B_{18}=0$	$B_{50}=0$
1	0	0	1	1	$B_{19}=1$	$B_{51}=0$
1	0	1	0	0	$B_{20}=0$	$B_{52}=0$
1	0	1	0	1	$B_{21}=0$	$B_{53}=0$
1	0	1	1	0	$B_{22}=0$	$B_{54}=0$
1	0	1	1	1	$B_{23}=1$	$B_{55}=0$
1	1	0	0	0	$B_{24}=0$	$B_{56}=0$
1	1	0	0	1	$B_{25}=0$	$B_{57}=0$
1	1	0	1	0	$B_{26}=0$	$B_{58}=0$
1	1	0	1	1	$B_{27}=1$	$B_{59}=0$
1	1	1	0	0	$B_{28}=0$	$B_{60}=1$
1	1	1	0	1	$B_{29}=0$	$B_{61}=1$
1	1	1	1	0	$B_{30}=0$	$B_{62}=1$
1	1	1	1	1	$B_{31}=1$	$B_{63}=1$

Figure 143: Two AND gates instantiated in the same LUT BEL. The AND functionality on both LUTs must be replicated in the truth table for every unused input state combination. For example, bottom LUT must have $B_0=B_4=B_8=B_{12}=B_{16}=B_{20}=B_{24}=B_{28}$; $B_1=B_5=B_9=B_{13}=B_{17}=B_{21}=B_{25}=B_{29}$ etc. so that the AND logic is respected whatever the state of port A5, A4 and A3.

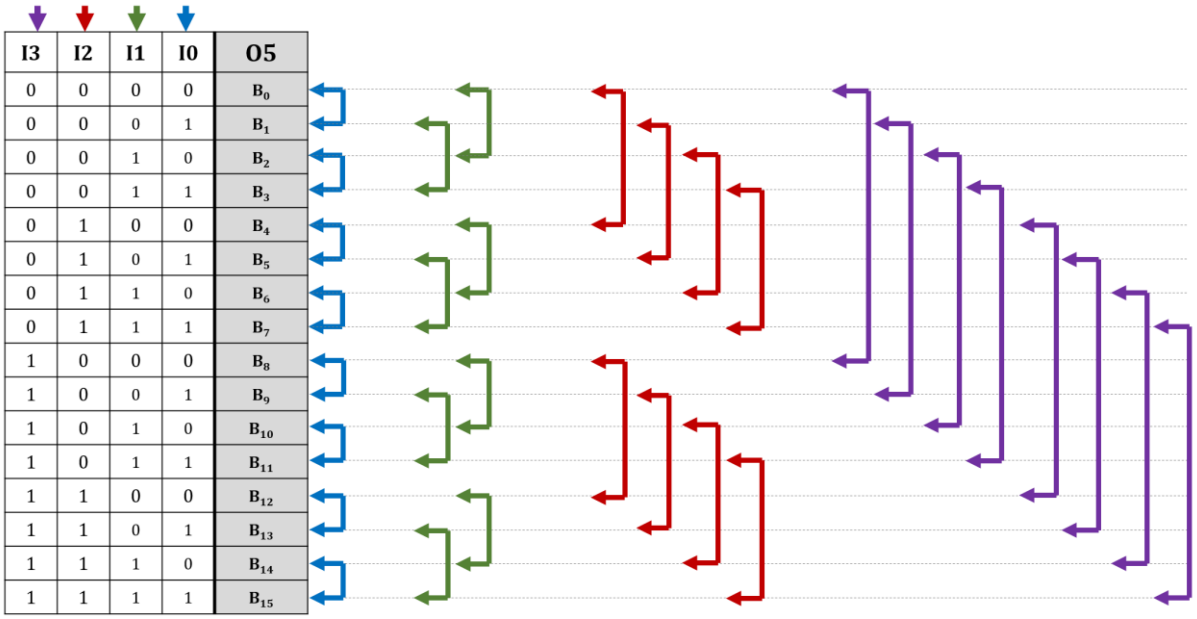
When the LUT is configured as route-through on one output while the remaining output is unused, two sensitive bits are accounted, one for each input state; their physical position in the truth table being defined by the used input and output ports. However, when the remaining output is used by another cell, the inputs of this cell are also considered to extract the number of sensitive bits and the associated propagation conditions.

To be noted that LUTs included in part of the FPGA slices (SLICEM) use two additional configuration bits, one to activate the "shift-register" feature (SRL) and another one used to activate their "distributed memory" feature (LUT used as dual port memory). The exact behavior of these two bits could not be clearly identified. Further investigation would be required to understand in which condition the corruption of these bits can alter the output state when the LUT is not using these features. At the moment, these bits are considered as sensitive and activated unconditionally.

5.2.7.2. PROPAGATION CONDITIONS

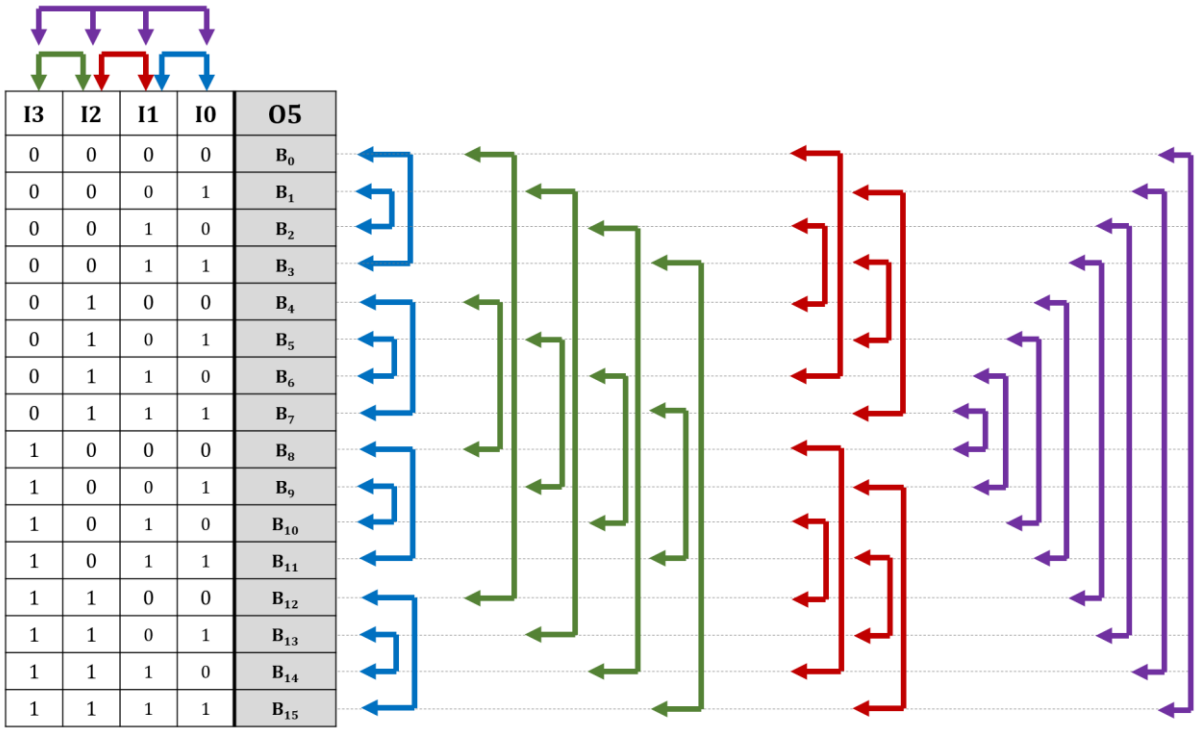
Propagation conditions can be defined using only the logical behavior of the LUT cell (its INIT property). Error masking on a LUT is entirely dependent on its configuration. As there are more than 10^{19} possible configurations with 64 configuration bits, the propagation conditions are calculated for each cell during the execution.

The propagation conditions calculation is based on the LUT truth table using the same principle as the one used in Figure 137. When an error is present in one of the inputs, each line of the truth table is paired with the line for which the state of the considered input is inversed as shown in Figure 144. For example, considering errors in the 1st input port (I0), the 1st line is paired with the 2nd one, the 3rd with the 4th, the 5th with the 6th etc. When multiple inputs are affected by the same error, every line of the truth table is paired based on the same principle (i.e. with the line for which the state of the considered inputs is inversed) as shown in Figure 145.



I3	I2	I1	I0	O5
0	0	0	0	B_0
0	0	0	1	B_1
0	0	1	0	B_2
0	0	1	1	B_3
0	1	0	0	B_4
0	1	0	1	B_5
0	1	1	0	B_6
0	1	1	1	B_7
1	0	0	0	B_8
1	0	0	1	B_9
1	0	1	0	B_{10}
1	0	1	1	B_{11}
1	1	0	0	B_{12}
1	1	0	1	B_{13}
1	1	1	0	B_{14}
1	1	1	1	B_{15}

Figure 144: Truth table lines pairing to determine single input error propagation conditions. For each input, the lines are paired with the lines for which the considered input state is inversed (double arrow).



I3	I2	I1	I0	O5
0	0	0	0	B_0
0	0	0	1	B_1
0	0	1	0	B_2
0	0	1	1	B_3
0	1	0	0	B_4
0	1	0	1	B_5
0	1	1	0	B_6
0	1	1	1	B_7
1	0	0	0	B_8
1	0	0	1	B_9
1	0	1	0	B_{10}
1	0	1	1	B_{11}
1	1	0	0	B_{12}
1	1	0	1	B_{13}
1	1	1	0	B_{14}
1	1	1	1	B_{15}

Figure 145: Truth table lines pairing to determine multiple inputs error propagation conditions. For each multiple input combination, the lines are paired with the lines for which the states of the considered inputs are inversed.

For each line pair, if the result of the truth table is different, both corresponding input values are added to the propagation conditions. The resulting conditions are then optimized using a logic minimization algorithm.

5.2.8. FLIP-FLOPS

Each slice of the *Xilinx 7 series* FPGAs embeds eight flip-flops. These flip-flops can be configured with different clock, reset and clock enable polarities using the previously mentioned static multiplexers. Two additional configuration bits, at the slice level (common to all 8 flip-flops) allow to define if the reset is synchronous or asynchronous and to define if the BEL is used as latch or flip-flop. To evaluate the criticality of this first bit, the timing information of the clock and reset paths should be taken into account to find whether the corruption of the bit can lead to an alteration of the flip-flop output when the reset signal is asserted. The current version of the proposed software does not consider this configuration bit as critical. The functionality of the second configuration bit has not been clearly identified, at the moment this bit is considered as sensitive and is activated unconditionally. In addition, each flip-flop is supplemented by two configuration bits to define the state of the flip-flop at initialization and after a reset. The bit defining the state at initialization is not considered critical given the low probability of error occurring between the component configuration and the global reset signal assertion. On the other hand, the configuration bit defining the state of the flip-flop after a reset is considered sensitive and is activated when the reset signal is asserted.

Regarding the propagation conditions, errors on CE and CLK ports propagate as soon as the input value is not constant and errors on the reset port propagate when the reset state of the flip-flop is different from the input value. As for the errors on the D port, they can propagate to the output when the clock enable signal is activated and the reset port is not asserted. As it will be explained in section 5.3.3, the propagation conditions for errors propagating from the D port are temporally incremented to account for the one clock cycle delay introduced by the flip-flop.

5.2.9. SHIFT REGISTER LUT

To reduce the flip-flops utilization, part of the FPGA slices (SLICEM type) integrates LUTs that can be configured as addressable shift registers (SRL). These LUTs integrate additional ports: clock (CLK), clock enable (CE) and data in (DIN) and the configuration bits of the LUT are then used as dynamic user bits as shown in Figure 146. At each clock cycle, the value on the DIN input is copied to the LSB of the LUT content (B_0) and all configuration bits are shifted toward the MSB ($B_0 \rightarrow B_1$; $B_1 \rightarrow B_2$; $B_2 \rightarrow B_3$ etc.). The standard LUT inputs (A[3:0]) are used to multiplex one of the stages of the shift register to the output in order to define the actual size of the of shift register. These SRL can be cascaded inside and across slices to form longer shift register.

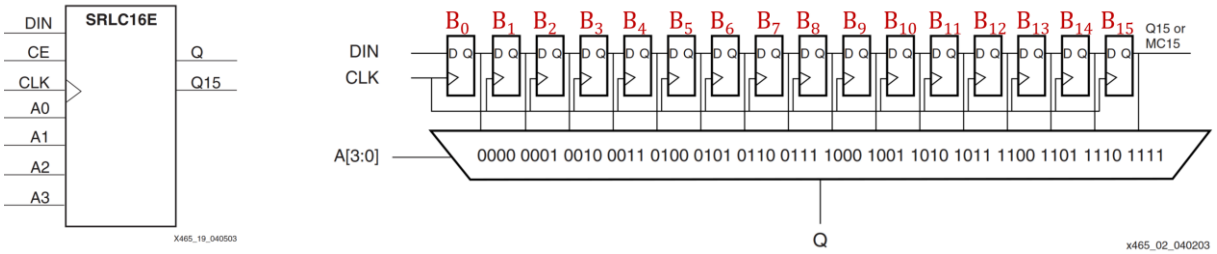


Figure 146: Shift Register LUTs uses the LUT configuration bits to delay a signal by up to 16 clock cycles. The A[3:0] input are used to select the number of clock cycles of delay using the multiplexing structure of LUTs, from [194].

5.2.9.1. FAILURE MODES

The configuration bits of the LUT content being shifted and gradually replaced by the DIN input values, their corruption does not create permanent fault, the effect would be similar to flip-flop SEUs which are not considered in this analytical approach. However, one configuration bit is activated to configure the LUT as SRL, this bit is considered as sensitive and activated unconditionally on the output port.

5.2.9.2. PROPAGATION CONDITIONS

Errors on CE and CLK ports propagate as soon as the input value is not constant. Errors on the DIN port propagate when the clock enable is activated. The propagation conditions for error from DIN ports are also incremented according to the size of the shift registers defined by the value on A[3:0] ports (only static size shift registers are currently supported). Errors on the A3, A2, A1 and A0 actually modify the length of the shift register by 8, 4, 2 and 1 respectively. The propagation of errors on these ports is defined by comparing the current output state with its previous or future values. For example, if the input A0, normally in state 1, is corrupted by a stuck-at-0 fault, the shift register is shortened by 1 bit. Consequently, the output of the SRL is erroneous only when the previous state is different from the current state. If A0 is normally in state 0 and corrupted by a stuck-at-1 fault, the shift register is lengthened by 1 bit, the output is thus erroneous only when the current output state is different from the next output state. The way in which the timing of the signal states and the propagation conditions are managed in the proposed software is described in section 5.3.1.

5.2.10. BLOCK RAM

Memory blocks in *Xilinx 7 series* FPGAs store up to 36Kbits of data and can be configured as either two independent 16Kb RAM, or one 36Kb RAM. The memory width and length can be freely configured from 1x32K to 72x512 in dual port mode. The content of the memory block is directly integrated in the configuration memory for content initialization and modification purposes.

5.2.10.1. FAILURE MODES

The content of the BRAM is user writable. This implies that the corruption of the BRAM content is only visible when the affected data word is read and that the error is corrected when its overwritten. The probability that the error is activated (revealed in the outputs) depends on the time at which the error is generated. If the error happens after the last write operation on the affected word and before the next read operation, the error will propagate to the output when it will be read. However, if a write operation is performed on the affected word before it is read, the error will be erased. The time window during which the error can be activated must be considered to estimate the activation probability of the error. This mechanism is not supported yet by the proposed software, so read-only memories are only considered. In another hand, the bitstream decoding of BRAM slices is not completely achieved, thus, particular configuration bits used to define the data length and width and the different BRAM operating modes (ECC, registering etc.) are not considered at the moment.

5.2.10.2. ACTIVATION CONDITIONS

When configured as read-only memory, errors on the memory bits of the BRAM are activated when the address of the affected word is applied to the address input port. A cycle delay should also be considered when the output data are registered. The error is then activated only on the output port corresponding to the position of the configuration bit in the data word. For example, a bitflip on the first bit of the first word is activated on the output data LSB when the input address value is 0, or one clock cycle later if the output is registered.

5.2.10.3. PROPAGATION CONDITIONS

The propagation conditions depend on the type of affected input. Errors on the address bus trigger the transmission of the wrong data word to the output. The current address value and the index of the error define the index of the word that is replaced and the index of the word that replaces it. By comparing both words, the indexes of the affected output ports are determined. For example, considering a stuck-at-1 fault on the address LSB, when the address value is 0, the output data, theoretically equal to the first word stored in the memory, is replaced by the second word. If the first and second word stored in the BRAM are *0xFF* and *0xF0* respectively, the fault only propagates to the first four output bits. When the address value is 1, the fault does not propagate (the LSB is already stuck-at-1), when the address value is 2, the process is repeated by comparing the 2nd and the 3rd words, etc.

Stuck-at fault errors on the clock and clock enable ports actually freeze the output value. The activation conditions are thus conditioned by the actual address value when the error is generated. This mechanism is not yet supported so these errors are currently propagated to every output port unconditionally.

5.2.11. UNSUPPORTED PRIMITIVES

The following primitives are currently unsupported by our software:

- Digital Signal Processing Blocks
- Latches (LDCE and LDPE)
- Distributed memory elements (LUTRAM)
- Dual port BRAM
- Phase-Locked Loops (PLL) and Mixed-Mode Clock Manager (MMCM)
- Input/Outputs buffers
- GTX transceivers
- Dual Data-Rate registers (DDR)
- Analog to digital converters (XADC)
- Internal configuration control elements (ICAP, BSCAN, FRAME_ECC, STARTUPE etc.)

Design integrating these resources can be analyzed, but the sensitivity of these resources will not be addressed. Further studies on bitstream composition and extensive localized fault injections would be required to analyze the different failures modes of these resources and integrate them in the proposed analytical approach.

5.3. NETLIST ANALYSIS AND CRITICAL BITS EXTRACTION

Once the failure model has been established, the different algorithms must be developed to analyze the circuit netlist, navigate through the different logic gates and routing paths to extract all the sensitive bits and compute their activation and propagation conditions until reaching the circuits outputs.

In this section, the different data structures used by the software will be firstly presented. Then, the procedure used to extract the states of all nets in the circuit and affect them to the corresponding Java objected is described. Finally, a detailed description is provided of the different algorithms used to navigate through the netlist, extract the sensitive bits and compute their activation and propagation conditions.

5.3.1. DATA STRUCTURES

When dealing with large circuits with hundreds of thousands of logic gates and nets, the choice of data structures used to store and manipulate the different types of circuit information has a major impact on the execution time and memory usage of the program.

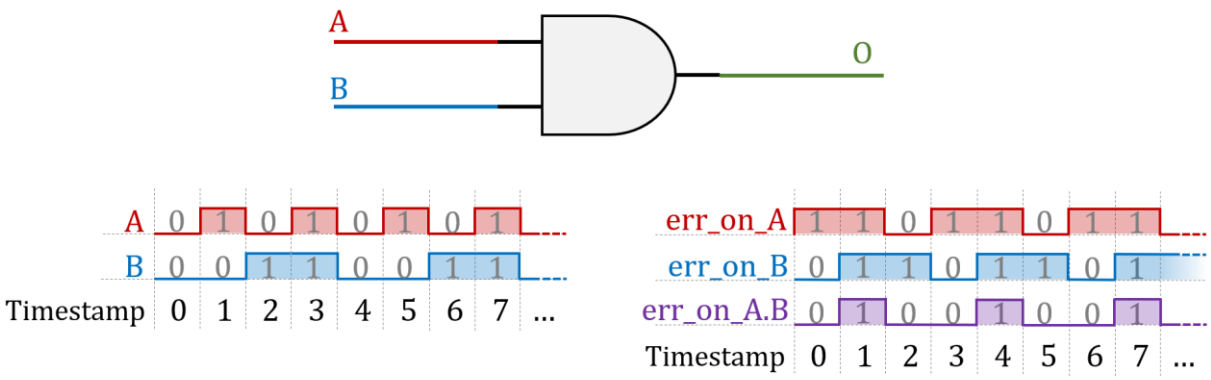
The number of sensitive bits in large designs can easily reach over one million. Each sensitive bit can cross several hundreds of logic gates before reaching one of the circuit primary outputs. Each time the error propagates through a logic gate, a new propagation condition is added serially, when the error has propagated to two or more inputs of the same logic gates, new propagation conditions should be added. The single input propagation condition should be added in parallel while multiple-inputs propagation conditions are also considered for the timestamps where the error did propagate to multiple inputs simultaneously. In this study, timestamps refer to the different time instants in the simulation where the signals value are extracted, namely, at each clock cycle. Different data structure solutions have been tested to represent these propagation conditions combinations.

The first attempt was to combine and minimize the conditions after each crossed logic gate to get one single set of conditions, defined by a list of net names and a list of possible values that enables the propagation. This approach resulted in endless computation when dealing with large circuits due to the heavy workload of the minimization algorithm.

The second approach was to use a dedicated data structure to store each individual propagation condition (the one from each logic gate the error has been through) and to record the history of how these conditions combined (serial vs parallel). The idea was to wait until each fault reaches one of the primary outputs to combine and minimize the different conditions. This approach also resulted in endless computation due to the large number of sensitive bits and the number of conditions to be stored for each one of them.

The solution finally adopted consists in directly assigning to each net object of the design, the different values they have throughout the simulation. The activation and propagation conditions can then be directly computed and verified on the fly for each logic gate the error passes through. Instead of storing the propagation conditions as a list of nets and the associated values, a list of timestamps at which the error propagates is stored. Using this representation, the timestamps where the error is present on one input only or on several inputs simultaneously can be easily extracted by comparing the lists of timestamps on each input. For each erroneous input

combination (single input only, double inputs, etc.), the corresponding propagation conditions are computed and the timestamps at which these conditions are verified are computed based on the values of the input nets. The common timestamps to both timestamp lists are then extracted. Once this process has been repeated for each erroneous input combination, the resulting timestamp lists are merged to define the timestamps where the error is propagated to the output. An example of this procedure applied to an AND gate is described in Figure 147.



1- Nets **V**alue **L**ist extraction:
VL(A)=[0,1,0,1,0,1,0,1]
VL(B)=[0,0,1,1,0,0,1,1]

2- Error **T**ime**s**tamps extraction:
TS^{err}(A only) = [0,3,6]
TS^{err}(B only) = [2,5]
TS^{err}(A.B) = [1,4,7]

3- **P**ropagation **C**onditions establishment:
PC(A→O) : B='1'
PC(B → O) : A='1'
PC(A.B → O): {AB}="00" or "11"

4- Compute confirmed propagation conditions **t**ime**s**tamps:
TS^{PCA} = [2,3,6,7]
TS^{PCB} = [1,3,5,7]
$TS^{PCA\&B}$ = [0,3,4,7]

5- Combine confirmed propagation condition TS and error TS:
$TS^{A\rightarrow O}$ = [3,6]
$TS^{B\rightarrow O}$ = [5]
$TS^{A.B\rightarrow O}$ = [4,7]

6- Merge all **t**ime**s**tamps :
TS^{err}(O)=[3,4,5,6,7]

Figure 147: Error propagation evaluation procedure applied to an AND gate. First, the list of values at each timestamp (from behavioral simulation) is extracted for all input nets. Secondly, the timestamps at which the considered error has actually propagated to the inputs (from previous stages) are extracted for the different cases: error in 1st net only, error in 2nd only and error on both inputs. Thirdly, the propagation condition for each erroneous input combination are computed. Fourth, for each propagation condition, the timestamps at which the condition is verified is extracted based on the input value lists. Fifth, the confirmed propagation timestamps are filtered among those present on the error time stamps from step 2. Finally, the resulting timestamp lists are merged to define the timestamps where the error is propagated to the output.

To clarify, **error timestamps** refer to the time instants (referring to the simulation time base) at which the error generated by a given sensitive bit is actually able to propagate to the considered net. When coupled with an erroneous input combination, the **error timestamps** refer to the time instant at which the error generated by the same sensitive bit is only present on one input (single input error) or to the timestamps that are common to several inputs (multiple input error). Finally, **confirmed propagation condition timestamps** refer to the timestamps at which a given propagation condition is met.

This propagation procedure is renewed for all sensitive bits propagated to the inputs of the logic gates. The computation steps related to the propagation conditions establishment and the associated timestamps can be reused for all sensitive bits to be propagated to reduce the execution time.

Two types of data structures are used to represent the value list of each net and the error timestamp list. The first data type is a simple ordered list of Booleans for value list and an ordered list of integers for timestamps. These list structures provide a good flexibility for manipulation but use a lot of memory space and require a lot of iterations when dealing with comparison, sorting, incrementing, extraction of common elements from two lists, etc. For these reasons a more efficient data structure is used to store both value list and timestamp list: the `BigInteger`. This data structure can store any integer with an arbitrary number of bits. Value lists can thus be stored by representing values at each timestamp as one bit of a `BigInteger`, considering that the LSB is the first timestamp and MSB the last one. Similarly, the timestamp lists are stored by initializing a `BigInteger` to 0 and setting to 1 each bit in the positions contained in the timestamp list as shown in Figure 148.

```
Value lists:
List<Boolean> valueList = [0,1,0,0,0,0,0,1]  → BigInteger valueInt = 130 (=b"10000010")
                           7 6 5 4 3 2 1 0                              76543210
Timestamp lists:
List<Integer> timestampList = [0,2,3,4]      → BigInteger tmstpInt =  29 (=b"00011101")
                                                                         4 3 2 0
```

Figure 148: Example of `BigInteger` variables used to represent list of Boolean and list of integers.

The length of `BigInteger` being scalable, it can be used whatever the number of timestamps analyzed during the behavioral simulation. Additionally, thanks to the use of bitwise operators, this structure is much more efficient for common operations:

- Incrementing: bit shifting towards left
- Extracting common elements of two lists: bitwise AND
- Extracting elements contained in at least one of the lists: bitwise OR
- Extracting elements that are exclusives between two list: bitwise XOR
- Extracting elements that are contained in 1st list but not in a 2nd list: bitwise AND NOT

These common operations are widely used for computing propagation conditions and associated timestamps while further benefits are provided by the fact that values and timestamps share the same data structure. For example, in the previous example of propagation conditions computation applied to an AND gate (Figure 147), the error timestamps that are exclusive to one port are computed with a bitwise AND NOT between the error timestamps from the two ports, the error timestamps that are common to both inputs are computed with a bitwise AND. The propagation conditions timestamps for `B=1` are a simple copy of the `BigInteger` holding the value of net B, for `A=1`, a copy of the `BigInteger` holding the value of net B. As for `AB="00"or"11"`, a bitwise NOT XOR is applied between values of net A and B. Similarly, to check the timestamps at which the error actually propagates, a bitwise AND is applied between the propagation condition timestamps and the corresponding error time stamps. The final result is obtained by applying a bitwise OR between the resulting `BigIntegers`. This procedure, adapted using `BigIntegers` variables, is illustrated by the pseudo code shown in Figure 149.

```
BigInteger A,B,tsA,tsB,tsAonly,tsBonly,tsAandB,tsPcA,tsPcB,tsPcAB,tsOfromA,tsOfromB,tsOfromAB,tsO
//step1: net values
A = 172    // b"10101010"
B = 314    // b"11001100"
//step2: error
tsA = 219 // b"11011011"
tsB = 182 // b"01101101"
tsAonly = A.andNot(B)
tsBonly = B.andNot(A)
tsPcAB = A.and(B)
//step3&4: propagation condition
tsPcA = A
tsPcB = B
tsPcAB = not(A.xor(B))
//step5: output confirmed error propagation
tsOfromA = tsAonly.and(tsPcA)
tsOfromB = tsBonly.and(tsPcB)
tsOfromAB = tsPcAB.and(tsPcAB)
//step6 output error
tsO = tsOfromA.or(tsOfromB).or(tsOfromAB)
```

Figure 149: Pseudo code of the procedure to compute the propagation timestamps of an AND gate using `BigIntegers` and associated bitwise operators.

The bit shifting is also a convenient operation to increment all error timestamps simultaneously when propagation through flip-flops or SRL. Indeed, as the `BigInteger` storing the values of each net are based on the same time scale, the error timestamps on a net must be incremented when delayed by one clock cycle. That way, the output propagation timestamps are synchronized with the net values when further propagating through the next logic gates. A cyclic bit shifting can also be performed when dealing with cyclic net values (the values repeat periodically with a cycle equal to the number of values extracted from the simulation).

The sensitive bits extracted from each logical and routing resource are stored in a dedicated Java object. This object contains different types of arguments to define its origin: the configuration bit type, the slice coordinates and the name of the BEL or PIP the configuration bit is related to. These arguments are used to build a string argument to differentiate all potential sensitive bits of the design. Each sensitive bit object is supplemented with a `BigInteger` to define the current error conditions. When dealing with multiple outputs cells or nets, the sensitive bit objects are cloned for each output. Each cloned object can carry its own error timestamps that will evolve by propagating through the different logic gates. When the cloned sensitive bits are propagated to the inputs of the same cell, their string arguments are used to combine them using the error propagation procedure for multiple erroneous inputs described earlier.

5.3.2. WORKLOAD EXTRACTION

To compute the activation and propagation conditions, the values of every net in the design must be extracted and associated with the corresponding net object. As explained earlier, these net values are stored as `BigInteger` where each bit correspond to the state of the net at a given timestamp of the simulation. The length of the simulation (i.e. the number of extracted timestamps) is user defined. The extraction of these values and their association with the right java object is constrained by different elements. Indeed, unlike other commercial simulation tools, the behavioral simulator integrated in Vivado does not support hierarchical referencing of the circuit nets through the HDL testbench file nor through the TCL commands. This constraint therefore prevents the extraction of signal values from being fully automated. The following procedure is then used:

1- The design is exported into design checkpoint file (.dcp) and imported in the java software.
2- The design is flattened into one single VHDL file using the `write_vhdl -cell *top_cell*` TCL command. This command generates a VHDL file describing a design with the same behavior as the original design while removing all level of hierarchy. Every net can then be referred to through TCL commands.
3- From the software side, all nets of the design are extracted, and a TCL script is automatically generated. This script, aimed to be launch during the simulation, is composed of a set of command that extract the current state of every listed net and write it into a text file as a binary string. This extraction is repeated by systematically incrementing the simulation time by one clock period until the number of timestamps defined by the user is reached.
4- The binary strings are transposed and formatted to obtain an integer value for each net where each bit of the integer represent the net value at a different timestamp.
5- The resulting file is finally imported back into the software and the integer values are associated to each net. In this purpose, the `EDIFNet` object from the RAPIDWRIGHT API was supplemented with a `BigInteger` attribute to store the net values.

5.3.3. CIRCUIT NAVIGATION ALGORITHMS

The analysis of the number of critical bits is based on the extraction of the sensitive bits of each logic gate and each net of the circuit and on the calculation of their propagation from their origin to one of the primary outputs. The objective of the navigation algorithm, the core of the critical bit extraction process, is then to traverse the whole circuit, to extract the sensitive bits of each crossed resource by applying their activation conditions and to propagate the set of sensitive bits coming from the downstream resources towards the outputs. The difficulty of the algorithm development process is to ensure that all the resources that can alter one of the considered outputs are traversed and computed while ensuring that each resource is computed only once. The algorithm is therefore based on the following principles:

- The output of a logical gate can only be computed when all its inputs have been computed.
- The sink port(s) of a net can only be computed if its source port has been computed.

To this end, the Java object `EDIFPortInst` (materializing the ports of the logical cells) of the RAPIDWRIGHT API, is supplemented with two arguments: a list of sensitive bits to hold the set of downstream sensitive bits propagated up to this port and a Boolean used to indicate if the port has already been computed. For an input port, this implies that all downstream resources and nets have been computed and their sensitive bits propagated to that port. For an output port, this implies that all input ports have been computed, their critical bits have been propagated to the output port and the intrinsic sensitive bits of the resource have also been added with their activation conditions.

The navigation algorithm is based on the recursion of two types of functions that mutually call each other: the `backwardScan` and the `forwardPropagate` function types. Starting from the user defined output ports of the circuit, the corresponding cells are checked, if the output port is not already computed (which is obviously the case at program startup), the `backwardScanCell`

function, detailed in Figure 150, is called. This function checks sequentially every input port of the cell. If a port is not already computed, the `backwardScanNet` function, detailed in Figure 151, is called for the corresponding net. Similarly, this function checks the source port of the net and calls another instance of the `backwardScanCell` function for the corresponding cell if this port is not already computed. This process of scanning backwards by alternating these two functions is repeated until an already calculated port or a primary input of the circuit is detected. When a computed port is detected during the `backwardScanNet` function, the list of sensitive bits is propagated to all the sink ports of the net while adding the sensitive bits of the different PIPs encountered on the routing path. This forward propagation is performed through the `forwardPropagateNet` function detailed in Figure 153. The sink ports of the net are finally set as computed to prevent the net and the upstream resources from being recalculated. On leaving this last function, the program falls back into the `backwardScanCell` function which will check the next input port. Once all the input ports have been calculated, all their sensitive bits are propagated to the output(s) of the cell through the `forwardPropagateCell` function. This function is available in different versions depending on the type of cell considered in order to apply its own error propagation rules while benefiting from different optimizations (particularly for logic multiplexer, flip-flops or CARRY4 cells). A generic version of this function is detailed in Figure 152. Once all the sensitive bits are propagated, the intrinsic sensitive bits of the resource are added to the output port(s) with their activation condition and the output port is set as computed. This back and forth between the backward scan functions and the forward propagation functions continues until all cells and nets have been calculated and propagated to the user defined outputs.

```
1  backwardScanCell(Cell cell):
2  // iterating through the input ports of the cell
3  FOR inPort IN cell.getInputPorts():
4      IF !inPort.isComputed():
5          Net net = inPort.getNet()
6          // collect the SBs from the net source port
7          backwardScanNet(net)
8          // propagate the source port SBs to each sink port
9          // by integrating the intrinsic net SBs
10         forwardPropagateNet(net)
11 RETURN
```

Figure 150: Pseudo code of the `backwardScanCell` function. Each input port of the cell is sequentially checked. For ports which are not already computed, the corresponding net is checked through the call of the `backawardScanNet` function. The sensitive bits of its source ports are propagated to every sink port while adding the sensitive bits of the PIP crossed during propagation through the `forwardPropagateNet` function.

```
backwardScanNet(Net net):
// get Net source port
EDIFPort srcPort = net.getSourcePort()
IF !srcPort.isComputed():
    // scan the cell inputs
    backwardScanCell(srcPort.getCell())
    // propagate the input sensitive bits (SBs)
    forwardPropagateCell(cell)
RETURN
```

Figure 151: Pseudo code of the `backwardScanNet` function. The source port of the net is checked. If not already computed, the cells inputs are checked through the call of the `backwardScanCell` function and the cells input sensitive bits are propagated to the output through the call of the `forwardPropagateCell` function.

```
forwardPropagateCell(cell):
EDIFPort outputPort = cell.getOutputPort()
// gather all sensitive bits (SBs) present on the input ports
List<Sb> inputSBs = NEW List<>()
FOR inPort IN cell.getInputPorts():
        inputSBs.addAll(inPort.getSensitiveBitList)
// iterating through the SBs
FOR sb IN inputSBs:
        // determine which ports are affected by the error
        List<EDIFPort> errInputPort = NEW List<>()
        FOR inPort IN cell.getInputPorts():
                IF(inPort.getSBList.contains(sb):
                        errInputPort.add(inPort)
        // create a list of inputPort list to represent  allpossible input combination
        List<List<inPort>> inputPortCombinations = getPortComb(erroneousInputPort)
        // use BigInteger to store the output propagation timestamps for this error
        BigInteger finalOutTS = 0
        //iterate through each combination
        FOR inPortList IN inputPortCombinations:
                //get error timestamps (TS) for this combination
                BigInteger errTS = getErrTS(inPortList,errInputPort)
                //get propagation conditions (PC)
                Map<List<Net>,List<List<Boolean>>> pCs = getPCs(inPortList,cell)
                //get validated propagation condition timestamps
                BigInteger validPcTs = getTimestamps(pCs)
                //combine error TS and validated PC timestamps
                BigInteger outTS = errTS.and(validPcTs)
                // add it to the final output timestamps
                finalOutTS = finalOutTS.or(outTS)
        // if the propagation TS list is not empty, add the SB to the output SB list
        IF finalOutTS!=0:
            Sb outSB = NEW Sb(sb)
            outSB.setErrorTS(finalOutTimestamps)
            outPort.addSB(outSensitiveBit)
        // compute and add all the intrinsic SB from the cell for the considered output
List<Sb> intrinsicSBs = getSBsFromCell(cell,outPort)
outPort.addSbs(intrinsicSBs)
outPort.setIsComputed(True)
RETURN
```

Figure 152: Pseudo code of the `forwardPropagateCell` function. In a first step all sensitive bits are gathered in a single list. Then, by iterating through each sensitive bit, the input ports that contain this bit are listed. From this list, all possible erroneous input combinations are generated (single and multiple). For each combination, the corresponding error timestamps are extracted. The propagation conditions for this input combination and the corresponding timestamps are computed and combined with the error timestamp list. The resulting timestamp list is accumulated in the final output timestamp for this sensitive bit. If this timestamp list is not empty, the sensitive bit is added to the output port. Once all sensitive bits have been propagated, the output port is set as computed.

```
forwardPropagateNet(Net net):
// get Net source port
EDIFPort srcPort = net.getSourcePort()
// get the SB list from sourcePort
List<Sb> srcSBList = srcPort.getSBList()
// get all net PIPs
List<PIP> pipList = net.getPIPs()
// create a Map linking each node to its downstream PIPs
Map<Node,List<PIP>> node2PipMap = getNode2PipMap(net)
// get the source node of the net
Node srcNode = srcPort.getNode()
// propagate the SB list through the node and extract PIP SBs
forwardPropagateNode(net,node2PipMap,srcNode,srcSBList)
RETURN
```

Figure 153: Pseudo code of the `forwardPropagateNet` function. The sensitive bits of the source port and the list of PIPs the net is composed of, are firstly extracted. From the list of PIPs, a Java Map is generated to link each node of the net to the list of downstream PIPs. Starting from the source node, the sensitive bits are propagated to each sink port through the call of the `forwardPropagateNode` function.

A net is a connection between one source port and one or several sink ports. A net is composed of node and PIPs. To extract the sensitive bits of the PIPs that compose a net and propagate those from its source port, the `forwardPropagateNode` detailed in Figure 154 is used. This function is also based on recursion: each time a new PIP is crossed, a new instance of the function is called by passing the next node and a clone of the current sensitive bits list as parameters. If the node has a single downstream PIP, the sensitive bits from the PIP are computed and simply added to the current list before calling the next instance of the function with the next node. When the node has several downhill PIPs (the net splits into several branches), a different instance is called for each branch. Once a sink port is reached, the current sensitive bit list is assigned to it and the port is set as computed. This process ensures that each sink node is only supplemented with the bits from the PIPs that are actually used between itself and the source port. It also ensures that each PIP is only computed once. An example of this process applied to a multiple branch net is shown in Figure 155.

```
forwardPropagateNode(Net net, Map<Node,List<PIP>> node2PipMap,
                             Node currNode, List<Sb> currSBList):
// clone the current SB list
List<Sb> newSBList = new List<Sb>(currSBList)
// when a sinkPort is reached, affect the cloned SB list to it
IF node2PipMap.get(currNode)==null:
    EDIFPort sinkPort = currNode.getSinkPort()
    sinkPort.setSensitiveBit(newSBList)
    sinkPort.setIsComputed(True)
// else, for each sink PIP, add the PIP SBs & continue propagation
ELSE:
    FOR pip IN node2PipMap.get(currNode):
        newSBList.addAll(getSBfromPIP(pip))
        Node nextNode = pip.getEndNode()
        forwardPropagateNode(net,node2PipMap,nextNode,newSBList)
RETURN
```

Figure 154: Pseudo code of the `forwardPropagateNode` function. Each time a PIP is crossed a new instance of the function is called. When a node has multiple downstream PIPs, the function is called one time for each PIP based on the same level of function instance.

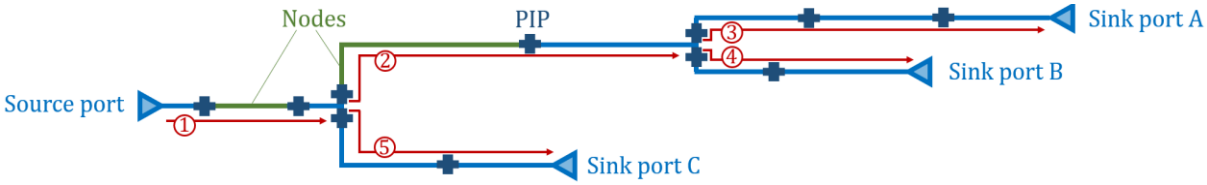


Figure 155: Example of forward propagation of critical bit through net with multiple sink ports using a recursive function. Each node that composes the net can have a single downhill PIP or several downhill PIPs when the net splits. For each PIP (blue crosses), a new instance of the function is called. When the net splits, different recursive branches are instantiated (one for each downhill PIP) to ensure that each sink port is only supplemented with the bits from the PIPs that are actually used to reach them. Numbers in red define the order in which the different net branches are computed.

An example of the full navigation procedure is illustrated in Figure 156. Once all primary outputs have been computed, the sensitive bits are gathered while removing the duplicates. Every sensitive bit that reached one of the primary outputs can be considered as critical. Indeed, during the forward propagation of sensitive bits across logic gates, the sensitive bits that have an empty timestamp list are removed to ensure that any error logically masked is filtered out.

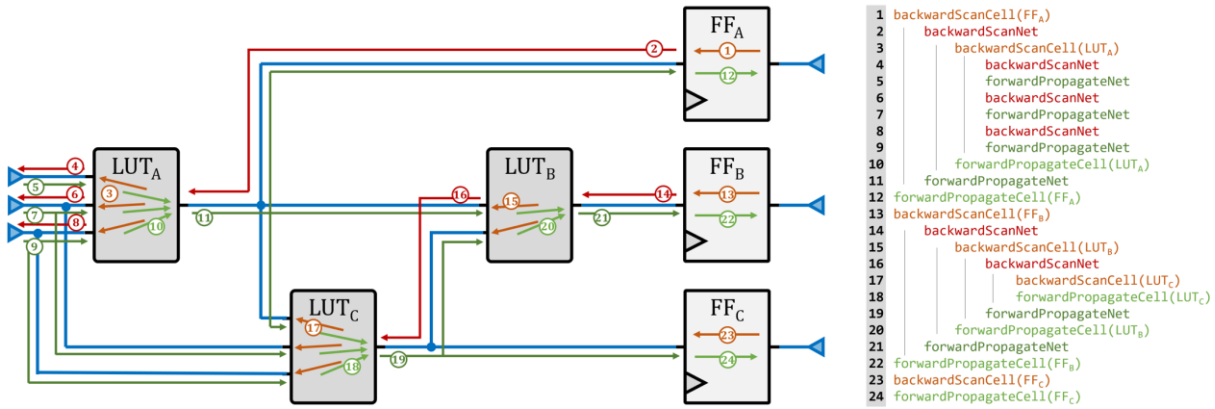


Figure 156: Example of the navigation process applied to a basic 6-cells circuit. Starting from the 1st output cell, the backward scan procedure goes through the FF_A and LUT_A cells before reaching the primary inputs. Once every input ports of the LUT_A cell have been computed, their sensitive bits are propagated to the output of the cell. These sensitive bits are then propagated to the LUT_C, LUT_B and FF_A cells while extracting and propagating the PIPs sensitive bits of the net. The input of FF_A being computed, the sensitive bits can finally be propagated to the first primary output. The process is repeated with the two remaining primary outputs taking advantage of the already computed ports.

The Java code is supplemented with different functionalities to deal with different types of exception and different optimizations have been implemented to reduce the memory usage of the program. For example, once the output of a cell has been computed, the list sensitive bits of its input port can be removed and once the sink port of a net has been computed, the sensitive bits of its source port can be deleted. This provides a good reduction on the memory usage when dealing with large circuits by avoiding the memory over-duplication of sensitive bits.

When dealing with circuits integrating feedback loops, the previously detailed code would result in endless loops of backward propagation. To overcome this issue, a list of cells under test is created at program initialization. When a backward scan function is called, the corresponding cell name is added to this list. Once the cell has been computed, the cell is removed from the list. When

checking if a source port of a net is already computed, if the cell is already contained in the list of cells under test, the port is considered as computed and the forward propagation is triggered.

To be noted that the propagation conditions in feedback loops structure cannot be formally determined as easily as with forward propagation circuits. Indeed, the modification of any net state in such structure can alter the state of the input values in the subsequent clock cycles. This input values being used to determine the propagation conditions, the exact time at which the error occurs can affect its propagation probabilities. These considerations are not supported by the proposed software at the moment. The application of this analytical approach on Finite State Machines, counters, and recursive arithmetical circuits can results in an overestimated number of critical bits. In the meantime, the results of the proposed analysis can be coupled with simulation-based fault injection to determine the failure propagation and recovery mechanisms in such circuits.

5.4. EVALUATION OF THE TOOL CAPACITIES

The software developed in this study allows to evaluate the number of critical bits of any design implemented on a *Xilinx 7 Series* FPGA by considering the workload of the circuit in the error propagation calculation. Beyond the drastic reduction of the execution time (discussed in section 5.4.3), this approach brings many advantages over emulation-based fault injection. Firstly, it can be applied to any type of circuit without requiring any built-in nor external testing system. Different input stimulus can be experienced without any modification of the design to evaluate the workload impact on the system reliability. In addition, this approach provides a great visibility over the predominant failure mechanisms. Based on their attributes, the type and the position in the FPGA fabric of every critical bit can be extracted. For each critical bit, the corresponding resource in the logical netlist can be identified. The software automatically generates a report providing the detailed number of critical bits for each type of configuration bit. Last but not least, this approach provides a great internal visibility over the reliability of each node. Indeed, the number of critical bits associated with each internal net of the device can be extracted. This feature can be used by designers to identify the most critical points of their designs and help them to wisely apply selective triplication schemes.

To validate this software, the results obtained on different circuits are compared with those from fault injection and proton experiments presented in section 4.5.3. Firstly, the results of the analysis are formally validated by comparing the results on a test circuit with an exhaustive fault injection performed on a component whose exact bitstream composition is known (Artix7-50T). The fault injection results are processed to identify, for each critical bit detected, the type and position of the affected resource as well as the role of the configuration bit. The same procedure is applied on the software side and the resulting critical bit lists are compared to study the missing or excess bits. The analytical approach performances are finally evaluated on wide panel of circuits, by confronting the results to those obtained through fault injection and proton beam testing presented in section 4.5.3.

5.4.1. EXHAUSTIVE FAULT INJECTION

The objective of this fault injection campaign is to verify that all critical configuration bits detected by fault injection for a given circuit can been detected with the proposed software. Conversely, the goal is also to ensure that the proposed software does not overestimate the

number of critical bits by verifying that all the critical bits extracted by this software are also detected through fault injection.

To identify the function of the critical bit extracted with fault injection, this experiment requires a complete knowledge of the bitstream composition of the FPGA used, at least on the types of resources used by the targeted design. The PROJECT X-RAY database currently provides the bitstream composition for all *Artix7* devices but only one device for *Spartan7*, *Zynq7* and *Kintex7*. The bitstream global architecture is identical for other devices but the memory address corresponding to a given slice cannot be determined and therefore conversely, the function of a configuration bit at a given address cannot be identified.

The circuit used for this experiment is a FIR filter with SOM multipliers previously used in the radiation tests described in chapter 4. The same BIST structure is used while simply replacing the RAM blocks with distributed memory (LUT configured as read-only memory). The fault injection procedure is also identical to the one use in previous experiments. However, as the SEM controller is provided as encrypted netlist, it cannot be analyzed by the developed software. The SEM controller and the structure under test are then implemented on two different parts of the fabric with a large physical separation to avoid that nodes of the SEM controller cross the part where the circuit under test is implemented. Additionally, the list of essential bits used to generate the memory addresses is filtered to remove all the configuration bits related to the part where the SEM controller is implemented.

On the fault injection side, a total of 467,524 bits has been injected (all essential bits after removing the SEM controller related part). For each critical bit detected, the Physical Frame Address (PFA), returned by the SEM controller, is used to determine the bit functionality and the coordinate of the slice to which it is linked. A common configuration bit naming is used between the fault injection reports and the reports from the developed software so that the critical bit list can be compared to identify the missing bits.

On each side, the critical bits have divided into different categories depending on their functionality:

- **LUT content**: the bits defining the LUT truth table.
- **Switchbox multiplexers**: bits driving the pass transistors of extra slice PIPs. For the analytical approach result, these bits are further divided into open faults and bridge faults.
- **Intra-slice routing multiplexers**: bits driving the pass transistors of intra-slice routing multiplexers.
- **Flip-flop Latch**: bit defining if the flip-flops are used as latch or as a regular flip-flop.
- **LUT parameter**: contain the LUT SRL and LUT RAM configuration bits, respectively used to activate the shift-register LUT and Distributed memory features of LUTs contained is SLICEM.

The comparison of the critical bit lists applied to this circuit is used to highlight and correct some defects in the program and to adjust the failure model presented previously. For example, when a missing critical bit is detected, the corresponding source cell is checked once the program is finished to verify that this sensitive bit has actually been generated. If the bit is not generated, the failure model is adjusted accordingly. If the sensitive bit is properly added to the cell's output, the different propagation paths of the error are monitored to detect the eventual presence of incorrect error masking in one of the downstream logic gates. After this adjustment phase, the current

version of the program allowed to obtain a very good agreement with the result from the fault injection as shown in Figure 157. Indeed, for each category, the number of estimated critical bit is very close to the one extracted from fault injection (1.0%, 0.3%, 3.9% of error for the three first categories respectively).

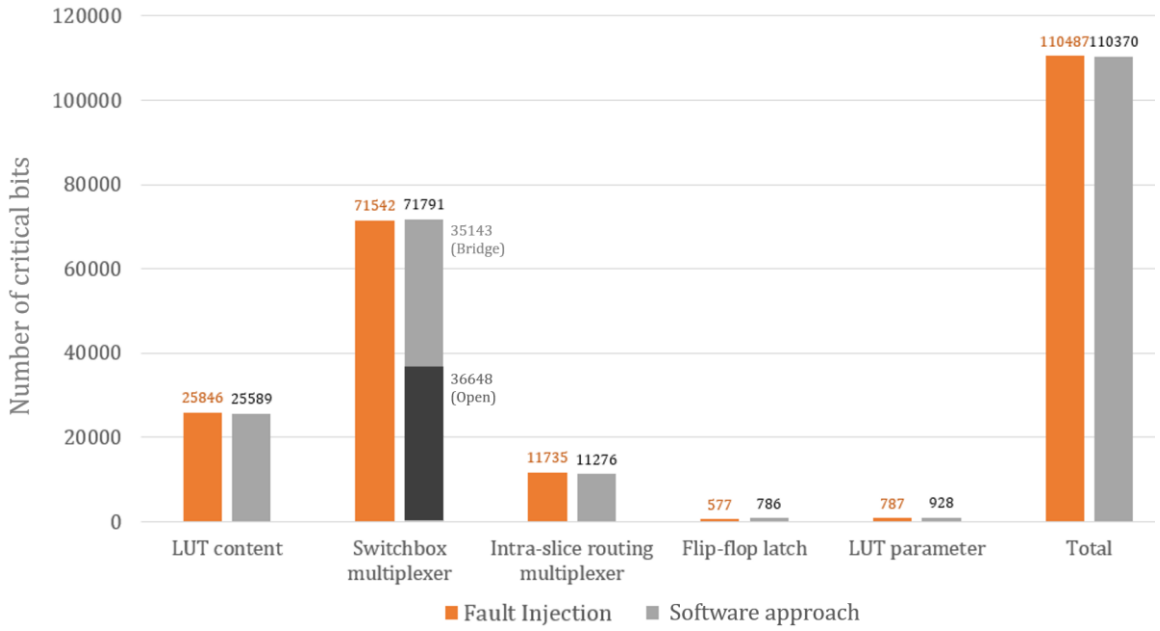


Figure 157:Comparison of the number of critical bits detected between exhaustive fault injection and the proposed analytical approach. Critical bits are divided into different categories related to their functionality.

Nevertheless, significant overestimation can be observed in the "Flip-flop Latch" and "LUT parameters" categories (+36% and +18% respectively). This can be explained by the fact that these errors are activated unconditionally while in some cases the corruption of these bits might not impact the state of the concerned signals. These categories of bits have nevertheless a very limited impact on the global critical bit count.

Further comparisons with fault injection results could be performed to continue improving the current failure model. The error level on the total number of estimated critical bits having reached a satisfactory level (<1%), the evaluation of the software performance is carried to the next step. Its results must then be confronted with the results of radiation experiments and fault injection on circuits that were not used to improve it.

5.4.2. Comparison with experimental results

In section 4.5.3, a radiation experiment was carried with 180MeV protons on Kintex7 embedding eight types of FIR filter with different structures and parameters. The different structures are recalled below:

- **DIR**: Direct form
- **TRA**: Transposed form
- **COM**: Combinatorial multiplier (no pipeline stages)
- **RCE**: Reset and clock enable signals added
- **TMR**: Triplicated and majority voted
- **PAA**: Parandeh-Afshar [143] multiplier

Through this radiation experiment, the cross section of each filter was extracted. The reports sent by the SEM controller were also used to extract the bit cross section of the configuration memory. As the proportion of multibit upsets was not negligible (34%), two types of bit cross sections were computed to represent the lower and upper bound of an equivalent single bit upset cross section (as explained in section 4.5.3.2). The lower bound only accounts for the total number of events whatever the number of corrupted configuration bits per event. The upper bound considers the number of corrupted bits of each event to represent the total number of affected bits. By carrying a fault injection campaign on the same design, the number of critical bits for each filter structure was extracted. The number of critical bits was finally multiplied to the bit cross section and compared to the experimental results.

The proposed software is now used to extract the number of critical bits for every filter structure used in this experiment and compute their cross section using the lower and upper bound of the configuration memory bit cross section. Results are compared to experimental and fault injections results in Figure 158.

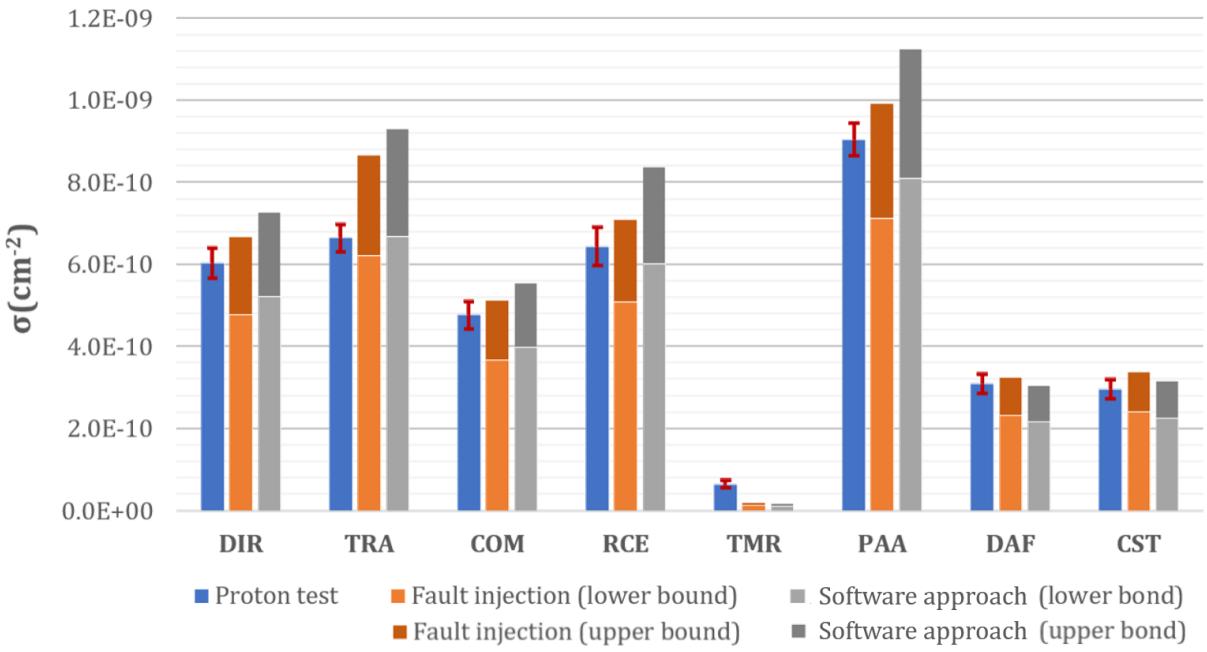


Figure 158: Comparison of the cross section of different FIR filter structures between proton tests, fault injection and the proposed software approach.

The results obtained with the software approach are in a good agreement with those obtained with fault injection and proton experiment. Except for the TMR structure, the lower and upper bounds of the cross section estimate actually surround the value obtained through proton testing. A slight overestimation of the sensitivity can nevertheless be observed over fault injection, excepted for DAF and CST structures. This overestimation can be attributed to the unconditional activation of certain type of faults.

Indeed, the fault activation mechanisms for configuration bits related to routing resources could not be clearly identified for every switchbox PIP. The tool currently considers that open faults are activated unconditionally and that bridge faults are activated when both nets carry different

values. A better agreement with fault injection results could be obtained by further investigating these effects and properly implementing their real behavior in the software.

The mismatch with TMR results has been discussed in section 4.6.3.3. It was partially attributed to bitflips accumulation related to high radiation flux experiments. However, the TMR failures related to MBUs are not addressed at the moment by the software. The knowledge of the bitstream composition could be exploited to identify among the configuration bits belonging to different replicas of the TMR scheme those sharing the same physical location.

Beyond providing the total number of critical bits, the analysis tool allows to differentiate the number of critical bits for each configuration bit category as shown in Figure 159. The proportions are extracted for each FIR filter structure type and shown in Figure 160. These two representations provide an insight view of the main sources of failures for each filter structure and allows to confirm and further investigate the assumption made in section 4.5.3 to explain the difference in sensitivity induced by the various structural parameters of the filters.

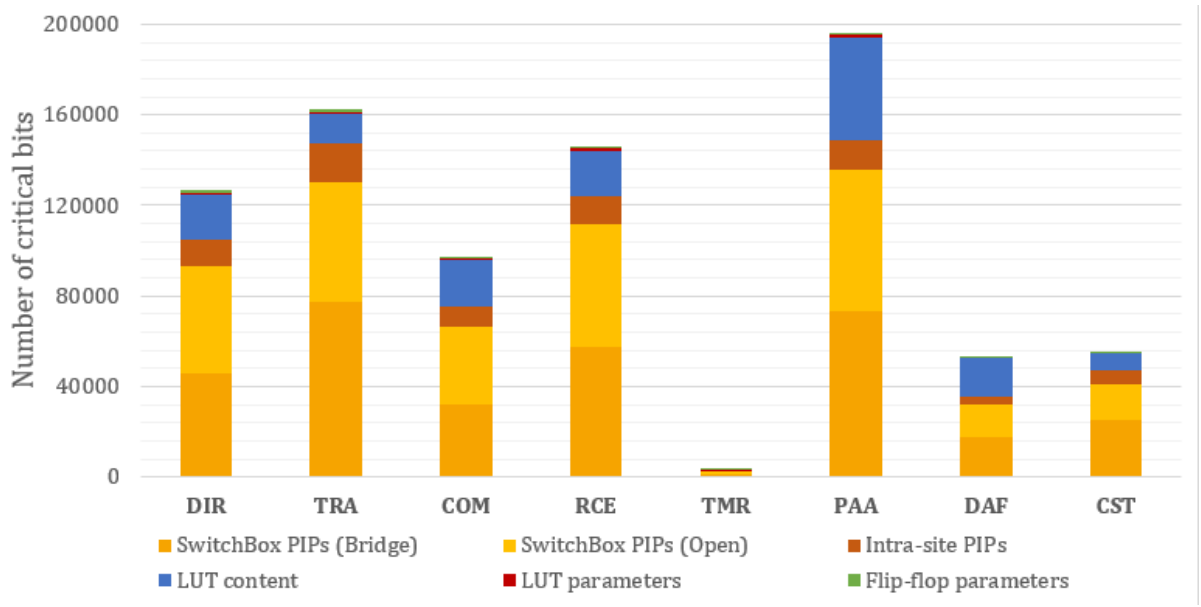


Figure 159: Number of critical bits among the different configuration bit categories, applied to eight FIR filter structures.

Transposed form (TRA) over direct form (DIR): the main difference lies in the high fanout nets used to distribute the input signals to the multipliers and to the adder tree carrying the addition of the multiplier output. The transposed form uses a set of binary adders separated by register stages while the direct form uses a multiple input adder to add all multipliers outputs at the same time. The critical bit proportion analysis reveals that the transposed form benefits from a reduction of the critical bits related to the LUT content as binary adder use only two input LUTs while the direct form uses bigger LUTs to carry the multiple input addition stage. However, this reduction is totally overwhelmed by an increase of the routing resources susceptibility due to the increased number of connections in the adder stage and in the input signal distribution network.

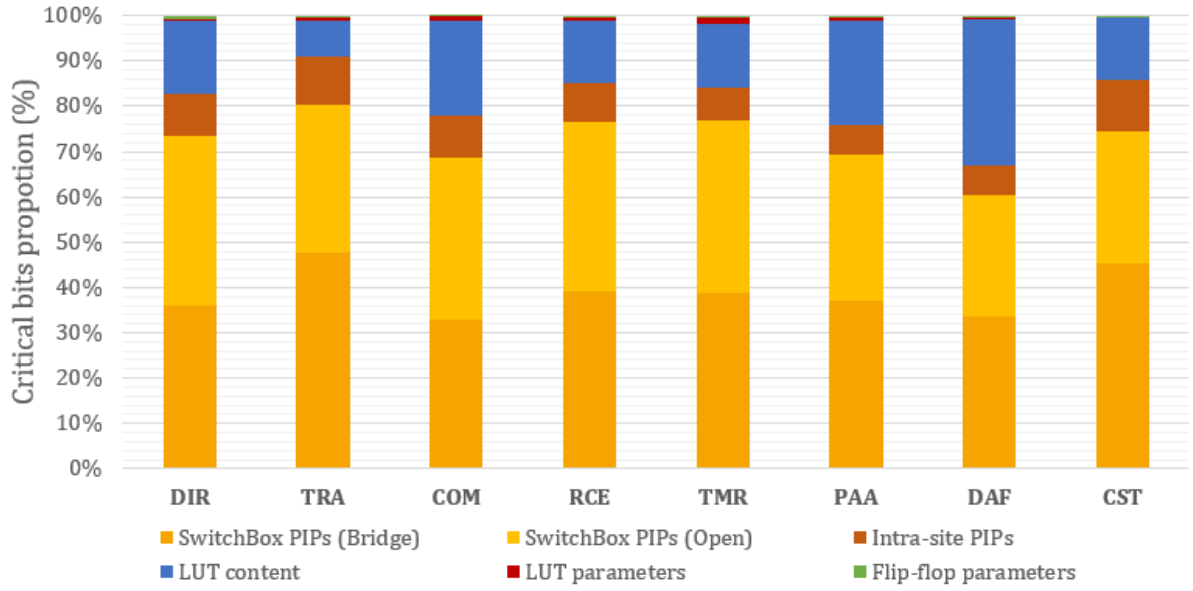


Figure 160: Proportion of critical bit among the different configuration bit categories, applied to eight FIR filter structures.

Combinatorial (COM) multiplier over pipelined multiplier (DIR): as expected, the number of critical bits related to intra-slice multiplexers and flip-flops is reduced but the main sensitivity reduction lies in the PIP saving due to the shortening of the net's lengths imposed by higher timing constraints.

Flip-flop control signals usage: results confirm that the increased sensitivity is due to the additional PIP required to distributed these control signals.

Triplication: an in-depth analysis of the location of fault inside the circuit was performed and reveled that most critical bits are related to the majority voter resources and to the clock distribution network.

PAA multiplier: the number of critical bits is increased for LUT content and switchbox PIP categories. This is simply explained by the increase number of LUT and the increased size of the LUTs.

Regarding the filter with constant coefficients, the assumption made previously are confirmed. The distributed arithmetic filter (DAF), while using less LUT than the constant multiplier filter (CST), has more critical bits related to the LUT content. This is due to the utilization of bigger LUT (mostly 6-inputs LUTs) to hold the precomputed coefficient combination while the constant multipliers use mainly 2-inputs LUTs. However, this high LUT count (for CST filter) comes at the expense of a higher routing network sensitivity due to and increase number of nets to be propagated.

More generally, the susceptibility to configuration memory upsets of arithmetic circuits is driven by two main trends. In one hand, the more compact the calculations are (by exploiting the 6 inputs of the instantiated LUTs), the more the sensitivity related to the contents of the LUT will be important. In another hand, the more we try to spread the calculation on a higher LUT count of small size, the more PIPs related CRAM bits are required to propagated this increased net count. As a general observation, the routing resources represent between 66% and 91% percent of the critical bits and close to 99% when including the LUT content bits. However, the remaining bits

should not be omitted as they can be the source of single point of failures on TMR protected circuits.

5.4.3. EXECUTION TIME

The execution time of the program has been measured for all the filter circuits tested previously executed on a desktop computer with an *Intel* Xeon Bronze 3204 and 64Go of RAM. This execution time is then correlated to the number of cells contained in each design as shown in Figure 161.

A linear trend in execution time as a function of the number of cells can be observed. However, strong variations of the execution time can be noted for designs having approximately the same number of cells. The execution time is indeed impacted by many factors related to the number of critical bits to propagate. The computational load is dominated by the propagation of errors from the inputs to the outputs of the logic gates because the calculation of timestamps for each input combination must be calculated for each sensitive bit present on the inputs. The number of logic stages that separate the inputs and outputs of the circuit as well as the error masking rate are therefore predominant factors for the execution time.

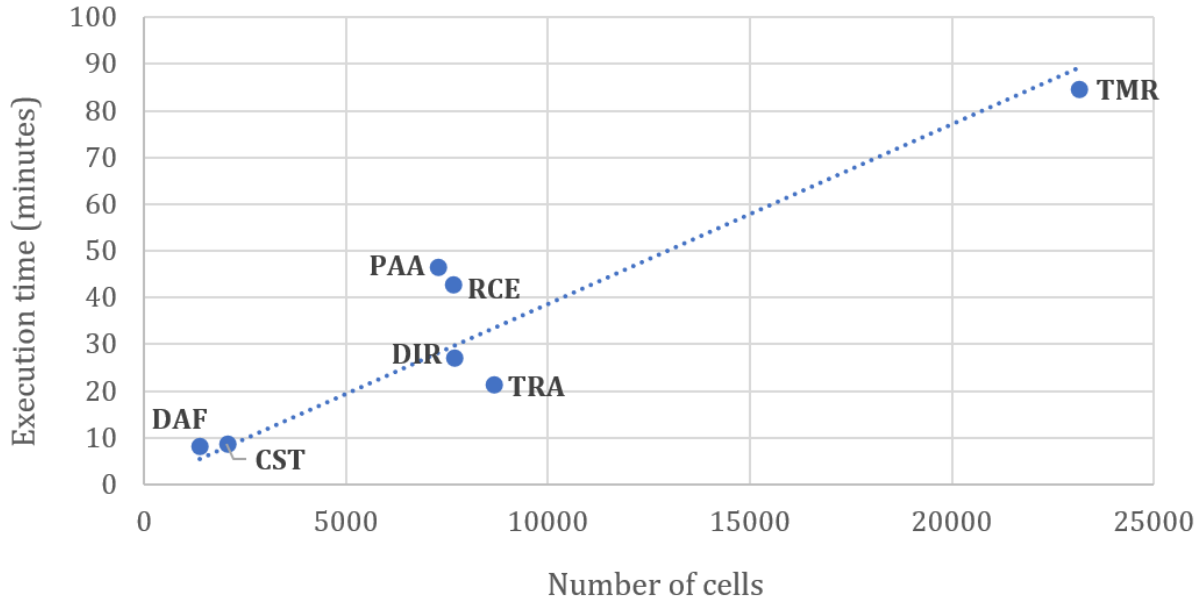


Figure 161: Execution time of the program to extract the number of critical bits of the different filter structures.

To be noted that in this study, a simulation length of 128 timestamps was chosen. No investigation on the influence of the number of timestamps over the execution time have been carried but the simulation length is expected to have a major influence on the execution time of the program. Indeed, it has been observed that the calculation time is highly impacted by the propagation of errors across LUTs. Unlike multiplexers, flip-flops and CARRY4 gates (which have a fixed logic function and can thus benefit from important optimization in error propagation calculation thanks to the use of `BigInteger` and bitwise operators) the propagation condition of LUT must be computed "on-the-fly". Therefore, the `BigInteger` data structure cannot be used to optimize the propagation timestamps computations. The timestamps are then represented by list of integers for which the computation is expected to be proportional to the simulation length. The propagation across LUT algorithm (described in section 5.3.3) could be further optimized by pre-

computing `BigInteger` to represent the timestamps for each possible input combination. Parallelization could also be implemented to compute the propagation of several sensitive bits simultaneously.

Nevertheless, the computation time of the program remains very limited confirming a clear advantage over fault injection.

5.5. CONCLUSION

In this chapter, a new software, based on an analytical approach, has been proposed to estimate the susceptibility to SEU on the configuration memory of circuits implemented on *Xilinx 7 series* SRAM-based FPGAs.

First, the state of the art on bitstream reverse engineering techniques, on failure analysis for configuration memory related errors and on susceptibility analysis tools has been described.

Then, based on the decoding of the *7 series* FPGA bitstream and by performing localized fault injections on specific memory locations, a failure model has been established for the majority of the logic and routing resources that compose the fabric of these FPGAs. For each type of resource, the configuration bits that can possibly induce an error have been identified along with the conditions in which the error can be activated. Additionally, for each logic gate, the conditions for the propagation of errors on the inputs to the output have been defined.

Based on this failure model, a navigation algorithm was developed to parse the circuit netlist file, navigate through all the logic gates and nets of the circuit to extract the associated sensitive configuration bits. Based on the user-defined workload of the circuit, extracted from behavioral simulation, the criticality of each sensitive bit is then evaluated by identifying the conditions that allow errors to propagate to the primary outputs of the circuit.

As a result, this approach allows to identify and count the number of critical bits of any circuit. The tools performances have been validated by confronting its results with those obtained through fault injections and through radiation experiments over a wide panel of arithmetic circuits. These validation steps have proven the ability of the proposed tool to predict the sensitivity of these design while providing the following advantages:

- It provides a drastic reduction of the execution time over fault injection (max 84 mins for 20.000 logic gates).
- It can be applied to any type of (unencrypted) circuits without requiring any built-in nor external testing system.
- Different input stimulus can be experienced without circuit modification to evaluate the workload impact on the system reliability (this feature could also be useful to evaluate ATPG tool performances).
- The type and the position in the FPGA fabric of every critical bit can be extracted along with the associated resource.
- It automatically provides a report with the detailed number of critical bits for each type of configuration bits.
- It provides a great visibility over the reliability of each node as the number of critical bits associated with each internal net of the device can be extracted. This feature can be used

to identify the most critical points of their designs and selectively apply a triplication scheme.

This tool could be very useful to all FPGA designers who want to quickly evaluate the reliability of their circuits, compare the sensitivity of different implementations, identify critical points in their design and evaluate the effectiveness of the applied fault-tolerance strategy.

Several improvements could be made to improve the results accuracy and the range of application that can be analyzed. For example, the real activation conditions of PIPs could be properly considered by further investigating the routing faults behavior. Similarly, by extending the localized fault injection on hardcoded blocks such as PLL, DSP and IOs, their failure model could be completed and fully integrated is the analysis. The error propagation model could also be improved to relate the real behavior of errors propagation in feedback loops structures. In addition, new features could be enabled such as the detection of possible MBU sensitivities of circuits protected by triplication schemes.

The extension of this tool to more recent *Xilinx* FPGA series (*Ultrascale* and *Ultrasacale+*) could be easily implemented as these FPGAs are already supported by the RAPIDWRIGHT API. However, even if the configuration memory architectures are apparently close to the one of *7 series* FPGAs, the exact failure model of these new series cannot be updated at the moment due to lack of information on the bitstream composition. While the *7 series* FPGAs failure model could be identically applied to these new components, the analytical results might be inaccurate. The establishment of the failure model of *Ultrascale* and *Ultrascale+* FPGAs could be enabled in the future by the currently ongoing bitstream decoding within the framework of the PROJECT U-RAY [195].

General Conclusion

In this study, several methodologies to assess the reliability of FPGA-based systems against radiation have been proposed. The test methodologies provide various advantages in terms of standardization of test results, visibility on degradation and failure mechanisms, and how test results can be exploited to provide designers with the insight needed to address the radiation reliability constraint.

In the first chapter, the different environments in which the radiative constraints must be considered to ensure the reliability of electronic systems have been described. The interactions of particles and photons with matter have been discussed along with a general overview of the radiation effects on electronic components. Two main types of effects are to be considered: the cumulative effects (TID and TNID) resulting in a progressive degradation of the component parameters, mainly translated at system level by a propagation delay drift and an increase of the power consumption. The Single Event Effects (SEE) that are generated by the interaction with a single particle are subdivided in different categories depending on the type of effect and the affected resources. For digital devices, the predominant SEEs are Single Event Upset (SEU), corrupting the content of memory elements, Single Event Transient (SET) generating current pulses in the combinatorial logic and Single Event Latchup (SEL), characterized by a sudden increase in power consumption that can lead to a deactivation of all or parts of the component.

In the second chapter, the architecture of FPGAs has been detailed along with a description of the main radiation effects and the associated failure mechanisms on these components. Depending on the type of technology used to store the configuration memory, the behavior under radiation can be very different. Flash-based FPGAs are particularly sensitive to dose effects that can easily compromise the contents of memory cells and the associated programming circuitry. As for SRAM-based FPGAs, the main concern is the sensitivity of the SRAM cells to SEU. The configuration memory corruption can result in a modification of the implemented circuit topology. Different state-of-the-art hardening technique at process, layout and design level have also been discussed.

In the third chapter, a new testing methodology has been proposed to assess the TID-induced parametric degradation in FPGAs. This approach is based on the development of dedicated benchmarking structures allowing the extension of the timing degradation evaluation to all the logical and routing resources of the device. In addition, a new cost-effective technique for in-situ delay measurement have been proposed based on the use of reprogrammable PLLs embedded in the device. The proposed approach brings a step forward towards the standardization of TID test results by extracting the individual degradation of each type of resource independently from the test structure in which they are integrated. The standardization of the results is pushed even further by proposing a method to reduce the impact of thermal experimental conditions on the test results. The proposed methodology has been validated through X-ray testing on different FPGA families.

In the fourth chapter, an FPGA SEE testing methodology based on a benchmarking approach has been proposed. This methodology allows to reproduce the diversity and complexity of interactions found in real circuits while providing good visibility on the predominant failure

mechanisms. Additionally, the benchmarking approach can be used to compare the radiation sensitivity of different components and to identify their main vulnerabilities while providing device specific recommendation regarding the mitigation techniques to be applied. Dedicated structures have been developed around multipliers. By proposing several types of implementation of the same arithmetic function, these test structures allow to test circuits with a large architectural diversity and to evaluate the influence of different parameters on the SEE sensitivity: the number and type of instantiated resources, the number and type of connections, etc. This methodology has been applied to different FPGA components across three irradiation campaigns with neutrons beam and protons beam. For each experiment, the benchmark was supplemented with additional test structures to improve its diversity and identify the influence of new implementation parameters. The experimental results revealed different trends on the sensitivities of circuits implemented on FPGA. First, the sensitivity on SRAM-based FPGAs are largely dominated by SEUs on the configuration memory making them much more sensitive than their flash-based counterparts. The susceptibility characterization of a circuit to SEUs in configuration memory turned out to be a complex task: the type, the size and number of resources used, their flexibility and the way they are configured must be considered. The number of used extra-slice routing segments also appears to be an important factor due to the possibility of errors on the programmable interconnection points. This susceptibility to SEUs in the configuration memory has also been addressed through fault injections. Indeed, for the *Xilinx* FPGAs, the use of the soft error mitigation (SEM) controller during the test campaigns allowed not only to limit the accumulation of bitflips in the configuration memory but also to extract its cross section. Fault emulation in the configuration memory can then be performed with this same controller to extract the number of critical bits and estimate the susceptibility of the circuit using the cross section extracted during radiation tests. The correlation of the fault injection results with those from the radiation experiments has also contributed to a better understanding of the involved failure mechanisms.

To go further in the analysis of SEU susceptibility in the configuration memory of FPGA-based system, a new software has been developed to analyze the netlist of circuits implemented on FPGA and identify its configuration memory related vulnerabilities. First, a detailed failure model has been established based on bitstream composition analysis and localized fault injections. For each type of resource, the different configuration bits likely to generate errors have been identified and the conditions of activation and propagation of these errors have been established. Based on this failure model, an algorithm was developed to scan all the nets and logic gates of the circuit and identify for each resource encountered, the type and number of sensitive configuration bits. By integrating the workload of the circuit (extracted from the behavioral simulation), the activation and propagation conditions are checked on-the-fly to determine which of the detected sensitive bits may actually generate errors that propagate to the system outputs (critical bits). The results of the proposed methodology have been confronted with proton tests and fault injection results from previous experiments to prove its effectiveness. This approach allows a drastic reduction of the execution time (83 minutes for a 20.000 cells circuit) compared to fault injection (days to weeks) and gives a better visibility on the most sensitive area of the design and on the predominance of the different failure mechanisms. The developed software could then provide FPGA designers with the necessary tools to measure, from the early design phases, the sensitivity

of their circuit, identify the most sensitive areas (potentially usable to apply a selective triplication) and evaluate the effectiveness of the implemented soft error mitigation techniques.

This software is currently based on *Xilinx 7 series* FPGAs but the proposed methodology could be extended to other FPGA families. It currently focuses on the effects on the configuration memory which are predominant for SRAM FPGAs but the algorithms developed to explore a circuit and determine the conditions of activation and propagation of errors could be used more widely to evaluate the influence of the other types of radiation effects. For example, the failures related to SEUs on flip-flops could be addressed statistically: the probability of a SEU on a given flip-flop can be determined by counting the number of timestamps where the generated error can propagate to the outputs of the system and dividing it by the total number of timestamps used in the simulation. Based on the propagation delay information contained in the checkpoint file, the probabilities of SEU captures and temporal masking phenomena could also be integrated into the failure probability calculation. Concerning the SETs, further investigations could be conducted through radiation tests to analyze the effects of generation, broadening/narrowing and amplification/attenuation of electrical pulses for each type of logic gate of the FPGA. Based on these experimental results, an equivalent model for each element could be built and integrated into the software to estimate the overall sensitivity of the circuit to SET. Such a failure model taking into account SEUs on user flip-flops and SETs in the combinatorial logic could thus complete the reliability analysis on SRAM FPGAs but also extend the use of this type of analytical tool to Flash FPGAs and digital ASICs.

More generally, this work proposes several improvements to the standard FPGA radiation test methodologies to overcome the dependency of the test results on the component configuration. A step forward is made regarding the standardization of the test results and their reusability to estimate the reliability of final applications. The radiation sensitivity, accurately estimated on a large panel of representative circuits, supplements the timing and power constraints to provide designers with all the information required to carefully select the types of implementations tailored to their mission profiles. Finally, the analytical approach is proposed as a complementary tool to improve the insight on the error generation and propagation mechanisms and allows to gain precision on the reliability estimation of any circuit in the early design phases. Designers can use this tool to quickly and easily understand the sources of failure within their circuits and thus improve their reliability.

Whatever the maturity of these analysis tools, radiation testing will probably remain mandatory and will continue to evolve in the upcoming years to study the reliability of new components put on the market and to identify the failure mechanisms and their predominance. The benchmarking approach proposed in this study could then be extended by integrating other types of test structures to follow the rapid evolution and increasing complexity of FPGA components. The extensive use of system-on-chip will also require new test methodologies to take into account the new integrated features and the on-chip coupling of the FPGA fabric with hardwired processing units (CPU, GPU, DSP, artificial intelligence engines, etc.).

Publications

International Conferences:

- G. Tsiligiannis, Antoine Touboul, Gaetan Bricas, Tadec Maraine, Jerome Boch, Frederic Wrobel, Alain Michez, Frederic Saigne, Alain Godot, Asenath Etile, Thibaud Durand, Romain Sueur, Didier Farigoule, Philippe Girones., "Evaluation and Analysis of Technologies for Robotic Platforms for the Nuclear Decommissioning," in 2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

- G. Bricas, G. Tsiligiannis, A. Touboul, J. Boch, M. Kastriotou, and C. Cazzaniga, "Novel FPGA Radiation Benchmarking Structures", RADECS 2020

- G. Bricas, G. Tsiligiannis, A. Touboul, J. Boch, M. Kastriotou, C. Cazzaniga, C.D. Frost, L. Dilillo, L. Matana Luza., "On the evaluation of FPGA radiation benchmarks," ESREF 2021

- G. Bricas, G. Tsiligiannis, A. Touboul, J. Boch, and T. Maraine, "FPGA Benchmarking structures dedicated to TID parametric degradation evaluation," RADECS 2021

Journal Papers:

- G. Bricas G. Tsiligiannis, A. Touboul, J. Boch, M. Kastriotou, C. Cazzaniga, C.D. Frost, L. Dilillo, L. Matana Luza, "On the evaluation of FPGA radiation benchmarks," Microelectronics Reliability, vol. 126, p. 114276, Nov. 2021, doi: 10.1016/j.microrel.2021.114276.

- G. Bricas, G. Tsiligiannis, A. Touboul, J. Boch, T. Maraine and F. Saigné, "FPGA Benchmarking structures dedicated to TID parametric degradation evaluation," in *IEEE Transactions on Nuclear Science*, doi: 10.1109/TNS.2022.3180107.

References

[1] J. T. Wallmark and S. M. Marcus, "Minimum Size and Maximum Packing Density of Nonredundant Semiconductor Devices," *Proceedings of the IRE*, vol. 50, no. 3, pp. 286–298, Mar. 1962, doi: 10.1109/JRPROC.1962.288321.

[2] "White Paper: Is an ASIC Right for Your Next IoT Product?," *AnySilicon*, Jun. 27, 2017. https://anysilicon.com/asic-right-next-iot-product/ (accessed Sep. 08, 2022).

[3] Sé. Bourdarie and M. Xapsos, "The Near-Earth Space Radiation Environment," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 4, pp. 1810–1832, Aug. 2008, doi: 10.1109/TNS.2008.2001409.

[4] H. Schlaepfer, "Cosmic rays," *Spatium, 11*, 2003. Accessed: Feb. 09, 2022. [Online]. Available: http://www.issibern.ch/PDF-Files/Spatium_11.pdf

[5] "Monthly and smoothed sunspot number | SILSO." https://wwwbis.sidc.be/silso/monthlyssnplot (accessed Feb. 09, 2022).

[6] B. H. Mauk, N. J. Fox, S. G. Kanekal, R. L. Kessel, D. G. Sibeck, and A. Ukhorskiy, "Science Objectives and Rationale for the Radiation Belt Storm Probes Mission," *Space Sci Rev*, vol. 179, no. 1–4, pp. 3–27, Nov. 2013, doi: 10.1007/s11214-012-9908-y.

[7] K. K. Robert Baumann, "Radiation Handbook for Electronics," p. 118.

[8] "Our Radiation Software," *TRAD*. https://www.trad.fr/en/space/omere-software/ (accessed Mar. 18, 2022).

[9] J. F. Ziegler, "Terrestrial cosmic ray intensities," *IBM J. Res. & Dev.*, vol. 42, no. 1, pp. 117–140, Jan. 1998, doi: 10.1147/rd.421.0117.

[10] E. Normand, "Single-event effects in avionics," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 461–474, Apr. 1996, doi: 10.1109/23.490893.

[11] T. C. May and M. H. Woods, "A New Physical Mechanism for Soft Errors in Dynamic Memories," in *16th International Reliability Physics Symposium*, Apr. 1978, pp. 33–40. doi: 10.1109/IRPS.1978.362815.

[12] R. C. Baumann and E. B. Smith, "Neutron-induced boron fission as a major source of soft errors in deep submicron SRAM devices," in *2000 IEEE International Reliability Physics Symposium Proceedings. 38th Annual (Cat. No.00CH37059)*, Apr. 2000, pp. 152–157. doi: 10.1109/RELPHY.2000.843906.

[13] G. Santin, P. Truscott, R. Gaillard, and R. G. Alía, "Radiation environments: space, avionics, ground and below," p. 142.

[14] D. J. Fitzgerald and E. H. Snow, "Comparison of surface and bulk effects of nuclear reactor radiation on planar devices," *IEEE Transactions on Electron Devices*, vol. 15, no. 3, pp. 160–163, Mar. 1968, doi: 10.1109/T-ED.1968.16154.

[15] K. Roeed *et al.*, "Method for Measuring Mixed Field Radiation Levels Relevant for SEEs at the LHC," *IEEE Transactions on Nuclear Science*, vol. 59, no. 4, pp. 1040–1047, Aug. 2012, doi: 10.1109/TNS.2012.2183677.

[16] "ISIS ChipIr." https://www.isis.stfc.ac.uk/Pages/Chipir.aspx (accessed May 18, 2022).

[17] "Plateforme Technologique PRESERVE," *IES - Institut d'Electronique et des Systèmes*. https://www.ies.umontpellier.fr/la-recherche-et-linnovation/equipements-et-plateformes/plateforme-technologique-preserve/ (accessed Sep. 08, 2022).

[18] F. Wrobel, "Fundamentals on Radiation-Matter Interaction," *IEEE RADECS 2005, Cap d'Agde, France, 19-23 septembre 2005*.

[19] J. F. Ziegler, "SRIM-2003," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 219–220, pp. 1027–1036, Jun. 2004, doi: 10.1016/j.nimb.2004.01.208.

[20] T. R. Oldham and F. B. McLean, "Total ionizing dose effects in MOS oxides and devices," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 483–499, Jun. 2003, doi: 10.1109/TNS.2003.812927.

[21] R. C. Lacoe, "Improving Integrated Circuit Performance Through the Application of Hardness-by-Design Methodology," *Nuclear Science, IEEE Transactions on*, vol. 55, pp. 1903–1925, Sep. 2008, doi: 10.1109/TNS.2008.2000480.

[22] A. Bacchini, G. Furano, M. Rovatti, and M. Ottavi, “Total Ionizing Dose Effects on DRAM Data Retention Time,” *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3690–3693, Dec. 2014, doi: 10.1109/TNS.2014.2365532.
[23] R. C. Baumann, “Radiation-induced soft errors in advanced semiconductor technologies,” *IEEE Trans. Device Mater. Relib.*, vol. 5, no. 3, pp. 305–316, Sep. 2005, doi: 10.1109/TDMR.2005.853449.
[24] P. E. Dodd and L. W. Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics,” *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 583–602, Jun. 2003, doi: 10.1109/TNS.2003.813129.
[25] M. Berg, “Field Programmable Gate Array (FPGA) Single Event Effect (SEE) Radiation Testing,” p. 54.
[26] N. A. Dodds *et al.*, “Selection of Well Contact Densities for Latchup-Immune Minimal-Area ICs,” *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3575–3581, Dec. 2010, doi: 10.1109/TNS.2010.2082562.
[27] R. Ecoffet, “Overview of In-Orbit Radiation Induced Spacecraft Anomalies,” *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 1791–1815, Jun. 2013, doi: 10.1109/TNS.2013.2262002.
[28] R. Singh, “FPGA vs ASIC: Differences between them and which one to use?,” *Numato Lab Help Center*. https://numato.com/blog/differences-between-fpga-and-asics/ (accessed Mar. 24, 2022).
[29] E. Ahmed and J. Rose, “The effect of LUT and cluster size on deep-submicron FPGA performance and density,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 3, pp. 288–298, Mar. 2004, doi: 10.1109/TVLSI.2004.824300.
[30] “7 Series FPGAs Configurable Logic Block User Guide (UG474),” p. 74, 2016.
[31] I. Kuon, R. Tessier, and J. Rose, “FPGA Architecture: Survey and Challenges,” *FNT in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007, doi: 10.1561/1000000005.
[32] “7 Series DSP48E1 Slice User Guide (UG479),” p. 58, 2018.
[33] “7 Series FPGAs Memory Resources User Guide,” p. 88, 2019.
[34] “Vivado Design Suite User Guide: Partial Reconfiguration (UG909),” p. 147, 2018.
[35] “Flash*Freeze Control Using JTAG,” p. 7.
[36] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael, and P. Alfke, “The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs,” *IEEE Trans. Device Mater. Relib.*, vol. 5, no. 3, pp. 317–328, Sep. 2005, doi: 10.1109/TDMR.2005.854207.
[37] X. Yao, N. Hindman, L. T. Clark, K. E. Holbert, D. R. Alexander, and W. M. Shedd, “The Impact of Total Ionizing Dose on Unhardened SRAM Cell Margins,” *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3280–3287, Dec. 2008, doi: 10.1109/TNS.2008.2007122.
[38] M. S. Gorbunov *et al.*, “Design of 65 nm CMOS SRAM for Space Applications: A Comparative Study,” *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1575–1582, Aug. 2014, doi: 10.1109/TNS.2014.2319154.
[39] S. Vartanian, G. R. Allen, and D. Thorbourn, “SRAM-Based FPGA: High Dose Test Methods Using Evaluation Boards,” p. 6.
[40] L. Ding *et al.*, “Analysis of TID Failure Modes in SRAM-Based FPGA Under Gamma-Ray and Focused Synchrotron X-Ray Irradiation,” *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1777–1784, Aug. 2014, doi: 10.1109/TNS.2014.2314530.
[41] H. Ito and M. Watanabe, “Total ionizing dose tolerance of the serial configuration on cyclone II FPGA,” in *2015 IEEE International Conference on Space Optical Systems and Applications (ICSOS)*, Oct. 2015, pp. 1–4. doi: 10.1109/ICSOS.2015.7425067.
[42] E. G. Stassinopoulos, G. J. Brucker, O. Van Gunten, and H. S. Kim, “Variation in SEU sensitivity of dose-imprinted CMOS SRAMs,” *IEEE Transactions on Nuclear Science*, vol. 36, no. 6, pp. 2330–2338, Dec. 1989, doi: 10.1109/23.45444.
[43] L. A. Tambara *et al.*, “Soft error rate in SRAM-based FPGAs under neutron-induced and TID effects,” in *2014 15th Latin American Test Workshop - LATW*, Mar. 2014, pp. 1–6. doi: 10.1109/LATW.2014.6841920.
[44] J. Benfica *et al.*, “Analysis of SRAM-Based FPGA SEU Sensitivity to Combined EMI and TID-Imprinted Effects,” *IEEE Transactions on Nuclear Science*, vol. 63, no. 2, pp. 1294–1300, Apr. 2016, doi: 10.1109/TNS.2016.2523458.

[45] S. Gerardin *et al.*, “Radiation Effects in Flash Memories,” *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 1953–1969, Jun. 2013, doi: 10.1109/TNS.2013.2254497.
[46] E. S. Snyder, P. J. McWhorter, T. A. Dellin, and J. D. Sweetman, “Radiation response of floating gate EEPROM memory cells,” *IEEE Transactions on Nuclear Science*, vol. 36, no. 6, pp. 2131–2139, Dec. 1989, doi: 10.1109/23.45415.
[47] T. R. Oldham *et al.*, “SEE and TID Characterization of an Advanced Commercial 2Gbit NAND Flash Nonvolatile Memory,” *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3217–3222, Dec. 2006, doi: 10.1109/TNS.2006.885843.
[48] D. N. Nguyen, S. M. Guertin, G. M. Swift, and A. H. Johnston, “Radiation effects on advanced flash memories,” *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, pp. 1744–1750, Dec. 1999, doi: 10.1109/23.819148.
[49] G. Cellere *et al.*, “Radiation effects on floating-gate memory cells,” *IEEE Transactions on Nuclear Science*, vol. 48, no. 6, pp. 2222–2228, Dec. 2001, doi: 10.1109/23.983199.
[50] G. Cellere, A. Paccagnella, A. Visconti, and M. Bonanomi, “Transient conductive path induced in floating gate memories by single ions,” in *2005 International Conference on Integrated Circuit Design and Technology, 2005. ICICDT 2005.*, May 2005, pp. 29–32. doi: 10.1109/ICICDT.2005.1502583.
[51] N. Z. Butt and M. Alam, “Modeling single event upsets in Floating Gate memory cells,” in *2008 IEEE International Reliability Physics Symposium*, Apr. 2008, pp. 547–555. doi: 10.1109/RELPHY.2008.4558944.
[52] A. V. Microsemi CA, USA, “Single Event Effects - A Comparison of Configuration Upsets and Data Upsets.” Nov. 2015.
[53] N. Rezzak, D. Dsilva, J.-J. Wang, and N. Jat, “SET and SEFI Characterization of the 65 nm SmartFusion2 Flash-Based FPGA under Heavy Ion Irradiation,” in *2015 IEEE Radiation Effects Data Workshop (REDW)*, Boston, MA, USA, Jul. 2015, pp. 1–4. doi: 10.1109/REDW.2015.7336733.
[54] J. M. Benedetto and C. C. Hafer, “Ionizing radiation response of an amorphous silicon based antifuse,” in *1997 IEEE Radiation Effects Data Workshop NSREC Snowmass 1997. Workshop Record Held in conjunction with IEEE Nuclear and Space Radiation Effects Conference*, Jul. 1997, pp. 101–104. doi: 10.1109/REDW.1997.629806.
[55] J.-J. Wang *et al.*, “A Novel 65 nm Radiation Tolerant Flash Configuration Cell Used in RTG4 Field Programmable Gate Array,” *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, Dec. 2015, doi: 10.1109/TNS.2015.2495262.
[56] M. Berg *et al.*, “Characterizing the Effects of Single Event Upsets on Synchronous Data Paths,” *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2697–2703, Aug. 2013, doi: 10.1109/TNS.2013.2273938.
[57] E. Keren, S. Greenberg, N. M. Yitzhak, D. David, N. Refaeli, and A. Haran, “Characterization and Mitigation of Single-Event Transients in Xilinx 45-nm SRAM-Based FPGA,” *IEEE Transactions on Nuclear Science*, vol. 66, no. 6, pp. 946–954, Jun. 2019, doi: 10.1109/TNS.2019.2916151.
[58] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, “Single Event Transients in Digital CMOS—A Review,” *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 1767–1790, Jun. 2013, doi: 10.1109/TNS.2013.2255624.
[59] P. Adell and G. Allen, “Assessing and Mitigating Radiation Effects in Xilinx FPGAs,” p. 36.
[60] L. Ding, Z. Wang, W. Chen, Y. Li, Q. Zhang, and D. Yu, “Bitstream-based simulation for configuration SEUs in Xilinx Virtex-4 FPGAs,” in *2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Sep. 2016, pp. 1–4. doi: 10.1109/RADECS.2016.8093217.
[61] “Device Reliability Report Second Half 2021,” p. 90, 2022.
[62] Xilinx, “virtex5qv-product-table.” https://www.xilinx.com/content/dam/xilinx/publications/prod_mktg/virtex5qv-product-table.pdf (accessed Jul. 12, 2022).
[63] “Space-Grade Virtex-4QV Family Overview (DS653),” p. 8, 2014.
[64] G. Allen, G. Swift, and C. Carmichael, “Virtex-4VQ Static SEU Characterization Summary,” p. 22.

[65] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "Static Proton and Heavy Ion Testing of the Xilinx Virtex-5 Device," in *2007 IEEE Radiation Effects Data Workshop*, Jul. 2007, pp. 177–184. doi: 10.1109/REDW.2007.4342561.
[66] J. Pellish, "Xilinx Virtex-5QV (V5QV) Independent SEU Data," p. 76.
[67] D. M. Hiemstra and V. Kirischian, "Single Event Upset Characterization of the Virtex-6 Field Programmable Gate Array Using Proton Irradiation," in *2012 IEEE Radiation Effects Data Workshop*, Jul. 2012, pp. 1–4. doi: 10.1109/REDW.2012.6353716.
[68] G. Tsiligiannis and S. Danzeca, "SmartFusion2 and Artix 7 radiation test results for the new developments." https://slideplayer.com/slide/13055387/ (accessed Jul. 13, 2022).
[69] D. S. Lee, M. Wirthlin, G. Swift, and A. C. Le, "Single-Event Characterization of the 28 nm Xilinx Kintex-7 Field-Programmable Gate Array under Heavy Ion Irradiation," in *2014 IEEE Radiation Effects Data Workshop (REDW)*, Paris, France, Jul. 2014, pp. 1–5. doi: 10.1109/REDW.2014.7004595.
[70] G. M. Swift and S. Engineering, "Overview of the XRTC Single-Event Test Results on the Xilinx 7-Series FPGAs," p. 22.
[71] A. M. Keller, T. A. Whiting, K. B. Sawyer, and M. J. Wirthlin, "Dynamic SEU Sensitivity of Designs on Two 28-nm SRAM-Based FPGA Architectures," *IEEE Transactions on Nuclear Science*, vol. 65, no. 1, pp. 280–287, Jan. 2018, doi: 10.1109/TNS.2017.2772288.
[72] P. Maillard, "Total Ionizing Dose and Single-Events characterization of Xilinx 20nm Kintex UltraScale™," p. 4, 2019.
[73] "Radiation Tolerant Kintex UltraScale XQRKU060 FPGA Data Sheet," p. 101, 2022.
[74] C. Johansson and T. Månefjord, "Characterization and Considerations for Upset in FPGA," in *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, Oct. 2018, pp. 1–4. doi: 10.1109/NORCHIP.2018.8573506.
[75] M. Glorieux *et al.*, "Single-Event Characterization of Xilinx UltraScale+® MPSOC under Standard and Ultra-High Energy Heavy-Ion Irradiation," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, Jul. 2018, pp. 1–5. doi: 10.1109/NSREC.2018.8584296.
[76] P. Maillard, M. Hart, J. Barton, J. Arver, and C. Smith, "Neutron, 64 MeV Proton & Alpha Single-event Characterization of Xilinx 16nm FinFET Zynq® UltraScale+™ MPSoC," in *2017 IEEE Radiation Effects Data Workshop (REDW)*, Jul. 2017, pp. 1–5. doi: 10.1109/NSREC.2017.8115449.
[77] P. Maillard, Y. P. Chen, J. Barton, and M. L. Voogel, "Single Event Latchup (SEL) and Single Event Upset (SEU) Evaluation of Xilinx 7nm Versal™ ACAP programmable logic (PL)," in *2021 IEEE Radiation Effects Data Workshop (REDW)*, Jul. 2021, pp. 1–6. doi: 10.1109/NSREC45046.2021.9679343.
[78] S. L. Clark, K. Avery, and R. Parker, "TID and SEE testing results of Altera Cyclone field programmable gate array," in *2004 IEEE Radiation Effects Data Workshop (IEEE Cat. No.04TH8774)*, Jul. 2004, pp. 88–90. doi: 10.1109/REDW.2004.1352911.
[79] B. Cheynis and L. Ducroux, "Radiation effects on V0 detector elements," p. 15.
[80] A. B. Sanders, K. A. LaBel, C. Poivey, and J. A. Seely, "ALTERA STRATIXTM EP1S25 FIELD-PROGRAMMABLE GATE ARRAY (FPGA)," p. 9, 2005.
[81] G. R. Allen and G. M. Swift, "Single Event Effects Test Results for Advanced Field Programmable Gate Arrays," in *2006 IEEE Radiation Effects Data Workshop*, Jul. 2006, pp. 115–120. doi: 10.1109/REDW.2006.295478.
[82] C. Färber, U. Uwer, D. Wiedner, B. Leverington, and R. Ekelhof, "Radiation tolerance tests of SRAM-based FPGAs for the potential usage in the readout electronics for the LHCb experiment," *J. Inst.*, vol. 9, no. 02, pp. C02028–C02028, Feb. 2014, doi: 10.1088/1748-0221/9/02/C02028.
[83] G. R. Allen, G. Madias, E. Miller, and G. Swift, "Recent Single Event Effects Results in Advanced Reconfigurable Field Programmable Gate Arrays," in *2011 IEEE Radiation Effects Data Workshop*, Jul. 2011, pp. 1–6. doi: 10.1109/REDW.2010.6062511.
[84] C. Poivey, M. Grandjean, and F. X. Guerre, "Radiation Characterization of Microsemi ProASIC3 Flash FPGA Family," in *2011 IEEE Radiation Effects Data Workshop*, Jul. 2011, pp. 1–5. doi: 10.1109/REDW.2010.6062510.
[85] "Radiation-Tolerant ProASIC3 FPGAs Radiation Effects".

[86] N. Rezzak, J.-J. Wang, D. Dsilva, and N. Jat, "TID and SEE Characterization of Microsemi's 4th Generation Radiation Tolerant RTG4 Flash-Based FPGA," presented at the 2015 IEEE Radiation Effects Data Workshop (REDW), Boston, MA, USA, Jul. 2015. doi: 10.1109/REDW.2015.7336739.
[87] "RTG4_Proton_Test_Report," p. 12.
[88] J. J. Wang, N. Rezzak, F. Hawley, G. Bakker, J. McCollum, and E. Hamdy, "Radiation Characteristics Of Field Programmable Gate Array Using Complementary-Sonos Configuration Cell," p. 6.
[89] "NanoXplore Radiative test Brave-FPGA."
[90] O. Musseau, "Single-event effects in SOI technologies and devices," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 603–613, Apr. 1996, doi: 10.1109/23.490904.
[91] J. Verbeeck, P. Leroux, and M. Steyaert, "Radiation effects upon the mismatch of identically laid out transistor pairs," in *2011 IEEE ICMTS International Conference on Microelectronic Test Structures*, Apr. 2011, pp. 194–197. doi: 10.1109/ICMTS.2011.5976845.
[92] "Mitigation of Radiation Effects in RTG4 Radiation-Tolerant Flash FPGAs WP0191 White Paper," p. 10.
[93] J.-J. Wang *et al.*, "A Novel 65 nm Radiation Tolerant Flash Configuration Cell Used in RTG4 Field Programmable Gate Array," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 3072–3079, Dec. 2015, doi: 10.1109/TNS.2015.2495262.
[94] J. L. Andrews, J. E. Schroeder, B. L. Gingerich, W. A. Kolasinski, R. Koga, and S. E. Diehl, "Single Event Error Immune CMOS RAM," *IEEE Transactions on Nuclear Science*, vol. 29, no. 6, pp. 2040–2043, Dec. 1982, doi: 10.1109/TNS.1982.4336492.
[95] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2874–2878, Dec. 1996, doi: 10.1109/23.556880.
[96] F. Wrobel, "Prepare for the next decade!," p. 204.
[97] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *2006 IEEE International Reliability Physics Symposium Proceedings*, San Jose, CA, USA, 2006, pp. 226–232. doi: 10.1109/RELPHY.2006.251221.
[98] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, Oct. 2004, doi: 10.1109/TNS.2004.834955.
[99] A. Sánchez, L. Entrena, and F. Kastensmidt, "Approximate TMR for selective error mitigation in FPGAs based on testability analysis," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Aug. 2018, pp. 112–119. doi: 10.1109/AHS.2018.8541485.
[100] "FPGA Design Solution for High-Reliability Applications," p. 4.
[101] "Precision® Hi-Rel Advanced FPGA Synthesis Datasheet," *Mentor Graphics Corp.*, 2018.
[102] "Functional Triple Modular Redundancy (FTMR)," p. 56.
[103] M. Zheng, Z. Wang, and L. Li, "DAO: Dual module redundancy with AND/OR logic voter for FPGA hardening," in *2015 First International Conference on Reliability Systems Engineering (ICRSE)*, Beijing, China, Oct. 2015, pp. 1–5. doi: 10.1109/ICRSE.2015.7366414.
[104] F. G. de Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis, "Designing fault-tolerant techniques for SRAM-based FPGAs," *IEEE Design Test of Computers*, vol. 21, no. 6, pp. 552–562, Nov. 2004, doi: 10.1109/MDT.2004.85.
[105] M. Berg, N. Program, and M. D. Berg, "Single Event Effects in FPGA Devices," p. 49.
[106] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain Crossing Errors: Limitations on Single Device Triple-Modular Redundancy Circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2037–2043, Dec. 2007, doi: 10.1109/TNS.2007.910870.
[107] T. Bates and C. P. Bridges, "Single event mitigation for Xilinx 7-series FPGAs," in *2018 IEEE Aerospace Conference*, Mar. 2018, pp. 1–12. doi: 10.1109/AERO.2018.8396520.
[108] "pg036_sem.pdf • Viewer • Documentation Portal." https://docs.xilinx.com/v/u/en-US/pg036_sem (accessed Apr. 29, 2022).
[109] NanoXplore, "Configuration Memory Integrity Check (CMIC) Application Note for NG-MEDIUM," Feb. 2021.
[110] "Intel® Stratix® 10 SEU Mitigation User Guide," p. 45.

[111] L. Ding *et al.*, "Analysis of TID Failure Modes in SRAM-Based FPGA Under Gamma-Ray and Focused Synchrotron X-Ray Irradiation," *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, Art. no. 4, Aug. 2014, doi: 10.1109/TNS.2014.2314530.
[112] J. J. Wang *et al.*, "Total ionizing dose effects on flash-based field programmable gate array," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3759–3766, Dec. 2004, doi: 10.1109/TNS.2004.839255.
[113] F. L. Kastensmidt, E. C. P. Fonseca, R. G. Vaz, O. L. Goncalez, R. Chipana, and G. I. Wirth, "TID in Flash-Based FPGA: Power Supply-Current Rise and Logic Function Mapping Effects in Propagation-Delay Degradation," *IEEE Transactions on Nuclear Science*, vol. 58, no. 4, pp. 1927–1934, Aug. 2011, doi: 10.1109/TNS.2011.2128881.
[114] S. Rezgui *et al.*, "Investigation of Low Dose Rate and Bias Conditions on the Total Dose Tolerance of a CMOS Flash-Based FPGA," *IEEE Transactions on Nuclear Science*, vol. 59, no. 1, pp. 134–143, Feb. 2012, doi: 10.1109/TNS.2011.2179316.
[115] G. Lentaris *et al.*, "TID Evaluation System With On-Chip Electron Source and Programmable Sensing Mechanisms on FPGA," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 312–319, Jan. 2019, doi: 10.1109/TNS.2018.2885713.
[116] R. Ichimiya *et al.*, "Radiation qualification of electronics components used for the ATLAS level-1 muon endcap trigger system," in *IEEE Symposium Conference Record Nuclear Science 2004.*, Oct. 2004, vol. 2, pp. 779-783 Vol. 2. doi: 10.1109/NSSMIC.2004.1462325.
[117] I. C. Lopes *et al.*, "Comparison of TID-induced Degradation of Programmable Logic Timings in Bulk 28nm and 16nm FinFET System-on- Chips under Local X-ray Irradiation," p. 5.
[118] N. Ma, S. Wang, D. Liu, and Y. Peng, "A run-time built-in approach of TID test in SRAM based FPGAs," *Microelectronics Reliability*, vol. 64, pp. 42–47, Sep. 2016, doi: 10.1016/j.microrel.2016.07.128.
[119] M. A. Kacou, F. Ghaffari, O. Romain, and B. Condamin, "FPGA static timing analysis enhancement based on real operating conditions," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, Beijing, Oct. 2017, pp. 3556–3561. doi: 10.1109/IECON.2017.8216602.
[120] "Cmod S7 Reference Manual - Digilent Reference." https://digilent.com/reference/programmable-logic/cmod-s7/reference-manual?redirect=1 (accessed May 05, 2022).
[121] "Cmod A7 - Digilent Reference." https://digilent.com/reference/programmable-logic/cmod-a7/start (accessed May 05, 2022).
[122] "DIPFORTy1 'Soft-Propeller' - Open Source Hardware - Trenz Electronic Wiki." https://wiki.trenz-electronic.de/pages/viewpage.action?pageId=20612010 (accessed May 05, 2022).
[123] "TEI0003 Resources - Public Docs - Trenz Electronic Wiki." https://wiki.trenz-electronic.de/display/PD/TEI0003+Resources (accessed May 05, 2022).
[124] N. Rezzak, J.-J. Wang, C.-K. Huang, V. Nguyen, and G. Bakker, "Total Ionizing Dose Characterization of 65 nm Flash-Based FPGA," in *2014 IEEE Radiation Effects Data Workshop (REDW)*, Paris, France, Jul. 2014, pp. 1–5. doi: 10.1109/REDW.2014.7004606.
[125] A. Michez *et al.*, "TCAD prediction of dose effects on MOSFETs with ECORCE," in *2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Oct. 2017, pp. 1–4. doi: 10.1109/RADECS.2017.8696230.
[126] A. Manuzzato, S. Gerardin, A. Paccagnella, L. Sterpone, and M. Violante, "On the Static Cross Section of SRAM-Based FPGAs," in *2008 IEEE Radiation Effects Data Workshop*, Jul. 2008, pp. 94–97. doi: 10.1109/REDW.2008.24.
[127] H. M. Quinn *et al.*, "A Test Methodology for Determining Space Readiness of Xilinx SRAM-Based FPGA Devices and Designs," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 10, pp. 3380–3395, Oct. 2009, doi: 10.1109/TIM.2009.2025469.
[128] L. Bozzoli, C. De Sio, B. Du, and L. Sterpone, "A Neutron Generator Testing Platform for the Radiation Analysis of SRAM-based FPGAs," in *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, May 2021, pp. 1–5. doi: 10.1109/I2MTC50364.2021.9459804.
[129] "7 Series FPGAs Configuration User Guide (UG470)," p. 180, 2018.

[130] "Soft Error Mitigation Controller v4.1 LogiCORE IP Product Guide • Soft Error Mitigation Controller Product Guide (PG036) • Reader • Documentation Portal." https://docs.xilinx.com/r/en-US/pg036_sem (accessed Jul. 19, 2022).
[131] P. Maillard *et al.*, "Single-Event Upsets Characterization Evaluation of Xilinx UltraScale™ Soft Error Mitigation (SEM IP) Tool," in *2016 IEEE Radiation Effects Data Workshop (REDW)*, Jul. 2016, pp. 1–4. doi: 10.1109/NSREC.2016.7891745.
[132] M. Berg, H. Kim, M. Friendlich, C. Perez, C. Seidlick, and K. Label, "Actel ProASIC A3PE3000L-PQ208 Field Programmable Gate Array Single Event Effects (SEE) High-Speed Test Plan- Phase II," p. 96.
[133] C. Leong *et al.*, "Fast radiation monitoring in FPGA-based designs," in *2015 Conference on Design of Circuits and Integrated Systems (DCIS)*, Nov. 2015, pp. 1–6. doi: 10.1109/DCIS.2015.7388590.
[134] M. Cannon, M. Wirthlin, A. Camplani, M. Citterio, and C. Meroni, "Evaluating Xilinx 7 Series GTX Transceivers for Use in High Energy Physics Experiments Through Proton Irradiation," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2695–2702, Dec. 2015, doi: 10.1109/TNS.2015.2497216.
[135] E. Gousiou, G. F. Penacoba, J. C. Cubillos, and E. Gousiou, "Radiation tests on the complete system of the instrumentation of the LHC cryogenics at the CERN Neutrinos to Gran Sasso (CNGS) test facility.," p. 4.
[136] N. J. Buchanan and D. M. Gingrich, "Proton Induced Radiation E ects on a Xilinx FPGA and Estimates of SEE in the ATLAS Environment." http://cds.cern.ch/record/684188/files/larg-2001-011.pdf?version=1 (accessed Jul. 22, 2022).
[137] H. Quinn *et al.*, "Using Benchmarks for Radiation Testing of Microprocessors and FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 6, pp. 2547–2554, Dec. 2015, doi: 10.1109/TNS.2015.2498313.
[138] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design & Test of Computers*, vol. 17, no. 3, pp. 44–53, Jul. 2000, doi: 10.1109/54.867894.
[139] G. Squillero, "PoliTo ITC99 (I99T)." Jun. 27, 2022. Accessed: Aug. 29, 2022. [Online]. Available: https://github.com/squillero/itc99-poli
[140] M. Kumm, S. Abbas, and P. Zipf, "An Efficient Softcore Multiplier Architecture for Xilinx FPGAs," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, Lyon, France, Jun. 2015, pp. 18–25. doi: 10.1109/ARITH.2015.17.
[141] M. Kumm and J. Wilkomm, "Ternary Adder IP Cores." http://www.martin-kumm.de/wiki/lib/exe/fetch.php?media=FPGA_Cores:ternary_adder.pdf (accessed Jul. 22, 2022).
[142] A. Booth, "A signed binary multiplcation technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, pp. 236–240, 1951.
[143] H. Parandeh-Afshar and P. Ienne, "Measuring and Reducing the Performance Gap between Embedded and Soft Multipliers on FPGAs," in *2011 21st International Conference on Field Programmable Logic and Applications*, Sep. 2011, pp. 225–231. doi: 10.1109/FPL.2011.48.
[144] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *2009 International Conference on Field Programmable Logic and Applications*, Aug. 2009, pp. 242–249. doi: 10.1109/FPL.2009.5272301.
[145] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *2013 23rd International Conference on Field programmable Logic and Applications*, Sep. 2013, pp. 1–8. doi: 10.1109/FPL.2013.6645544.
[146] M. Kumm and J. Kappauf, "Advanced Compressor Tree Synthesis for FPGAs," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1078–1091, Aug. 2018, doi: 10.1109/TC.2018.2795611.
[147] R. Amirtharajah, "Chapter 24 - Distributed Arithmetic," in *Reconfigurable Computing*, S. Hauck and A. Dehon, Eds. Burlington: Morgan Kaufmann, 2008, pp. 503–512. doi: 10.1016/B978-012370522-8.50032-7.
[148] J. M. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientos, and M. A. Aguirre, "FTUNSHADES2: A novel platform for early evaluation of robustness against SEE," in *2011*

12th European Conference on Radiation and Its Effects on Components and Systems, Sep. 2011, pp. 169–174. doi: 10.1109/RADECS.2011.6131392.
[149] A. Miczo, *Digital Logic Testing and Simulation*, Wiley. 2003.
[150] "Digital Systems Testing and Testable Design | IEEE eBooks | IEEE Xplore." https://ieeexplore.ieee.org/book/5266057 (accessed Jul. 26, 2022).
[151] M. Renovell, J. M. Portal, P. Faure, J. Figueras, and Y. Zorian, "Analyzing the test generation problem for an application-oriented test of FPGAs," in *Proceedings IEEE European Test Workshop*, May 2000, pp. 75–80. doi: 10.1109/ETW.2000.873782.
[152] M. Rebaudengo, M. S. Reorda, and M. Violante, "A new functional fault model for FPGA application-oriented testing," in *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings.*, Nov. 2002, pp. 372–380. doi: 10.1109/DFTVS.2002.1173534.
[153] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone, "UA2TPG: An untestability analyzer and test pattern generator for SEUs in the configuration memory of SRAM-based FPGAs," *Integration*, vol. 55, pp. 85–97, Sep. 2016, doi: 10.1016/j.vlsi.2016.03.004.
[154] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone, "Accurate simulation of SEUs in the configuration memory of SRAM-based FPGAs," in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct. 2012, pp. 115–120. doi: 10.1109/DFT.2012.6378210.
[155] C. Cazzaniga and C. D. Frost, "Progress of the Scientific Commissioning of a fast neutron beamline for Chip Irradiation," *J. Phys.: Conf. Ser.*, vol. 1021, p. 012037, May 2018, doi: 10.1088/1742-6596/1021/1/012037.
[156] I. da Costa Lopes, "Méthodologie d'évaluation d'effets des radiations dans les systèmes numériques : du niveau composant au niveau système," University of Montpellier, Montpellier, 2020.
[157] M. Rimén, J. Ohlsson, J. Karlsson, E. Jenn, and J. Arlat, "Design Guidelines of a VHDL-based Simulation Tool for the Validation of Fault Tolerance," p. 23.
[158] L. A. B. Naviner, J.-F. Naviner, G. G. dos Santos, E. C. Marques, and N. M. Paiva, "FIFA: A fault-injection–fault-analysis-based tool for reliability assessment at RTL level," *Microelectronics Reliability*, vol. 51, no. 9–11, pp. 1459–1463, Sep. 2011, doi: 10.1016/j.microrel.2011.06.017.
[159] R. Travessini, P. R. C. Villa, F. L. Vargas, and E. A. Bezerra, "Processor core profiling for SEU effect analysis," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, Mar. 2018, pp. 1–6. doi: 10.1109/LATW.2018.8347235.
[160] M. A. Aguirre, J. N. Tombs, A. Torralba, and L. G. Franquelo, "UNSHADES-1: An Advanced Tool for In-System Run-Time Hardware Debugging," in *Field Programmable Logic and Application*, vol. 2778, P. Y. K. Cheung and G. A. Constantinides, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1170–1173. doi: 10.1007/978-3-540-45234-8_146.
[161] "Vivado Design Suite User Guide: Programming and Debugging (UG908)," p. 353, 2017.
[162] A. Ullah, P. Reviriego, and J. A. Maestro, "An Efficient Methodology for On-Chip SEU Injection in Flip-Flops for Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 65, no. 4, Apr. 2018, doi: 10.1109/TNS.2018.2812719.
[163] L. A. Aranda, A. Sánchez-Macián, and J. A. Maestro, "ACME: A Tool to Improve Configuration Memory Fault Injection in SRAM-Based FPGAs," *IEEE Access*, vol. 7, pp. 128153–128161, 2019, doi: 10.1109/ACCESS.2019.2939858.
[164] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Automation & Test in Europe Conference & Exhibition 2009 Design*, Apr. 2009, pp. 502–506. doi: 10.1109/DATE.2009.5090716.
[165] "Project X-Ray Documentation," p. 111.
[166] C. Lavin and A. Kaviani, "RapidWright: Enabling Custom Crafted Implementations for FPGAs," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Boulder, CO, USA, Apr. 2018, pp. 133–140. doi: 10.1109/FCCM.2018.00030.
[167] Z. Wang, Z. Yao, H. Guo, and M. Lu, "A software solution to estimate the SEU-induced soft error rate for systems implemented on SRAM-based FPGAs," *J. Semicond.*, vol. 32, no. 5, p. 055008, May 2011, doi: 10.1088/1674-4926/32/5/055008.

[168] J.-B. Note and É. Rannaud, “From the bitstream to the netlist,” in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays - FPGA ’08*, Monterey, California, USA, 2008, p. 264. doi: 10.1145/1344671.1344729.
[169] F. Benz, A. Seffrin, and S. A. Huss, “Bil: A tool-chain for bitstream reverse-engineering,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Oslo, Norway, Aug. 2012, pp. 735–738. doi: 10.1109/FPL.2012.6339165.
[170] R. Le Roux, G. Van Schoor, and P. Van Vuuren, “Parsing and analysis of a Xilinx FPGA bitstream for generating new hardware by direct bit manipulation in real-time,” *SACJ*, vol. 31, no. 1, Jul. 2019, doi: 10.18489/sacj.v31i1.620.
[171] Wolfgang-Spraul, “Wolfgang-Spraul/fpgatools.” Apr. 27, 2022. Accessed: Aug. 05, 2022. [Online]. Available: https://github.com/Wolfgang-Spraul/fpgatools
[172] M. Jeong, J. Lee, E. Jung, Y. H. Kim, and K. Cho, “Extract LUT Logics from a Downloaded Bitstream Data in FPGA,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5. doi: 10.1109/ISCAS.2018.8350950.
[173] Z. Wang, Z. Yao, H. Guo, and M. Lv, “Bitstream decoding and SEU-induced failure analysis in SRAM-based FPGAs,” *Sci. China Inf. Sci.*, vol. 55, no. 4, pp. 971–982, Apr. 2012, doi: 10.1007/s11432-011-4396-3.
[174] S. A. Guccione, D. Levi, S. Guccione, and D. Levi, “JBits: A Java-Based Interface to FPGA Hardware,” p. 9.
[175] K. Dang Pham, E. Horta, and D. Koch, “BITMAN: A tool and API for FPGA bitstream manipulations,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, Mar. 2017, pp. 894–897. doi: 10.23919/DATE.2017.7927114.
[176] L. Bozzoli and L. Sterpone, “COMET: a Configuration Memory Tool to Analyze, Visualize and Manipulate FPGAs Bitstream,” in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, Apr. 2018, pp. 1–4.
[177] L. Bozzoli, C. De Sio, L. Sterpone, and C. Bernardeschi, “PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing,” in *2018 International Symposium on Rapid System Prototyping (RSP)*, Oct. 2018, pp. 70–75. doi: 10.1109/RSP.2018.8632000.
[178] M. Ceschia *et al.*, “Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2088–2094, Dec. 2003, doi: 10.1109/TNS.2003.821411.
[179] L. Sterpone and M. Violante, “A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2217–2223, Dec. 2005, doi: 10.1109/TNS.2005.860745.
[180] M. Sonza Reorda, L. Sterpone, and M. Violante, “Efficient estimation of SEU effects in SRAM-based FPGAs,” in *11th IEEE International On-Line Testing Symposium*, Jul. 2005, pp. 54–59. doi: 10.1109/IOLTS.2005.26.
[181] M. Desogus, L. Sterpone, and D. M. Codinachs, “Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs,” in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, Jul. 2014, pp. 111–115. doi: 10.1109/IOLTS.2014.6873681.
[182] G. Asadi and M. B. Tahoori, “An Analytical Approach for Soft Error Rate Estimation of SRAM-Based FPGAs,” p. 8.
[183] H. Asadi, M. B. Tahoori, B. Mullins, D. Kaeli, and K. Granlund, “Soft Error Susceptibility Analysis of SRAM-Based FPGAs in High-Performance Information Systems,” *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2714–2726, Dec. 2007, doi: 10.1109/TNS.2007.910426.
[184] A. Sari, D. Agiakatsikas, and M. Psarakis, “A soft error vulnerability analysis framework for Xilinx FPGAs,” in *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, Monterey California USA, Feb. 2014, pp. 237–240. doi: 10.1145/2554688.2554767.
[185] G. Asadi and M. B. Tahoori, “An accurate SER estimation method based on propagation probability [soft error rate],” in *Design, Automation and Test in Europe*, Mar. 2005, pp. 306-307 Vol. 1. doi: 10.1109/DATE.2005.49.
[186] G. Asadi and M. B. Tahoori, “Soft error rate estimation and mitigation for SRAM-based FPGAs,” in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-*

programmable gate arrays - FPGA '05, Monterey, California, USA, 2005, p. 149. doi: 10.1145/1046192.1046212.
[187] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder — A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS," in *2008 International Conference on Field Programmable Logic and Applications*, Sep. 2008, pp. 119–124. doi: 10.1109/FPL.2008.4629918.
[188] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: towards an open-source tool flow," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '11*, Monterey, CA, USA, 2011, p. 41. doi: 10.1145/1950413.1950425.
[189] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *2011 21st International Conference on Field Programmable Logic and Applications*, Sep. 2011, pp. 349–355. doi: 10.1109/FPL.2011.69.
[190] T. Haroldsen, B. Nelson, and B. Hutchings, "RapidSmith 2: A Framework for BEL-level CAD Exploration on Xilinx FPGAs," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey California USA, Feb. 2015, pp. 66–69. doi: 10.1145/2684746.2689085.
[191] Xilinx, "Vivado Design Suite Properties Reference Guide (UG912)," Jun. 06, 2018. https://docs.xilinx.com/api/khub/documents/XwCw4k_rU1QQyWHEvp0lKA/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=3.11.45&filename=ug912-vivado-properties.pdf (accessed Aug. 08, 2022).
[192] "RapidWright Overview — RapidWright 2022.1.2-beta documentation." https://www.rapidwright.io/docs/RapidWright_Overview.html (accessed Aug. 08, 2022).
[193] L. Sterpone *et al.*, "Experimental Validation of a Tool for Predicting the Effects of Soft Errors in SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2576–2583, Dec. 2007, doi: 10.1109/TNS.2007.910122.
[194] "xapp465.pdf • Viewer • Documentation Portal." https://docs.xilinx.com/v/u/en-US/xapp465 (accessed Aug. 15, 2022).
[195] "Project U-Ray." F4PGA, Jul. 24, 2022. Accessed: Aug. 26, 2022. [Online]. Available: https://github.com/f4pga/prjuray