Fault-mitigation strategies for reliable FPGA architecture

Chagun Basha Basheer Ahmed

UNIVERSITÉ DE RENNES 1

ANNÉE 2016

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

École doctorale MATISSE

présentée par

Chagun Basha BASHEER AHMED

préparée à l'unité de recherche IETR – UMR6164 – CNRS
Institut d'Électronique et de Télécommunications de Rennes

Fault-mitigation strategies for reliable FPGA architecture

Thèse soutenue à Nantes
le 31 March 2016

devant le jury composé de :

Lionel TORRES
Professeur, Université de Montpellier II / *Éxaminateur*
Alexandre NKETSA
Professeur, LAAS-Université de Toulouse III / *Rapporteur*
Loïc LAGADEC
Professeur, Lab-STICC, ENSTA Bretagne / *Rapporteur*
Arnaud TISSERAND
Senior Researcher, CNRS-INRIA-IRISA / *Éxaminateur*
Sébastien PILLEMENT
Professeur, Polytech'Nantes, UBL / *Directeur de thése*
Stanislaw J.PIESTRAK
Professeur, Université de Lorraine / *Co-directeur de thése*

Acknowledgements

I take this opportunity to acknowledge and thank all those people who supported me during my PhD.

First of all, I would like to express my sincere gratitude to my advisor Prof. Sébastien PILLEMENT for his encouragement both at educational and personal levels, for his patience, motivation, moral support, enthusiasm, and immense knowledge. His suggestions and remarks have always been productive and constructive that are extremely helpful for my research. I would equally like to thank my co-advisor Prof. Stanislaw J. PIESTRAK for his constructive feedback, comments and suggestions at various stages. I am deeply indebted of both my supervisors for guiding me through all the challenges throughout the span of this thesis work.

I also want to thank all the members of the jury who evaluated this work: Lionel TORRES (Professeur, Université de Montpellier II), Alexandre NKETSA (Professeur, LAAS-Université de Toulouse III), Arnaud TISSERAND (Senior Researcher, CNRS-INRIA-IRISA) and Loïc LAGADEC (Professor, Lab-STICC, ENSTA Bretagne).

I would like to express my deep sense of gratitude to the partners of the ARDyT project for all the scientific exchanges and discussions. I am very grateful to the Agence Nationale de la Recherche and the Conseil Régional de Bretagne for the research grant and supporting me to attend and to present papers at conferences which were very important to increase my knowledge and to help the state-of-the-art research. I am equally grateful to the research internship students who contributed to this project: Aurélien Fourny, Amit Karel, Joby Varghese, Meng Zhang, and Shubham Maheshwari.

I would like to take this opportunity to acknowledge the help and support provided by all the administrative staff and research team members at IRISA-ENSSAT Lannion and IETR-Polytech Nantes, for the warm welcome, cultural integration and all the memorable moments. I take this opportunity also to express my sincere thanks to all those who guided, advised and supported me at different times, in technical and non-technical issues. I am thankful to Dr. Daniel Philip, Dr. Pramod Udupa, Dr. Romain Brillu and Dr. Quang Hai Khuat for their consultation and advise on various matters.

Finally, I will always remain in debt to my parents, family and friends for the limitless love, care and support that they have always provided me with. A special thanks to my friend Digvijay Babar for the motivation, encouragement and moral support he provided, which helped me handle many difficult situations during this PhD.

Contents

List of Figures

List of Tables

Résumé

La fiabilité est devenue un facteur essentiel pour la plupart des systèmes de calcul intégrés. Malgré les avantages fournis par les circuits reconfigurables (FPGAs), à savoir une conception à faible coût et un délai rapide de commercialisation, l'importance de la fiabilité risque de limiter leur large utilisation dans les systèmes critiques. Par exemple, dans des missions spatiales non habités, la fiabilité d'un système de calcul a un impact majeur sur le coût de la mission vu qu'il s'avère difficile de remplacer un système s'il devient défectueux. Aujourd'hui, le besoin de systèmes informatiques fiables s'est développé au-delà des applications militaires et spatiales traditionnelles. Cette liste croissante de domaine inclut les systèmes de communication, les systèmes médicaux et de sauvetages (comme les machine cardio-pulmonaire, les machines de ventilation mécaniques, les pompes de perfusion, les machines de radiothérapie, machines de chirurgie robotique, ...), les réacteurs nucléaires et autres systèmes de contrôle de centrales électriques, la signalisation dans le transport, et cette liste n'est pas exhaustive.

Les progrès de la technologie CMOS engendrent de nouvelles contraintes en raison des limites physiques et économiques de ces process. En particulier, les tailles réduites des transistors impliquent une diminution du rendement et de la précision des *System-on- Chip* (SoC) en raison de la présence (variabilité) ou l'apparition (vieillissement) de défauts physiques dans le circuit. Ainsi, les défauts induits par les radiations consistuent une grande menace pour les architectures reconfigurables et surtout lorsqu'elles sont utilisées dans des environnements de rayonnement difficiles. La variabilité de fabrication impliquent aussi des complications telles que la fuite sous le seuil, la dissipation de puissance, la sensibilité accrue du circuit au bruit. Tous ces phénomènes induisent, soit des défaillances transitoires (par exemple, erreurs du logiciel radio-induites par un changement de valeur dans une mémoire), soit permanentes (par exemple, le vieillissement du transistor). Ce besoin de fiabilité apporte une révolution dans la pratique de la conception de circuits et requière la prise en compte de la tolérance aux pannes ou l'inclusion de capacités de détection de défauts. C'est un défi important pour les développeurs, quand ils utilisent des FPGAs du commerce en vue de concevoir des applications critiques. Des stratégies de conception prenant en compte l'atténuation des effets des pannes doivent être mises en œuvre, ainsi qu'un processus de conception adaptÃľ. Cela dépend considérablement de plusieurs paramètres incluant la sensibilité de l'application, l'atmosphère de déploiement et le niveau de fiabilité requis.

En termes de la mise en œuvre de systèmes complexes, les circuits FPGA reconfigurables font désormais partis de l'ordinaire grâce à leur flexibilité, leurs performances et leur nombre élevé de ressources intégrées. Les architectures recon-

figurables présentent un compromis subtil entre complexité et la flexibilité nécessaire. Les champs récents d'applications des FPGA correspondent à des environnements difficiles (rayonnement cosmique, ionisants, bruit électromagnétique) et avec de hautes exigences de tolérance aux fautes. Les FPGAs récents ne sont pas adaptés à ce type d'environnement à l'exception de circuits bien spécifiques ayant été durci mais avec un prix de revient très élevé, ce qui les rend moins intéressant d'un point de vue économique. Par conséquent, de nouvelles alternatives doivent être envisagées.

La plupart des FPGA du commerce (COTS pour Components Of the Shelf) ne parviennent pas à répondre aux exigences des systèmes critiques (sauf quelques dispositifs tels que les Virtex-5QV de Xilinx et le Microsemi RTG4). Ceci est dû à leur grande sensibilité aux événements qui créent des fautes et des incertitudes notamment dans la mémoire de configuration basée sur une technologie SRAM très sensible aux radiations. Beaucoup de développeurs ont compris que les systèmes essentiels à la mission doivent être conçus pour un fonctionnement fiable dans des conditions environnementales extrêmes, mais constatent que la plupart des technologies FPGA ne peuvent pas répondre à ces besoins. Pour supporter ces environnements agressifs, les phénomènes transitoires et les effets d'un seul événement, les systèmes critiques nécessitent des composants fiables.

Dans la majorité des cas, pour atteindre le niveau désiré de fiabilité dans les applications basées sur des circuits reconfigurables, deux stratégies traditionnelles sont suivies, i) faire une architecture entièrement durcie au rayonnement par le processus de fabrication lui-même et ii) par l'application de diverses stratégies de tolérance aux fautes intégrés aux architectures COTS au stade de la conception de l'application. Les deux solutions de durcissement et de conception sur lesquels les solutions sont basées ont leurs propres avantages et inconvénients. Une solution alternative plus intéressante, où les avantages des différentes approches pourraient être utilisés, en termes de surcoût matériel, consommation d'énergie, l'amélioration de la fiabilité et la souplesse de conception, etc. peut être envisagé. Nous proposons donc le développement de nouvelles architectures fiables, où divers mécanismes de fiabilité peuvent être intégrés à différents niveaux de l'architecture, y compris les processus de fabrication, l'architecture matérielle, le plan de configuration, la conception de l'application afin de soutenir le niveau de fiabilitÃl' requis mais sans payer le prix du durcissement technologique du circuit.

Contributions

Ce travail de thèse a pour objectif de remédier à ces inconvénients et à ces défis en proposant des stratégies de gestion de fautes appropriées, et adaptables. La recherche présentée dans cette thèse fait partie intégrante du projet de recherche ANR ARDyT ("Architecture Reconfigurable Dynamiquement Tolérante aux fautes"). Le but de ce travail est de développer des stratégies adhoc de tolÃl'rances aux fautes pour protéger les diverses ressources constitutives d'une architecture FPGA. Le travail dans cette thèse est principalement axé sur l'architecture matérielle et le processus de gestion des fautes. Les contributions de ce travail de recherche sont les suivantes :

- les différents modèles de fautes qui se produisent dans les circuits reconfigurables ont été étudiés. Ainsi différents modèles d'erreurs associés aux différents éléments constitutifs de l'architecture FPGA ont été étudiés montrant la nécessité d'adapter les stratégies aux différents scénarii. Les différents systèmes de détection et de correction de fautes de l'état de l'art sont ensuite analysés et leurs avantages et inconvénients sont comparés les uns aux autres, en fonction de différents paramètres important dans le cadre des architectures reconfigurables. Les paramètres étudiés comprennent le surcoût matériel, le coût de mise en œuvre, la complexité de la conception, les contraintes de temps, et l'adaptabilité. Puis, il a été répondu à la question : "comment faire une architecture FPGA fiable en faisant une classification générale significative des différentes approches ?". Les différentes approches ont été classées en trois grandes classes : i) conception de l'architecture durcie au rayonnement par fabrication, ii) réalisation de la fiabilité en mettant en œuvre au niveau architecture des techniques de tolérances aux fautes iii) intégration de mécanisme de fiabilité au niveau applicatif (i.e. sans modifier l'architecture).

- Par la suite, les blocs de base de construction de l'architecture matérielle ARDyT sont définis. Des ressources spÃľcifiques pour la détection des fautes, le diagnostic et la tolérance sont ajoutés aux composants logiques comme des fonctionnalités intégrées. Les ressources logiques sont définies afin de permettre des compromis entre densité d'intÃľgration et fiabilité fonctionnelle. L'architecture est adaptable est permet de supporter les diffétes stratégies de tolérance aux fautes proposées. Nous avons définie et spécifié une couche d'abstraction fonctionnelle de la tolérance aux fautes (FTAL - Fault-Tolerant Abstraction Layer) et l'algorithme de gestion et de tolÃľrance aux fautes pour les différents modes adaptés sont intégrés dans le R3M (Run-time Reconfigurable Resource Manager), gestionnaire de la fiabilité centralisé de notre architecture. La granularité de la détection des fautes et leur notification est déterminée sur la base de régions reconfigurables partielles groupées (GPRR). La première étape d'amélioration de la fiabilité concerne la détection de la dite faute. La remontée d'informations pertinentes sur l'événement survenu (changement de valeur / faute / erreur) contribue à une meilleure formulation de la stratÃľgie de prise en compte de cet événement. Pour faciliter la lecture de l'état de la faute (de sa notification) et de fournir le lien vers le R3M, un registre d'état de faute (FSR - Fault Status Register) est défini dans chaque GPRR. Un protocole d'interrogation est alors employé afin de remonter vers le R3M le type de faute identifié et d'adapter la technique de gestion de la faute appropriée. Ici, le terme "lecture de faute" se réfère à l'identification de la faute qui a eu lieu dans les modules matériel "fault-aware". La nature de la "lecture de faute" dans le cadre du projet ARDyT dépend de différents facteurs tels que la granularité de l'identification de la faute, le mode de représentation de la faute, le temps et le coût matériel requis par la technique de gestion de la faute. Selon les spécifications architecturales proposées et la définition de la FTAL, le protocole d'interrogation a l'accès aux FSR de chaque GPRR.

- Dans les FPGA à base de mémoire SRAM, Les cellules mémoires sauvegar-

dent principalement des bits de configuration, occupant plus de 98% de toute la mémoire de la plupart des composants du commerce. Ces bits de configuration SRAM sont sujets à des radiations induites de type SEU (provoquant des changements de valeur d'un seul bit SBU ou de plusieurs bits MBU). En raison du fait que la fonctionnalité d'un FPGA à base de SRAM est déterminée par le contenu de ces cellules de mémoire de configuration, chaque modification des bits de configuration par un SEU modifie la fonctionnalité du dispositif programmé dans le FPGA. Dans ce travail, un nouveau schéma de protection, construit sur un codage de Hamming 3D (3DH - 3-Dimensional Hamming) permet de gérer les erreurs binaires multiples (SBU et MBU) causées par le rayonnement dans la mémoire de configuration. L'idée est de réaliser la protection de la mémoire de configuration (bitstream) par le système 3D-Hamming proposé s'exécutant en arrière-plan pendant l'exécution de l'application, comme cela se fait dans les dispositifs Xilinx Virtex. Dans le système de protection de proposé, la détection et la correction des erreurs se produisent comme un processus cyclique continu, en raison du fait que les codes de Hamming ne détectent pas seulement l'erreur, mais localise sa position dans le mot binaire et permettent une correction si une seule erreur est présente dans le mot en cours de traitement. La principale différence et avantage du système proposé par rapport à la méthode classique utilisée dans les architectures Xilinx, est que la technique ne nécessite pas l'utilisation d'une copie externe des bits de configuration. Le schéma 3DH proposé est implémenté grâce aux ressources d'accès internes au bitstream de configuration (ICAP - Internal Configuration Access Port) et entièrement gérés par le gestionnaire centralisé de la fiabilité (R3M) dans la FTAL. Il supporte une reconfiguration rapide des zones de mémoire concernées par les erreurs multiples, parce que la correction peut être faite en utilisant le bus interne seul, contrairement aux méthodes les plus connues qui reposent sur la sauvegarde de la configuration en externe et qui nécessitent le transfert des données via les lignes d'E/S.

- La tolérance aux fautes présentée ci-dessus gère seulement les changements de valeurs qui affectent le flux binaire de configuration, malheureusement, il y a d'autres sources de défauts qui peuvent affecter directement les ressources matérielles du FPGA. Leurs natures et leurs conséquences sont différentes de celles qui se produisent dans bitstream et leurs effets ne peuvent pas être corrigés en effectuant une re-configuration. De fait, pour protéger ces ressources logiques une nouvelle architecture, le FA-CLB (Fault-Aware Configurable Logic Block), est proposée qui est capable de détecter en ligne (en fonctionnement normal) des défauts a un niveau fin de granularité (à savoir le niveau LUT pour Look-Up Table). L'approche proposée repose sur l'identification des défauts des circuits combinatoires et séquentiels séparément, ce qui aide à trouver un défaut et à adapter sa gestion en fonction de sa nature. En ce qui concerne les multiplexeurs des blocs logiques combinatoires, ils ne peuvent pas être directement affectés par un SEU radio-induit qui pourrait provoquer des SBU et / ou MBU, car ils ne contiennent pas d'éléments de stockage. Les modèles de fautes affectant les circuits combinatoires et les circuits séquentiels sont différents. Par conséquent, différents schémas de détection de fautes sont

proposés pour les éléments combinatoires et séquentiels. L'architecture globale du FA-CLB résultante proposée est différente de la structure CLB classique utilisée dans les FPGA commerciaux. Elle intègre notamment les ressources de détection des fautes mais aussi aide la stratégie de gestion de ces fautes grâce notamment á sa structure permettant la redondance temporelle d'un calcul.

Conclusions :

1. La constante mise à l'échelle de la technologie et les changements de caractéristiques des environnements de fonctionnement des applications augmentent les erreurs dans la mémoire de configuration des architectures reconfigurables. Les niveaux élevés de rayonnement implique l'apparition de fautes multiples de type MBU (multi-bit upset). Ces niveaux de rayonnement déclenchant des changements de valeurs ne se limite plus seulement à l'espace et aux hautes altitudes. Même au niveau du sol, le rayonnement naturel et artificiel des particules est maintenant observé. De même les architectures sont elles même plus sensible à ces rayonnements du fait de la diminution des tailles de transistors. Par conséquent, les systèmes électroniques de haute fiabilité prennent de plus en plus d'importance dans les applications au niveau du sol également.

2. Comme le marché est en pleine expansion au-delà des applications spatiales, acceptant des coût élevé, il y a un besoin de développer une architecture fiable et flexible, ciblant une fiabilité à faible coût et permettant l'intégration d'applications critiques. Les architectures durcies par un processus de fabrication spécifique représentent un coût élevé qui ne peut pas être abordable dans de nombreuses applications grand public (comme l'automobile par exemple). Les approches de la tolérance aux fautes utilisées dans les FPGA COTS à base de redondance impliquent une complexité de conception supplémentaire et réduisent la flexibilité de conception. Par conséquent, le développement de nouveaux modèles d'architectures intégrant des mécanismes de support de la fiabilité adaptées à différents niveaux (architecture, configuration, application et logiciel) est nécessaire pour avoir des implémentations d'applications moins complexes, flexibles et à des coûts acceptables pour le grand public.

3. Les effets du rayonnement cosmique ont différents modèles de défaut sur les différents éléments des circuits reconfigurables qui entraînent des conséquences diverses selon leur nature. De ce fait, les techniques de détection et de prise en compte des défauts doivent être adaptées.

4. Le développement de modules logiques adaptables, où les circuits logiques sont personnalisés pour soutenir les stratégies de détection et de correction des fautes est nécessaire et a été réalisé dans ce travail pour le développement d'une architecture reconfigurable. Cela permet de réduire considérablement la complexité de la phase de développement de l'application et le temps d'accès au marché, vu que le concepteur n'a pas à se concentrer sur "l'allocation et l'utilisation des ressources" pour les aspects de fiabilité (comme dans le cas de la conception d'applications fiables sur des architectures non fiables).

5. Un compromis peut être obtenue entre "le niveau de fiabilité" et "le surcoût matériel, la latence, l'efficacité énergétique". Il peut être représenté collectivement par le facteur d'évaluation de la fiabilité et de l'efficacité. Il est basé sur divers aspects, y compris la sensibilité des différentes tâches ; le suivi et la non surveillance dynamique de zones du circuit ; et la granularité de la détection et de la correction des fautes. Dans le cadre du projet ARDyT un mécanisme de compromis entre densité d'intégration et fiabilité est introduit permettant tout un panel de niveau de fiabilité vs éfficacité (au sens large).

Abstract in French

Les circuits reconfigurables (Field Programmable Gate Arrays - FPGAs) sont largement utilisés dans divers domaines d'application en raison de leur flexibilité, de leur haute densité d'intégration, de leur niveau de performance et du faible coût de développement associé. Toutefois, leur grande sensibilité aux défauts dus aux rayonnements électromagnétiques tels que les "Single Event Effets" (SEE), est un défi qui doit être abordé pendant la conception du système. Ces SEE sont une préoccupation majeure dans la sécurité et pour les systèmes critiques tels que les systèmes automobile et avionique. En général, la plupart des FPGA d'aujourd'hui ne sont pas conçus pour fonctionner dans ces environnements difficiles, sauf pour les circuits spécifiques qui ont été durcis par construction au niveau du processus de fabrication. Ces circuits ont un surcoût très élevé et des performances moindres, ce qui les rend moins intéressants que leurs équivalents non protégés.

Le projet ARDyT vise à développer une architecture FPGA fiable à faible coût avec une suite d'outils de conception, offrant un environnement complet pour la conception d'un système tolérant aux fautes. Ce travail de thèse présente une contribution à l'architecture du FPGA ARDyT, qui intègre des stratégies de prises en charge des fautes adaptées aux différents éléments de l'architecture. L'un des principaux objectifs du projet ARDyT est de gérer les changements de valeurs multiples (multi bit upsets (MBUs)) dans le flux binaire de configuration du FPGA. Ces stratégies de tolérance aux fautes pour protéger les ressources logiques et le flux binaire de configuration sont discutées en détail. Une architecture spécifique du bloc logique élémentaire configurable est proposée afin de simplifier la stratégie de prise en compte des fautes dans les ressources logiques. Un nouveau système de correction d'erreur intégrée (3-Dimensional Hamming - 3DH) est proposé pour gérer les MBU dans le flux binaire de configuration. L'ensemble de cette stratégie de gestion de fautes est implémenté dans l'architecture au travers d'un manager de la fiabilité centralisée nommée R3M (Run-time Reconfigurable Resource Manager).

Abstract

Reconfigurable Field Programmable Gate Arrays (FPGAs) are extensively employed in various application domains due to their flexibility, high–density functionality, high performance and low–cost development compared to ASICs (Application Specific Integrated Circuits). However, the challenge that must be tackled during system design is their high susceptibility to the radiation induced faults such as Single Event Effects (SEEs). These radiation induced faults are a major concern in safety and mission critical systems such as automotive and avionics systems. In general, most of today's commercial off-the shelf (COTS) FPGAs are not designed to work under these harsh environments, except for specific circuits that have been radiation–hardened at the fabrication process level, but at a very high cost overhead, which makes them less interesting from an economic and performance point of view.

Design based techniques and architectural customization are the other ways to achieve desired level of reliability in a system design. This thesis work is a part of a multi-partner project–ARDyT, which aims to develop a low–cost reliable FPGA architecture with supporting EDA tool-suite that offers a complete environment for a fault tolerant system design. The ARDyT FPGA architecture plans to incorporate appropriate fault mitigation strategies at different level of the architecture. The work carried-out in this thesis focus mainly on developing reliability strategies at hardware and configuration level. A fault-aware customized configurable logic block architecture is proposed to support fault mitigation process in configurable logic resources. One of the main objectives of ARDyT project is to handle multi-bit upsets (MBUs) in the configuration bitstream. A new built–in 3–Dimensional Hamming (3DH) error correcting scheme is proposed to handle MBUs in the configuration bitstream. Proposed schemes are made adaptable in such a way that they are integrated in the ARDyT architectural framework to support the global (centralized) reliability management strategy.

Chapter 1

Introduction

Prompted by the development of new types of sophisticated field-programmable devices (FPDs), the process of designing digital hardware has changed dramatically over the past few years. The most compelling advantages of FPDs are instant manufacturing turnaround, low start-up costs, low financial risk and ease of design changes. Field Programmable Device is a general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end-user to realize different designs. The FPD market has grown over the past decade to the point, where there is now a wide assortment of devices to choose from. To choose a product, designers face the daunting task of researching the best uses of various chips and learning the intricacies of vendor-specific CAD software. Adding to the difficulty is the complexity of the more sophisticated devices. User-programmable switches are the key to user customization of FPDs.

A programmable logic device (PLDs) refers to any type of integrated circuit used to build user-configurable digital circuits. A PLD has an undefined function at the time of manufacturing. Since these logic devices can be programmed in the field, they are also called field programmable logic devices (FPLDs). PLDs come in two forms, complex programmable logic devices (CPLDs) and field programmable gate arrays (FPGAs), both having their advantages and disadvantages with respect to the specific application or design they are to be used in. The primary differences between CPLDs and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers, which results in less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnections, which makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for.

In practice, the distinction between FPGAs and CPLDs is often one of size, as FPGAs are usually much larger in terms of resources than CPLDs. Typically, only FPGAs contain more complex embedded functions such as adders, multipliers, and memory. Being the only type of FPD that supports very high logic capacity, FPGAs have been responsible for a major shift in the way digital circuits are designed [1]. FPGAs provide many advantages such as:

- **Field programmability:** FPGAs in contrast to traditional computer chips

are completely configurable. Updates and feature enhancement can be carried out in the field, even after deployment.

- **Extremely short time to market:** Through the use of FPGAs, the development of hardware prototypes is significantly accelerated since a large part of the hardware development process is shifted to developing the core design, which can be done in parallel. Additionally, because of the early availability of hardware prototypes, time-consuming activities like the start-up and debugging of hardware are brought forward concurrently to the overall development.

- **Fast and efficient systems:** Available standard components address a broad user group and, consequently, often constitute a compromise between performance and compatibility. With FPGAs, systems can be developed that are exactly customized for the designated task, which henceforth can be highly efficient.

- **Performance gain for software applications:** Complex tasks are often handled through software implementations in combination with high-performance processors. In this case, FPGAs provide a competitive alternative, which by means of parallelization and customization for the specific task even establishes an additional performance gain.

- **Massively parallel data processing:** The amount of data in contemporary systems is ever increasing, which leads to the problem that systems working sequential are no longer able to process the data on time. Especially by means of parallelization, FPGAs provide a solution to this problem which, in addition, scales excellently.

- **Real time applications:** FPGAs are perfectly suitable for applications in time-critical systems. In contrast to software based solutions with real time operating systems, FPGAs provide real deterministic behavior. By means of the featured flexibility even complex computations can be executed in extremely short periods.

In modern circuits, design flexibility is mandatory, as it enables fast evaluation of design changes during the lifetime of applications, enhancement of functionality, and so on. In counterpart, most of current applications are requiring more and more high computation capability to offer advanced services. In terms of complex systems implementation, programmable FPGA circuits are now part of the mainstream implementation solutions: thanks to their flexibility, good performances and high number of integrated resources. Besides, FPGAs are entering new fields of applications such as aeronautics, military, automotive or confined control, thanks to their ability to be remotely updated [2].

1.1 Field Programmable Gate Arrays (FPGAs)

FPGAs contain programmable logic blocks that can be wired in different configurations. These blocks create a physical array of logic gates that can be used to perform

different operations. Because the gates are customizable, FPGAs can be optimized for any computing task.

Based on implementation technology, FPGA architecture could be broadly classified into three types: i) *Antifuse-based*, ii) *Flash-based* and iii) *Static Random Access Memory (SRAM)-based*.

Antifuse-based FPGAs

They can be programmed only once. The antifuse is a device that doesn't conduct current initially, but can be "burned" to conduct current (the antifuse behavior is thus opposite to that of the fuse, hence the name). The antifuse-based FPGAs can't be then reprogrammed anymore, since there is no way to return a burned antifuse into the initial state. Antifuse-based device families include Axcelerator®produced by Microsemi [6].

Flash-based FPGAs

First of all, these type of FPGAs shouldn't be confused with SRAM-based FPGAs, as the internal flash memory of the latter ones uses flash only during startup to load data to the SRAM configuration cells. On the contrary, a true flash-based FPGA uses flash as a primary resource for configuration storage and doesn't require SRAM. The main advantages of this technology are low power consumption and better tolerant to radiation effects. Flash-based FPGA families such as Igloo [7] and ProASIC3 [8] are manufactured by Microsemi.

SRAM-based FPGAs

SRAM-based FPGAs store logic cells configuration data in the static memory (organized as an array of latches). Since SRAM is volatile and can't keep data without power source, such FPGAs must be programmed (configured) upon start. There are two basic modes of programming:

- Master mode, when an FPGA reads configuration data from an external source, such as an external flash memory chip.
- Slave mode, when an FPGA is configured by an external master device, such as a processor. This can be usually done via a dedicated configuration interface or via a boundary-scan (JTAG) interface.

SRAM-based configuration memory is more commonly used in today's advanced reconfigurable architectures. In this case, each configuration bit is presented as a field-effect transistor (FET) that is controlled by an SRAM cell. Xilinx and Altera are two major SRAM-based FPGA manufacturers. The *Virtex* family FPGAs from Xilinx (V5 [13], V6 [14], and V7 [15]) and *Stratix* family FPGAs from Altera [16] are examples of SRAM-based FPGAs. One disadvantage of this technology is that an SRAM-based FPGA always has to be re-programmed at the power up of the circuit board. There are a variety of techniques by which this programming may be achieved; a very common alternative is to use an external serial flash memory chip,

and for the FPGA to instigate the configuration process by reading the contents of this flash memory and using it to program its SRAM based configuration cells. An important advantage of SRAM-based FPGAs is that they can be manufactured using a standard CMOS process, hence they become available right at the forefront of each new technology, thereby offering the highest performance and lowest power consumption. By comparison, flash-based and antifuse-based FPGAs require extra processing steps during the manufacturing process, which means they typically lag the state-of-the-art by one or two technology feature sizes.

Antifuse-based FPGAs have to be configured using a special programming device before being attached to the circuit board. Flash-based FPGAs can be configured off-board—using a special programmer, or on-board—using additional circuitry on the circuit board. And, as previously noted, SRAM-based FPGAs always have to be re-programmed when the board is powered up.

Some devices [10–12] have both a flash transistor and an SRAM cell associated with each configuration bit [2]. Hence, on power-up, the contents of all of the flash transistors are copied (in a massively parallel fashion) into their corresponding SRAM cells. In addition to providing the advantages of non-volatility and instant-on, this also means that the flash portion of the FPGA can subsequently be re-programmed "on-the-fly" whilst the rest of the FPGA is performing its allotted tasks.

1.1.1 Building Blocks and Architecture Details

Figure 2.1 shows a sample FPGA architecture. In general, FPGAs comprise an array of uncommitted circuit elements, called *programmable logic blocks, programmable routing (interconnects)* and *programmable I/O blocks*. A programmable logic block provides the basic computation and storage elements used in digital systems. The basic logic element contains some form of programmable combinational logic, a flip-flop and some fast arithmetic carry logic. The programmable routing provides connections among logic blocks and I/O blocks to complete a user defined design. It consists of multiplexers, pass transistors and tri-state buffers, which form the desired connections. Generally, pass transistors and multiplexers are used within a logic cluster to connect logic elements together, while all three types of connection elements are used for more global routing structures.

Configurable Logic Blocks (CLBs)

A configurable logic block (CLB) is the basic building block of an FPGA, capable of realizing arbitrary logic functions. It contains a small memory for creating arbitrary combinatorial logic functions, also known as look-up table (LUT). It also contains flip-flops as clocked storage elements as well as multiplexers used to route the logic within the block and to and from external resources. The multiplexers also allow polarity selection as well as reset and clear input selection. Figure 2.5 shows a simplified sample structure of a CLB, which comprises only a 4-input LUT, a multiplexer and a register. The multiplexer requires an associated configuration cell to specify input to be selected. The register requires associated cells which allow to specify whether it acts as an edge-triggered flip-flop or a level-sensitive latch,

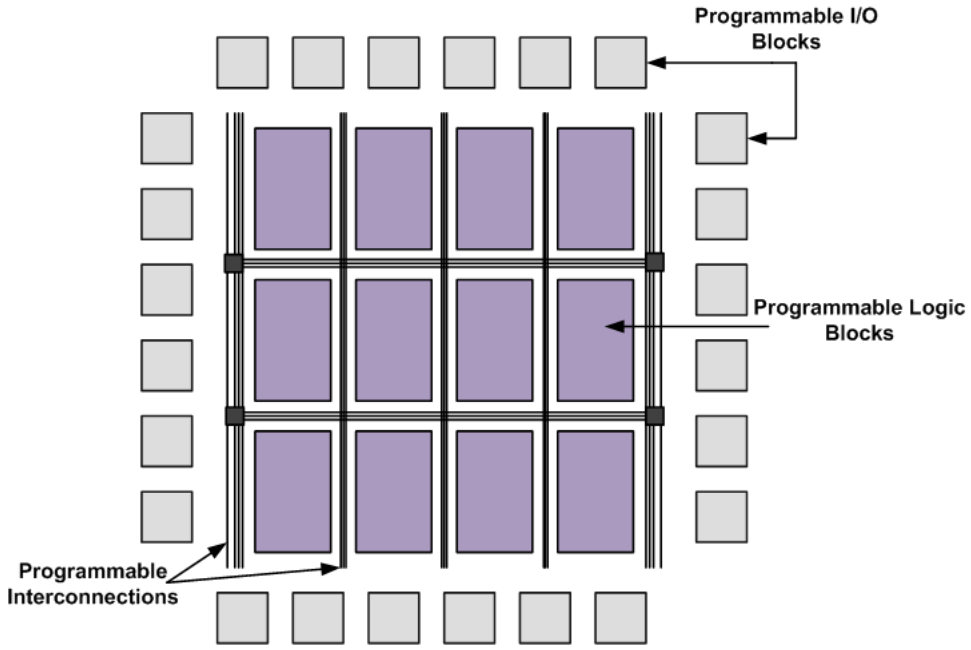


Figure 1.1 – Generic structure of an FPGA [1].

whether it is positive- or negative-edge triggered (in the case of the flip-flop option), whether an enable signal is active-low or active-high (if the register is instructed to act as a latch), and whether it is initialized with a logic 0 or a logic 1. The 4-input LUT is itself based on 16 configuration cells. In reconfigurable architectures these memory cells are SRAM cells.

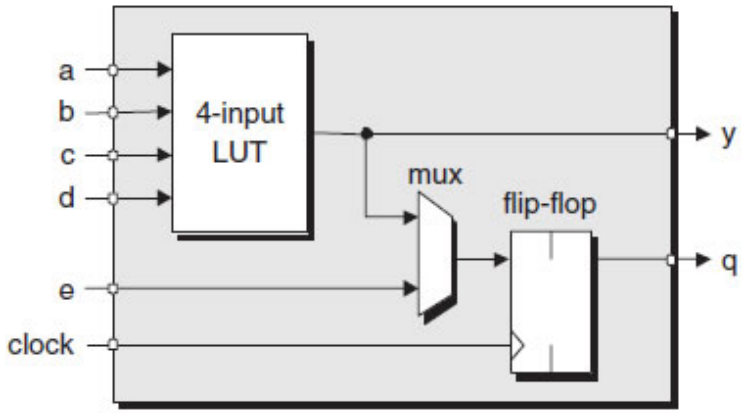


Figure 1.2 – Basic structure of a sample configurable logic element (CLB) [1].

In Xilinx Virtex 7 family FPGA devices, a CLB unit contains a pair of slices. These two slices do not have direct connections to each other, and each slice is organized as a column. Every slice contains: four logic-function generators (or LUTs), eight storage elements, wide-function multiplexers and a carry chain logic. Figure 2.6 shows a slice architecture of Xilinx 7 Series FPGA.

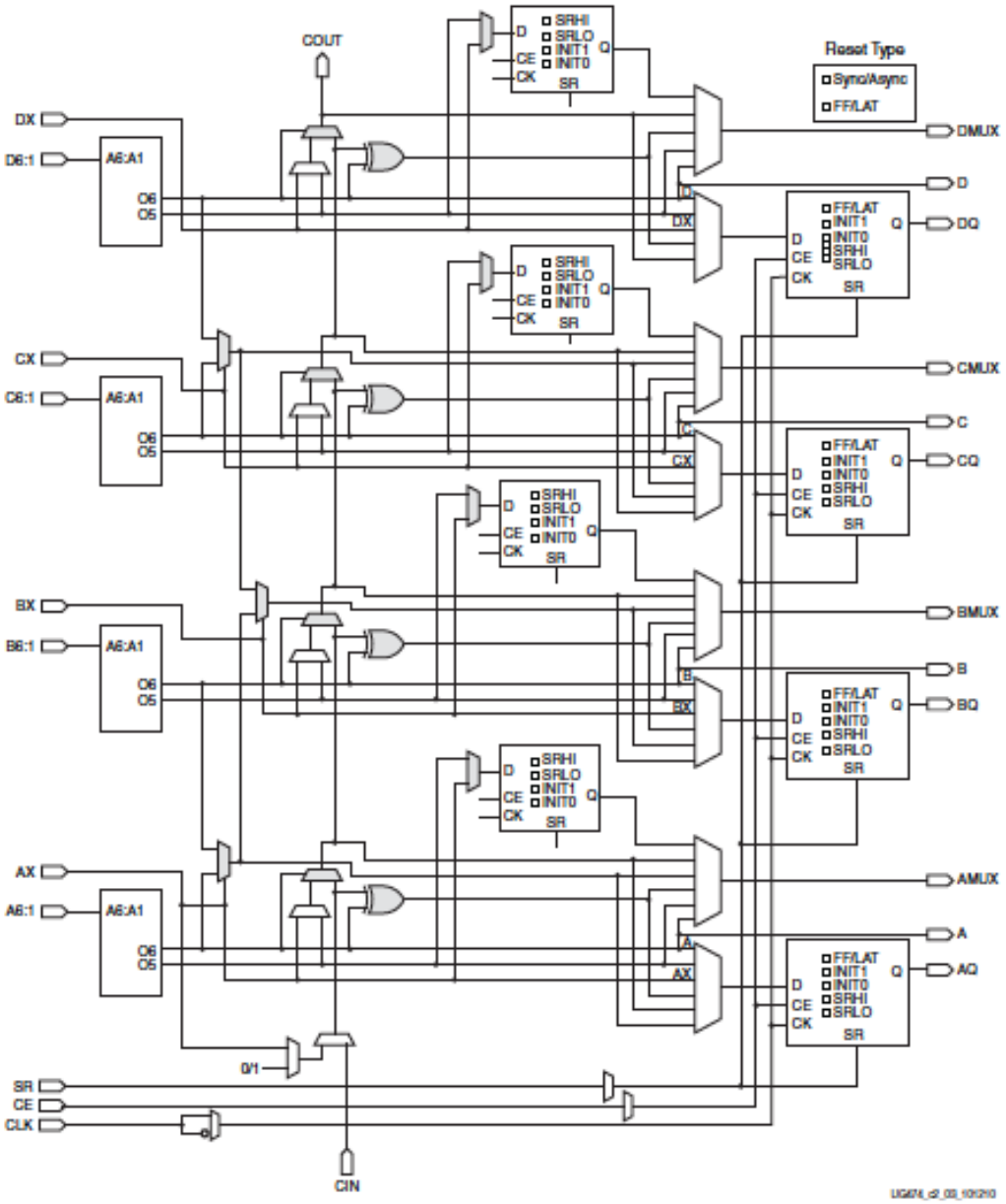


Figure 1.3 – Slice architecture of Xilinx 7 Series FPGA [119].

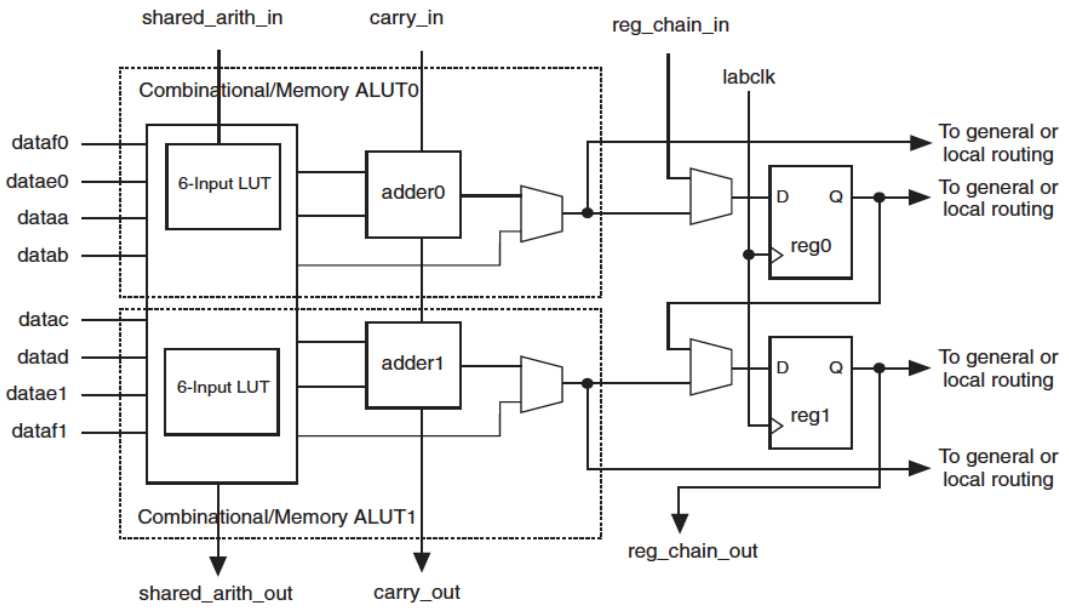


Figure 1.4 – High-level block diagram of the Stratix III ALM [9].

Similarly, Altera FPGAs have their *logic array block (LAB)* as their programmable logic block with a small number of adaptive logic modules (ALMs) inside. Figure 1.4 shows a high-level block diagram of Stratix III ALM FPGA device. For example, in Stratix III family of devices, each LAB consists of ten ALMs, carry chains, shared arithmetic carry chains, LAB control signals, local interconnect, and register chain connection lines. The local interconnect transfers signals between ALMs within the same LAB. The direct link interconnect allows a LAB to drive into the local interconnect of its left and right neighbors. Register chain connections transfer the output of the ALM register to the adjacent ALM register in an LAB. Each ALM contains a variety of LUT-based resources that can be shared by two combinational adaptive LUTs (ALUTs) and two registers. With up to eight inputs to the two combinational ALUTs, one ALM can implement various combinations of two functions. This adaptability allows an ALM to be completely backward compatible with 4-input LUT architectures. One ALM can also implement any function of up to six inputs and certain 7-input functions. In addition to the adaptive LUT-based resources, each ALM contains two programmable registers, two dedicated full adders, a carry chain, a shared arithmetic chain, and a register chain. Through these dedicated resources, an ALM can efficiently implement various arithmetic functions and shift registers. Each ALM drives all types of interconnects: local, row, column, carry chain, shared arithmetic chain, register chain, and direct link interconnects. Apart from these two (Xilinx and Altera) LUT-based architectures, there are also available some multiplexer-based architecture (Microsemi FPGAs). However, compared to multiplexer-based architectures, LUT-based logic block structures have the advantage of implementing any of 2^n n-input logic functions.

These commercially available FPGA's logic block architectures do not have any built-in fault-tolerance capabilities to support higher reliability. One axis of this thesis work investigates the possibilities to design a customized logic block to support

adapted fault mitigation scheme.

Programmable Routing

Programmable logic elements must be interconnected to implement more complex digital functions. An SRAM-based FPGA uses SRAM to hold the information used to program the interconnect. As a result, the interconnect can be reconfigured, just as the logic elements can. A programmable connection between two wires is made by a CMOS transistor (a pass transistor). The pass transistor's gate is controlled by a static memory program bit (cf. Fig. 1.5). When the pass transistor's gate is high, the transistor conducts and connects the two wires; otherwise, when the gate is low, the transistor is off and the two wires are not connected.

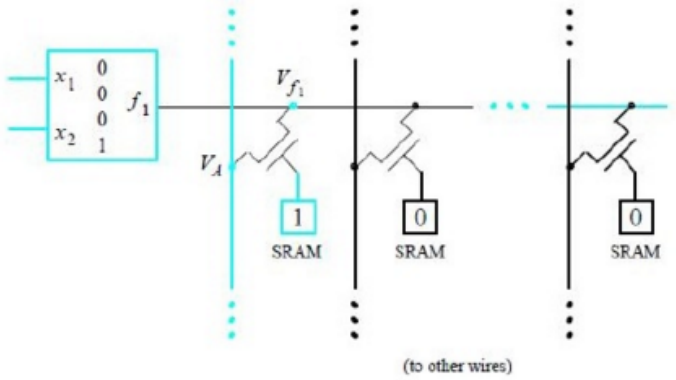


Figure 1.5 – Interconnect points controlled by SRAM cells [98].

A CMOS transistor has a good off-state, although off-states worsen with shrinking chip geometries. In this simple circuit, i.e. of Fig. 1.5, the transistor also conducts bidirectionally, as it doesn't matter which wire has the signal driver. However, the pass transistor is relatively slow, particularly on a signal path that includes several interconnection points in a row. FPGA wiring with programmable interconnect is slower than typical wiring in a custom chip for two reasons: the pass transistor and wire lengths. The pass transistor is not a perfect on-switch, so a programmable interconnection point is somewhat slower than a pair of wires permanently connected by a via. In addition, FPGA wires are generally longer than would be necessary for a custom chip. In a custom layout, a wire can be made just as long as necessary. In contrast, FPGA wires must be designed to connect a variety of logic elements and other FPGA resources. A net made of programmable interconnect may be longer, introducing extra capacitance and resistance that slows down the signals on the net.

An FPGA requires a large number of programmable wires to connect CLBs. FPGAs use wires of varying lengths in order to minimize the delay through wires. Wiring is often organized into different categories depending on its structure and intended use:

- Short wires connect only local logic elements, so they don't take up much space and introduce less delay.

- Global wires are specially designed for long-distance communication. As with high-speed highways with widely spaced exits, they have fewer connection points than local connections, which reduces their impedance. Global wires may also include built-in electrical repeaters to reduce the effects of delay. Also, wire lengths differ thus creating a hierarchy in the global routing structure.
- Special wires may be dedicated to distribute clocks or other register control signals.

Programmable Input/Output (I/O) Blocks

Input/Output (I/O) cells provide interface between internal FPGA circuits and external environment. An I/O cell can be configured as an input, output, or bidirectional port. D flip-flops are normally included in I/O cells to provide registered inputs and outputs. A generic programmable I/O cell is shown in Fig. 2.7.

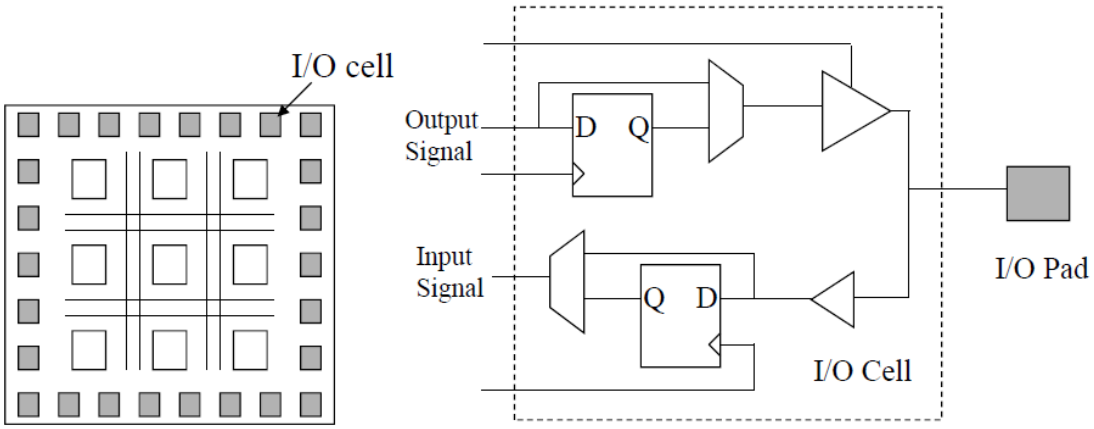


Figure 1.6 – Programmable input/output cell [1].

There is a programmable delay element on the input path, used to eliminate variations in hold times from pin to pin. Propagation delays within the FPGA cause the I/O block control signals to arrive at different times, causing that the hold times of various pins vary. The programmable delay element is matched to the internal clock propagation delay and, when enabled, eliminates skew-induced hold time variations. The output path has a weak keeper circuit that can be selected by programming. The circuit monitors the output value and weakly drives it to the desired high or low value. The weak keeper is useful for pins that are connected to multiple drivers; it keeps the signal at its last valid state after all the drivers have disconnected.

Dedicated Functional Resources

Apart of array of CLBs and routing resources, most of today's FPGAs have also some additional functional resources, such as: arithmetic & logic circuits (ALCs),

dedicated multiplexers, block random access memory (BRAM), carry logic and embedded processors. Sometimes the LUTs are used as distributed RAMs or as storage elements or as shift registers.

- Embedded Block RAM is available in most FPGAs, which allows for on-chip memory in the design. Xilinx FPGAs provide up to 10 Mbits of on-chip memory in 36 Kbit blocks that can support true dual-port operations.
- In addition to general-purpose interconnect resources, FPGAs have fast dedicated lines in between neighboring logic cells. The most common type of fast dedicated lines are carry chains, which allow to realize arithmetic functions (like counters and adders) efficiently (low logic usage and high operating speed).

  The carry chains are cascadable, to form wider add/subtract logic. The propagation delay for an adder increases linearly with the number of bits in the operand, as more carry chains are cascaded. The carry chain can be implemented with a storage element or a flip-flop in the same logic element.

Finally, several FPGA devices offer various implementations of embedded processors. Compared to typical microprocessors, they enjoy many exceptional advantages like: 1) customization, 2) obsolescence mitigation, 3) component and cost reduction, and 4) hardware acceleration. Both Xilinx and Altera offer FPGA devices that embed a dedicated physical processor core into the FPGA silicon, referred to as a 'hard' processor. On the other hand, a 'soft' processor can be configured using FPGA's general-purpose logic. The soft processor is typically described in a Hardware Description Language (HDL) or as a netlist. Unlike the hard processor, a soft processor must be synthesized and fit into the FPGA fabric.

Configuration Bitstream

State-of-the-art commercial FPGAs offer several hundred thousand logic cells along with specialized function units connected via a configurable network. In SRAM-based FPGAs, the functionality is specified by the contents of configuration memory. To configure a circuit, the user needs to load configuration data into the SRAM of the device. This data is generated by the CAD tools and is most often externally loaded onto the device via a configuration port. The FPGA's reconfiguration involves updating the entire or a part of the configuration memory. Reconfiguration time is roughly proportional to the amount of configuration data to be loaded onto an FPGA. Most of memory bit cells in SRAM-based FPGA are configuration bits, occupying more than 98% of memory. The configuration memory is organized into a series of frames. A frame is the smallest unit of the configuration memory that can be written to or read from the device. Because SRAM memory is volatile, the SRAM cells must be loaded with configuration data each time the device powers up. Once the FPGA device is configured, its registers and I/O pins must be initialized, and afterwards the device enters user mode for in-system operation. Some SRAM-based FPGAs with an internal flash memory (like for example Xilinx Spartan-3AN family) do not need to use an external non-volatile memory. Using internal non-volatile memory can be also useful to prevent unauthorized bitstream copying.

1.2 Dependable Embedded Systems

Dependability has become an essential factor for most of computing systems. Although FPGAs provide the advantages of low-cost design and fast time-to-market, the importance of dependability issues limit their widespread use in mission-critical applications [5]. For example, in unmanned space environments, dependability of a computing system has a major impact on the cost of a mission, because unless designed with some fault-tolerance mechanisms, it is difficult or even impossible to replace the system, once it becomes faulty. Today, the need for dependable computing systems has expanded beyond traditional military and aerospace applications. This steadily growing list includes telecommunications infrastructure systems, medical intensive care and life-support systems (such as heart-lung machines, mechanical ventilation machines, infusion pumps, radiation therapy machines, robotic surgery machines), nuclear reactor and other power station control systems, transportation signaling and control systems, amusement ride control systems, and the list goes on.

For clear and unambiguous understanding of dependability issues, we present define some of the key terms and concepts according to [19, 20], whose relationships are graphically visualized in Figure 2.2.

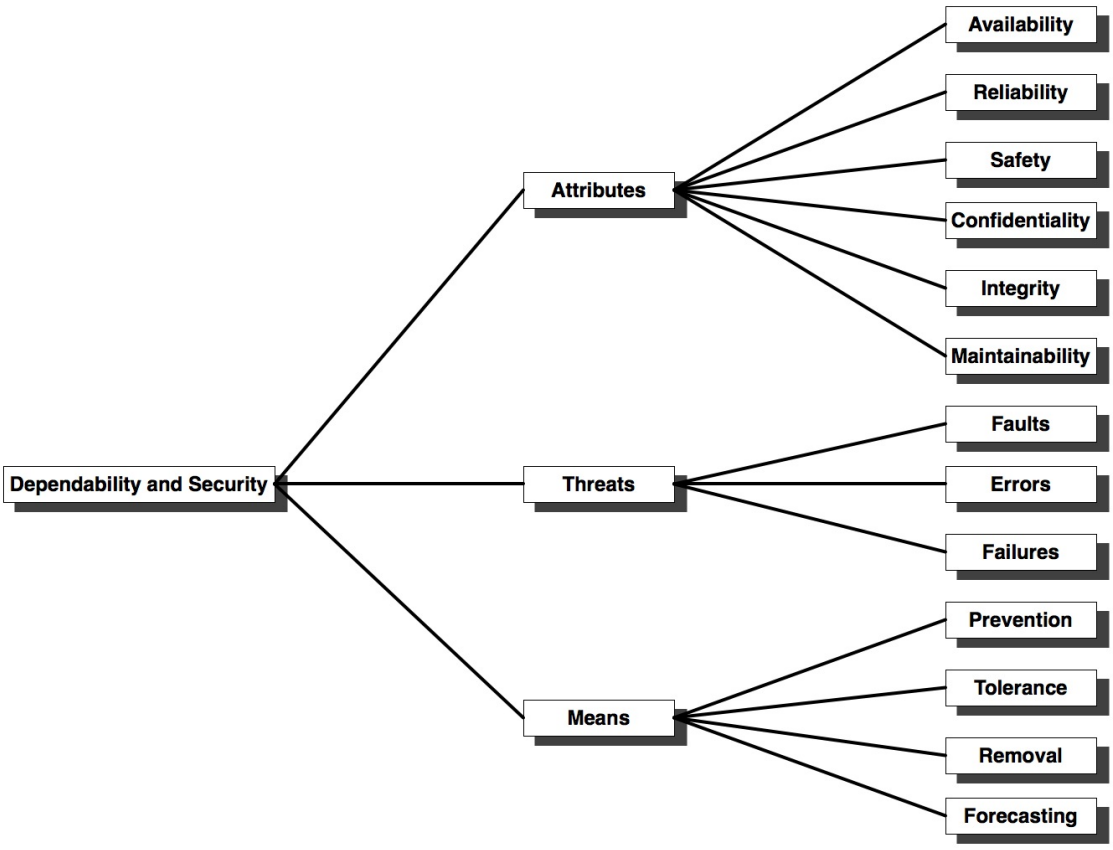


Figure 1.7 – Characterization of dependability and security by their attributes, threats and means [19].

Dependability is defined as the quality of being able to be counted on or relied

upon. In system engineering, dependability is characterized by the following system's attributes: availability, reliability, safety, confidentiality, integrity and maintainability [19]. These attributes can be assessed to determine overall dependability of a system using qualitative or quantitative measures.

- **Availability:** is the probability that a system is functioning correctly at a given time. The term "downtime" is used to refer to periods when a system is unavailable for use. Availability is usually expressed as a percentage of up-time over some specified period of functioning.
- **Reliability:** is the probability of a component or a system functioning correctly over a given period of time under given set of operating conditions; the latter are usually defined in the technical specification of a component or a system.
- **Safety:** is a property of a systems that will not endanger human life or the environment.
- **Confidentiality:** is the property characterizing absence of unauthorized disclosure of information. The term *confidentiality* is also used when addressing security.
- **Integrity:** involves maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle (i.e., absence of improper system's alterations). Data must not be changed in transit and steps must be taken to ensure that data cannot be altered by any unauthorized/unexpected control.
- **Maintainability:** is the probability that a failed system can be repaired or modified within a specified time (it characterizes how easily a system can be repaired or modified). A highly maintainable system may also show a high degree of availability.

Threats are events that can affect a system and negatively affect dependability. The following three main threats are distinguished.

- **Fault:** A fault is a defect in a system. The presence of a fault in a system may or may not lead to a failure. In the latter case, despite a system may contain a fault, its input and state conditions may never cause this fault to be activated, so that an error occurs; so, that particular fault do not causes a system failure.
- **Error:** An error is a discrepancy between the intended behaviour of a system and its actual behaviour inside the system boundary. Errors occur at run-time when some part of the system enters an unexpected state due to the activation of a fault. Since errors are generated from invalid states, they are hard to observe without special mechanisms, such as error detectors or debuggers.
- **Failure:** A failure is an instance in time when a system displays behaviour that is contrary to its specification. An error may not necessarily cause a

failure because, for instance, an exception may be signalled within a system, but this may be caught and handled using fault-tolerance techniques, so the overall operation of the system will still conform to the specification.

As a general rule: a fault, when activated, can lead to an error (which is an invalid state) and the invalid state generated by an error may lead to another error or a failure (which is an observable deviation from the specified behaviour at the system boundary).

There are the following means to attain dependability of a system.

- **Fault forecasting:** is the process of estimating the presence, occurrence, and the consequences of faults. It predicts likely faults, so that they can be removed or their effects can be circumvented.
- **Fault prevention (avoidance):** is the process of preventing the fault occurrence. It increases reliability of a system by conservative design and use of highly reliable components.
- **Fault removal:** is a process of minimizing the presence of faults in a system. Once a system has been deployed, a mechanism is needed to record failures and remove them via a maintenance cycle.
- **Fault tolerance:** relies on providing the service complying with the specification in spite of faults having occurred or occurring. It deals with putting mechanisms in place that will allow a system to still deliver the required service in the presence of faults, although that service may be at a degraded level.

Safety and Mission Critical Systems

- **Mission-Critical:** A mission-critical design refers to those portions of a system that are absolutely necessary. The concept originates from NASA, where mission-critical elements were considered those items that had to work or a billion dollar space mission would blow up. Mission-critical systems must be able to handle peak loads, scale on demand and always maintain sufficient functionality to complete the mission.
- **Safety-Critical:** A safety-critical or life-critical system is one whose failure or malfunction may result in death or serious injury to people, loss of or severe damage to equipment or damage to the environment. The main object of safety-critical design is to prevent a system from responding to a fault with wrong conclusions or wrong outputs. If a fault is severe enough to cause a system failure, then the system must fail "gracefully", without generating bad data or inappropriate outputs. For many safety-critical systems, such as medical infusion pumps and cancer irradiation systems, the safe state upon detection of a failure is to immediately stop and turn the system off. A safety-critical system is one that has been designed to lose less than one life per billion hours of operation.

Most of the commercial-off-the-shelf FPGAs fail to address the key mission and safety critical application requirements (except a few devices such as Xilinx's Virtex-5QV and Microsemi's RTG4). It is due to their (SRAM-based FPGAs) high susceptibility to the events that create faults and uncertainties. Many developers understand that mission-critical systems must be designed for reliable operation even in extremely harsh environmental conditions, but find that most FPGA technologies are stressed to meet these needs. In addition to traditional mission and safety critical applications like aerospace, nuclear and chemical processing, there are many other application fields that are joining the list, such as automotive, home automation, military and civil infrastructure.

However, most of current applications are requiring more and more computation capabilities to offer advanced services. In terms of complex systems implementation, reconfigurable FPGA circuits are now part of the mainstream thanks to their flexibility, performances and high quantities of integrated resources. Reconfigurable architectures exhibit a subtle (and potentially domain-dependent) trade-off between extra area and required flexibility. Extra area can be estimated by physical synthesis tools (used to design a reconfigurable device), whereas flexibility is scored using applicable synthesis tools (preforming the resource allocation in order to map the tagged portion of the application to the reconfigurable architecture). Recent fields of applications that the FPGAs seem to address, correspond to harsh environments (cosmic radiation, ionizing, electromagnetic noise) and with high fault-tolerance requirements. Current FPGAs are not adapted to these environments, except for specific circuits that have been hardened but at a very high cost overhead, which makes them less interesting from an economic point of view. As a consequence, new alternatives should be considered.

1.2.1 Threats, Uncertainties and Challenges

Advances in CMOS technologies are hampered because of physical and economic limits. In particular, shrinking transistor sizes imply a reduction in yield and reliability of System-on-Chip (SoC) due to the presence (variability) or appearance ("aging") of physical defects in the circuit. Also, radiation-induced faults are a great threat to reconfigurable architectures, not only when they are used in radiation-prone harsh environments but also in terrestrial applications. Specifically, these issues include manufacturing variability, sub-threshold leakage, power dissipation, increased circuit noise sensitivity and reliability concerns, due to transient (e.g., radiation-induced soft errors) and permanent (e.g., transistor aging) failures. These changes bring a revolution in design practices and impose designing of circuits with fault detection or even fault-tolerant capabilities. Hence, application designers face great challenges while designing systems for mission critical applications using COTS FPGAs. Additional fault mitigation design strategies have to be implemented and integrated into application design process. This task greatly depends on various parameters including sensitivity of the design, deployment atmosphere and required level of reliability.

1.2.1.1 Aging

With CMOS technology aggressively scaling towards the 16-nm feature size, modern FPGA devices face tremendous aging-induced reliability challenges. Major aging mechanisms of CMOS technology include bias temperature instability (BTI), hot carrier injection (HCI), electro-migration (EM), and time-dependent dielectric breakdown (TDDB) [28]. All of these mechanisms are responsible for the gradual oxide wear-out or interconnects failures that cause circuit performance degradation and transistor failures. Furthermore, all of these mechanisms can be worsened by high switching rate of a circuit, excess supply voltage or high operational temperature. Eventually, with continuous usage, circuit components gradually undergo structural degradation, resulting in hard faults. In standard circuits, these faults cannot be rectified and make a chip unreliable and out of use. Ultimately, such aging mechanisms will shorten the lifetime of the devices.

- *Negative Bias Temperature Instability (NBTI):* due to the applied electric field across the gate oxide, dangling bonds are developed at the interface of the channel and the oxide layer. This affects a transistor by increasing the threshold voltage thus making switching difficult. The NBTI is enhanced by high temperature and high supply voltage.

- *Hot Carrier Injection (HCI):* when carriers with high energy collide with the gate oxide layer and remain trapped there, the oxide layer is damaged, resulting in alteration of the transistor characteristics. High switching rate of a circuit as well as excess supply voltage enhance this effect.

- *Electro-migration (EM):* it is an aging effect taking place in interconnect wire(s), contact(s) and via(s) in an integrated circuit. The effect causes material transport by gradual movement of the ions in a conductor due to the momentum transfer between conducting electrons and the diffusing metal atoms. Integrated circuits are very prone to this effect.

- *Temperature-Dependent Dielectric Breakdown (TDDB):* due to the voltage applied across the gate oxide, conduction starts through it using the trapped charges, resulting in gradual break-down of the oxide layer. A high operating voltage as well as higher temperature accelerate TDDB.

1.2.1.2 Variability

As transistor densities continue to grow, the minimum feature sizes of semiconductor devices are approaching scales for which it is difficult and expensive to achieve uniformity in manufacturing. This results in variability in the critical dimensions of features such as transistor gate length and oxide thickness, which then manifest themselves in the spread of parameters such as propagation delay and leakage current.

Process variation causes the physical and electrical parameters of transistors in fabricated ICs to have different values from the intended nominal values. Such parameters include threshold voltage (V_{th}), effective gate length (L_{eff}) and width

(W_{eff}), oxide thickness (T_{ox}), etc. This variation in parameters stems from different factors including fluctuations in dopant concentration and the inability to precisely print the geometric features on silicon. There are two types of process variations: (i) Die-to-die (D2D) variations affect all transistors in the same die by the same amount; and (ii) Within-die (WID) variations cause transistors in the same die to have different characteristics; these variations are more difficult to address, and they are the main causes of yield reduction.

Like any other high-performance semiconductor device, advanced FPGAs are also affected by process variability. However, the reconfigurable nature of FPGAs gives them a unique advantage. It enables the actual performance variation in each device to be measured and characterized through Built-In Self Test (BIST), typically with ring oscillators [29] or critical path tests [31]. As this thesis work focuses more on radiation-induced faults in reconfigurable architectures, the effects and classifications of the latter faults are discussed in detail in the following section.

1.3 Radiation-induced Faults in Reconfigurable FPGAs

SRAM-based (as opposed to anti-fuse) FPGAs are especially appealing in many application domains due to their in-situ reprogrammability and high performance for signal processing tasks. However, Due to high radiation susceptibility nature of SRAM cells, the use of commercial SRAM-based FPGAs in satellites and spacecrafts presents unique challenges in the presence of the space radiation environment. Sensitivity to radiation effects depends on many factors, including transistor geometry and cell layout. Radiation effects generally include, but may not be limited to: *Total Ionizing Dose (TID) and Single Event Effects (SEE).* The TID represents the cumulative effect of many ionized particles hitting a device throughout the course of its mission life, slowly degrading the device until it ultimately fails. The second case involves high-energy particles that penetrate deep into materials and components, leaving a temporary trail of free charge carriers in their path. If these particles hit sensitive nodes in the circuit, they can produce adverse effects, generically described as SEEs).

1.3.1 Total Ionizing Dose (TID)

Ionization is a process of adding or removing electrons (or other charged particles) from atoms. The creation of electron-holes pair in the semiconductor may cause long term effects in the oxide (charge trapping), and thus alter electrical characteristics of electronic devices.

The cumulative damage of the semiconductor lattice (lattice displacement damage) caused by ionizing radiation over the exposition time (measured in rads) causes slow gradual degradation of the device's performance. Electronic devices suffer long-term radiation effects, mostly due to electrons and protons. The main sources of these particles are solar energetic particle events, which usually occur in association with solar flares. Cumulative long term ionizing damage due to protons and elec-

trons can cause devices suffer threshold shifts, increased device leakage (and power consumption), timing changes, decreased functionality, etc.

The TID radiation has the capability to damage semiconductor materials due to its ionizing capability. The energetic ions can cause damage to semiconductor materials by breaking and/or rearranging atomic bonds. After exposure to sufficiently large total-dose radiation, most insulating materials such as capacitor dielectrics, circuit-board materials, and cabling insulators become less insulating or become more electrically leaky. Certain conductive materials, such as metal-film resistors, can change their characteristics under exposure to total-dose radiation. As semiconductor devices exhibit a number of effects, it is important to choose materials and components for application electronics that have the necessary radiation tolerance for the required mission. It is also necessary to design in margins or allowances for the expected component changes induced by the radiation environment.

1.3.2 Single Event Effects (SEE)

Single event effects (SEEs), caused by a single, energetic particle, are the most common source of faults in SRAM-based FPGAs. There are various events that fall under the general category of SEEs, which can be divided into two broad categories: soft errors and hard errors. Soft errors are those events that have no damaging effects and are cleared by normal device operation. Hard errors are events that generally result in lasting damage to the circuitry. Highly energetic ions such as cosmic rays can easily penetrate the structure of the device, pass through internal components, and exit the structure in a straight line. This single particle impact is often referred as an SEE. Shielding against SEEs is simply not practical. Because heavy particles are omnidirectional, they impinge on an integrated circuit at random times and locations, with random angles of incidence. SEEs are of far greater concern to military avionics systems than total ionizing dose (TID) [32].

An energetic ion passes through a semiconductor device in a few picoseconds, leaving behind a "track" or column of ionized material, typically ranging from a few tenths of a micron to a few microns in diameter. The ionized track contains equal numbers of electrons and holes and is therefore electrically neutral. The total amount of charges is proportional to the linear energy transfer of the incoming particle. It is as if a conducting wire were suddenly inserted into the semiconductor device, disturbing the electric fields and normal current paths.

If a cosmic ray passes through the drain region of an NMOS transistor, a short path (short circuit) is momentarily created between the substrate (normally grounded) and the drain terminal (normally connected to a positive power supply voltage). If this happens, a spike of current flows for an instant at the event site. The amount of charge that is "collected" from the ion track before it dissipates or disappears by recombination is significant: every device has a certain critical charge, which, if exceeded, results in a single-event upset (SEU), burnout, or other undesirable phenomenon.

The impact of a high-energy particle is shown in Figure 2.4. When a high-energy particle, such as a neutron, strikes the silicon substrate of an integrated circuit, it collides with atoms in the substrate, liberating a shower of charged particles that

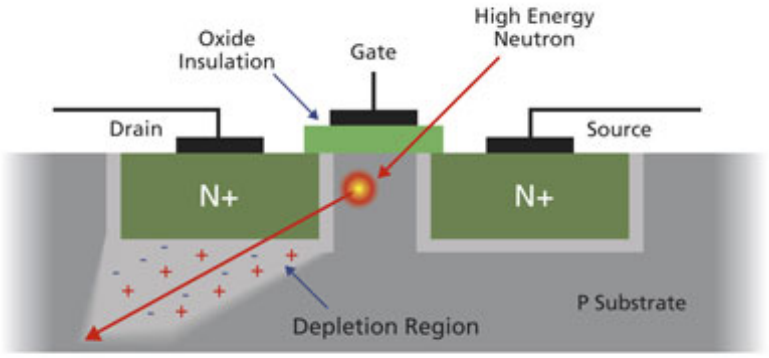


Figure 1.8 – Impact of a high-energy particle: when a high-energy particle (such as a neutron) strikes the silicon substrate of an integrated circuit, it collides with atoms in the substrate [32].

leave an ionization trail. For example, a neutron striking a silicon atom can release energy through elastic and inelastic scattering events or via spallation events that release magnesium and aluminum ions along with alpha particles and protons.

If a device is large, it presents a greater target for cosmic rays. It is therefore more likely to receive a "hit" than a smaller device. Two important parameters must be considered to determine the sensitivity of a device to SEEs: the threshold linear energy transfer, above which upsets or single events are seen, and the saturation cross section. Various types of SEEs, varying in their degree of seriousness, have been identified.

1.3.2.1 Recoverable and Non-recoverable Effects

The errors due to SEEs can be broadly classified as recoverable and non-recoverable, depending on the impact of those effects on circuit elements. Recoverable errors are cleared or their effects are mitigated by adapting appropriate fault mitigation strategies. On the other hand, non-recoverable errors create firm impact in the circuit element, called hard errors. Transients, upsets and functional interrupts are some of the recoverable fault events and latch-ups, gate rupture and burn-outs are non-recoverable fault events. Different SEEs are classified as shown in Figure 2.5 and their definitions are given below.

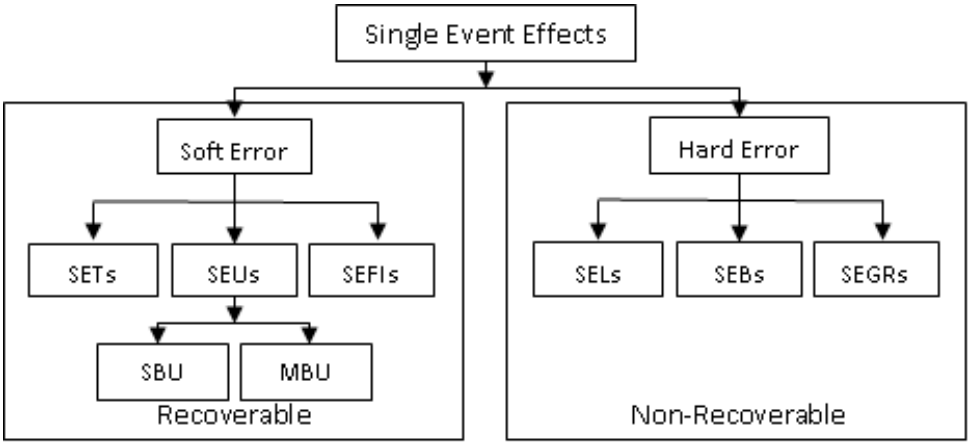


Figure 1.9 – Classes of Single event effects (SEEs) [3].

- *Single Event Transient (SET):* is a temporary spike or signal caused by a heavy ion. In some cases, this spike can excite analog circuits into temporary or permanent oscillation. In digital circuits, the spike may propagate through many logic gates, causing system malfunction. In mixed-signal devices, a transient generated in the analog part of the device can propagate into the digital part, causing logic-level shifts.

- *Single Event Upset (SEU):* usually manifests itself as a "bit-flip" or change of state in a logic circuit. If sufficiently large number of these upsets occur, or if a single critical node is affected, a computing system can freeze up and must be rebooted. SEUs also occur in computer memories, microprocessors, controllers, and almost any digital circuit containing latches or memory elements. They do not cause lasting damage to the device, but may cause lasting problems to a system which cannot recover from such an error. The consequence of an SEU could be either a single bit upset (SBU), where a single bit is corrupted or a multiple bit upset (MBU), where two or more number of bits are corrupted. In very sensitive devices, a single ion can cause a multiple-bit upset (MBU) in several physically adjacent memory cells.

- *Single Event Functional Interrupt (SEFI):* is a disruption to normal device operation that falls beyond a simple corruption of user data. These types of effects alter the functionality of the circuit and typically require reconfiguration/reset or power cycling for recovery. SEUs can become SEFIs when they upset control circuits, such as state machines, placing the device into an undefined state, a test mode, or a halt, which would then need a reset or a power cycle to recover.

- *Single Event Latch-up (SEL):* is triggered when a heavy ion causes current to flow uncontrolled between components of an integrated circuit. When PMOS and NMOS transistors which are integrated into the same area of a silicon substrate are struck by an energetic ion, they can form a parasitic or undesired circuit element (called a thyristor). A thyristor is an interconnected n-p-n and

p-n-p bipolar transistor; the current amplified by the n-p-n transistor supplies the p-n-p transistor, which in turn supplies it back to the n-p-n transistor, thus creating a feedback loop. When an energetic particle traverses the region of a CMOS integrated circuit containing the parasitic transistors, it can generate enough current to trigger the thyristor, if this happens the affected portion of the CMOS integrated circuit will be driven into what is called a "latch-up".

- *Single Event Induced Burnout (SEB):* is a short-circuiting caused when a high-energy ion impacts a transistor source, causing forward biasing. SEBs are typically a threat to power MOSFETs but are also seen in IGBTs, high-voltage diodes, and similar circuits. They may occur in power MOSFETs when the substrate right under the source region gets forward-biased and the drain-source voltage is higher than the breakdown voltage of the parasitic structures. The resulting high current and local overheating then may destroy the device.

- *Single-Event Gate Rupture (SEGR):* is a plasma spike caused by a high-energy ion impact, resulting in rupture of the gate oxide insulation. SEGR leads to damage of the gate oxide and the resulting current path.

1.3.3 Multiple Bit Upsets (MBUs)

As stated, a single particle strike can alter the content of several memory cells (usually physically adjacent), called multi-bit upsets (MBUs), which are significantly more difficult to handle than SBUs [21]. The results presented in [22] indicate that the percentage of MBUs continues to increase with each generation of FPGA devices. Recent experimental results on Xilinx Kintex7 FPGAs indicate that 9.9% of events cause multiple upsets within a frame (7.5% are double upsets); i.e., the estimated Configuration RAM (CRAM) MBU rate is $1.02 \cdot 10^{-11}$, which corresponds to one MBU every 1515 s (about 25 min) [23].

This problem is also common to any other FPGA device family and it will continue to worsen, as devices increase in density and geometries continue to shrink. The necessity of handling MBUs has become a serious problem, because an MBU may affect redundancy-based fault mitigation schemes deployed in FPGA devices (explained in Chapter 2). In particular, the so-called domain crossing events (the special case of MBUs, when an SEE affects configuration bits of different redundant modules) are the major threat to circuits protected using triple modular redundancy (TMR) [21].

MBUs can be induced by direct ionization or nuclear recoil. The energy of the particle is more likely to provoke double bit upsets, whereas MBUs of higher multiplicity are caused by an increase of the particle incident angle. In [25], the authors propose a methodology to quantify the occurrence of proton and heavy ion-induced MBUs. Induced upset patterns are correlated to a physical layout of the programming data to determine the adjacency of upset bits. The physical layout is used to classify adjacent upset bits and their affected resources. A bit is classified as adjacent to another if it lies within one of the eight neighboring memory cells surrounding that bit. Figure 2.3(a) illustrates the adjacency neighborhood used. Any adjacent upsets are classified as MBUs. In Figure 2.3(b) three upset bits are

grouped together in a single MBU. In this way, maximally sized MBUs are found to give an understanding of the size of MBU events. MBUs in 150nm and 65nm SRAM technology are respectively discussed in [27] and [26] and provide insight about different MBU patterns along with their occurrence probabilities.

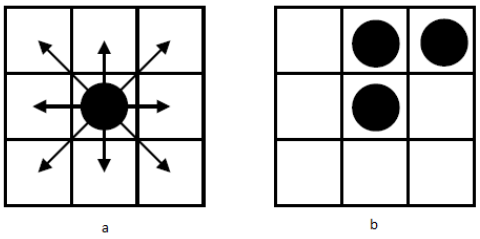


Figure 1.10 – a) Upset adjacency neighborhood; b) MBU of 3 upset bits [25].

There are two quite similar yet a little different multiple upsets: MBUs and multiple cell upset (MCUs). Good understanding of the differences between them will be useful in the context of addressing the upsets of the configuration bitstream. An MBU is a multiple upset in a single logical word, resulting from a single strike. An MCU is a multiple upset of various storage locations, also resulting from a single strike. Thus, an MCU is considered more like a circuit level fault whereas an MBU is treated as information corruption.

1.4 Contributions

Using unreliable COTS FPGAs for critical applications in radiation–prone environments by adapting design–based fault mitigation strategies may have several drawbacks, as discussed in this chapter. At the same time, fabrication–based rad–hard FPGA devices are not affordable in many cases, usually due to excessive manufacturing costs. The ARDyT project aims to address those drawbacks and challenges by developing a new dependable FPGA architecture platform, incorporating various suitable fault mitigation strategies, to have an acceptable trade-off between reliability and cost. ARDyT project is a multi-partner research consortium funded by the French National Research Agency (ANR) under the project identification number - ANR-11-INSE-015. *ARDyT* stands for Reliable and Reconfigurable Dynamic Architecture (in French: Architecture Reconfigurable Dynamiquement Tolérante aux fautes). The research presented in this dissertation is a part ARDyT framework, focused on developing reliability strategies for configurable building blocks, configuration bitstream and linking those strategies to the centralized reliability manager in the fault tolerant abstraction layer (presented in Chapter 3). The contribution of this research work is summarized as follows.

- Different fault models that could occur in reconfigurable FPGAs are investigated. Fault models associated with different building blocks of FPGA architecture are studied. Different state-of-the-art fault detection and correction schemes are analyzed and their advantages and disadvantages are compared

with each other, according to different parameters, including: hardware overhead, implementation cost, design complexity, timing constrains, adaptability, etc. Then, the fundamental question: *"How to make an FPGA architecture reliable?* is answered by presenting a classification of three broad essentially different approaches: i) designing radiation hardened architecture (by fabrication), ii) achieving reliability by implementing design based fault-tolerant techniques and iii) designing a reliable architecture without changing the fabrication process.

- As the unified ARDyT architecture model is already been defined collectively by the project partners, a systemic view of hardware building blocks and their granularity is presented in this thesis work. Fault detection and notification granularity is precised by defining *grouped partial reconfigurable regions (GPRRs)*. Functional specification of the dedicated fault tolerant abstraction layer and their link to hardware architecture to establish an interaction is introduced. Fault mitigation process flow algorithm for configurable logic blocks and configuration bitstream is associated with the centralized reliability management strategy. To facilitate the fault status reading (notification) and to provide link to the centralized reliability manager–R3M, a dedicated register is defined at GPRR level, called *fault status register (FSR)*. An interrogation protocol (with access to FSRs) is outlined with an objective to adapt appropriate fault-reading technique and to notify the fault status to the R3M. Here, the term fault-reading refers to getting knowledge about the fault occurrence in the fault-aware hardware/application modules.

- The reconfiguration process based fault mitigation schemes can remove only the upsets affecting the configuration bitstream. Unfortunately, there are other sources of faults that might directly affect hardware resources of reconfigurable FPGAs. Their nature and consequences differ from those which occur in the configuration bitstream and their effects cannot be corrected by performing configuration write-back. A new fault-aware configurable logic block (CLB) is proposed, which is capable of on-line detection (i.e. during normal functioning) of faults at the fine granular level (i.e. LUT level). The approach proposed relies on identifying combinational and sequential circuit faults separately, which helps in finding a fault and its handling according to its nature. As far as the multiplexers in CLBs are concerned, they cannot be directly affected by radiation-induced SEUs which could cause SBUs and/or MBUs, because they do not contain any storage elements. Nevertheless, as any other combinational circuits, they can be affected by radiation-induced temporary faults called Single Event Transients (SETs). Because fault models affecting combinational circuits and sequential circuits are different, hence different fault detection schemes are proposed to combinational and sequential circuit elements, according to their fault models.

  The proposed fault-aware CLB architecture is different from the conventional CLB structure used in commercial off-the shelf (COTS) FPGAs. The proposed CLB structure is customized in a way which supports ARDyT FPGA architecture. Additionally, the proposed scheme provides flexibility in apply-

ing the reliability mechanism, i.e., the designer can choose between either using the complete resource in CLB for functional implementation or making a fault-aware CLB by using internal resources.

- In SRAM-based FPGAs, memory bit cells are predominantly configuration bits, occupying even more than 98% of all memory in most FPGA devices. These configuration SRAM bits are prone to radiation-induced SEUs (both SBUs and MBUs). Because the functionality of an SRAM-based FPGA is determined by the contents of the configuration memory cells, any alteration of the configuration bits by an SEU could change the functionality of the FPGA device. Therefore, in this dissertation, a new built-in 3-dimensional Hamming (3DH) multiple bit error correcting scheme is proposed to mitigate the effects of SBUs and MBUs caused by radiation in the configuration memory. The idea is to perform the configuration bitstream protection by the proposed 3D-Hamming scheme in the background during run-time, as it is done in Xilinx Virtex FPGA devices. In the proposed configuration bitstream protection scheme, detection and correction of errors happen as a continuous cyclic process, by taking advantage that Hamming error detection and correcting code, not just detects the error but also locates its bit position in the (configuration) word being processed. The primary difference and advantage of the proposed scheme over the conventional methodology used in Xilinx architectures is that this technique does not uses the external golden copy of the configuration bitstream. The proposed 3DH scheme is incorporated with the internal configuration bitstream access resources (internal configuration access port (ICAP), background read-back and write-back, etc.) and completely managed by the centralized reliability manager (R3M) in the FTAL. It provides faster reconfiguration of frames affected by multiple errors/upsets, because correction can be done using internal bus alone, unlike most known methods that rely on the external configuration backup and the I/O lines.

1.5 Outline

This dissertation summarizes the propositions and results obtained in developing fault mitigation strategies for configurable logic resources (CLBs) and configuration bitstream, in the context of ARDyT framework. Detailed descriptions are presented in the corresponding chapters.

- Chapter 2 presents a comparative study and deeper analysis about conventional and state-of-the-art strategies used to achieve required level of reliability in FPGA architectures. It covers techniques of fault detection, diagnosis, containment, and masking as well as error correction. Fault mitigation strategies are discussed under three broad classifications: i) fabrication process based, ii) design based and iii) custom architecture.

- Chapter 3 introduces the ARDyT framework, which aims to design a low-cost reliable reconfigurable FPGA architecture with built-in fault-tolerant strategies. Conceptual overview of ARDyT architecture is presented including func-

tional description of hardware building blocks, features of dedicated reliability management layer and fault mitigation decision making flow related to configurable logic block and configuration bitstream protection.

- Chapter 4 focuses on fault mitigation in configurable logic resources. Fault occurrence in primitive elements of a reconfigurable architecture and various fault models affecting those primitive elements are discussed. A new fault–aware customized configurable logic block (FA-CLB), proposed to support the ARDyT architecture is presented. Fault–model–aware fault detection strategies are proposed for combinational and sequential logic elements.

- Chapter 5 discusses the configuration bitstream protection in reconfigurable FPGAs. It presents the proposed 3-dimensional hamming (3DH) based error correcting scheme to tackle the radiation-induced multiple bit upsets (MBUs) in the configuration bitstream. Possibilities and challenges of implementing the proposed 3DH scheme in COTS as well as in the ARDyT FPGA architecture are discussed.

- Chapter 6 concludes the dissertation by summarizing the results obtained and with some conclusions. It presents some directions of future research which are relevant to the work presented in this dissertation. It includes: i) the definition of a unified model for the design of reliable FPGA architecture with adaptive strategies, ii) identifying appropriate fault mitigation strategy for routing resources, and iii) adapting the proposed 3DH scheme in real 3D architectures.

Chapter 2

Related Work

To endure functioning in harsh environments, including total dose radiation, transient phenomena and SEUs, critical systems require highly reliable components. Reliability in FPGA based critical applications could be attained by implementing the design:

- in radiation-hardened target devices (or)
- in radiation-tolerant architecture platforms (or)
- using design based radiation-tolerant (fault-tolerant) mechanisms.

There is a minor difference between the definition of the terms *radiation-hardened* and *radiation-tolerant*. As its name implies, radiation-hardened means that the device is immune to the effects of radiation (up to a particular limit). A device that: i) is immune to one-mega rad of dose, ii) exhibits immunity from single event destructive effects and iii) upsets at the rate inferior to $1x10^{-10}$ per bit-day in a geostationary orbit (GEO) is almost universally considered radiation-hardened [33]. Radiation-tolerant means that the device can operate as expected, in a particular radiation environment, as long as certain precautions are taken. In other words, changes of the fabrication process to mitigate the effects of radiation is called and *radiation-hardening or rad-hard*, whereas changing the underlying architecture in terms of logic design structure to mitigate the radiation effects is known as *radiation-tolerant or rad-tolerant*. Table 2 in [34] shows the characteristics of the Actel Rad-Hard and Rad-Tolerant and Xilinx Rad-Tolerant 4000XL series. Atmel's ATF280E, Actel's RH1020 and RH1280 are examples of rad-hard FPGAs. Xilinx's XQR4013XL, XQR4036XL and XQVR300 are examples of rad-tolerant FPGAs. According to programming technology, both rad-hard and rad-tolerant FPGAs can either be 'one-time configurable' or 'reconfigurable'.

As reliability factors are not considered during the fabrication and architecture designing process, 'design based radiation tolerance' is completely different from the above two categories. It employs design based fault-tolerant strategies ranging from hardware redundancy, time redundancy to partial reconfiguration based approaches, during the process of application design and implementation in any COTS FPGA device. Indeed, most of the COTS FPGA architectures usually come with 'no' or 'little' built-in reliability support.

2.1 Physical Radiation-Hardening

Physical radiation-hardening techniques rely on various physical means, such as using insulating substrates and adopting rad-hard SRAM cells, etc., to achieve the hardening. In the case of radiation hardening schemes based on *insulating substrates*, hardened integrated circuits are often manufactured on insulating substrates instead of the usual semiconductor wafers. Silicon on insulator (SOI) and silicon on sapphire (SOS) are commonly adopted insulating substrate techniques.

2.1.1 Silicon on Insulator (SOI) [35]

Silicon on insulator (SOI) is an alternative way of chip making process, by replacing the bulk silicon wafers (approximately 0.75 mm thick) with wafers which have three layers: i) a thin surface layer of silicon (from a few hundred Angstrom to several microns thick), where the transistors are formed, ii) an underlying layer of insulating material, and iii) a support or "handle" silicon wafer. Another example of the rad-hard FPGA based on the SOI process technology is Atmel ATF280E [37]. The insulating layer is created by flowing oxygen onto a plain silicon wafer and then heating the wafer to oxidize the silicon, thereby creating a uniform buried layer of silicon dioxide. Transistors are encapsulated in SiO_2 on all sides. The differences between SOI-based devices and conventional silicon-built devices lie in that the silicon junction is above an electrical insulator. Figure 2.1 shows a typical NMOS transistor with bulk CMOS process and with SOI process.

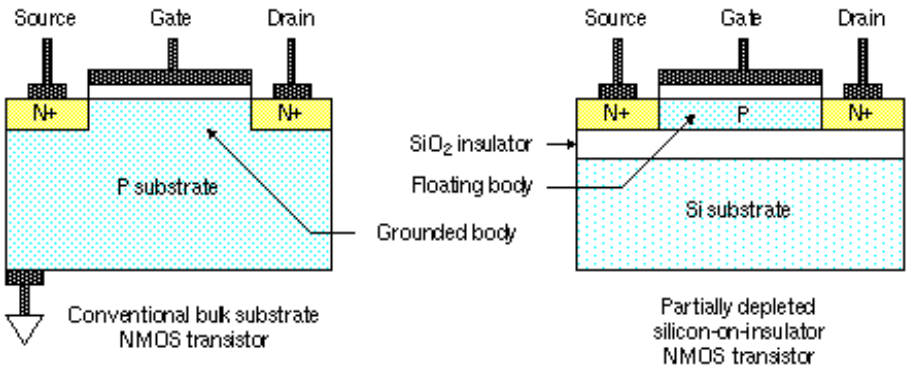


Figure 2.1 – NMOS transistor with bulk CMOS process and with SOI process [38].

The advantages of various aspects of the SOI technology can be summarized as follows.

- **Substrate Noise:** Due to today's increased digital density, substrate noise issue is dominant in the bulk process (especially the digital noise can affect the sensitive circuits). In SOI technology, the buried oxide layer acts as a die-electric barrier and it helps in reducing the substrate noise.

- **Latch-up:** Bulk CMOS relies on junction isolation between devices, while SOI uses dielectric isolation to surround the entire device sides and the bottom. SOI has no wells into the substrate and therefore has no latch-up or leakage paths.

- **Temperature Sensitivity:** SOI CMOS is much less sensitive to temperature than bulk silicon process. In all SOI processes, the leakage to the substrate is obviously suppressed. Furthermore, in SOI process, the threshold voltage (Vt) varies by about twice less with temperature than in the bulk CMOS process.

In summary, the benefits of the SOI technique compared to conventional silicon processing include: (1) lower parasitic capacitance due to isolation from the bulk silicon; and (2) resistance to latch-up due to complete isolation of the n-and p-well structures. From a manufacturing viewpoint, SOI substrates are compatible with most conventional processes.

The results presented in [39, 40] show design and fabrication of a radiation-hardened SRAM-based FPGA VS1000 with a 0.5μm partial-depletion SOI logic process. The radiation test results of [39] indicate that the VS1000 chip has the total dose tolerance of 100 krad(Si), a dose rate survivability of 1.5x10^{11} rad(Si)/s and a neutron fluency immunity of 1x10^{14} n/cm^2. The higher reliability of SOI devices is mainly due to eliminating latch up effects. Referring to hardness, it has been proven that SOI MOSFETs are extremely robust to radiation effects and other physical exposure. This is supported by the fact that in exposed circuitry, most of the electron-hole pairs are generated in thick silicon. The primary barrier to SOI implementation is the drastic increase in substrate cost, which contributes an estimated 10–15% increase to total manufacturing costs [36].

2.1.2 Silicon on Sapphire (SOS) [41]

As discussed above, CMOS technology can be hardened against radiation by fabricating its doped single-crystal silicon substrate over an insulating layer. When the insulator used is sapphire, the fabrication technology is called silicon on sapphire (SOS). It is a hetero-epitaxial process for integrated circuit manufacturing that consists of a thin layer (typically less than 0.6 μm) of silicon grown on a sapphire (Al_2O_3) wafer. A film of single crystalline silicon film is grown over the substrate, then etched into islands and doped to make a bipolar or FET transistor.

As the region between active devices is etched away in this technology, and the devices sit on an insulating layer of sapphire, complete electrical isolation is created between active devices as well as between the devices and the silicon substrate. As the sensitive regions around the channel are insulated from the substrate, the funneling effect that adds to the charge collection in bulk silicon can be neglected in SOS. The only charge collection is in the silicon, as no charge can get collected in the sapphire, thus the overall charge collection volume is smaller than that in bulk CMOS processes. Figure 2.2 illustrates the difference between bulk CMOS and SOS CMOS technology. The guard rings that are normally used to limit leakage current between transistors are unnecessary. Further, there are no deep well diffusion, removing the need for the additional separation and overlap rules associated with this

feature, so neither well nor substrate contacts are necessary. The overall result is higher packing density of the active devices, which can be equivalent to the density gain from shrinking one full technology node. There are no parasitic transistors to cause latch up and the interconnection capacitances are greatly reduced compared to bulk CMOS.

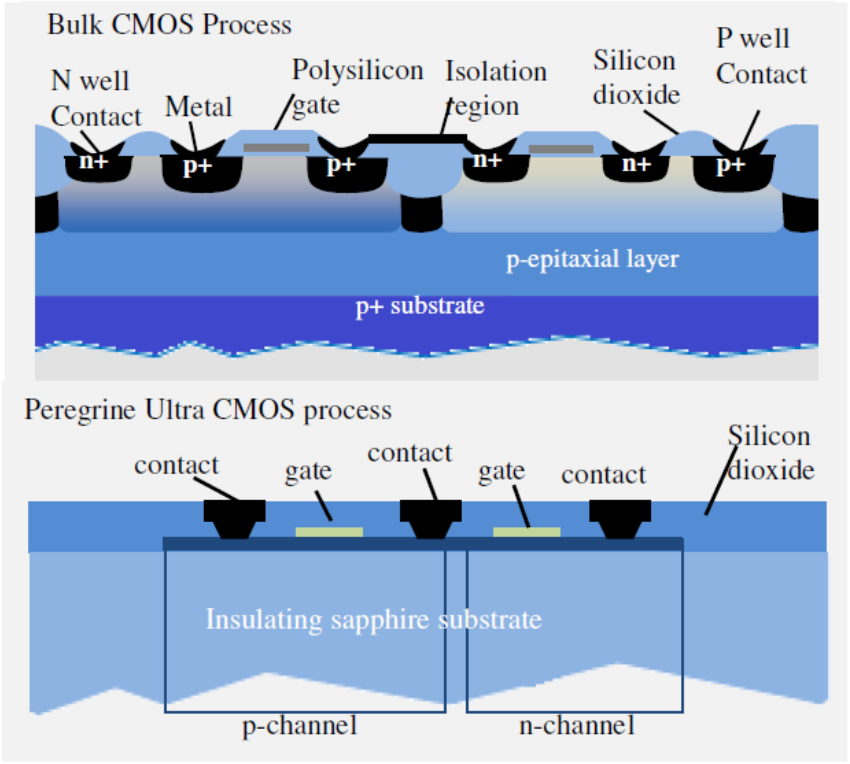


Figure 2.2 – Bulk CMOS and Ultra CMOS (SOS) process [43].

Finally, its inherent resistance to radiation, high input impedance characteristics, high noise immunity and relative insensitivity to voltage variation, which make CMOS-SOS a good solution for high radiation environments. The sapphire protects the device against transients, neutrons, and SEEs. Radiation-induced leakage currents cannot flow between devices because of the insulating substrate. A SOS process technology based radiation hardened SRAM FPGA is presented in [42].

The SOS is mainly used in aerospace and military applications, because of its inherent resistance to radiation. The first advantage of sapphire is that it is an excellent electrical insulator, preventing stray currents caused by radiation from spreading to nearby circuit elements. The second advantage of silicon on sapphire over exotic technologies is that it is manufactured in the same factories that produce common bulk silicon wafers. A further advantage is that, because of its better performance, it can be manufactured in a less advanced factory than similar devices in bulk silicon. One disadvantage of SOS over bulk silicon is that it is by nature a more complex process and sapphire substrates are expensive. SOS has seen little commercial use to date because of difficulties in fabricating the very small transistors used in modern high-density applications. They are physically heavy, causing

problems with manufacturing machines not designed for their mass.

2.1.3 Rad-hard SRAM

Government and private agencies deploying spacecraft are turning increasingly to SRAM-based FPGAs to perform vital processing tasks, because their capabilities have dramatically outpaced the traditional custom application specific integrated circuits (ASIC) or one-time programmable (OTP) FPGA approaches. Rad-hard SRAM-based FPGAs offer space system designers a compelling set of performance, feature, and flexibility advantages for their new projects. The all-programmable nature of SRAM-based FPGAs enables the capabilities of a system to be remotely improved and expanded even after launch, when physical access is no longer possible. The hardware-programmable nature of FPGAs makes them even more flexible than stand-alone processors such as microprocessors, DSPs, and microcontrollers that are only software programmable. The largest FPGAs can provide the functionality of multiple processors and other devices, all on a single chip.

SRAM-based devices enjoy a multiple-generation advantage in process geometry, thus offering greater capacity and performance, while consuming less power per gate. However, they require configuration each time they are powered-on and has to be radiation-tolerant. Currently, leading vendors (namely Xilinx, Atmel, and Altera) offer SRAM-based FPGAs as fully rad-hard components.

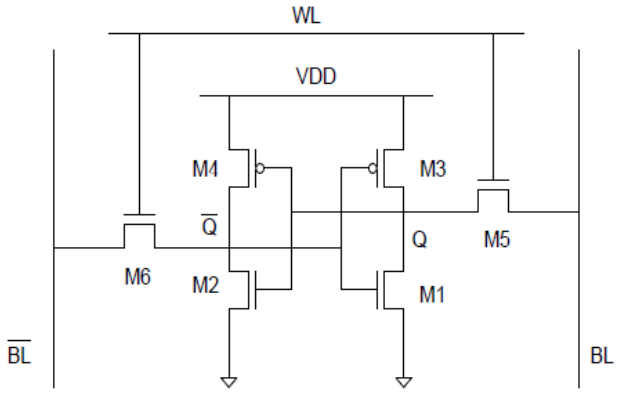


Figure 2.3 – Conventional 6T-SRAM bit-cell [44].

Figure 2.3 shows the conventional 6T-SRAM bit-cell structure, with: M1 and M2—driver, M3 and M4—load, and M5 and M6—access transistors, respectively. The two cross-coupled PMOS pull-up devices retain a value in the cell indefinitely without any external refresh mechanism. Data storage nodes directly connect with bit-line, which makes the voltage shared between access transistors and pull-down transistors, so that the storage node data is susceptible to interference. Such direct read-write mode can make the cell influenced by external noise, which may lead to logical errors. The worst case is observed when a cell is read-accessed. Suppose that the node Q carries a "zero". When the cell is read-accessed, the load transistor M3 is off and the saturated access transistor M5 effectively takes its place. The current

of the driver transistor M1 now has to contend with the saturation current of the access transistor M5, which degrades the level of logic "zero" stored in the node.

To create rad-hard SRAM-based FPGA devices, a variety of techniques are used to implement the underlying FPGA sequential and combinational elements in the device. One such technique involves implementing rad-hard SRAM cells. Rad-hardening solutions include modification of the SRAM bit-cell, which is one the most common circuit-level techniques used to achieve improved robustness to SEUs.

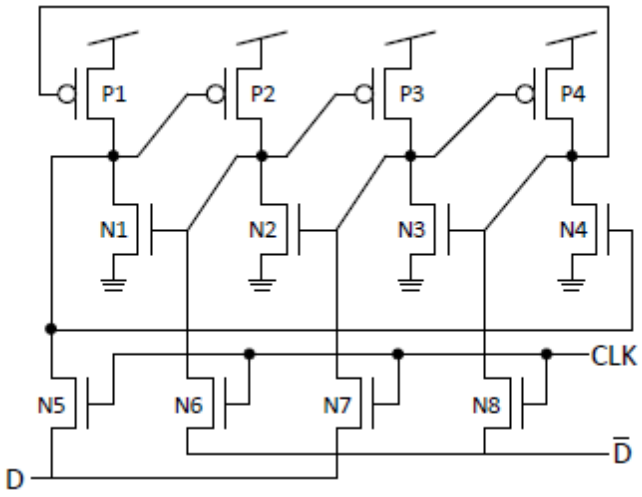


Figure 2.4 – The DICE memory bit-cell [47].

- **DICE [47]:** The dual interlocked storage cell (DICE) is the best known SEU hardened bit-cell. The design concept of the DICE relies on using the dual modular redundancy (DMR) of its internal circuit nodes to achieve immunity to errors affecting a single node. This is achieved with 12 transistors, implementing a dual node feedback control mechanism, as seen in Figure 2.4. It has an area overhead close to l00%, compared to a standard 6-transistor static RAM cell. The storage element utilizes four internal circuit nodes to store one memory bit. When an SEE temporarily upsets one of these four nodes, only one additional node is affected by the upset through positive feedback. In this way, a single node upset (SNU) will not propagate the error to the other nodes, and the unaffected nodes can correct the logic state stored by the cell. However, it still remains sensitive to multi-node upsets (MNU) and also suffers from high power consumption, due to its many transistors and leakage paths.

- **Quatro-10T [48]:** Quad-node 10T or Quatro-10T bit-cell is shown in Fig. 2.5. Two access transistors, N5 and N6, connect the bit lines (BL and BLB) to the storage nodes A and B. If the stored bit is '0', the logic values at nodes A, B, C, and D are '0', '1', '1', and '0', respectively. Each of these nodes is driven by an NMOS and a PMOS transistor, their gates being connected to

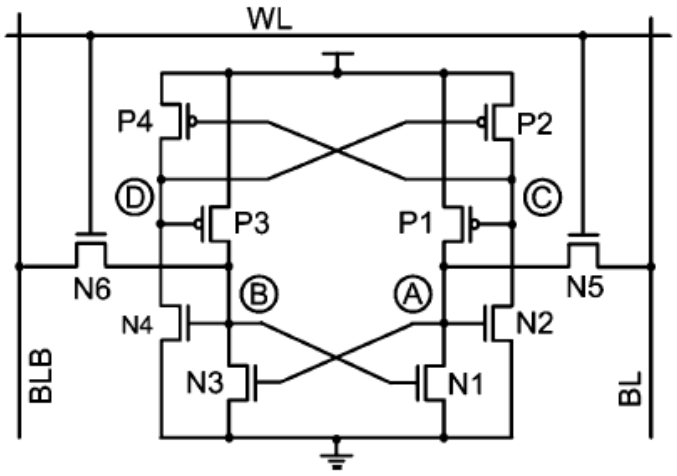


Figure 2.5 – Quad-node 10T or Quatro-10T bit-cell [48].

two different nodes. As a result, if an SET pulls down (up) a node voltage, the node voltage is restored by the 'ON' PMOS (NMOS) transistor connected to the node and driven by an unaffected node. If the SEU turns on a PMOS (NMOS) transistor, the transistor has to work against an unaffected 'ON' NMOS (PMOS) transistor to pull up (down) its drain voltage. Thus, a negative feedback prevents any accidental flipping of the cell. On the other hand, in a standard 6T cell (Figure 2.3), a positive feedback comes into play when an SEU changes a node voltage, thus helping to flip the cell.

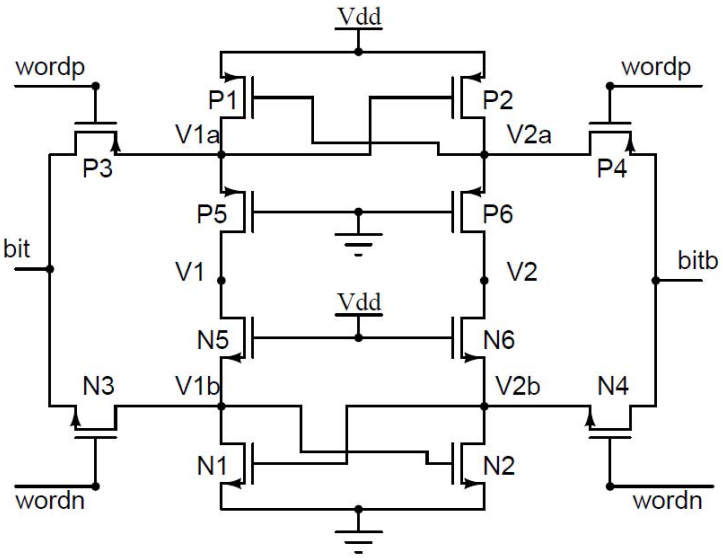


Figure 2.6 – 12T rad-hard SRAM bit-cell [47].

Accordingly, if the 6T bit-cell is sized up to occupy the same area as the Quatro-10T cell, the 6T bit-cell will still remain more vulnerable to soft errors. As opposed to the DICE which relies on four access transistors, the 10T bit-

cell uses only two access transistors for functionality, as can be seen in the circuit schematic of Fig. 2.5. This decreases the area and the leakage current of the bit-cell through the access transistors but, unfortunately, it also results in a much higher write access time and requires careful sizing for functionality. In spite of the multiplication of the storage data nodes, the 10T bit-cell still has a sensitive node that can flip it after a radiation particle hit. While it still enjoys lower SEU rate than the standard 6T SRAM bit-cell, it is mainly a candidate for sea-level SEU hardening, as its error resilience is insufficient for space applications.

- **12T Rad-hard SRAM Cell [47]:** According to the authors in [47], the 12T SRAM bit-cell of Figure 2.6 brings in the benefit of filtering not just mere voltage division. It overcomes the need of a separate well for PMOS and it does not need periodic refresh signals. To improve its SEU tolerance, the design of Figure 2.6 uses two additional transistors compared to the 10T SRAM bit-cell. The transistors P5, P6, N5, and N6 are always turned on, thus acting as a low pass filter to reduce the magnitude of a transient pulse. Indeed, the amplitude of the noise pulse is limited, thus ensuring that one side of the symmetric cell will always have approximately the same potential on the drain and body of one of its devices, which provides desired immunity to SEUs.

  The main disadvantage of the 12T bit-cell is its high static power consumption, caused by four always-on middle transistors P5, N5, P6 and N6, and four weakly gated lateral transistors P1, N1, P2 and N2. Consequently, the 12T bit-cell is unsuitable for use in low power applications.

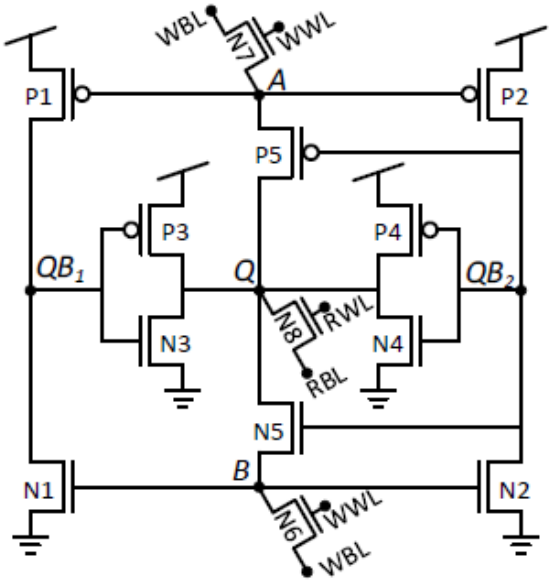


Figure 2.7 – 13T rad-hard SRAM bit-cell [50].

- **13T Rad-hard SRAM Cell [50]:** This bit-cell targeted robust low-voltage operation under SEUs for ultra-low power applications. The 13T bit-cell of

Figure 2.7 achieves radiation hardening by employing a dual-feedback, separated storage mechanism to overcome the increased vulnerability due to supply voltage scaling. The storage mechanism of this circuit comprises five separate nodes: Q, QB_1, QB_2, A, and B, with the acute data value stored at Q. This node is driven by a pair of CMOS inverters made up of transistors N3, P3, N4, and P4 that are respectively driven by the inverted data level, stored at QB_1 and QB_2. The nodes QB_1 and QB_2 are respectively driven to VDD or GND through devices P1, P2, N1, and N2 that are controlled by the weak feedback nodes A and B, that are connected to Q through a pair of complementary devices (P5 and N5) gated by QB_2. By driving the acute data level with a pair of equi-potentially driven, but independent, inverters, a strong, dual-driven feedback mechanism is applied with node separation for SEU protection. This setup effectively protects Q from an upset on QB_1 or QB_2, while achieving a high critical charge at node Q.

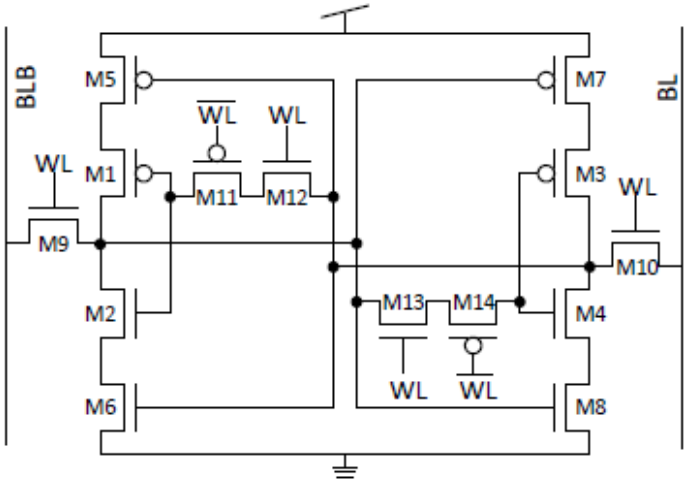


Figure 2.8 – Extremely low power SRAM bit-cell (SHIELD) [49].

- **The SHIELD SRAM Cell [49]:** The SHIELD SRAM bit-cell of Figure 2.8 also targeted low-power critical applications. To mitigate SEU susceptibility, SHIELD uses gated inverters M5-M1-M2-M6 and M7-M3-M4-M8. (A gated inverter is an inverter with an additional input gate.) If both inputs are in the same logical state, the output will be equivalent to that of a regular inverter. When the inputs differ from each other, the output floats with the logical state of the previous output. In the SHIELD bit-cell, a novel radiation tolerant 'cut-off' network M11-M12 and M13-M14 is located between two gates of each gated inverter. The SHIELD SRAM bit-cell has two of these upgraded gated inverters, which are cross-coupled. This results in two sets of separate dual-data nodes, which exhibit high SEU tolerance under scaled supply voltages.

In summary, rad-hard SRAM-based FPGAs reduce or eliminate the compulsion of resource triplication in many critical applications, thus freeing resources of the FPGA devices to implement more functionality with lower payload weight, power

consumption, and system cost, thus allowing to implement more sophisticated systems. Unfortunately, rad-hard SRAM-based FPGAs are much more expensive than COTS FPGAs, hence, the COTS devices are preferred when the cost is the major issue,. Also, the total estimated area of the hardened SRAM configuration memory is approximately 2.5 times larger than that of the basic memory using un-hardened bit-cells [51]. Also, there is a considerable amount of performance degradation in hardened memory cells compared to basic SRAM memory cells.

2.2 Design Based Fault Tolerant Techniques

Most of the aforementioned physical hardening techniques have their drawbacks. Process hardening is expensive and often higher protection is not mandatory for the entire design. Solutions at the circuit level depend on the tolerated critical charge Q_{crit} which, in turn, depends on technology, design and electrical parameters. System design level solutions, such as error correcting codes and modular redundancy schemes, better leverage their costs by adding soft error resilience features where required and are technology independent.

Design based fault mitigation techniques represent a general approach which is potentially easier to apply to achieve a desired reliability level of an FPGA-based design intended for critical applications. In contrast to fabrication process level strategies, design based techniques do not require any process level variations and can be applied directly to any COTS FPGAs [52]. Important factors such as design complexity, overhead, latency and power utilization, greatly depend on the chosen fault mitigation strategy and implementation methodologies. *Redundancy* and *Rewriting* are two most common approaches used in design based fault mitigation, which can be applied at different levels and with different perspectives. Redundancy is simply the addition of resources, information, or time beyond what is needed for a normal system operation. Rewriting based techniques are mostly used for clearing errors in memory elements. The use of redundancy can provide additional capabilities within a system, although it can have important impact on a system's performance, size, weight and power consumption. The following two classes of design based techniques can be distinguished.

- **Static techniques** rely on the concept of fault masking. They achieve fault-tolerance by means of passive redundancy and comparator/voting mechanisms, without requiring any action on the part of the system.
- **Dynamic techniques** achieve fault-tolerance by detecting the existence of faults and performing some action to remove the faulty hardware (or fault) from the system (that is why they are also called active redundancy). These techniques rely on using concurrent fault detection, fault location, and fault recovery.

The goal of using design based techniques can vary from simple detection of the presence of an upset in the system to more complex detection and correction of the system error in the presence of a fault. All design-based techniques rely on some kind of redundancy, which can be provided by extra components (hardware

redundancy) or by an extra execution time or by different instants of data sampling (time redundancy). Obviously, very often, a combination of both approaches is used. In today's programmable logic components, such as SRAM-based FPGAs, due to their advanced features such as partial reconfiguration, various *rewriting-based* techniques are used. Examples of this technique are configuration scrubbing, dynamic partial reconfiguration and rerouting design. They are able to clean out quickly an upset from the programmable matrix, so that accumulation of multiple upsets (which are significantly more difficult to handle) can be avoided.

Identifying the most appropriate fault mitigation solution for a specific application design is a challenging task. It requires a comparative trade-off study focusing on fast turnaround time, low cost, high performance and high reliability. A proper set of fault mitigation strategies must deal perfectly with both SETs (which occur in combinational circuitry) and SEUs (which occur occur in sequential elements). In this way, transient faults in the combinational logic will not produce errors subsequently stored in memory cells, and bit flips in the storage cells will be immediately corrected. Each technique has some advantages and drawbacks, and there is always a compromise between area, performance, power dissipation and fault-tolerance efficiency.

2.2.1 Hardware Redundancy

Hardware redundancy relies on addition of extra hardware blocks for detecting and/or tolerating faults. Functional blocks are physically replicated, so that the system could monitor its functioning state on its own. The N-tuple ($N \geq 2$) modular redundancy, also known as parallel redundancy, refers to the approach of having functionally equivalent multiple units running in parallel and executing the same task. All units are highly synchronized and receive the same input information at the same time. Here, we will concentrate on three typologies the most frequently used in real-world designs: dual modular redundancy (DMR), triple modular redundancy (TMR), and quadruple modular redundancy (QMR).

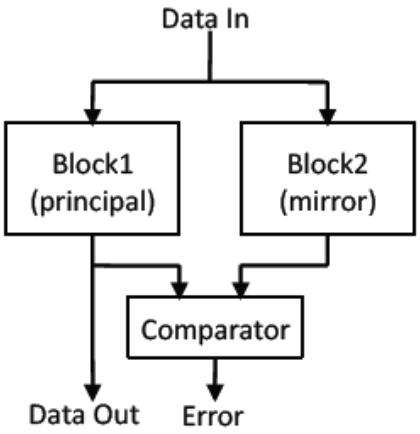
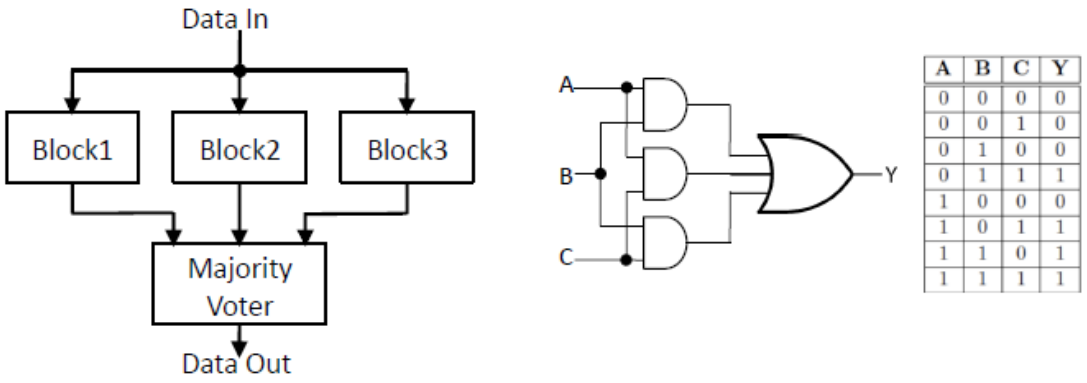


Figure 2.9 – Dual modular redundancy (DMR) [53].

Dual Modular Redundancy (DMR): is the simplest hardware redundancy scheme, only capable of detecting errors. The DMR uses two identical copies of a circuit and compares their outputs to determine if an error has occurred. The comparator circuit signals any disagreement in the operation of the two circuits and signals an error to the system. As shown in Figure 2.9, both primary and mirrored blocks are identical functional elements, served with the same input signal (*Data In*) and the output of both the blocks are connected to a logic comparator to find the mismatch. The presence of a fault/error revealed by any output mismatch is indicated by the status signal (*Error*). The functional output (*Data Out*) can be derived from any of the duplicated blocks. The DMR scheme can be applied at different granularity levels: from the bit level to device level as well as from the fine to coarse granular level. It can be used for both combinational and sequential logic respectively to detect SETs and SEUs. The cost increase of a DMR system over a non-redundant system is over 100% for the additional hardware (redundant block and the comparator) and possibly some extra software development time.



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 2.10 – Triple modular redundancy (TMR) and the majority 2-out-of-3 voter with its truth table [53].

Triple Modular Redundancy (TMR): is the most common hardware redundancy based mitigation technique in many critical application such as aerospace, nuclear, and medical, where the cost of failure could be extremely high. TMR is more practical than DMR, when immediate fault masking and continuous operation (without any interruption) are required. In the case of TMR hardware redundancy approach, all blocks are triplicated and voters are placed at their outputs to determine the correct value by means of voting. As shown in Figure 2.10, three identical functional blocks are served with the same input (*Data In*) and their outputs feed the majority voter. When there is no fault, all three outputs agree. When there is a fault in one of replicated blocks, the majority voter chooses as the correct output the identical output produced by two fault-free functional modules. The circuit can be a mere flip flop or an entire logic design. The majority voter produces the logic value ("1" or "0") output that corresponds to at least of two of its inputs. For example, if two or more of the voter's three inputs are a "1", then the output of the voter is a "1". If the inputs of the voter are labeled A, B, and C, and the output

V, respectively, then the Boolean equation for the voter is: V= AB + AC + BC, according to the truth-table also shown in Figure 2.10.

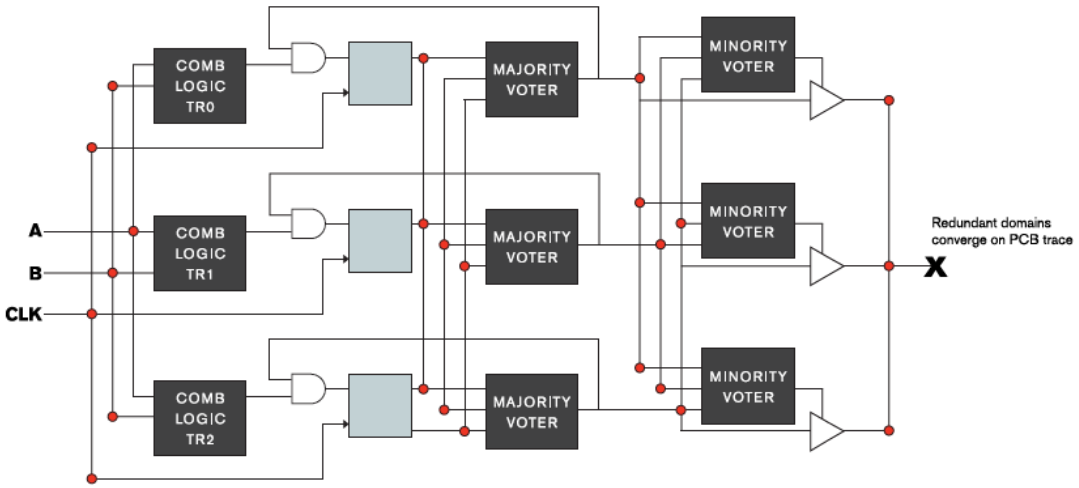


Figure 2.11 – The Xilinx TMR—XTMR scheme [54].

XTMR: TMR deals with transients and upsets in both combinational and sequential logic, respectively. However, if an upset occurs in the voter, the TMR scheme could be ineffective and a wrong value will be present at the output. Another problem of this approach is the possible accumulation of upsets, hence an extra mechanism is necessary to correct the upset in each of triplicated modules before the next fault occurs. The second upset in the same section of the design, but in a different redundant module can lead to incorrect majority voter output. Common mode failures (CMFs) [53] result from failures that affect more than one element of a redundant system at the same time, generally due to a common cause. They may be design faults or operational faults due to external (such as electromagnetic interference (EMI) and radiation) or internal causes.

To facilitate the possibility of designing fault-tolerant systems by any users and overcome the issues related to CMFs, Xilinx proposes the XTMR (Xilinx TMR) supported by the CAD tool TMRTool [54], applicable to Xilinx FPGAs. TMRTool can partially or fully triplicate a design, insert voters, synchronize feedback path loops, and allow customized user-triplicated module insertion. A triplicated design can mitigate SEU impact on the user design. However, a TMR design in a single FPGA device is still vulnerable to SEFI. A TMR design also consumes significantly more resources and power, and can suffer a significant performance degradation. For example, system clock frequencies over 100 MHz in a Virtex-II device might be difficult to achieve. Some other design considerations such as board layout complexity, signal integrity analysis, and asynchronous applications are documented in the TMRTool user guide which can be obtained with the evaluation TMRTool. To note that Xilinx TMR approach supports: i) triplicating all inputs, including clocks and throughput (combinational) logic; ii) triplicating feedback logic and inserting majority voters on feedback paths; and iii) triplicating all outputs, using so called

minority voters to detect and disable incorrect output paths.

The major difference between traditional TMR approaches and Xilinx TMR approach is that the majority voter itself is also triplicated. Also, the minority voters are also added, as shown in Figure 2.11. Thanks to these minority voters, the output of the block which behaves differently will be disconnected by the tri-state buffer (TBUF). If an upset occurs in the throughput logic or in a state machine somewhere in the design, one of the redundant design domains will behave differently from the others. Then, the output voter for that domain will detect that its domain is behaving differently and disable the three-state buffer for that domain, placing its pin in a high impedance state. The other two domains will continue to operate correctly, driving the correct output off the chip. XTMR addresses various shortcomings of a traditional TMR, because it eliminates single points of failure and makes the design immune to SEUs and SETs, as well as is supported by means allowing to avoids accumulation of errors. Although this technique is considered highly efficient, it utilizes significantly larger design area (area overhead). Also, it is difficult to implement this solution in large systems due to limited number of TBUFs in the FPGA devices. Generally, the XTMR based designs require approximately from 3X to 4.5X hardware overhead.

Quadruple Modular Redundancy (QMR): is fundamentally similar to TMR but using four units instead of three to increase reliability. One of a very few designs using QMR-based approach to reliability improvement is *Honeywell's* 2oo4D architecture [55].

2.2.2 Time Redundancy

Techniques based on time redundancy are usually used to detect a SETs, by taking advantage of the transient pulse characteristic to compare the same signal at different moments. Hardware redundancy based schemes require large amount of extra hardware. In those applications in which time is less important than hardware, time redundancy is a means to reduce the amount of extra hardware at the expense of additional time. Due to their nature, transient faults can be detected by repeated computations with comparison of the results obtained. Time redundancy schemes can be classified as: 1) simple time redundancy and 2) full time redundancy, depending on the actual implementation and how a system handles its transient faults.

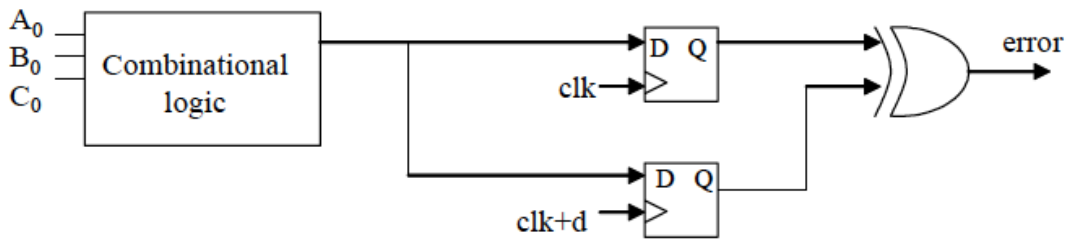


Figure 2.12 – Simple time redundancy scheme [84].

Simple time redundancy takes advantage of the transient character of the spurious pulse (SET) generated by the particle strike and relies on comparison of the output signals at two different moments. The output of the combinational logic is latched at two different times, where the clock edge of the second latch is shifted by time d, which is assumed larger than the SET duration. Should a transient pulse be present, a comparator would indicate its occurrence (an error detection signal), because it does not arrives simultaneously at its two inputs. A single bit comparator is nothing else but the 2-input XOR gate, as shown in Figure 2.12. Such a scheme allows only to detect SETs and fault recovery can be done e.g. by re-execution of the last operation.

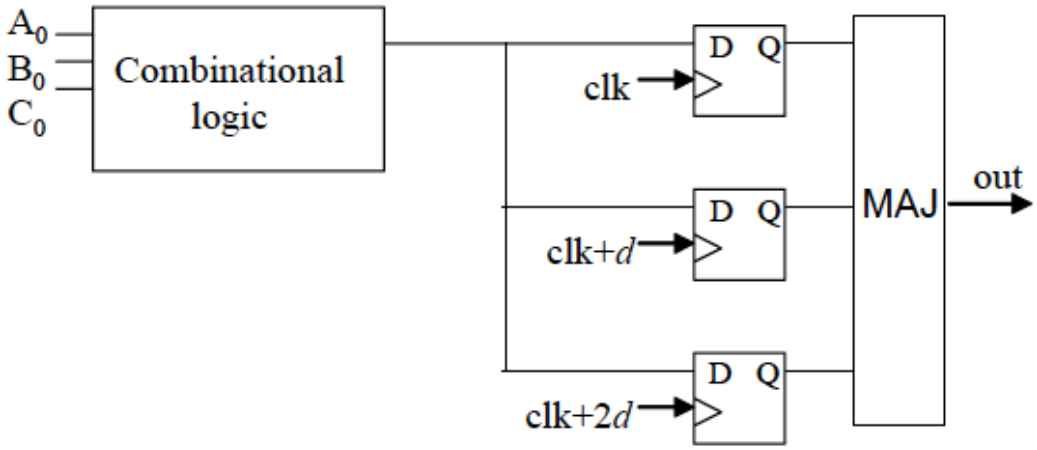


Figure 2.13 – Full time redundancy scheme—An example [84]

Full time redundancy applied to the combinational logic relies on voting the correct output value in the presence of a SET. The name full redundancy comes from the complete N-modular redundancy, where usually $N = 3$, i.e. it is a kind of TMR. In this case, the output of the combinational logic is latched at three different moments, where the clock edge of the second latch is shifted by the time delay d and the clock of the third latch is shifted by the time delay $2d$. A voter chooses the correct value. The full time redundancy scheme is shown in Figure 2.13. The area overhead comes from the extra sample latches and the delay penalty is given by $clk + 2.d + tp$, where d depends on the duration of the transient current pulse and tp is the delay of the majority voter. Unlike simple time redundancy, where only detection of transients is possible, the full time redundancy allows to tolerate the effects of transients.

Actually, time redundancy can be applied to protect logic circuits in two different ways. One case already described and illustrated by the examples shown in Figures 2.12 and 2.13 requires some hardware support (the duplication or triplication of registers). Another way of applying time redundancy is known as re-computation approach. In this case, transient effects are cleared from the circuit nodes by re-computing the task with same set of inputs in the next operating cycle. This method

helps to distinguishing between transients and permanent faults: if re-computing of the task after the detection of the fault succeeds (i.e. a fault disappears0, then it was transient; otherwise, the fault is declared permanent.

2.2.3 Configuration Scrubbing and Partial Reconfiguration

The *re-programmability*, *read-back* and *run-time partial re-programmable* capabilities of modern FPGAs (e.g. of Xilinx Virtex family devices) allow for fast detection and correction of SEUs in configuration memory [60].

Scrubbing: Programmable FPGAs contain a large number of memory cells which are all susceptible to radiation-induced upsets.

Amongst them, the vast majority of memory cells are those of the configuration memory (typically over 90%). Configuration memory upsets are particularly troublesome, since they may change the user's circuit, possibly altering the function of configurable logic, I/O, or other resources as well as changing the structure of the routing network. An important and most common technique used to reduce the effects of upsets within the configuration memory is called scrubbing [60]. Configuration scrubbing does not contain any error detection mechanism but it relies on periodic refreshing of the FPGA's configuration memory during its normal operation. Its goal is to clean up configuration errors caused by SEUs and thus to prevent the build-up of multiple configuration errors [63]. In this case, the refresh frequency must be superior to the expected appearance rate of the SEUs. A through study on upset rates should be done before applying scrubbing, which requires also to take into account the device manufacturing technology as well as the environment conditions. Scrubbing requires the availability of a golden backup copy of the whole configuration data either inside or outside of the device. Obviously, memory which holds the golden copy must be protected against radiation effects. The effectiveness of internal and external scrubbing is discussed in [56].

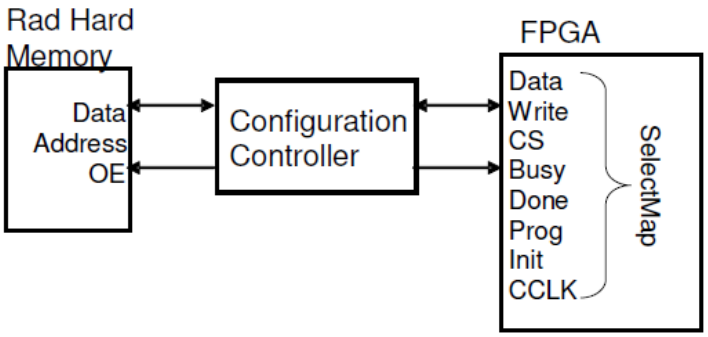


Figure 2.14 – Architecture of a traditional scrubbing scheme [57].

Configuration scrubbing requires more infrastructure than that needed by an unprotected FPGA-based systems. As shown in Figure 2.14, external memory and a processor or a configuration controller are needed to support the scrubbing process. The external memory is required to hold the "golden copy" (i.e. error-free) of the

configuration bitstream, whereas the configuration controller is required to sequence through the partial reconfiguration steps. More sophisticated scrubbing techniques require a fully programmable processor. Because these components manage the configuration of the FPGA, it is essential that they are also protected against SEUs through appropriate mitigation or rad-hard by design techniques. The scrubbing technique of Figure 2.14 is referred to as the "blind scrubbing" since it configures the FPGA whether or not upsets have occurred (i.e. blindly). A variation of blind scrubbing is called "read-back with correction".

Partial Reconfiguration: The read-back function is an efficient means for SEU detection. If a particle penetrates the susceptible portion of a configuration SRAM cell and thus alters its state, a read-back and verification of the configuration data will detect the upset. To perform a verification (SEU detection), the configuration data which is read back from the device, is compared with the configuration bitstream. Whenever an upset is detected, it is corrected by executing reconfiguration operation [60]. In this method of error detection, bit by bit comparison of configuration data is performed. This requires the use of a mask file and read-back file, each of which has the same size as the original bitstream used to configure the FPGA. Because this method would effectively triple amount of system memory required for configuration and read-back operations, it is not generally considered applicable for space applications [60]. Moreover that the read back interval, which is calculated in function of the structure of the device as well as the space environment, could be a crucial factor to be considered [64].

The time required for SEU correction may be dramatically decreased by using partial reconfiguration. This is a significant point of consideration because complete reconfiguration implies "de-configuration" which means bringing the part "off-line" during the correction cycle and thus losing all internally stored data. Partial reconfiguration allows individual frames to be written to the configuration memory. The frame is the smallest reconfigurable unit for a given FPGA device. The number of frames and the bits per frame vary for different devices). Therefore, only the frame whose cell is affected by the SEU would need to be corrected. The fault mitigation technique presented in [62] exploits this partial reconfiguration feature and is based on using a cyclic redundancy code (CRC) checker for each FPGA frame. In this method, CRC is periodically generated for each frame (during the read-back) and compared to the expected CRC value. This method greatly reduces the amount of system memory required to perform SEU detection. However, the above read-back based techniques have some important limitations. An error-free read-back of the configuration bitstream does not guarantee that an SEU did not occur. The FPGA contains hidden state that cannot be read back, and upsets affecting the hidden state can conceivably cause errors in the design without any bitstream errors being detected [61]. For example, SEUs in the flip-flop states can occur without disturbing the bitstream.

2.2.4 Error Detection and Correction Codes

Hardware redundancy based techniques are effective against errors affecting a single module, but they could be prone to MBUs and accumulated SEU-induced multiple

errors producing erroneous outputs when more than one copy of any redundant module is affected by SEUs at the same time [84]. To avoid accumulation of SEU-induced multiple errors, fault-tolerance techniques supported by some form of configuration scrubbing can be the simplest way. An alternative to scrubbing is configuration read-back which enables verification of the bitstream frame data by performing a bit-by-bit comparison. However, the latter requires a mask and a read-back files, a size of each is equal to the size of the original bitstream used to configure the FPGA, which is time-consuming and triplicates the memory required to perform the read-back and reconfiguration process.

To perform multi-bit error correction, scrubbing/reconfiguration based techniques are considered more efficient. They can handle multiple upsets but require continuous access to an external storage device which contains the original (golden) configuration bitstream. Majority of reconfiguration based techniques rely on external non-volatile memory devices which also must be protected against SEUs. Such radiation-hardened memories can be expensive. Additionally, excessive access delay for the external memory can degrade performance of the system. In the case of multi-bit error detection, the reconfiguration process involves excessive delay in accessing external storage device to recover the original configuration data through limited I/Os in FPGAs. For instance, the internal FPGA system clock runs at several hundred megahertz, while the slower I/O clock can use only one tenth of the system clock.

Error detection and correction (EDAC) codes represent *data redundancy*, as a number of redundant bits is added to the data, to facilitate error detection and correction. EDAC schemes are also used in reconfigurable FPGAs to mitigate internal faults, especially SEUs. SEUs occur in memory elements in the form of single bit and multi-bit upsets (SBU and MBU). Memory elements in reconfigurable FPGAs can be classified as: 1) user data memory and 2) configuration memory. Most of the memory bit cells in SRAM-based FPGA are configuration bits, occupying more than 98% of memory in the device. Thus, the probability of SEU-induced errors in configuration bits is much higher compared with that in user data.

There exists several multi-bit error correction codes, used in communication and massive storage applications such as Hamming codes [68], low-density parity check (LDPC) codes [69], turbo codes [70], Viterbi codes [72], and Reed-Solomon (RS) codes [71]. Figure 2.15 illustrates hardware complexity of encoding and checking circuitry and bit error rate (BER) efficiency of those error correcting codes [66]. Clearly, the *Hamming code* involves minimal hardware complexity compared to other codes.

Hamming Codes: are an extension of simple parity checking method that can be used to detect and correct errors. The basic idea is to have several parity bits (called check bits in Hamming codes) and assign different bits to several overlapping groups. If some parity bits are correct and others are not, the bit in error can be deduced. The simple Hamming code permits correcting single-bit errors. Let k be the number of information bits and m the number of check bits used. Because the check bits must check themselves as well as the information bits, the value of p, interpreted as an integer, must range from 0 to which is distinct values. Because m

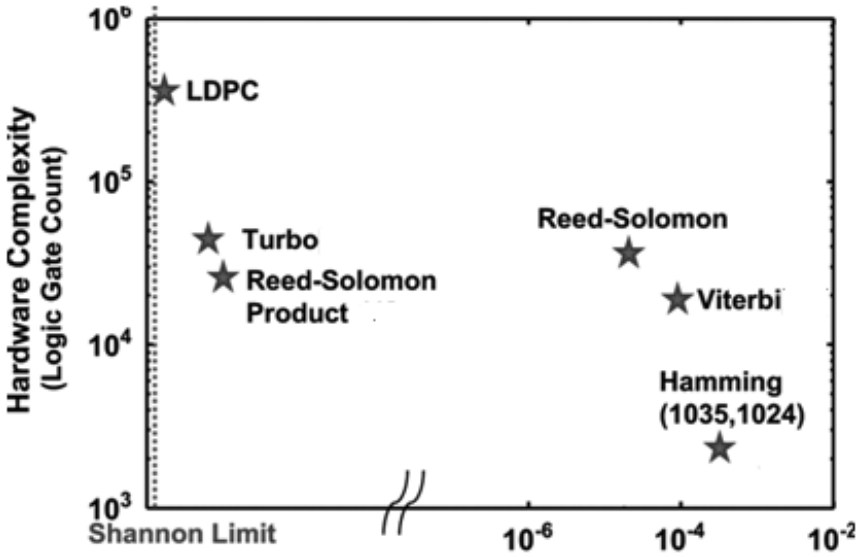


Figure 2.15 – EDAC: hardware complexity (logic gate count) vs. BER [66].

bits can distinguish 2^m cases, the following inequality (known as the Hamming rule) must hold:

$$2^m \geq m + k + 1 \tag{2.1}$$

It applies to any single error correcting (SEC) binary forward error correcting (FEC) block code. The Hamming SEC code is the most common and the simplest error correcting scheme, which is implemented by adding check bits to the data according to the following pattern:

1. The data bits are numbered from left to right, starting with 1.
2. Every bit whose number is a power of 2 (i.e. 1, 2, 4, 8, ...) is a check bit.
3. The other output data bits (i.e. 3, 5, 6, 7, 9, 10, ...) contain the ordered data bits.

Each check bit establishes even parity over itself and a group of data bits. A data bit is in a check bit's group if the binary representation of the data bit's number contains a 1 in the position of the check bit's weight. For instance, the data bits associated with the check bit 2 are all those with a 1 in the 2's position of their binary bit number-bits 2, 3, 6, 7, and so forth. More details about *Hamming* code syndrome generation and error detection and correction can be found e.g. in [68].

Hamming SEC/DED: For many applications a SEC code could be considered unsatisfactory, because it has been observed that more than single errors are also likely. The Hamming code can be extended to correct single bit errors and detect double errors in a data word by adding one global parity check bit (we assume even parity) on all the bits of the SEC code word. Such a *Single Error Correction and Double Error Detection (SEC/DED)* code, called an extended Hamming code, has been used the most often in computing systems.

Number of data bits k	m for SEC	m for SEC/DED
1	2	3
2 to 4	3	4
5 to 11	4	5
12 to 26	5	6
27 to 57	6	7
58 to 120	7	8
121 to 247	8	9
248 to 502	9	10

Table 2.1 – Check bits for SEC and SEC/DED [68]

The number of check bits required for both *Hamming SEC* and *Hamming SEC/DED* codes is shown in Table 2.1. Obviously, the rightmost column simply shows that one more bit is required for a SEC/DED code than for the simple Hamming SEC. For example, from this table one can easily find that to protect a 64-bit memory word using the SEC/DED ECC, eight check bits are required, giving a total memory word size of 72 bits.

Perfect Hamming Code: Answering the question: "What is the minimum number of check bits required to protect the maximum number of data bits under the occurrence of single bit error?" leads to the notion of the perfect Hamming code. Table 2.1 shows the minimal and the maximal numbers of data bits k which can be protected by a given number of check bits m. For instance, 7 check bits are required to protect 27-bit data, whereas the same 7 bits are also required to protect the maximum of 57-bit data. Therefor, the perfect Hamming code condition is

$$2^m = m + k + 1 \tag{2.2}$$

By respecting the perfect Hamming code condition, given by equation (2.2), the check bit overhead can be maximally reduced.

There are some effective ways of constructing a built-in EDAC scheme using perfect Hamming codes to mitigate radiation induced faults in configuration memory, whose overhead is reduced compared to other EDAC schemes.

- In [65], the proposed detection/correction scheme is called Matrix codes (MC), since the check bits are arranged in a matrix format. One important restriction of error correction based on the MC is that if there are more than two errors in each code word, MC can correct them if and only if there are only two errors in each row of the matrix and one in each column.

- In [66], the Hamming based 2-D product code (2-D HPC) is introduced, which performs SEC/DED in two different axis (two directions: row-wise and column-wise) to deal with MBUs. It is shown that multi-bit error correction capability of this built-in 2-D HPC can improve the reliability, and hence, system availability, by orders of magnitude. However, there are the cases when

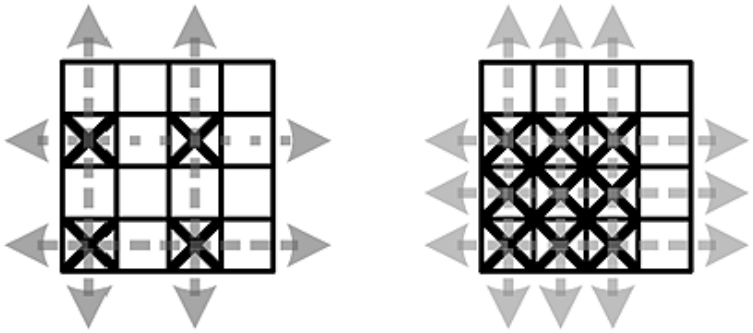


Figure 2.16 – 2-D HPC: Non-detectable and non-repairable case [66].

the 2-D HPC fails to recover the data. As shown in Figure 2.16, when there are multi-bit errors in both directions, the 2-D HPC may not be able to recover the data. Due to advanced high-scaling technologies and shrinking device sizes, the probability of occurrence of MBUs has highly increased and it will be even higher in the future. So there is a strong urge to develop some error protection techniques to attain 100% built-in multi-bit error recovery capability.

2.2.5 Hybrid Approaches

Traditional fault detection and mitigation techniques based on *hardware redundancy* (such as DMR and TMR), *time redundancy*, and *error detection and correction codes* can be combined with advanced features of today's reconfigurable FPGA architectures such as *configuration read-back and write-back* and *run-time partial reconfiguration*, to obtain more sophisticated fault mitigation strategies allowing to achieve maximal reliability and reduced area, time and power overhead. Such hybrid approaches helps in finding an acceptable trade-off between the overhead and the reliability.

Redundancy with Partial Reconfiguration (PR)

Many fault mitigation approaches, which adopts combined techniques based on partial reconfiguration and redundancy to achieve a reliable system, have been devised in recent years. Generally, redundancy is used mainly to detect and/or mask errors whereas the partial reconfiguration is used to reconfigure only the faulty portion (faulty replica) of the design.

(i) Resource Duplication and PR: DMR is a simple hardware redundancy scheme which uses two replicas (of the same logic) whose outputs are checked by a comparator. However, the standalone DMR scheme cannot identify neither the fault-free module—whose output is correct and can be used by a system, nor the faulty module—whose state must be corrected or which must be reconfigured. It needs an additional circuitry or mechanism to locate the affected resource.

- As in [79], a dedicated hardcore processor is used to detect the error as well

as to decide when and which part need to be reconfigured.

- In [83], a dependable DMR system is developed based on dynamic reconfigurable FPGAs, in which fault detection and mitigation is equipped with redundant comparison test units and redundant encoding detection logic.

- Some techniques use combined approaches of DMR with concurrent error detection (CED) based on time redundancy to achieve desired reliability as in [84]. However, the possibility of using this method depends on the logic of the circuit that is mapped onto the FPGA.

There are some approaches which use higher granular redundancy schemes as in the case of fault-tolerant dynamic multi-processor system on chip (FT-DyMPSoC) [81]. The FT-DyMPSoC uses a processor based redundancy and lockstep scheme. This methods provides increased system performance by the utilization of parallel computing. As for fault-tolerance, each processor in the device needs access to the partial bitstream of the other processors, but only one processor has connectivity to the compact flash memory (which holds the configuration bitstream). Hence, the system is forced to use another shared memory to provide bitstream access to other processors. Eventually, the size of consumed on-board memory has increased [90]. Also in such paired, mutually reconfigurable systems, reliability is ensured only until the interfaces between the partially reconfigurable modules (PRM) are stable. Generally, in resource duplication based techniques, the restoration time and input rate are being crucial factors, which are application dependent [79].

(ii) Resource Triplication and PR: Initially, resource triplication based techniques were developed just to mask the effects of the error without the aim of identifying and correcting the fault. The results presented in [75, 76, 78] show that dynamic reconfiguration and TMR seems to be the most effective way to mitigate the effects of radiation induced faults by means of efficiently filtering the errors at the voter stage. Such schemes avoid fault propagation, which has been considered the most important issue to be addressed in the context of fault mitigation. TMR can be adopted at different granularity levels. Area overhead, performance degradation and reconfiguration timing characteristics change accordingly. Finding the faulty module (faulty replica) and more precise localization of the fault is considered as an important key point in a fault-tolerant system based on TMR with partial reconfiguration. It requires the complete design space exploration and it involves partitioning of the entire system into independently controlled sub-systems, and then placing this sub-systems into different reconfigurable regions of the FPGA [75].

EDAC with Partial Reconfiguration (PR):

In many architectures, EDAC codes are coupled with run-time partial reconfiguration technique to protect configuration bitstream against SBUs and MBUs. In order to detect errors in the content of configuration memory, some kind of parity bits for error detection and correction codes are embedded within the memory content. In the case of Xilinx's FPGAs, SEC/DED parity bits are used [73]. Xilinx provides a primitive, called FRAME_ECC, with the Virtex 5 devices. The FRAME_ECC

primitive examines the data read back from the configuration memory and uses the SEC/DED check bits to detect and correct errors in the bitstream. This scheme significantly simplified the design of a scrubber (configuration scrubbing) with read-back capabilities, as read-back is a background process. As the frame is the smallest reconfigurable unit in the configuration bitstream plan of reconfigurable FPGAs, it is sufficient to reload only the erroneous frame with the help of 'run-time partial reconfiguration'. *Internal* and *Hybrid* scrubbing based configuration bitstream protection schemes make use of 'EDAC + PR' combination.

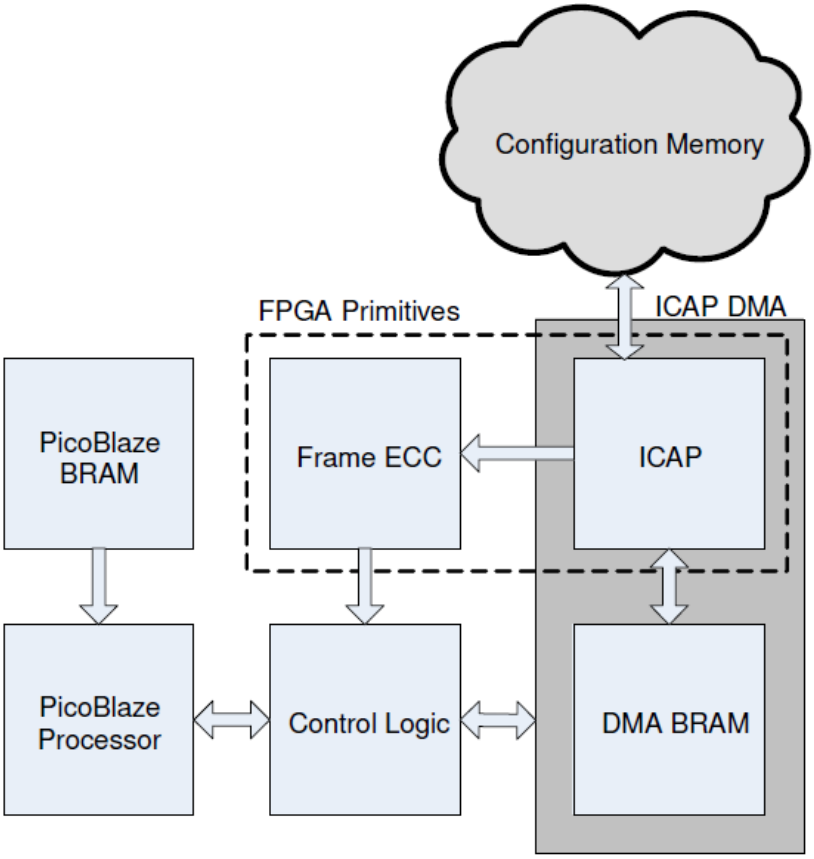


Figure 2.17 – ICAP-based internal scrubbing scheme [58].

- **Internal Scrubbing [58]:** it takes advantage of internal configuration access port (ICAP) to implement the scrubbing controller in the FPGA fabric. Thanks to the ICAP, the FPGA is capable of reading and modifying the configuration internally. The ICAP ports are built into several FPGAs from Xilinx (Virtex II and above). Internal scrubbing process performs read-back of each frame via ICAP interface: it uses Frame_ECC to detect errors, it corrects errors using the Frame_ECC syndrome value, and writes corrected frame back via ICAP interface. ICAP based internal scrubbing scheme architecture is shown in Figure 2.17. This scheme does not require external memory, external controller and external I/O pins.

  The internal scrubber of Virtex 7 series FPGAs is capable of checking the

entire device configuration every 30 ms. This hardware block provides an excellent means to speed up device scrubbing speed, though multi-bit errors will still need to be handled. Unfortunately, FPGA designs using the ICAP are susceptible to the same radiation upsets they are developed to fix. For such designs multiple copies of the scrubbing state machine will be needed. There will still be a single point of failure where the logic connects with the ICAP interface at which a single upset could hang the system.

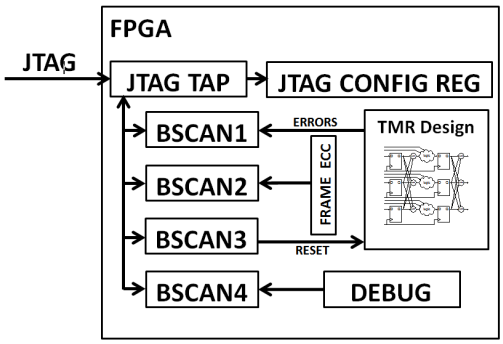


Figure 2.18 – The hybrid scrubbing system [59].

- **Hybrid Scrubbing [59]:** By combining internal scrubbing and traditional JTAG-based external scrubbing techniques, a fast yet robust scrubbing system was developed [59]. In this hybrid scrubbing system, the internal scrubber unit is configured to correct all single bit errors it encounters. During this operation the internal scrubber also notifies if there is any multi-bit errors detected. The internal scrubber will freeze on any error it does not correct until the configuration is accessed through another port (external). Once the configuration has been accessed externally, the internal scrubber begins scrubbing again starting at the first frame of the device. A block diagram of the scrubber internal to the FPGA is shown in Figure 2.18. For this design, many hardware components have been connected to the BSCAN ports of the JTAG. The scrubber then reads in information from these JTAG ports to get the status of the system.

2.2.6 Selective or Partial Mitigation

Although TMR based techniques impose greater overhead in-terms of area and power, for some highly-critical applications TMR schemes are still considered more efficient in terms of higher reliability than other mitigation methods [85]. Due to resource constraints and/or system constraints, TMR of an entire user FPGA design (full TMR) is not always feasible. If this is the case, partial triplication of the FPGA design may be the next best alternative [86]. Mitigation of part of the design can increase the overall reliability of the design at a lower cost than full TMR.

One such technique, called selective TMR (STMR) was proposed in [82]. It identifies SEU "sensitive" gates in a given circuit and then applies TMR selectively to those gates. The sensitivity of a gate to an SEU is determined by the signal probabilities of its inputs. A gate is sensitive if an SEU on any one of the inputs is

likely to be propagated to the output of the gate. The advantage of this technique is that the area overhead is typically much smaller than that of the full TMR.

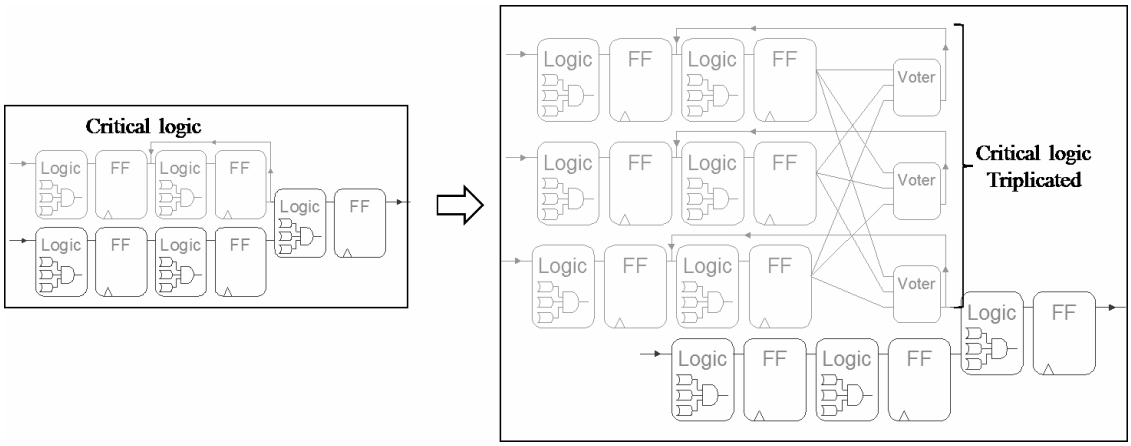


Figure 2.19 – Implementation of partial TMR: An example [63].

Another attractive alternative to full mitigation is to mitigate only the most critical sections of a design. As proposed in [63], partial TMR classifies circuit structures depending on their importance and applies TMR selectively according to this classification. An example of a partial TMR implementation is shown in Figure 2.19, where the 'red' color indicates critical logic and 'blue' indicates non-critical or less critical logic. Higher priority is given to structures causing "persistent" errors within the design. (Persistent errors are caused by SEUs within the configuration memory corresponding to sequential circuit structure; an incorrect state remains until appropriate reset measures are taken.) By selectively applying mitigation to a design, the most effective balance between mitigation cost and reliability can be found. For certain applications, applying selective mitigation to components with persistent errors can yield higher returns in reliability per unit cost than full mitigation.

Any partial mitigation technique using TMR is based on the idea that only a subset of a design's components will be protected with TMR by triplicating them when necessary. This subset must be carefully selected so that the resulting partially mitigated design would be as reliable as possible. Basically, some other strategies like time redundancy, EDAC and other hardening techniques, etc., can also be adapted to obtain partial fault mitigation. Performance and overhead trade-off radio varies case-to-case, according to the chosen fault mitigation technique and the critical nature of implemented application.

2.2.7 Fine Granular vs. Coarse Granular Strategies

Fault mitigation schemes can be adopted at different granularity level but the effectiveness, implementation complexity and overall reliability of the system greatly depend on the chosen granularity level. Many recent works discuss the effectiveness of fault mitigation strategies with respect to the granularity in which they are applied.

High-granularity level (or coarse granularity) mitigation techniques do not provide simple and robust recovery mechanism after an error has been detected in one of its internal modules. They can guarantee that an error will be detected only when it is manifested on the output of the module, where it is compared/voted with the outputs of the redundant modules, provided that some redundancy based scheme was employed. The internal state of the erroneous module can at that stage be very much different from the state of the remaining redundant modules. Any further execution will be meaningless since the erroneous state will not be automatically recovered from. The likely consequence is that the application has to be reset or some other means of action has to be taken to recover the faulty portion/module (the decision can be made only by analyzing the outputs of the modules). It is required to save the correct context and restore it after the error is corrected. Then the module could resume the execution at the saved point. Normally, to bring the faulty portion back to its recent stable state, additional mechanisms like *check-pointing* with *roll-back* or *roll-forward* are needed [90]. Also, once the recovery process terminates, an immediate resynchronization of the modules must be attempted, which is really a challenging task. Even a small delay in doing this, will lead to loss of data and extra operational downtime [89].

According to [86], the coarse level of granularity does not give much flexibility and yields lower reliability. At this level of granularity, there could be a significant amount of unused resources on the FPGA device. The BLTmr tool proposed in [86], works at the possible fine granular logic (LUT) level rather than a higher-granular level and shows how effectively the entire FPGA is used for comparatively higher possible reliability at this particular fine granular level. Suitable redundancy granularity level for fault mitigation is examined in [88], where it is shown that it results that the homogeneous nature of reconfigurable architectures makes it easy to achieve higher reliability using redundancy at fine grain level.

The authors of [88] propose *Fine Grained TMR (FGTMR)*, which regards SRAM-based FPGAs as a system composed of fine grains which include LUTs and corresponding flip-flops. It applies TMR to these fine grain parts of the system. Obviously, every TMR-ed fine grain part needs a voter to select the correct output value, which results in a longer critical path. To overcome this bottleneck, the methodology based on *quadded-logic* is presented in [88], which uses inherent voting in fine grain parts of FPGAs by wiring different copies of a grain in a special manner and using them to vote for the correct value.

2.3 Architectural Customization for Reliability

To achieve the desired level of reliability in reconfigurable FPGA based applications, two mainstream strategies have been followed: i) making the architecture fully radiation hardened by the fabrication process itself (section 2.1), and ii) applying various mitigation strategies to the COTS architectures at the application design stage (section 2.2). Both *radiation hardening* and *design based* solutions have their own advantages and disadvantages as discussed in previous sections. An alternative solution would be more interesting, where advantages of different approaches could be availed in terms of hardware overhead, power consumption, reliability improve-

ment and design flexibility, etc. The latter is possible by developing new reliable architectures, where various reliability mechanisms can be integrated at different levels of architecture, including fabrication process, hardware architecture, configuration bitstream plan, application design and supporting software tool level. There are several research groups who work on designing reliable reconfigurable architectures with the same goal like, for example, Xilinx's space grade Virtex 5QV [92] and DeSyRe project [93].

2.3.1 Xilinx Space-grade Virtex-5QV FPGA [91]

Virtex-5QV FPGA architectures are developed by Xilinx to meet the requirements of space applications that demand both high performance and high reliability. It uses second generation Advanced Silicon Modular Block (ASMBL) column-based architecture. Virtex-5QV devices are user-programmable gate arrays with various configurable elements and embedded cores optimized for high-density and high-performance system designs. They implement various fault mitigation techniques including process level variations, hardened memory elements, hardening latch, TMR, SET filters and EDAC codes.

- At fabrication level, Virtex-5QV FPGA technology incorporates a thin epitaxial layer in the wafer manufacturing process for immunity against *Single Event Latch-ups (SELs)*.

- CLBs, the basic logic elements of Xilinx FPGAs, provide combinational and sequential logic as well as distributed memory and SRL32 shift register capability. The CLB of Virtex-5QV FPGA is based on real 6-input LUT technology and contains eight user registers implemented with RHBD dual-node latches, thus providing the same protection from static SEU as with the configuration latches. Additionally, protection from SET during dynamic operation is provided by transient filters placed on the inputs of each register, which provide up to 800 ps of glitch filtration on the data, clock, clock enable, and set/reset input paths of each register.

- I/O blocks (IOBs) provide the interface between package pins and the internal configurable logic. The Virtex-5QV FPGA IOBs have RHBD dual-node latch registers, which do not have SET filters. Thus, transient filtration is available on CLB registers only.

- The Virtex-5QV FPGA configuration control logic and JTAG controller have been hardened to SEUs and SETs with embedded TMR. Each control register is implemented with independent and redundant EDAC circuits for autonomous state correction. This mitigation combination eliminates most of SEFIs.

- Fault mitigation in the configuration memory is handled in two ways, as the uncertainty in the configuration bit could be either due to a fault in the memory cell or because of data corruption. In Virtex-5QV FPGA, the configuration memory is implemented with RHBD dual-node latches that provide the SEU

hardness nearly 1000 times higher than of the standard cell latches used in the commercial devices. Additionally, the RHBD configuration latch is nearly impervious to upsets caused by proton interaction. The programming data corruption is handled by ECC and partial reconfiguration based approaches.

- Block RAM modules provide flexible 36 Kb true dual-port RAM that are cascadable to form larger memory blocks. In addition, Virtex-5QV FPGA block RAMs contain optional programmable FIFO logic for increased device utilization. Each block RAM can also be configured as two independent 18 Kb true dual-port RAM blocks, thus providing lower memory granularity for designs needing smaller RAM blocks. In Virtex-5QV FPGAs, block RAM contains an integrated ECC and write-back function to autonomously detect and correct SEU in the block memory content.

2.3.2 DeSyRe—On-Demand System Reliability [93]

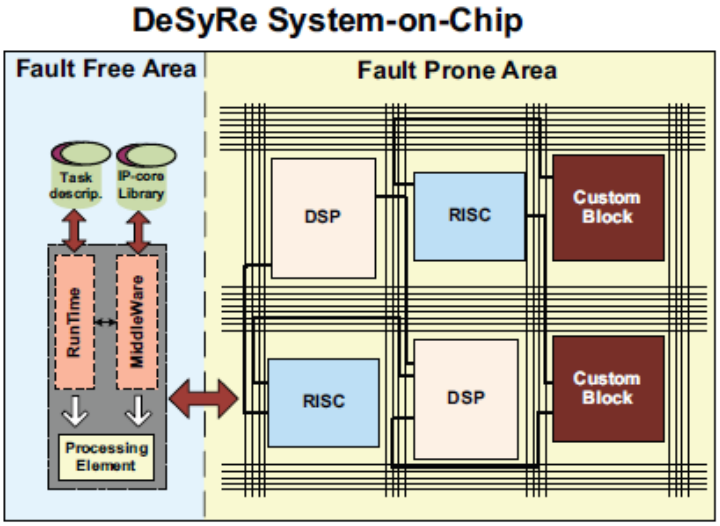


Figure 2.20 – DeSyRe physical partitioning: fault free and fault prone area [93].

DeSyRe project builds an on-demand adaptive and reliable architecture, which is partitioned across two design dimensions: a physical and logical abstraction. The physical partitioning is based on different technological substrates with different fault densities. The logical partitioning considers functional viewpoint.

Figure 2.20 illustrates the physical partitioning of the DeSyRe architecture. The design area is physically divided into two, fault-free and fault-prone sections. The fault-free section provides overall system management whereas the fault-prone section provides the actual system functionality (application). The logic employed in the fault-free section is responsible for: i) on-line testing and fault-tolerance, ii) run-time task scheduling (being aware of task characteristics such as safety-criticality), iii) resource allocation (under varying availability of computational resources) and

iv) scrubbing/reconfigurable schemes (to achieve flexible and fault-tolerant operation).

The fault-prone area is under the direct control of the fault-free area. The fault-prone section has various reconfigurable components to implement desired functionality based on target application. The resources in the fault-prone area are supervised by the monitoring logic scheme implemented by the fault-free area. The resources in the fault-prone area are designed with self-checking mechanisms to detect and/or correct the faults by themselves. The fault detection and correction mechanisms are managed and controlled by the fault-free section.

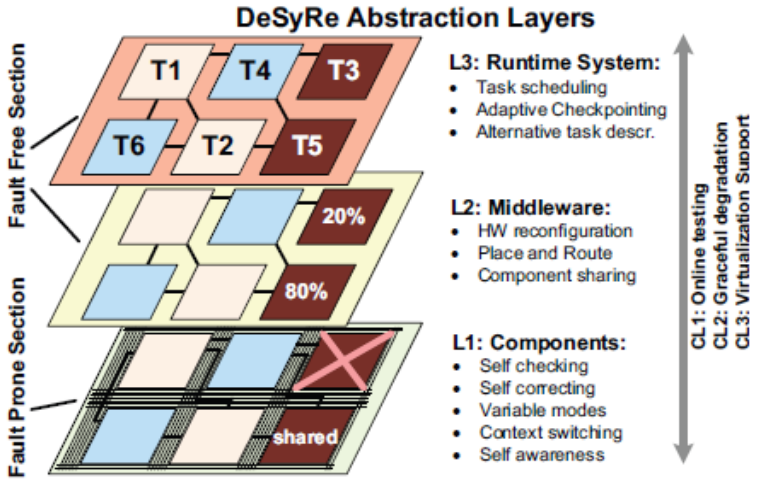


Figure 2.21 – DeSyRe logic partitioning: abstraction layers [93].

The logic partitioning organizes the DeSyRe architecture in three layers, as shown in Figure 2.21: component, middle-ware and run-time. The division is based on the abstraction level involved and the task handled by each layer. The bottom layer (component) deals with the fault-tolerance issues of each functional component present in the fault-prone section of the architecture. The middle layer is responsible for hardware synthesis and reconfiguration of the components in order to provide correct functioning of the underlying hardware to upper layers. Finally, the upper layer (run-time management system) handles run-time issues of the system. Its basic functionality is to schedule tasks to the components and to adapt the system to function normally in the presence of a fault. In DeSyRe, permanent faults are detected by 'on-line testing' and mitigated by task re-positioning. Transient faults are detected at software level ECC and corrected with help of check-pointing and rollback mechanism.

2.4 Summary

Fault-tolerance must be guaranteed at an affordable cost by making use of simple detection and efficient recovery techniques. Physical hardening techniques require changes of the fabrication process whereas rad-hard SRAM cells require excess hard-

ware causing power and performance degradation. Redundancy based techniques are quite simple to implement but their effectiveness is questionable in today's high fault-rate scenario. Hardware redundancy schemes such as DMR and TMR are very costly, because they involve respectively more than 100% and 200% hardware overhead. Time redundancy is costly in terms of latency and cannot be applied several applications. EDAC codes are applicable only to handle SEUs. EDAC codes prove to be promising when they are coupled with partial reconfiguration to address SBUs and MBUs in the configuration bitstream. Most of configuration bitstream protection schemes that rely on the complete or partial reconfiguration are dependent on 'golden copy' of the configuration data. Hamming codes based error correcting schemes are quite interesting when an application cannot afford to spend resources on keeping the configuration bitstream in the 'golden memory'. As MBUs are the major reliability concern of memory systems, it is very important make sure that the chosen EDAC scheme is capable of correcting the maximum of error patterns.

However, all the aforementioned techniques, physical hardening based and design based, focus on specific fault models, restricted to the logic element or a module to which they are applied. When COTS FPGAs are used for critical applications, choosing various fault mitigation strategies (according to expected fault models) and to adapt them to the application implementation is a time consuming and tedious task. To simplify and fasten the design cycle of mission-critical applications, unified reliable reconfigurable architecture models are required.

Section 2.3 discussed various issues of such reliability aware customized reconfigurable architectures. Virtex-5QV is the only reprogrammable and highest density space-grade FPGA in the industry now. High-blend reliability mechanisms, applied at various levels of the architecture, keep Virtex-5QV in a strong market place as far as highly critical applications are concerned. However, to protect the configuration bitstream, both Virtex-5QV and DeSyRe architectures adapt the concept of 'golden memory'. Besides them, there are only a few other reliability-aware customized architectures, such as those discussed in [94, 95]. The PAnDA architecture of [94] is focused specifically to deal with physical substrate variations. The authors of [95] propose customized fault-tolerant pipe-lined sequential and combinational circuit architectures, which makes use of error detection circuits (EDC) instead of DMR or TMR. Unfortunately, designing specific fault-aware logic circuits without supporting tool-set and architectural adaptability will make fault-tolerant functional implementation a complex task.

Chapter 3

Dynamically Reconfigurable Reliable Architecture—The ARDyT Project

3.1 The ARDyT Framework

The purpose of the ARDyT project is to provide a complete environment for the design of a fault-tolerant and self-adaptable platform for a system development. In other words, the ARDyT project is a framework, which includes the design of a reconfigurable architecture, its associated programming environment and management methodologies for field programmability, diagnosis, testability, and reliability. The techniques considered are focused on providing low-cost design solutions for critical applications, such as aerospace, nuclear, and medical applications. In particular, its aim is to develop a dynamically reconfigurable embedded architecture with specific and flexible support mechanisms for the management of reliability. The recovery aspect of dependable systems are studied in Chapter II, accordingly, fault detection and diagnosis strategies are developed. An appropriate strategic coupling of partial reconfiguration technology and advanced fault detection techniques promises a self-adaptable fault-tolerant architecture (i.e., a self-healing architecture that is capable of detecting as well as correcting the faults, and then to recover the system functionality from its faulty state as early as possible). This ability of keeping the system available with correct working functionality, even during the presence of a fault, is what most of the present day safety and mission critical applications need.

The project is organized around three main axes shown in Figure 3.1. The first axis is the study and implementation of reconfigurable hardware with integrated (read dedicated) resources for improving reliability and physical diagnosis of the circuit. The architecture design requires development of a software environment enabling its exploitation, including design space exploration, which helps in tasks such as rapid prototyping, optimization, and system integration. The second axis of the project concerns the definition and development of a set of design tools (synthesis, placement, and routing) for the proposed architecture. This software framework also enables to synthesize the applications with the insertion of high-level diagnostic mechanisms (such as duplication) to improve reliability at the system level. The last axis of the project deals with the definition of test and fault mitigation methodologies dedicated to the proposed dynamic architecture. Adaptability and various

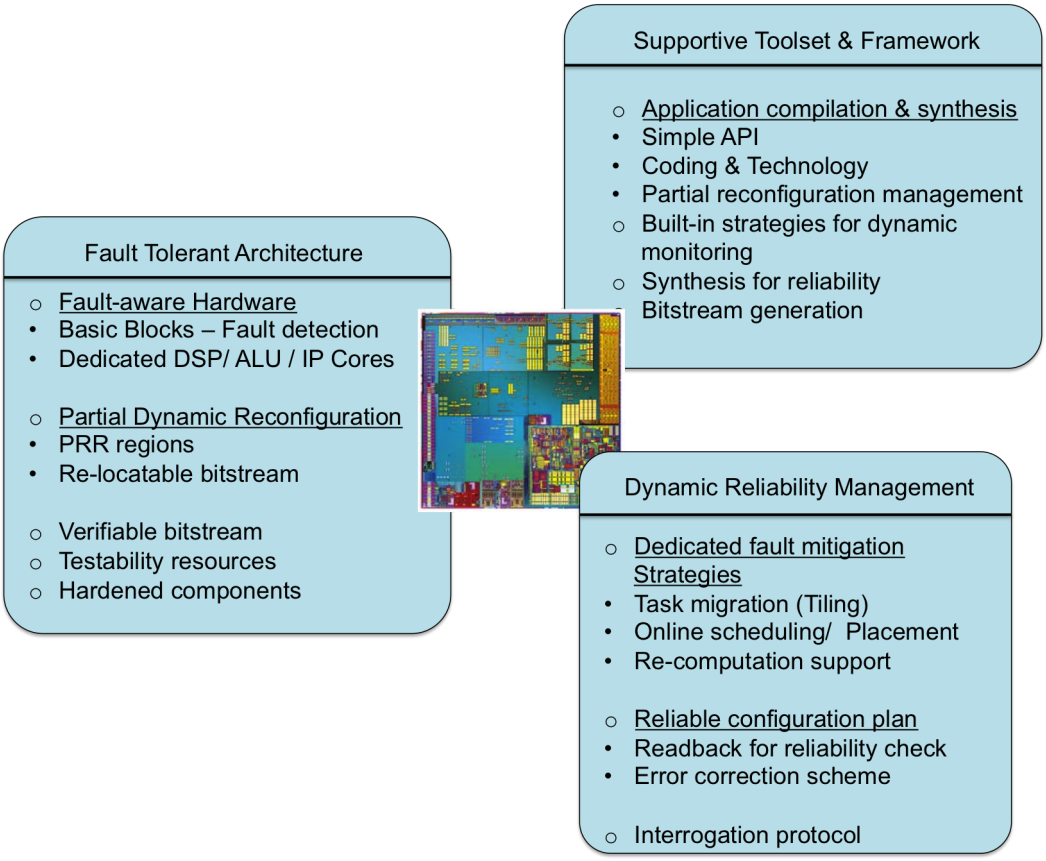


Figure 3.1 – Introducing the vision of the ARDyT architecture.

ways of making use of these intrinsic properties are also studied. This study has the following impact on architecture as well as its associated tools: on one hand, the dedicated architecture resources simplifies the fault-tolerance implementation, and on the other hand, it allows the tools to achieve synthesis for reliability.

The research work presented in this Thesis contributes to the definition of reliable hardware architecture and dynamic management of reliability mechanisms involved in the ARDyT project. It includes definition of a new fault-aware logic block, modified basic building block structures, customized configuration bitstream plan and strategies to mitigate the effects of various fault models through the centralized reliability manager. More details of these aspects will be presented in the following sections of this chapter.

3.1.1 Fault-tolerant Hardware Architecture

FPGA's hardware architecture can be defined with the classical two-layer system (computation and configuration layer). The computation layer includes all the logical resources required for functional computations. It contains basic building blocks of the hardware architecture comprising Configurable Logic Blocks (CLBs), which are interconnected via flexible channel of wires and switch matrices. This layer also includes dedicated resources such as memory elements, DSP blocks and/or hard-wired processor. The second layer has the configuration plan, which specializes the

computation layer for a particular application. For the design of a fault-tolerant architecture these two layers must be protected by more precise mechanisms. The goal is to define the architecture with basic built-in capabilities, which would enable dynamic management system to perform required operations to ensure reliability and thereby the overall system availability. It also includes proper structuring and efficient distribution of the bitstream. The configuration bitstream plan also has a great impact on the proposed error correcting mechanism to be adapted by the dynamic management system. The dynamic management system has significant impact on the definition of underlying fault-aware hardware architecture. Logic resources in the hardware architecture must be customized (or a new structure need to be proposed) in such a way that the underlying hardware architecture would comply with the strategies being carried out by the dynamic management system.

Homogeneous architecture has been assumed in order to promote the reallocation of tasks and mechanisms for supporting the primitive configuration manager (fast context switching, lock granularity of the configuration page etc.,), which is integrated into hardware. It requires the design of a specific block embedding primitives for reconfiguration process control and choice of an adapted recovery technique according to the detected fault. The structures of different building blocks are adapted in such a simplistic manner which would increase fault detection capabilities and diagnosis in the overall architecture. Dedicated resources for fault-detection, monitoring and fault-mitigation are included with a trade-off in efficiency/cost integration.

3.1.2 Supporting Tool-set and Framework

Proposing a new architecture alone, without any supporting computer aided design (CAD) tools, would be largely insufficient. Such a scenario would limit the impact of the architecture even for the specially targeted groups. Therefore, it is mandatory to set up a complete design framework allowing designers to program and use this architecture. The framework includes defining and developing a customized tool-set (synthesis, placement, and routing tools, etc.) for the targeted architecture. Then, re-targetable compilation flow techniques can be adapted in the design process by keeping some of the already available commercial architectures as a reference target. In addition to all the aforementioned capabilities, the tool-set has been enhanced to include in it the self-healing capabilities of the architecture. Also, special mechanisms are incorporated to implement reliability methods at compile time by supporting debug-friendly synthesis, which refers to the ability to inject some diagnosis mechanisms (such as probes, breakpoints etc.,) into the netlist.

Designing and developing a new FPGA architecture is a tedious, expensive and time-consuming task. Instead, the ARDyT approach relies on virtual prototyping—which means the development of hardware/software systems without using a real hardware prototype appears as an enabler for the domain space exploration. As a result, rather than investing time to set-up a real prototype, the efforts are focused on software development and model designs. These models exhibit agility and support fast re-factoring. Besides, reuse grows up, which contributes to cost reduction. On the other hand, despite virtual prototyping allows taking early sound decisions and cuts off development costs, it should also be associated with a second run that

refines the design decisions in order to reach a real prototype prior to manufacturing of the final product. Obviously, because this path may create a discontinuity risk, it stresses up the necessity of sizing down any manual development phase, to preserve the time-to-market gain. The proposed FPGA is intended to be fault-tolerant with no (or with minimum) support of radiation hardening. This requires the need for multilevel policies in order to guarantee self-healing capabilities: error detection through bitstream and application monitoring, development of dedicated fault-aware logic blocks and/or ALUs with specific encoding, bitstream relocation, etc. These features drive the whole design process. For instance, ensuring that bitstream relocation is possible but it has a strong impact on the target architecture. The virtual prototyping flow relies on the Biniou framework [96] that offers FPGA modeling capabilities along with the programming environment. Biniou is a framework that supports modeling of reconfigurable platforms. The platform designer has to describe the platform (structure, topology, logic elements, etc.) using a proprietary architecture description language (ADL), as illustrated by the code listed in Fig. 3.2. On return, the Biniou framework generates adapted and customized low level programming tools offering floor-planning, placement-routing, visualization, editing, and other functionality. The framework is being extended to support the new architectural patterns investigated in the ARDyT project.

```
(((ARRAY
   (DOMAIN  1 1 50 50) "END of DOMAIN"
    (((COMPOSITE "FA-CLB"
(((FUNCTION
    (INPUTS ((WIRE(WIDTH 5)) NAMED in))
    (OUTPUTS ((WIRE (WIDTH 3)) NAMED o)))
   NAMED f)
((WIRE (WIDTH (VALUE chW '30')) EXPANDED )
   NAMED south)
...
```

Figure 3.2 – Modeling code for a 50*50 FA-CLBs FPGA, with 5 inputs and 3 outputs logic elements, and 30-bit interconnection channels.

The ADL supports the uses of parameters and variables. The first benefit is the possibility to link values (e.g. interconnection channel width). In the code of Fig. 3.2, the variable CHW is set and can be further reused to tailor another architectural element. Another usage is to associate a variable to a list of possible values (similar to enumerated typing); hence, in this scheme, exploration can be automatically performed by itemizing the values. Figure 3.3 illustrates the global flow of the supporting tool-suite.

In the Biniou framework, a reconfigurable unit is described as two different parts. The modeling separates the resources specification (left side) from the configuration

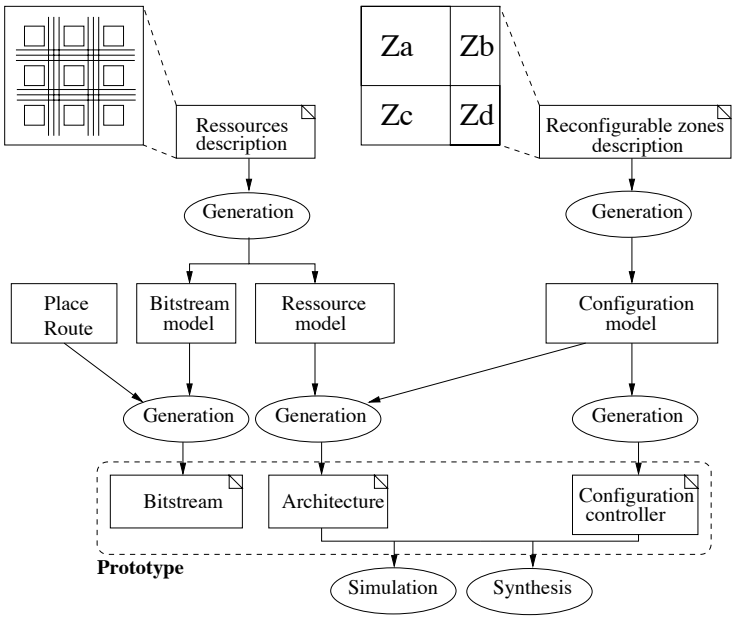


Figure 3.3 – Global flow supporting generation of a complete environment for reconfigurable unit prototyping.

plan specification (right side) in order to perform orthogonal DSE in each case. The resource model corresponds to the routing and computing resources, which are sized in terms of granularity, according to exploration results. The configuration model gives the structure of the configuration plan as a set of zones. Every zone is independently reconfigurable and supports multiple contexts implementations.

The programming flow starts with application partitioning. Applications are broken down into partitions based on the sizing of zones. As the circuit in a zone can be swapped in/out, a smart sizing prevents internal fragmentation, although at the cost of increased placement complexity. Every partition is then implemented (resources allocation) using place&route algorithms that explore dynamically the architecture model. From it, a collection of partial bitstream (binary representation) is issued. The bitstream model is automatically derived and is used for generating configurations from placed&routed applications. As a result, dynamic partial and multi-context reconfiguration modes can be explored to quantify their impact on the execution performance.

From a practical point of view, both models contribute to deriving back-end applicative tools, including a synthesizer that produces net-list, a placer-router in charge of resources allocation and a bitstream model with a generator that outputs the configuration file.

Prototype implementation relies on a generated VHDL design taken as the input of cycle-accurate simulators, such as Modelsim, or synthesized on the state-of-art FPGA. The configuration plan of the hardware prototype is connected to a specialized controller that supports both partial and multi-context reconfiguration modes.

3.1.3 Dynamic Reliability Management

The architecture and design tools incorporate advanced and optimal mechanisms to achieve high degree of flexibility and usage, and, more importantly, reliability of the design. The third axis of the project concentrates more deeply on the definition and the implementation of test methodologies and fault-tolerance approaches adapted to the proposed dynamic architecture, and making use of built-in properties provided in the architecture to support the reliability mechanisms to be carried out by the decision making bodies of the system. The dedicated and focused study on dynamic reliability management yields two results: i) an adaptive architecture that simplifies the implementation of fault-tolerant design and ii) a dedicated tool-set to achieve synthesis for design reliability.

There are various fault types that could occur in the design, which can create different set of unexpected and unwanted consequences. Depending on each targeted fault type, several strategies are developed and suitable decision making policies are included. Those dedicated fault mitigation strategies include task migration, online/dynamic scheduling, and re-execution support. To ensure reliable configuration bitstream, run-time read-back and verification techniques are adapted, as used in the conventional Xilinx's and Altera's configuration bitstream protection plans, with adequate changes in the error correction phase to handle the maximal number of MBUs. Apart from such issue specific fault mitigation strategies, ARDyT incorporates some system level strategies as well. All the reliability management strategies adapted in the targeted architecture are supported by a built-in introspection plan to have a bounded reliability monitoring.

To handle all the dynamic reliability management mechanisms in the architecture, the third layer is proposed (besides the hardware and configuration layers), called Fault Tolerant Abstraction Layer (FTAL). This layer accumulates all the resources for fault-tolerance and includes a dedicated processor to manage reliability (i.e. implementing the reliability policies) in the chip.

3.2 ARDyT Overall Architecture

Figure 3.4 shows the general scheme of the proposed architecture. Unlike the conventional FPGA architecture, which is usually defined by the computation and configuration layers, the proposed architecture is defined by the computational, configuration, and an additional layer called FTAL, dedicated for reliability management. In Figure 3.4, the configuration layer is not shown separately, as both computational and configuration layers are coupled together and named as "targeted hardware". The reason for representing it in such a manner is that in the proposed architecture, the hardware basic building blocks are viewed as collective units, which include logic circuit components and their associated configuration bits.

The dedicated layer for reliability management (FTAL) is shown in Figure 3.4 with its functional representations. The main novelty of the proposed architecture in the ARDyT framework is the definition of this new layer dedicated for the dynamic reliability management. This additional layer takes the functional responsibility of maintaining the required level of reliability in the underlying architecture. In other

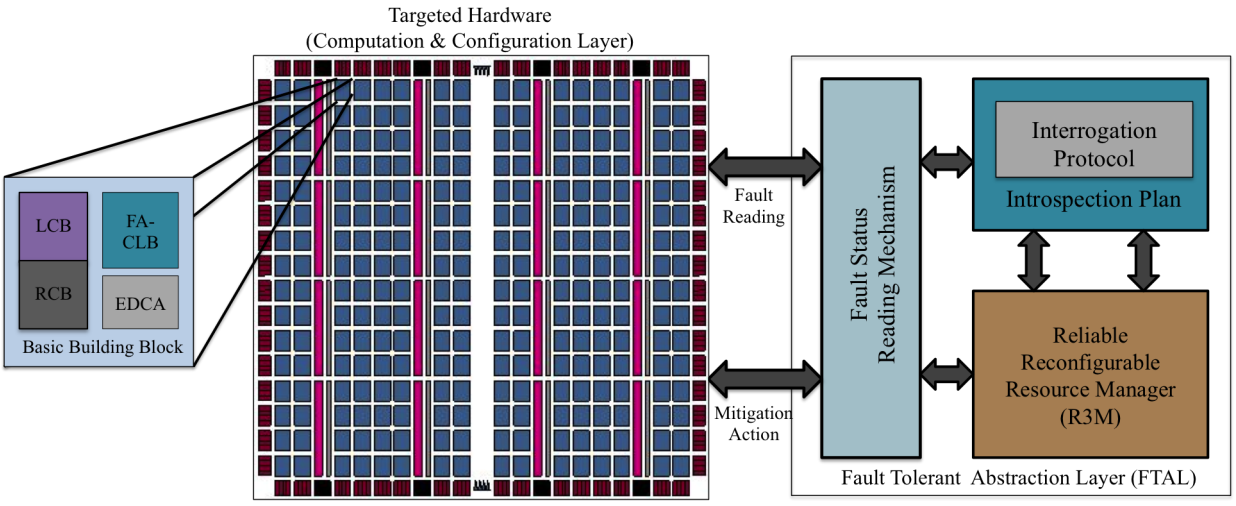


Figure 3.4 – Complete vision of the proposed ARDyT FPGA architecture.

words, the "fault reading" and "fault mitigation" are two principal operations that the FTAL layer carries upon the hardware architecture in coordination with the upper management control. The fault reading is performed by the interrogation protocol employed, as a part of the introspection plan. "Fault mitigation" refers to the set of strategies employed by the centralized reliability manager to deal with various fault types that occur in the underlying architecture. The centralized dynamic reliability manager is named as R3M — *Reliable Reconfigurable Resource Manager*. The three primary functional units of the FTAL: i) the fault status reading mechanism, ii) the introspection plan, and iii) the R3M, work closely with each other. As a whole, the proposed three-layer (computational, configuration, and FTAL) system represents a unified architectural model for highly reliable applications.

3.3 Hardware Architecture and Building Blocks

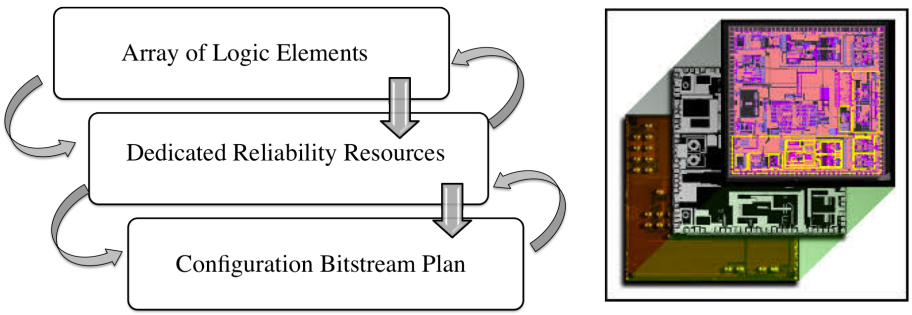


Figure 3.5 – Physical architecture integration.

This section describes the architectural fundamentals of the proposed hardware framework. In a consolidated view, the physical architecture of the proposed ARDyT FPGA is an appropriate proportional integration of three required key resources, as

shown in Figure 3.5. The array of logic elements is the primary resource to hardware architecture plan. Dedicated reliability resources represent additional hardware stock to support the reliability management mechanisms and configuration bitstream plan to attain the required functional and reliability management of the basic architecture.

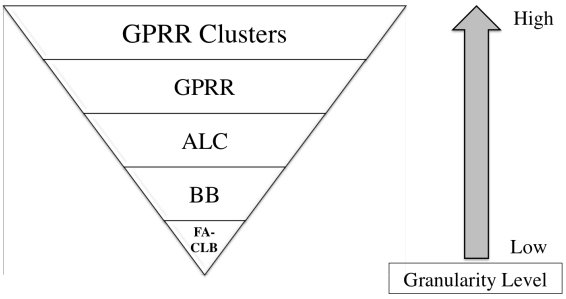


Figure 3.6 – Hardware architectural hierarchy.

The proposed architecture's hardware building block hierarchy based on its granularity (from the lowest to the highest level of granularity) is as follows:

- Computational (Configurable) logic block (FA-CLB)
- Basic building blocks (BB)
- Array of logic cells (ALC)
- Grouped partial reconfigurable region (GPRR)
- Clusters of grouped partial reconfigurable regions (monitoring and non-monitoring GPRR clusters)

A simplified conceptual view of the hardware architectural hierarchy is shown in Figure 3.6. Other dedicated hardware resources such as DSP blocks, ALUs, multipliers, dedicated IPs and supportive hardware units for reliability and testability are not shown in the simplified hierarchical diagram. These dedicated hardware resources are the intrinsic part of any one of these hierarchical levels. For instance, dedicated hardware functional resources like DSPs and ALUs, and dedicated reliability supportive register FSR are part of GPRRs. More detailed descriptions of hardware building blocks of the proposed architecture are given in the following sections.

3.3.1 Basic Building Blocks

In any typical FPGA architecture, the basic building blocks combine combinational and sequential circuit elements which are put together in a desired way, whereas the configuration bitstream is used to define the functionality of those basic building blocks. The most common basic building block of an FPGA is a Cell or a Slice. Typically, a slice has a few inputs, contains a Look-up Table (or LUT) which can

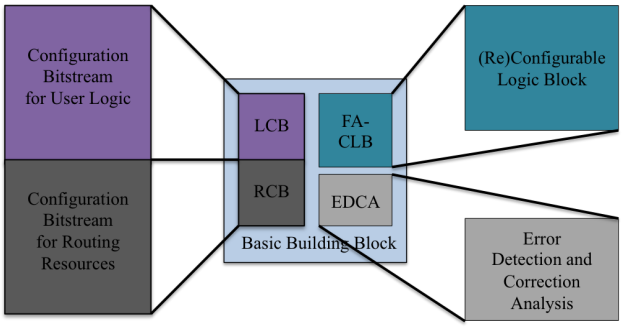


Figure 3.7 – Architecture of basic building block.

be programmed to realize any Boolean function over those inputs, and has one or more outputs, each of which can be configured to either update immediately when the input updates (asynchronous) or update only on the next clock tick, using a flip-flop built into the slice (synchronous). The architecture of basic building blocks in ARDyT FPGA is quite different from those used in conventional FPGAs (for example, CLB architecture of Xilinx or LAB architecture of Altera).

Unlike conventional FPGA architectures, the basic building blocks of the ARDyT FPGA are not just compositions of combinational (LUT/MUX) and sequential logic (D-FF) elements. The proposed ARDyT FPGA's basic building block, shown in Figure 3.7, contains three units: configurable logic elements, error detection and correction analysis block (EDCA) and dedicated (separately identifiable) bitstream for logic as well as routing resources of that particular basic building block.

- FA-CLB is used to implement a desired logic function (as in conventional architectures), but with a modified structure to support the adaptability. FA-CLBs of the proposed ARDyT architecture can be compared to CLBs of Xilinx architectures. A CLB in Xilinx FPGAs has several slices in it. For instance, in Xilinx Virtex 7 devices, a CLB has 2 slices named *Slice(0) and Slice(1)*. Similarly, the FA-CLB of the ARDyT architecture consists of four configurable logic elements (CLE). The detailed description will be given in the following section.

- EDCAs accompany CLEs to implement the reliability management mechanisms, provided by the centralized reliability manager.

- The dedicated building block specific bitstream identification facilitates fault localization and fault separation between logic and routing resources.

3.3.1.1 Fault-aware Configurable Logic Block (FA-CLB)

As a primary module of the basic building block, the FA-CLB's structure and its associated design parameters play a vital role in providing performance advantages of the proposed architecture. Since reliability is of primary importance in the newly proposed architecture, the FA-CLB fabric is customized to support reliability management mechanisms along with the required functionality implementations. Figure

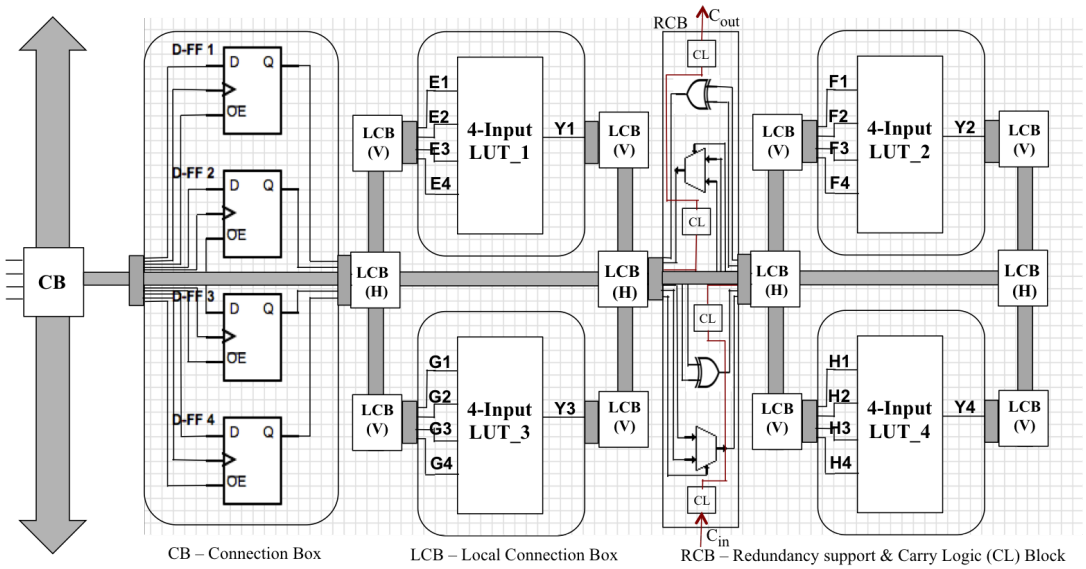


Figure 3.8 – Fault-aware Configurable logic block (FA-CLB).

3.8 shows the internal structure of the proposed FA-CLB in the ARDyT architecture. It comprises functional a equivalent of four CLEs (i.e., LUT-D-FF pairs and additional circuits like selection multiplexers, comparator circuitry, internal glue logic and carry chain logic), arranged as shown in Figure 3.8.

The conventional FPGA's logic cell structure (usually referred as the classic logic cell structure) has combinational and sequential circuit elements in the same hardware closet (a LUT-FF pair). However, in the proposed FA-CLB structure we prefer to separate combinational and sequential circuit element(s). Such arrangement helps in achieving flexible reliability mechanism implementation in the proposed architecture. Each of sequential circuit elements (D-FF) arranged in a column, in the first local zone of the FA-CLB associates itself with the corresponding combinational circuit element (LUT) in the following local zones of the same FA-CLB. The $D-FF_1$ is associated with LUT_1 to form CLE_1. Similarly, $D-FF_2$, $D-FF_3$ and $D-FF_4$ are respectively associated with LUT_2, LUT_3, and LUT_4 to form CLE_2, CLE_3 and CLE_4. The D-FF is placed in such a way that it could be used for two purposes:

1. to hold the input value of the LUT to reinforce and re-execute the task (in-case of fault identification), and

2. to hold the computed data output to assist in logic pipe-lining and data storage.

The logical advantages, detailed functional modes, and mechanisms supporting reliability of the proposed FA-CLB architecture are further elaborated in Chapter 4.

In Figure 3.8, it is seen that a set of sequential circuit elements (D-FFs) are arranged in a column manner in the first local zone and followed by the same number of local partitions containing combinational circuit elements (LUTs), and separated

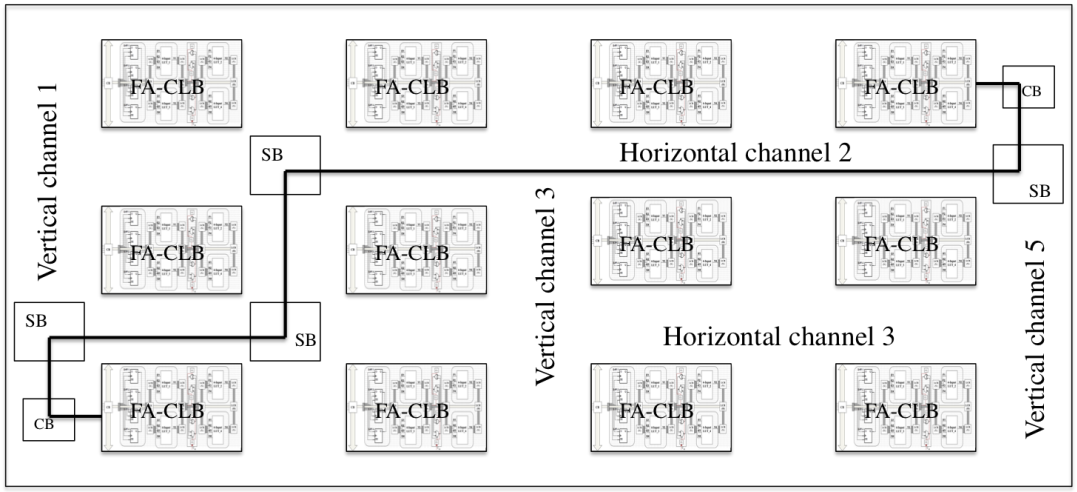


Figure 3.9 – Interconnections among FA-CLBs.

by a rail of local connection lines. The local connection box (L.C.B) helps in making appropriate internal connections between the sequential and combinational circuit elements, to form the LUT-FF pair as in the conventional logic cell structure. Apart from local connection boxes, there are programmable connection boxes (C.Bs) and switch boxes (S.Bs) associated to the design implementation of the architecture. The interconnection system of any FPGA is one of the most complex blocks, because wiring is a global property of a logic design. Connections between logic elements may require complex paths, since the FA-CLBs are arranged in a sort of two-dimensional structure, as shown in Figure 3.9. Hence, it is necessary to make connections not just between FA-CLBs and wires but also between the wires themselves. Wires are typically organized in wiring channels or routing channels that run horizontally and vertically through the chip. Each channel contains several wires; the designer or a program chooses which wire will be used in each channel to carry a signal to make a design connection. Configurable connection boxes (C.Bs) and switch boxes (S.Bs) serve to this purpose. CLEs are connected to the routing network through connection boxes (CB). Horizontal and vertical routing tracks of the programmable routing network are interconnected through switch boxes (SB) which handle the connections through various wire structures, like segmented wiring and offset segments, according to the routing structure topology. Connections must be made between wires in order to carry a signal from one point to another. For example, the net in the Figure 3.9 starts from the output of the FA-CLB in the upper-right-hand corner, travels down the vertical channel 5 until it reaches the horizontal channel 2, then moves down the vertical channel 2 to the horizontal channel 3. Finally, it uses the vertical channel 1 to reach the input of the CLE at the lower-left-hand corner.

3.3.1.2 Built-in Error Detection and Correction Analysis (EDCA)

The built-in error correction and detection analysis (EDCA) unit is a conceptual design block introduced as a part of the basic building block. The EDCA block holds collective responsibility of handling reliability related issues in the hardware

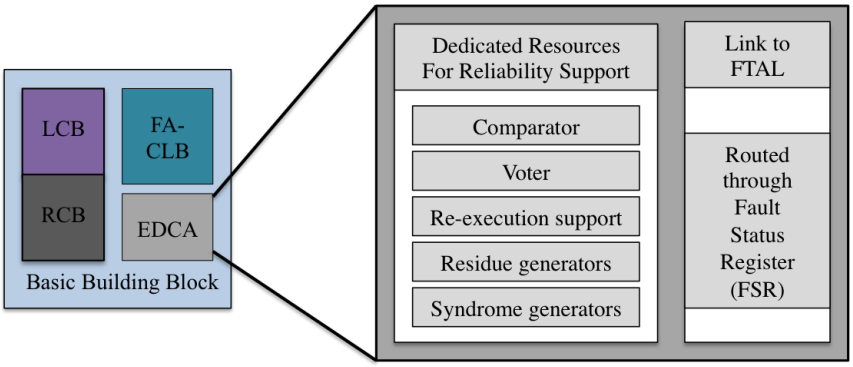


Figure 3.10 – Error detection and correction analysis (EDAC) unit.

functional units. Its detailed internal structure is shown in Figure 3.10. To note that the EDCA block is actually not as isolated from the configurable circuit resources as suggests Figure 3.10. Either a part or the whole EDCA unit makes part of the architecture of the fault-aware configurable hardware (FA-CLB, DSP, multipliers, etc.). It enables the logic circuit to detect an error in the presence of a fault in the circuit. The content of the EDCA varies according to:

1. the architecture of the functional hardware (FA-CLB, DSP, ALU, multiplexer, etc.) and
2. the reliability mechanism proposed to handle faults occurring in that particular circuit element.

EDCA is closely associated to FA-CLB of that particular basic build block in all aspects. To ensure an uniform, homogeneous structure of the basic building blocks, the content of the FA-CLB is as proposed in Figure 3.8. Hence, the EDCA content would also be the same and uniform. However, when the EDCA is associated to other dedicated hardware functional units such as DSPs, ALUs, multipliers and other hard IPs, its content is adapted, according to the error handling mechanisms proposed to those dedicated hardware functional units. The dedicated fault handling resource in EDCA would be:

- a simple **comparator**, in case of the duplication with comparison (DWC) scheme;
- a **majority voter**, in case of the triple modular redundancy (TMR) scheme;
- a **residue generator**, in case of arithmetic operations;
- a **syndrome generator**, in case of error correction codes (ECC); or
- any other dedicated approach according to the selected fault mitigation strategy.

The choice of the EDCA content also depends on the reliability level required through the adapted fault handling mechanism. In the minimal case, a simple comparator is used as the dedicated EDCA resource, just to detect the fault by comparing two identical circuits and activating a mismatch signal.

Besides that, the EDCA must also provide the 'link to FTAL'. As mentioned earlier, fault-tolerance abstraction layer is an additional layer meant for centralized reliability management in the proposed architecture. The reliability management by FTAL is handled through various means. However, the initial step is to getting notified about the fault occurrence status. The communication must be established from the functional units of the underlying architecture, so that the notification could reach the FTAL. To mark the beginning and to facilitate this feature, the EDCA unit of basic building blocks has a dedicated resource to notify the fault occurrence status. In ARDyT architecture, according to the proposed hardware hierarchy, GPRR is a reconfigurable partition comprising a group basic building blocks and dedicated functional units. Each GPRR has a dedicated register called Fault Status Register (FSR) to store the fault occurrence status from various hardware blocks and to link the fault-aware hardware blocks to the FTAL. From the FSR, the fault occurrence status is communicated to the centralized reliability manager (R3M) in the FTAL. Updating fault occurrence status in the FSR is a hierarchical process. The status signals from various functional units/EDCAs are encoded to carry only the minimum of information required, and recoverable amount of status signals of that GPRR are updated in the FSR.

3.3.1.3 Dedicated Identification of Logic and Routing Bitstream

The ARDyT architecture has a feature of identifying configuration bitstream associated to a particular basic building block. Additionally, it is capable of distinguishing the configuration bitstream corresponding to logic and routing resources:

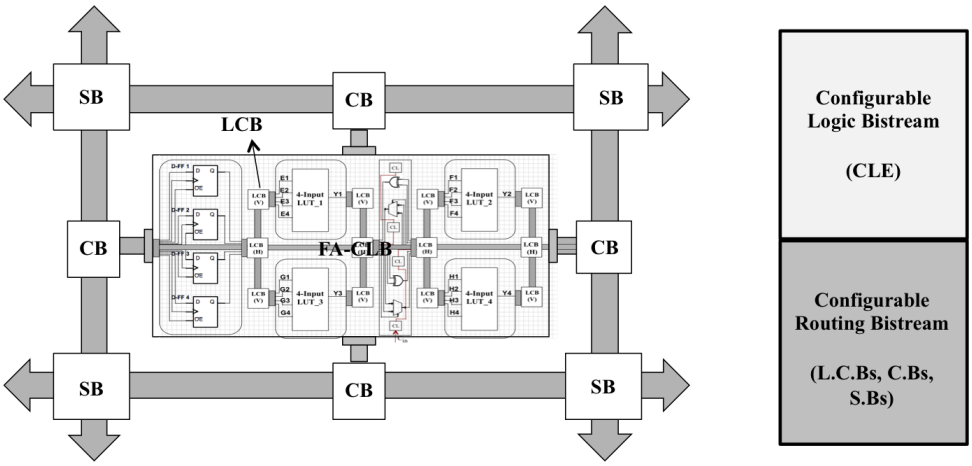


Figure 3.11 – Fault-aware configurable logic block (FA-CLB) with associated routing elements.

- **Logic configuration bitstream:** a group of configuration bits used to implement logic design in FA-CLBs.

- **Routing configuration bitstream:** a group of configuration bits associated with the implementation of routing structures/patterns in the FPGA architecture. The routing configuration bitstream involves the configurable bits in the horizontal as well as vertical local connection boxes (LCB), connection boxes (CB) and switch boxes (SB) associated to a FA-CLB.

Identifying a configuration bitstream belongs to a particular basic block. The possibility to distinguish the bitstream of implemented logic functionality and implemented routing/switching associated with that implemented functionality significantly helps in detecting and localizing faults at a required granularity level. This is because upsets in the bits of 'configurable logic' create different consequences than upsets in the bits of 'configurable routing'. Distinguishing them greatly helps to identify the nature and consequence of the fault.

For instance, consider Figure 3.11, where a simplified view of one FA-CLB along with its associated routing elements are shown. The LCBs are those which make local connections internal to the FA-CLB. CBs connect the FA-CLBs to the external routing network and SBs are used to make further horizontal and vertical cross connections between FA-CLBs and/or other elements of the architecture. Programmable bits in LCBs, CBs and SBs are identified as *configurable routing bitstream*, whereas other programmable bits in FA-CLB are identified as *configurable logic bitstream*.

3.3.2 Grouped Partial Reconfigurable Regions

A grouped partial reconfigurable region (GPRR) is a group of: (i) fault-aware basic computational elements (FA-CLBs), (ii) dedicated functional hardware blocks such as DSPs, ALUs, multipliers and other necessary IP cores, and (iii) dedicated fault status register (FSR).

Array of Logic Cells (ALC): Fault-aware configurable logic blocks assembled in an array format, in two-dimensional plane, with horizontal and vertical routing channels running across, with provision to make connections among them, is called array of logic cells (island-style). A little abstracted view of array of logic cells (ALC) is shown in Figure 3.12, which depicts the arrangements of connection boxes (CBs) and switch boxes (SBs) along the routing lines of the architecture. The internal functional structures of both CB and SB remain the same as those used in conventional FPGA architectures, such as shown in Figure 3.13.

- **Connection Box (CB):** the CBs connect the channel wires with the input and output pins of the FA-CLBs. A CB has two major properties that can affect the routability of a design: its flexibility, F_c, which is the number of wires that each logic block pin can connect to; and its topology, which is the pattern of switches that make the connection (especially if F_c is low).

- **Switch boxes (SB):** the SBs allow wires to switch between vertical and horizontal wires. The flexibility of the SB, F_s, defines for a wiring segment

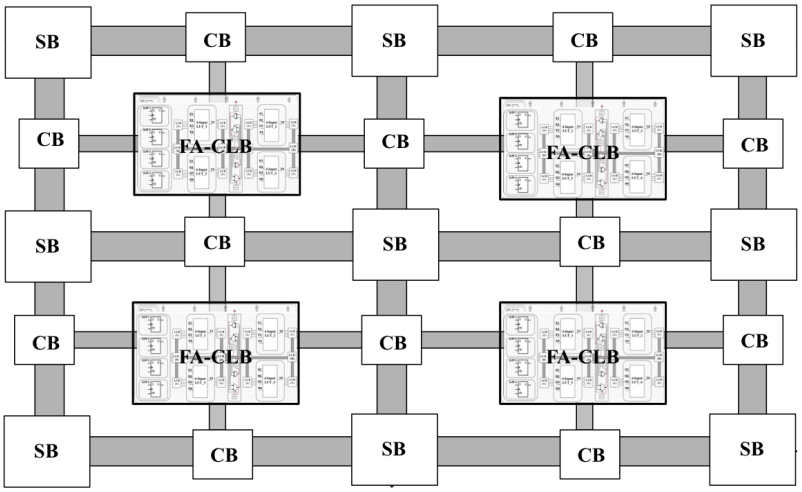


Figure 3.12 – Array of logic cells.

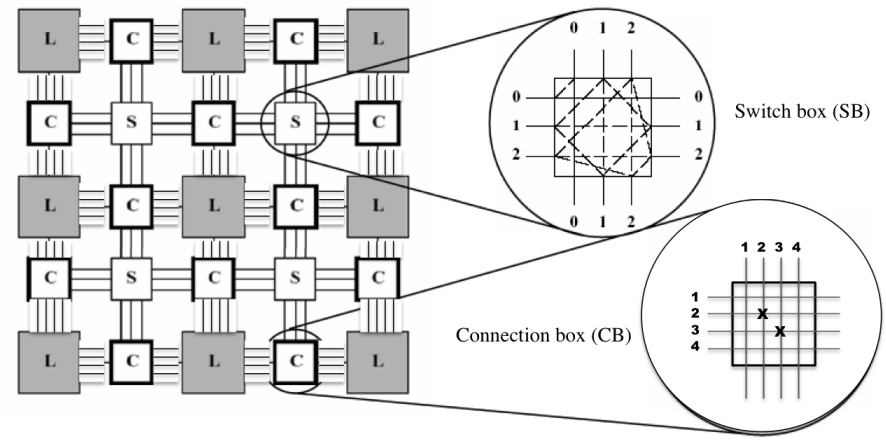


Figure 3.13 – Connection box (CB) and switch box (SB) structure.

entering the SB the number of other wiring segments it can be connected to. The topology of the SBs is very important, since it is possible to choose two different topologies with the same flexibility F_s that result in very different routability.

The CB is implemented with pass gates rather than multiplexers for input connections. This allows two or more tracks to be electrically connected via the input pin by turning on individual switches in the connection box. This is called input pin doglegs [98]. The significance of this scenario will be explained in Chapter 4, where the proposed FA-CLB architecture is presented. Each wiring segment spans only one block before it terminates in a switch box. By enabling some of the programmable switches within a switch box, longer paths can be constructed. Whenever vertical and horizontal routing channels intersect, there is a switch box.

GPRR is a dynamic reconfigurable partition (DRP) in the architecture. This hardware partition can be reprogrammed during the application run-time without

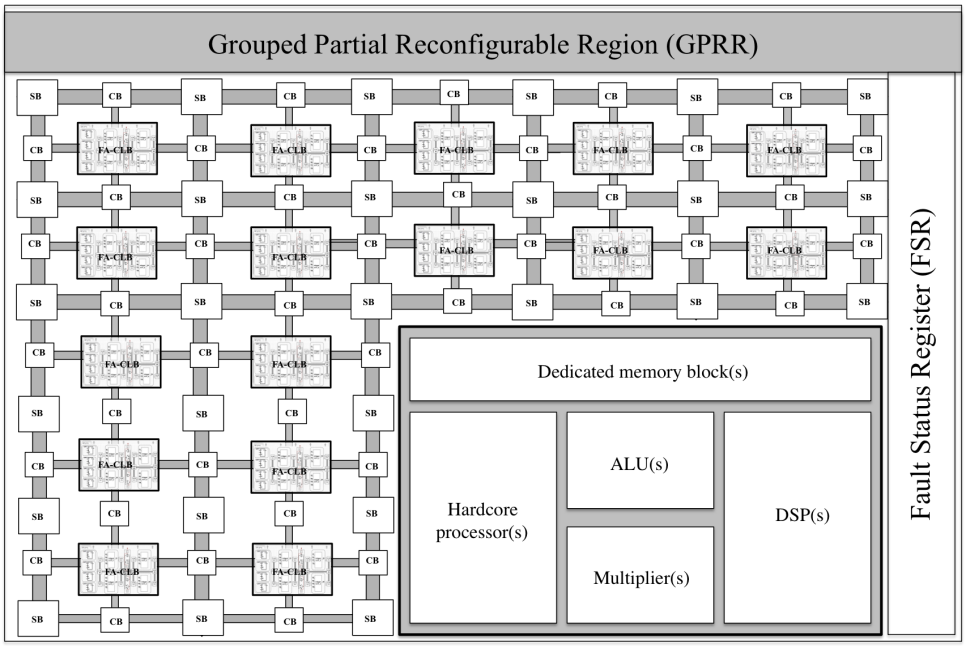


Figure 3.14 – Grouped partial reconfigurable regions (GPRR).

halting the system. The size of the GPRR is determined by various factors, including: the required number of basic building blocks, additional dedicated functional resources, and the granularity of fault status reading. The frame is the smallest unit of (re)configuration in any FPGA architecture. The number of frames required to form a dynamic reconfigurable partition in the architecture could be a decisive factor, in accordance with the size of the GPRR. Also the GPRR content size, its internal versatility and the associated reliability support mechanism, play an important role in defining the fault localizing granularity level in the hierarchical reliability management scheme. Moreover, the size of the GPRR (the number of configuration frames/bits involved in a single GPRR) must be a factor of the size of configuration bits handled by the proposed configuration bitstream protection plan to achieve optimized end results.

FSR: This register is an integral part of the GPRR to support the fault status reading process through the interrogation protocol. It establishes communication between the fault detection and the fault mitigation resources. All the fault-aware functional blocks in the GPRR update their health status (fault status) in the corresponding bits of the FSR. An unique set of configuration bits (partial bitstream) for each defined GPRR is generated during the process of configuration bitstream generation by the CAD tools. The health status of the corresponding partial bitstream of each GPRR is updated in the FSR. As a whole, the FSR holds the health status of the complete GPRR. The overall health status bit (FSR[8]) informs (through interrogation protocol) the R3M about the uncertainty in the corresponding GPRR's functionality.

3.3.2.1 Dedicated Functional Units

Nowadays, to address the challenges of large scale safety and mission critical applications, FPGA devices integrate more and more dedicated advanced functional units such as DSPs, ALUs, multipliers and hardcore processors. Most of the present day applications demand the inclusion of microprocessors, DSPs and SRAM memory resources, which are highly optimized structures, so they are radically more efficient for computations that fit within their target model than any LUT-based implementation. The proposed ARDyT architecture aims to have an inclusive architecture, to fulfill most of the requirements demanded by a self-reliable, high density, large-scale applications.

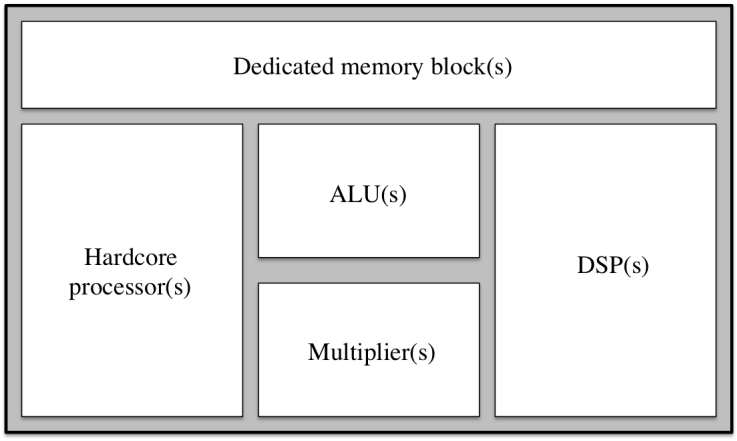


Figure 3.15 – Dedicated functional units integrated in the FPGA device.

As discussed before, any functional unit of the proposed architecture has a dedicated EDCA unit which provides reliability management support by various means. Hence, it makes the dedicated functional blocks such as DSP, ALU and built-in processors to be fault-aware—a feature that ARDyT FPGA proposes. The fault detection (or any other reliability management support) circuitry provided by the EDCA unit depends on the dedicated functional unit with which it is associated. For instance, the DWC scheme is used make to the proposed FA-CLBs fault-aware, hence a comparator circuitry is added to the EDCA unit; residue code generators are used to handle faults in the DSP and other arithmetic blocks, therefore, the residue code generators make part of the EDCA unit associated with any such a block. At the moment, not all the resources shown in Figure 3.15 are expected to be included in the final ARDyT architecture. As of now, it is aimed to included minimum required dedicated functional units (such as DSPs). The fault-aware DSPs are being developed by ARDyT consortium partners at IRISA laboratory (Lannion).

3.3.2.2 Fault Status Register (FSR)

As shown in Figure 3.14, the fault status register (FSR) is an integral part of the GPRR. The FSR plays an important role in the introspection plan, which is crucial to support the reliability management mechanisms included in the underlying architecture. The detailed bit-wise organization of the FSR is shown in Figure 3.16. To

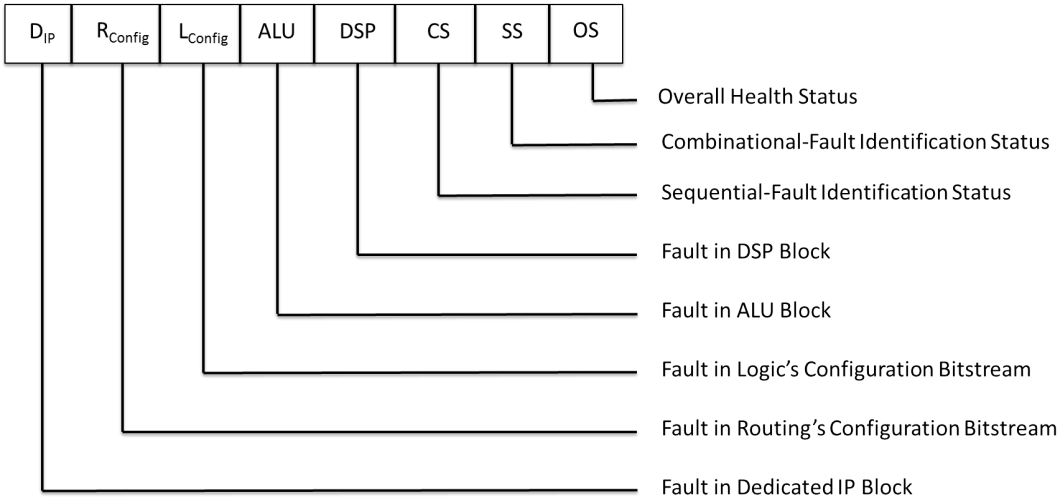


Figure 3.16 – Bit definition of the fault status register (FSR).

update their 'health status' (i.e., fault detection status), various fault-aware functional elements, present the corresponding bits of the FSR to the GPRR. Different level of encoding is followed for different hardware functional units, to update the *fault detection status* signal in the corresponding register (FSR) bit. For instant, the fault detection status signals from all the FA-CLBs (considered 16 FA-CLBs as in Figure 3.14) in GPRR are encoded into 2 bits to be updated in CS and SS bit of the FSR.

- *OS* bit indicates the overall health status of a particular GPRR. This indicates a condition of uncertainty in the GPRR. *OS* bit is updated, whenever there is a change in the remaining bits of the FSR. Additional deeper interrogation has to be done, once the *OS* bit is found asserted. It is possible through the interrogation protocol which can access these registers with the help of the ICAP.

Table 3.1 – Fault occurrence status and interpretation in FA-CLBs

State	SS (FSR[1])	CS (FSR[2])	Interpretation
S0	0	0	Fault-free FA-CLBs
S1	0	1	Fault occurred in combinational logic resources (e.g., transient effect)
S2	1	0	Fault occurred in sequential logic resources (e.g., user data upset)
S3	1	1	S3(i) - Recoverable fault in FA-CLB
			S3(ii) - Non-recoverable fault in FA-CLB

- *SS* and *CS* bits are associated with the FA-CLBs present in the GPRR. The encoded combinational and sequential fault status signals are updated

in FSR[2] and FSR[1] bit locations, respectively. These two status bits support the fault mitigation process developed for the proposed FA-CLBs. Table 3.1 shows four possible combinations of these status bits and their interpretations. Accordingly, the action needed to be taken by the centralized reliability manager is determined.

- FSR[3] bit indicates the faulty status of the dedicated DSP units present in the GPRR. The fault-aware DSP block, employed with built-in error detection capability, raises a warning flag whenever it detects a faulty output. That flag signal is updated in the FSR[3], in case of a single dedicated DSP block; otherwise, warning flag signals from more than one dedicated DSP block undergoes an encoding process.

- FSR[4] bit indicates the fault occurrence status of dedicated ALU present in the GPRR. The fault-aware ALU block, adapted with built-in fault detection operands, produces a mismatch signal to indicate a faulty operation in it.

- FSR[5] bit indicates the error occurrence in the configuration bitstream which belongs to logic resources present in the GPRR (FA-CLBs, EDCA circuits, dedicated DSPs, ALUs and other dedicated hardware units).

- FSR[6] bit indicates the error occurrence in the configuration bitstream which belongs to routing resources present in the GPRR (local connection boxes, switch matrices and connection boxed, horizontal and vertical routing channels, etc.).

- FSR[7] bit indicates the fault occurrence in dedicated IP blocks such as a hardcore processors. Fault identification of such hardcore processors involves schemes such as lock-step to identify the faulty behavior.

These FSRs serve as a linking medium between FTAL and the hardware blocks. Each of them notifies the status of the hardware blocks to R3M through the status bits. The FSRs are read with the help of the ICAP handled by the interrogation protocol. The details of fault reading and interrogation protocol will be presented in Section 3.4.2.

3.3.2.3 Cluster Based Health Monitoring

Granularity of fault detection in the architecture is one of the most important aspects of fault mitigation. Cluster based grouping is suggested in the proposed ARDyT architecture to have error notifications communicated to the decision making module, as shown in Figure 3.17. A number of GPRR units are grouped together to form a cluster, called *monitoring GPRR cluster*. Likewise, there can be more than one *monitoring GPRR cluster* in the architecture. Such clustering is helpful, when a sensitive process requires more than one GPRR unit to get its design implemented/mapped in the hardware. In the context of reliability management, GPRR clusters are on top of the granularity based reliability management hierarchy, as shown in Figure 3.6. When a sensitive process is implemented in such GPRR clusters, the aftermath

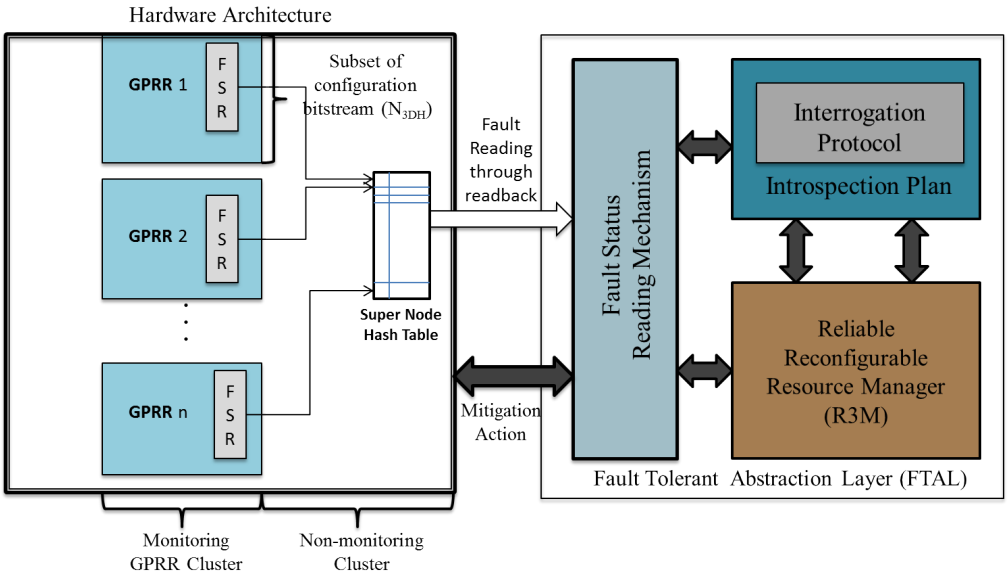


Figure 3.17 – Cluster based health monitoring.

process of fault mitigation becomes relatively easy. For example, when a process is divided among different GPRR units of a GPRR clusters and if one particular GPRR unit detects a fault and overcomes (resolves) it some time, the task (a part of the process implemented in the full GPRR cluster) implemented in that particular GPRR unit has to be synchronized with its fellow (associated) tasks running in other GPRR units. This aftermath process of fault mitigation (i.e., task synchronization) would be simpler when all the associated GPRRs are in an identifiable group/cluster.

Monitoring and Non-monitoring Clusters: The designers working on reliable system-on-chip architectures argued that it is not always necessary to make **the complete architecture** highly reliable. This is because if we breakdown the functionality of a critical application into various tasks, not all of them need to be monitored throughout their lifetime. Some tasks are real-time critical which requires complete monitoring, whereas some are less critical and non-real-time. As for reliability, the tasks can be classified on those with very high reliability requirements to those with zero (or negligibly small) reliability requirements. In such cases, a trade-off can be maintained by applying selective reliability measures in the architecture itself. The categorization of monitoring and non-monitoring clusters in the proposed reliable architecture is one such a trade-off measure with the flexibility provided to the designer, to place their application in the target architecture with a balance between resource usage, reliability requirement level and design overhead. Sensitive tasks can be implemented in any of the *monitoring GPRR clusters* having built-in support for the reliability management and non-sensitive tasks can find their place in the *non-monitoring clusters*. Partial reconfiguration process in the architecture is supported by R3M and 'monitoring and non-monitoring clusters' can be dynamically managed according to the functional requirement.

3.4 Dedicated Layer for Reliability Management

The discussion of achieving required reliability level for any critical application implemented in FPGA always comes with a counterargument of the cost of extra overhead, complexity and degraded overall performance. This is because commercially available COTS FPGAs have no (or very limited) built-in support to cope with reliability mechanisms developed by the application designer. Nevertheless, some FPGA manufacturers targeted high reliable applications market by providing some reliability support tools like *functional triple modular redundancy (FTMR)* [97] and *Xilinx triple modular redundancy (XTMR)* [54]. However, TMR'ed design alone could be insufficient in many critical applications, where different fault mitigation strategies have to be adapted, according to the nature and consequences of the fault occurrence. The newly proposed ARDyT architecture provides reliability support at various levels, including hardware architecture level, configuration bitstream plan, task/process level, application and system level, as shown in Figure 3.18. In addition to this distributed reliability support at various levels, the newly proposed architecture has a dedicated layer called FTAL, which is responsible for the overall reliability management.

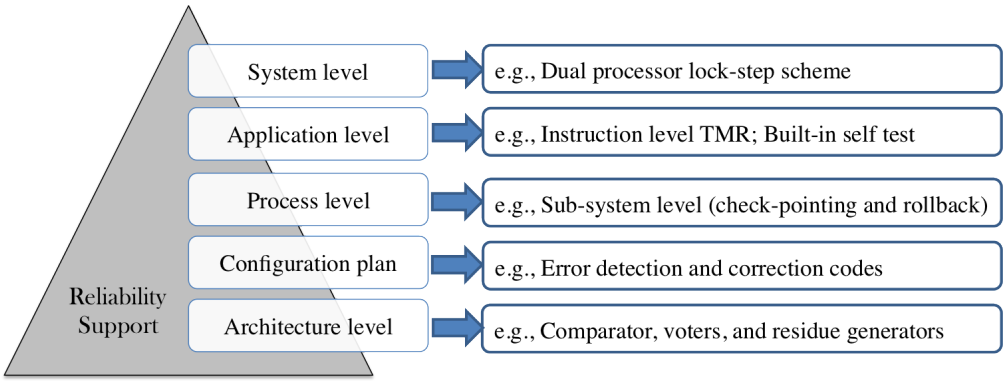


Figure 3.18 – Reliability support at different levels.

This additional layer (FTAL) was introduced to protect hardware architecture, configuration layer, and the implemented application. The FTAL coordinates the supportive resources (for fault management) present at various levels of the design. Proposing such a dedicated layer with a single large responsibility, called *reliability management*, is addressing the main challenge posed by the COTS FPGAs, when they are used for highly reliable and safe mission applications.

3.4.1 Fault Tolerant Abstraction Layer (FTAL) as Middle-ware

This fault-tolerant abstraction layer serves as a middle-ware between the hardware architecture, application and the software tools. The introspection plans provides interface between the hardware architecture and R3M with the help of ICAP and dedicated Application Programming Interfaces (APIs) for fault status reading.

The proposed layer decouples the underlying architecture from the applications and tools framework. Such a layer set-up facilitates generation of efficient tools using the properties described in the FTAL, whereas the architecture includes dedicated resources to efficiently support the desired properties. This dedicated layer supports the newly developed fault-tolerant strategies by defining correct API of methodologies. The tool set uses those APIs to implement specified algorithms at the compilation level for the usage on the underlying architecture. Fault-tolerant abstraction layer is a virtual component of the overall design flow which serves as a specification layer for the architecture and an API for the dedicated tools.

This additional layer serves to protect the resources in the configuration layer, the hardware layer and the application implemented in the architecture. Because not all the resources of the FPGA must be protected, a trade-off between the "hardware cost, timing overhead" and "level of reliability provided" can be maintained. This is where the concept of *monitoring and non-monitoring clusters* plays its role. The FTAL manages the reliability mechanism integrated in the *monitoring clusters* and coordinates *monitoring and non-monitoring clusters* in case of task of relocation and synchronization. The proposed hardware architecture has functional logic implemented with fault identification capabilities. The identified faults from various functional resources in the architecture are routed to the centralized controller (the reliability manager – R3M) through the *fault reading* process, involving FSRs. The hardware architecture (configurable resources + configuration bitstream) is dynamic and being supervised by the dynamic fault-tolerant layer. The centralized reliability manager in the FTAL performs decision-making and strategic planning for the appropriate fault-mitigation, whereas the upper layer provides supportive tools to keep control on the implemented fault-mitigation strategy through the reliability manager (R3M). In a broader context, fault-mitigation starts with the fault localization and faults are localized to their possible regional locality in the architecture.

3.4.2 Introspection Plan and Interrogation Protocol

The first step towards fault mitigation is to detect the presence of a fault (called fault reading). Receiving as much as relevant knowledge about the event that has occurred (upset/fault/error) helps in executing appropriate fault mitigation strategy. In the proposed ARDyT architecture, the reliable reconfigurable resource manager module (R3M) in the fault-tolerant abstraction layer (FTAL) is considered as the decision-making authority. It employs an introspection plan to monitor the reliability status of different building blocks in the architecture. The introspection plan runs an interrogation protocol to facilitate the *fault reading process*. Here, the term *fault-reading* denotes getting knowledge about the fault occurrence in the fault-aware hardware/application modules. The nature of the *fault-reading* in the ARDyT framework depends on various factors, like the granularity of fault identification, the mode of fault representation and timing and hardware overhead to be involved in the employed *fault-reading* technique.

The underlying architecture is provided with mechanisms to support the introspection plan. A simplified block diagram of the *fault-reading* process through the interrogation protocol is shown in Figure 3.19. It is an hierarchical process. As

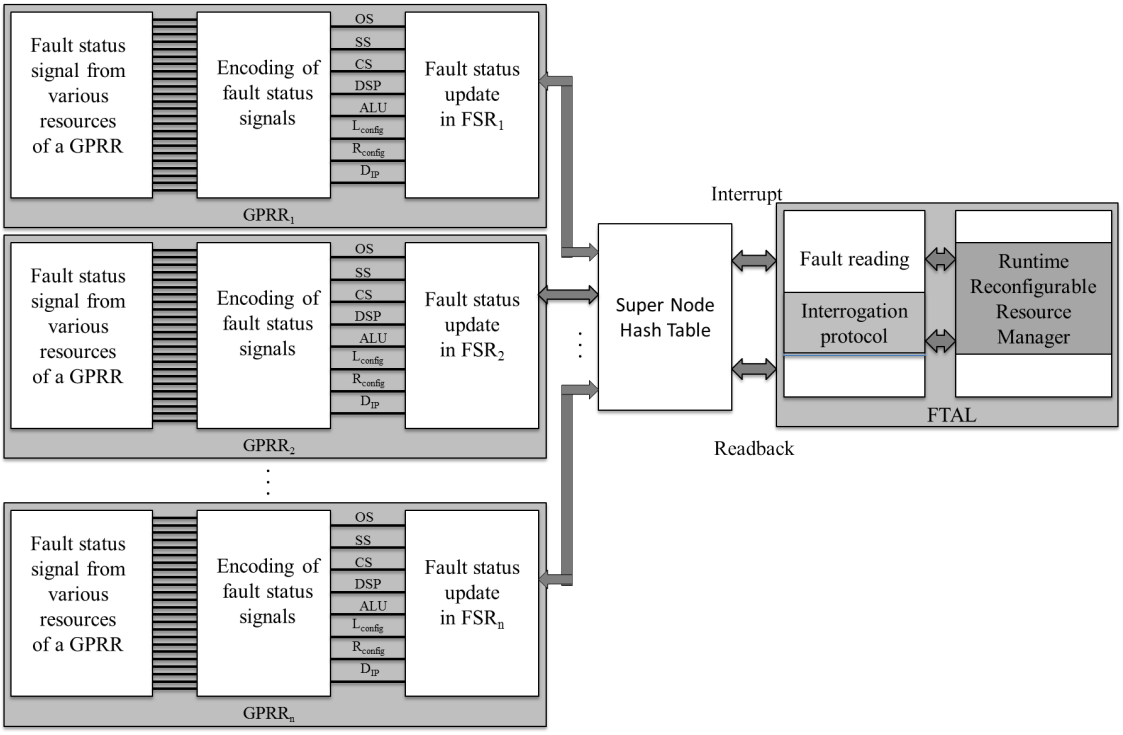


Figure 3.19 – Introspection plan.

explained before, each GPRR has a fault status register where the reliability status bits of different functional modules are updated. In the next level of the hierarchy, the overall health status bits (OS) of FSRs are mirrored to the *super node hash table*. The mirrored status bits stored in the *super node hash table* are called 'keys'. Each 'key' in the *super node* is associated with the corresponding address of the FSR in the *hash table*. Figure 3.20 shows the general structure of the *super node hash table*. For instance, assuming that a monitoring cluster has 16 GPRRs, the *super node hash table* length would be 16, with 4 address bits representing each FSR, ranging from '0000' to '1111'.

Key	Address
OS_{FSR1}	$Addr_{FSR1}$
OS_{FSR2}	$Addr_{FSR2}$
OS_{FSR3}	$Addr_{FSR3}$
⋮	⋮
OS_{FSRn}	$Addr_{FSRn}$

Figure 3.20 – Super node hash table.

The *super node hash table* interacts directly with the interrogation protocol.

Whenever there is a change in the key values of the super node, an interrupt is generated. The interrogation protocol reads the corresponding FSR address of the interrupted key from the hash table. Should it identify an uncertainty (a fault occurred) in the associated GPRR, it gains the access of the ICAP from the centralized manager and reads that particular FSR register. Because the single bit health status (OS) does not provide any more information about the faults and events which have occurred in the GPRR, hence the introspection plan proceeds with deeper interrogation. It checks all the remaining bits of a given FSR (FSR[1] to FSR[7]) to localize the fault to the specific functional block of the GPRR. At this level, the interrogation protocol notifies the fault occurrence to the R3M with relevant information such as the exact functional element in which a fault is detected (with the help of the fault status bit values registered in FSR). From here, R3M initiates the process of fault mitigation, starting from identifying the nature of the fault occurred.

3.4.3 Run-time Reconfigurable Resource Manager (R3M)

Decision making, reconfigurable resource management and reliability maintenance in the architecture is completely handled by this centralized intelligence, R3M. It monitors the hardware architecture and running application as well as provides run-time support. The functionality of R3M includes:

- Monitoring the hardware architecture and the implemented tasks for fault-free operation, with the help of different APIs and the introspection plan.
- Mechanisms to support the proposed configuration bitstream protection scheme, fault detection and mitigation in configurable logic resources, and reliability mechanisms in other dedicated functional modules of the architecture.
- Creating a link between the application tasks and the hardware architecture as well as managing sensitive and non-sensitive application tasks in monitoring and non-monitoring reconfigurable partitions of the architecture.
- Keeping record of different reconfigurable partitions of the architecture; and the set of frame addresses of different GPRRs' relocatable bitstream.
- Task management related operations such as task placement, task replacement, task relocation, co-ordination among different tasks and task synchronization, whose definitions are the following.

  Task placement: choosing an appropriate location to implement a particular task by considering different factors, including the sensitivity of the task, the physical location of other related tasks, etc. *Task replacement*: implementing a different task by replacing an existing task from a particular hardware location, with the help of partial reconfiguration. *Task relocation*: moving a particular task to another hardware location, due to permanent hardware failure in the current location. *Task co-ordination*: recall that tasks are sub-units of a bigger function, output(s) of a particular task might be input(s) to one or more other tasks, and some tasks need to be executed in parallel with certain

timing requirements. In such cases, making inter-task communication and co-ordinating different tasks to maintain internal state values intact is essential. *Task synchronization*: after the task fault mitigation or fault relocation, the state values of different inter-related tasks must be synchronized to avoid any uncertain results.

- Managing shared resources such as ICAP, background read-back and write-back mechanisms, among different fault mitigation strategies and APIs. Also, co-ordinating different fault mitigation strategies in terms of providing access to the shared resources.

Different APIs are employed in the R3M to support different reliability and resource management mechanisms. For instance, the LFM-API (explained in Chapter 4) is dedicated to handle the fault mitigation scheme for configurable logic resources, whereas the CFM-API (explained in Chapter 5) is dedicated to handle the configuration bitstream protection scheme. Upon getting notified about fault occurrence by the introspection plan, the R3M initiates the fault mitigation process. The fault mitigation strategies defined are unique and vary from one to another functional unit. The factors involving the definition of the fault mitigation strategies are: i) the nature of a fault (temporary, hard or soft), ii) the fault model and its consequences in that particular circuit/hardware, and iii) the severity of the malfunctioning.

In the case of fault mitigation in configurable logic resources, once the fault occurrence is detected, the R3M employs a 3-stage strategy. Initially, assuming that a fault is transient, it attempts to recompute the task to clear the results of the fault. If the presence of the fault is still signaled, assuming that some upsets (SBU or MBU) are present in the associated configuration bits, it attempts to perform partial reconfiguration to remove them. Finally, if the fault status still persists, the permanent hardware fault is declared and the task is relocated to another location. Through this strategy, R3M identifies the nature of the fault and it helps in distinguishing temporary and permanent faults. Within the framework of the configuration bitstream protection, it performs the proposed 3D Hamming based multiple-bit error correcting scheme, discussed in Chapter 5. Although the configuration bitstream protection scheme operates as a background process locally at the architecture level, it is controlled globally by the centralized reliability manager, R3M.

Handling Shared Resources

The stages two and three of the configurable logic resource fault mitigation strategy involve partial reconfiguration. Similarly, the configuration bitstream protection scheme involves partial reconfiguration. The configuration bitstream protection scheme handles the frames sequentially in the complete ARDyT FPGA architecture, and its a cyclic process. However, at stage two (when the fault is still detected after task re-computation) of configurable logic resource fault mitigation, the configuration bitstream protection scheme has to be performed on these particular set of configuration frames (associated to fault detected area) to clear the upsets.

These operations share the configuration frame access control, read-back and write-back resources. The R3M co-ordinates different strategies and provides access

to these shared resources among different APIs. For instance, if the LFM-API enters the second stage of operation (where upsets in the associated configuration bitstream have to be cleared), it sends an access request signal to the R3M, to get the access of the configuration bitstream protection scheme. The R3M interrupts the background configuration protection process and launches the same at the different frame address provided by the LFM-API. As soon as the configuration bitstream protection scheme completes serving the given set of configuration frames, it returns to its previous location (frame address) to continue operation. The current and the previous frame addresses are handled by the CFM-API. The R3M establishes communication between the LFM-API and the CFM-API, and co-ordinates the complete process.

3.5 Summary

The ARDyT framework is introduced and detailed in this chapter. The general overview of the overall architecture is presented, including details of the hardware and the dedicated FTAL layer. The functional description of the introspection plan and the R3M are detailed in this chapter. Fault mitigation in other dedicated hardware resources like DSP, ALU and BRAM, etc., relocatable bitstream and task management strategies, and supporting reliability-aware CAD tool-suits are developed by other partners in the ARDyT project consortium.

Chapter 4

Fault Aware Configurable Logic Block (FA-CLB)

Configurable logic blocks (CLBs), which are the basic building blocks in reconfigurable FPGAs, comprise a set of combination and sequential logic circuit elements, which are programmed by built-in SRAM cells to realize a desired function. These circuit elements and programming memory cells are primitive elements of a CLB. The behavior of these primitive circuit elements differs in case of radiation particle hit so do the consequences of the latter. To mitigate the effects of radiation induced faults in primitive elements of the CLB, here a fault-aware (FA) architecture (FA-CLB) is proposed, which is aimed at fine granular level (i.e at the level of LUT and D-FF inside the CLB). Obviously, the proposed architecture is customized to adapt it to the ARDyT architectural framework. The fault mitigation process is handled by the centralized reliability manager (R3M) in the ARDyT architecture. The proposed FA-CLB architecture is fault-aware and mechanisms provided along with the basic structure of the FA-CLB architecture support fault mitigation process managed by the R3M. Each FA-CLB consists of a functional equivalent of four "LUT:D-FF" pairs, called configurable logic elements (CLE).

As presented in Chapter III, the goal of the ARDyT architecture is to integrate reliability strategy in different layers of the design: from hardware to application, based on different abstraction levels. Fault tolerance in configurable logic elements are realized by adapting techniques in hardware layer as well as in the fault tolerant abstraction layer. The techniques adapted in the hardware layer facilitate fault detection and fault mitigation processes. On the other hand, the mechanism integrated in the FTAL manages the fault status reading and initiates the fault mitigation process. The combined approach of the 'architectural support' and the 'fault management support by R3M', aims to keep better trade-off between overhead (hardware, time and power) and reliability.

Table 4.1 – Fault models and their consequences in combinational and sequential circuit elements

	Combinational Circuit Elements	**Sequential Circuit Elements**
Effect of Radiation Induced Charged Particle Hit	Radiation induced transistor/ logic circuit faults	Upset in the user data memory elements
Fault Model	Single Event Transients, Transistor Bridging and Stuck-at-1/0 faults	Single Bit Upsets (SBUs) and Stuck-at-logic value
Consequence	Unpredictable/ undesired combinational logic output	User data corruption (State change in the user logic)

4.1 Primitive Elements, Fault Models and Their Consequences

To apply efficiently fault mitigation schemes to any given logic resource, it is crucial to understand the fault models and the consequences of their occurrence. Before proposing FA-CLB architecture, traditional logic cell architecture is analyzed, to classify its primitive elements with respect to their behavior to different fault models. Table 4.1 summarizes the effects of radiation induced charged particle in combinational and sequential elements of reconfigurable FPGAs. In configurable logic resources, combinational and sequential circuit elements are programmed by configuration memory bits. As the configuration memory bitstream protection will be introduced in Chapter V, this chapter focuses on fault models and the mitigation schemes of combinational and sequential circuit elements of the FA-CLB. The nature of a fault and its consequences differ for combinational and sequential circuit elements.

4.1.1 Consequences in Combinational Circuit Elements

As far as combinational logic is concerned, it does not contain any storage element to get affected by radiation-induced SEUs and therefore to create single bit or multi-bit upsets. Faults in combinational circuits are more of transient effects. As discussed in [99–103], the circuit level faults in combinational logic include radiation-induced transients, stuck-at-faults and transistor bridging faults. Unlike fabrication faults, the effect of these radiation induction faults could be temporary as well as permanent, depending on the nature and intensity of the radiation particle.

4.1.2 Consequences in Sequential Circuit Elements

To implement sequential circuit functionality and store user data bits, D-flip flops (D-FFs) are commonly used in FPGA architectures. Unlike configuration memory cells (SRAM), these user data memory cells are updated during task execution. The effects of a radiation particle hit in these D-FFs can cause different faults with different consequences.

A radiation induced charged particle hit in a D-FF can cause: i) circuit level internal transient fault and/or ii) user data corruption, which can also be called a single bit upset (SBU) as it causes state change in the user logic.

4.2 Fault-aware Configurable Logic Element (CLE)

A logic cell/element, denoted here a CLE, refers to a simple functional equivalent of a 'combinational and sequential element pair'. Conventionally, it is a 'LUT:D-FF' pair, where LUT is for combinational and D-FF is for sequential circuit implementations. As discussed in Chapter I, a simple CLE structure is realized by coupling a combinational circuit element and a sequential circuit element, provided with a simple 2:1 output selection multiplexer. In contemporary architectures, the basic building block (CLB in Xilinx and LAB in Altera FPGA devices) contains more than one 'LUT:D-FF' pair (CLE), due to today's increased functional density requirements. Similarly, in the ARDyT architecture the proposed FA-CLB has functional equivalents of four CLEs. However, the reason is not just to meet increased requirements of the functional density but also to provide reliability support.

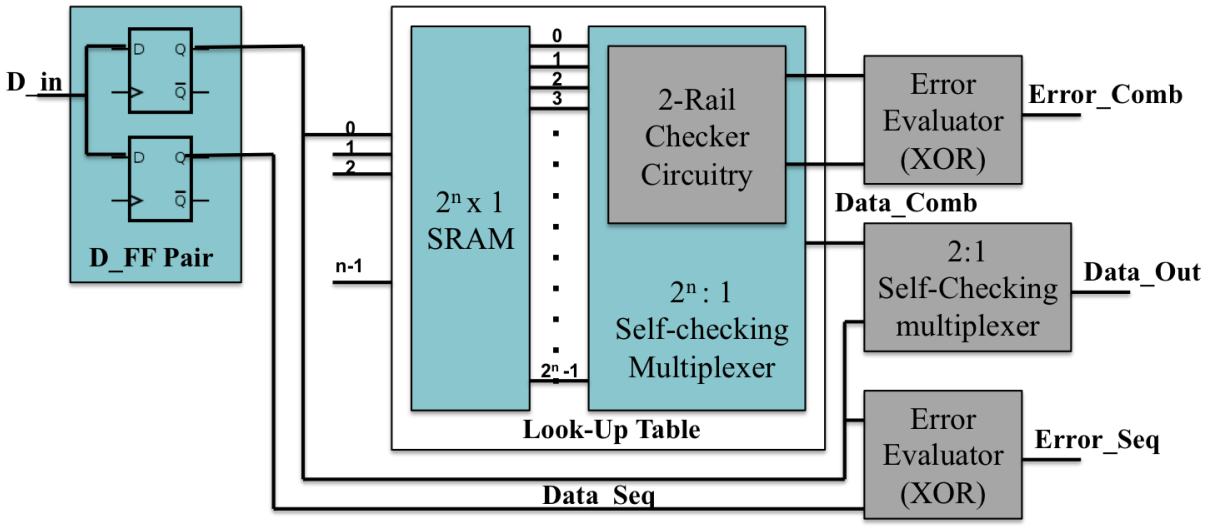


Figure 4.1 – Configurable logic element (CLE) architecture of the ARDyT FA-CLB

By using fault-aware logic elements, configurable logic resources can be made fault-aware, which can hence support reliability mechanism managed by R3M. The fault-aware configurable logic blocks (FA-CLBs) are designed on the basis of:

- categorizing their primitive elements according to their logical nature (sequential or combinational) and
- analyzing the fault models and their consequences in the primitive elements (as discussed in Table 4.1.

According to the discussion of the details listed in Table 4.1, it is understood that each primitive element has a different circuit nature, experiencing different faults

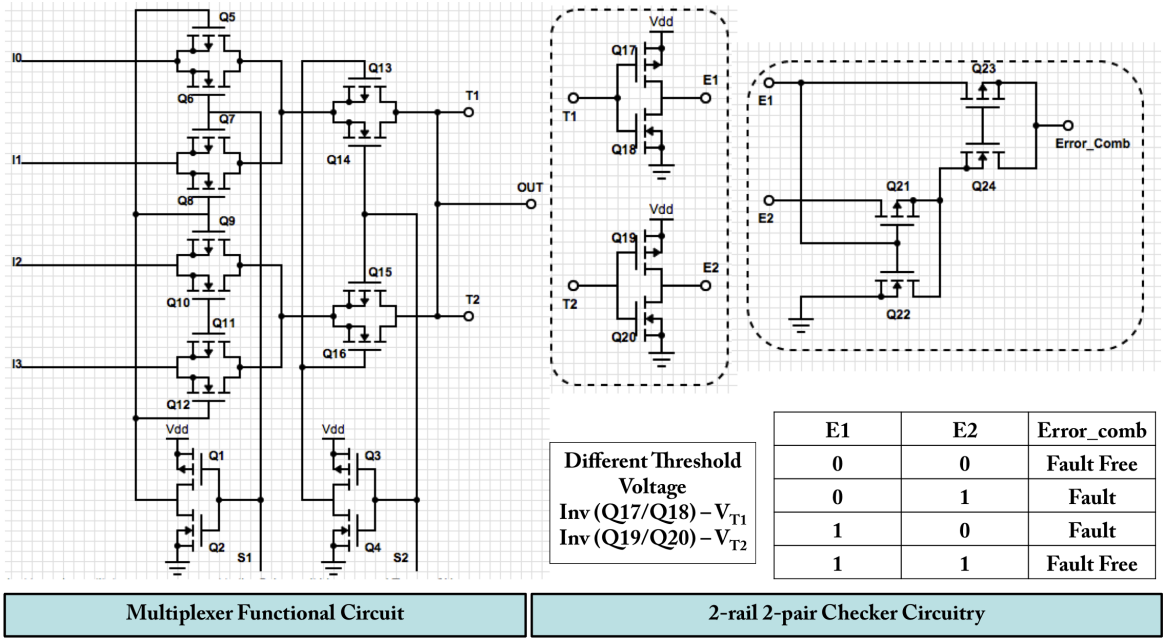


Figure 4.2 – Transistor-level structure of the 4-input self-checking multiplexer

which have different consequences. Hence, the well-adapted fault-model-aware fault detection schemes are suggested for different primitive elements in the proposed FA-CLB.

Figure 4.1 shows the functional diagram of the proposed fault-aware CLE (Each FA-CLB has 4 CLEs; CLE is a simple 'LUT:D-FF' pair logic element), for a general case. The n-input logic element comprises conventionally available combinational and sequential logic elements: an $n:1$ LUT, a pair of D-FFs, and the $2:1$ multiplexer (to choose between combinational and sequential data output). In the proposed CLE, the primitive elements are implemented as self-checking circuits by using 2-rail codes for multiplexers (similarly e.g. to [104, 105]) and the DMR approach to protect the D-FF.

4.2.1 Fault-aware Combinational Logic Element

A LUT-based logic element structure is used in the proposed scheme (similar to Xilinx Virtex FPGA architectures), due to the possibility of implementing all 2^n n-bit logic functions in it, unlike, multiplexer-based architectures. However, the proposed approach can also be adapted to multiplexer-based logic blocks, such as Microsemi's FPGA architectures [106], with smaller transistor overhead but limited functional efficiency. The configuration data of LUT SRAM cells are part of FPGA's configuration bitstream and they can be protected using configuration bitstream protection scheme discussed in Chapter 5 and the references therein. Here, we focus on dealing with faults occurring in the selection multiplexers attached to the LUT SRAM cells.

The self-checking $2:1$ multiplexer built using four transmission gates and an inverter can be found in [105]. Figure 4.2 shows a pass-transistor scheme of a self-checking $4:1$ multiplexer protected using 2-rail code, used in [104]. The transistor

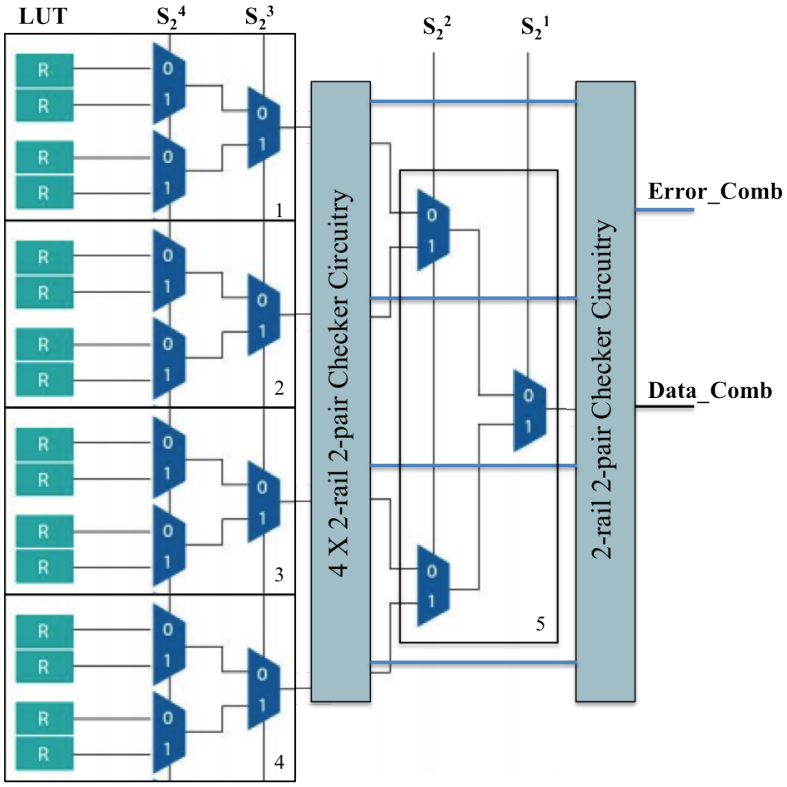


Figure 4.3 – Proposed structure of the $2^4 : 1$ self-checking multiplexer

pairs (Q17/Q18 and Q19/Q20) used in the checker are characterized by different aspect ratios to achieve different voltage thresholds V_{T1} and V_{T2}, which can be achieved in CMOS technology by various methods discussed in [107, 119]. If the 'OUT' node has an intermediate voltage, the checker outputs 'E1 and E2' are with different voltage levels; otherwise, E1 and E2 have the same voltage level. Indeed, the 'OUT' node has an intermediate voltage when there is a fault in the circuit. Hence, it is interpreted as in the absence of a fault, when the output of the checker (E1 E2) produces the values (11) or (00). If a fault occurs in the multiplexer, (E1 E2) shall assume either (01) or (10). According to (E1 E2), the comparator (i.e. the 2-input XOR) generates the error status signal 'Err_comp' to indicate the occurrence of a fault in the combinational logic.

The self-checking 4-input multiplexer of Figure 4.2 is the basic circuit for the proposed fault-aware combinational logic element. The $4 : 1$ multiplexer has two functional logic levels 2^2 and 2^1 connected to the final 2-rail checker. The LUT selection multiplexers of the 4-input, 6-input, and 8-input LUTs can be constructed by introducing the 2-rail checker circuitry after every second functional logic level of the multiplexer; i.e., larger self-checking multiplexers can be constructed using the set of $4 : 1$ self-checking multiplexers by connecting them appropriately. For example, the $2^4 : 1$ self-checking multiplexer can be constructed using five $4 : 1$ multiplexers using 2-rail checker circuitry at two logic levels 2^1 and 2^3, as shown in Fig. 4.3. The $2^4 : 1$ or $16 : 1$ combinational logic element has 16 SRAM cells containing the LUT data (through configuration bits), to realize a desired Boolean function, followed by a 16×1 selection multiplexer. The 16×1 multiplexer has 4

logic levels 2^4, 2^3, 2^2, and 2^1. At the 2^3 logic level, it is evident that four distinct (non-overlapping) outputs are coming from four $4:1$ multiplexers. Similarly, at the end of the $2^4:1$ multiplexer, in the 2^1 logic level, the output is from a simple $4:1$ multiplexer, if the circuit is considered after 2^3 logic level. Considering these five $4:1$ multiplexers independently and implementing each of them as self-checking will result in a self-checking $2^4:1$ multiplexer.

4.2.2 Fault-aware Sequential Logic Element

D-flip flops are the most commonly used user data memory elements in FPGAs, to support sequential circuit function implementations. As D-FFs are storage elements, it is not possible to adapt self-checking circuit as in multiplexer checking, as it can deal only with circuit faults but not with bit upsets. The contents of D-FFs is updated during computation (run-time), hence the user data bits cannot be protected using any of the built-in configuration bitstream protection schemes.

To deal with circuit faults as well as with user data upsets, it is worth to consider some redundancy based technique. A simple duplication of D-FF with output comparison suffices to identify/detect the fault. By taking advantage of the ratio between basic combinational and sequential logic resources (e.g. LUTs and D-FFs, respectively) in the contemporary FPGA logic block architectures, a simple DMR could be an effective method to implement fault-aware sequential element. Often the ratio of combinational and sequential elements is like $1:2$ (for example, a standard CLB slice of Xilinx 7 Series architecture consists of 4 LUTs and 8 D-FFs [119]). To note also that it is reported in [119] that the number of unused flip-flops is high in most of applications. Hence, unused flip-flops of the logic block can be utilized to detect faults in sequential elements by performing duplication with comparison, as it is shown in Figure 4.1. The comparator is nothing else but the 2-input XOR gate. In the presence of a fault causing an error, once the mismatch is detected on the outputs of duplicated D-FFs, the comparator generates 'Err_seq' signal to indicate the presence of error in the sequential logic. The cost of this strategy is one additional XOR gate for each pair of D-FFs and associated routing wires inside the FA-CLB.

4.2.3 Hardware Overhead Comparison

Table 4.2 shows the transistor count for the individual circuit elements used in the proposed fault-aware CLE structure. These data will be helpful to estimate the total hardware overhead.

The circuit elements in the proposed 4-input CLE include 16 SRAM LUT memory cells, the $16:1$ self-checking multiplexer, two D-flip flops and the 1-bit comparator (2-input XOR), and the $2:1$ self-checking output multiplexer. The $16:1$ self-checking multiplexer is constructed using 5 $4:1$ multiplexers (the $4:1$ multiplexer is constructed of 3 $2:1$ multiplexers).

Table 4.2 – Transistor count of modules used

Module Name	**Transistor Count**
SRAM cell of LUT	6T
2-pair 2-rail checker + Error evaluator	8T
D flip-flop	8T
XOR/Comparator	4T
2 : 1 multiplexer	6T

Table 4.3 – Hardware overhead: proposed scheme vs. TMR-protected scheme

LUT Input size	Unprotected	Proposed structure			TMR-Protected		
	Transistor count	Transistor count		Overhead %	Transistor count		Overhead %
		Total	Overhead		Total	Overhead	
4	178	238	60	33.70	560	382	214.60
6	662	850	188	28.39	2012	1350	203.92
8	2587	3287	700	27.05	7787	5200	201.00

Tables 4.3, 4.4, and 4.5 show hardware complexity estimations which allow to compare the proposed fault-aware CLE against its unprotected version as well as its three fault-aware counterparts: TMR-based, DMR-based and the scheme proposed in [104]. Table 4.3 compares the estimated hardware complexity between the TMR-based scheme and proposed CLE structure, by keeping the unprotected design transistor count as reference.

Table 4.4 – Hardware overhead: proposed scheme vs. DMR-protected scheme

LUT Input size	Unprotected	Proposed structure			DMR-Protected		
	Transistor count	Transistor count		Overhead %	Transistor count		Overhead %
		Total	Overhead		Total	Overhead	
4	178	238	60	33.70	360	182	102.24
6	662	850	188	28.39	1328	666	100.60
8	2587	3287	700	27.05	5178	2591	100.15

The evaluation is performed for three logic block sizes: 4-input ($2^4 \times 1$ SRAM and $2^4 : 1$ MUX), 6-input ($2^6 \times 1$ SRAM and $2^6 : 1$ MUX), and 8-input ($2^8 \times 1$ SRAM and $2^8 : 1$ MUX). The TMR-based scheme triplicates every single circuit element in the logic cell and the "supposedly" correct output is voted by the majority voter at the output. On the other hand, the DMR-based scheme simply duplicates the complete CLE resources and compares output results. The hardware complexity comparison of the proposed fault-aware CLE and the DMR-based scheme is given in Table 4.4. To note that both TMR and DMR schemes do not rely on any particular fault-model of a cell, because they are directly applied to the entire CLE structure, whereas the scheme proposed here relies on fault-model-aware detection.

Table 4.5 – Hardware overhead: proposed scheme vs. scheme in [104]

LUT Input size	Unprotected	Proposed structure			Scheme proposed in [104]		
	Transistor count	Transistor count		Overhead %	Transistor count		Overhead %
		Total	Overhead		Total	Overhead	
4	178	238	60	33.70	312	134	75.28

The proposed fault-aware CLE applies self-checking at the fine granular level (circuit level fault detection). Also, it applies different fault detection techniques to combinational and sequential logic, according to their fault models and consequences. Hence, the proposed scheme is able to detect and identify combinational faults and sequential faults individually. Unlike redundancy based techniques such as DMR and TMR, the proposed scheme does not have to wait till the signal reaches the module outputs, so that a comparator or a voter can handle errors; thus it helps to avoid accumulation of errors and enables faster fault detection.

Table 4.5 compares hardware complexity of the proposed fault-aware CLE and the self-checking logic cell circuit proposed in [104]. Compared to the 4-input self-checking logic cell proposed in [104] which requires a total of 312 transistors, our fault-aware CLE requires only 238 transistors, i.e. 23.71% less overhead, due to design optimization.

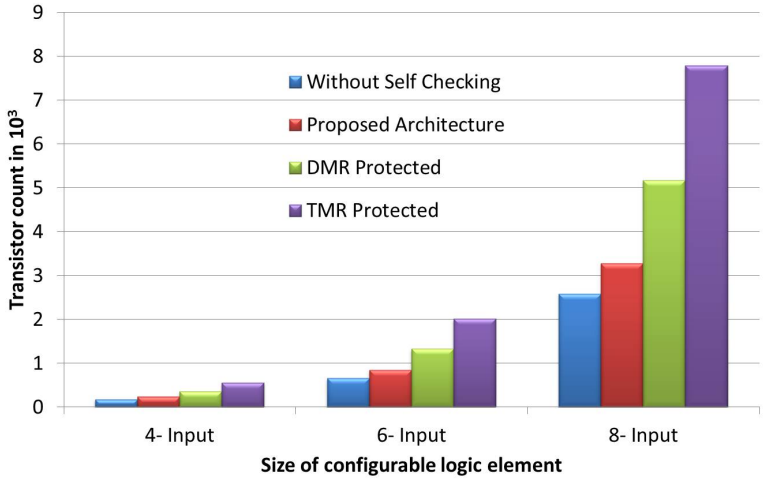


Figure 4.4 – Hardware complexity of various schemes of CLE

The hardware complexity characteristics given in Fig. 4.4 clearly show that the proposed fault-aware CLE is significantly more hardware efficient compared to DMR and TMR based schemes which require respectively about three and six times extra hardware than the proposed architecture. For example, to implement the 6-input CLE using the proposed fault checking mechanism, only extra 188 transistors are required, i.e., 28.39% more, whereas in case of DMR and TMR the overhead reaches 100.60% and 203.92%, respectively.

4.3 Fault-aware CLB for ARDyT FPGA

The detailed architecture of the proposed *Fault-aware Configurable Logic Block (FA-CLB)* is shown in Figure 4.5. One can notice significant changes of its structure compared to conventional COTS FPGA's logic block architectures. In particular, it comprises four CLEs ('LUT:D-FF' pairs) with a unique arrangement of circuit elements, as shown in Figure 4.5. The repositioning of the combinational and sequential logic elements in the proposed FA-CLB supports: i) built-in fault detection scheme and ii) the fault mitigation scheme managed by the R3M.

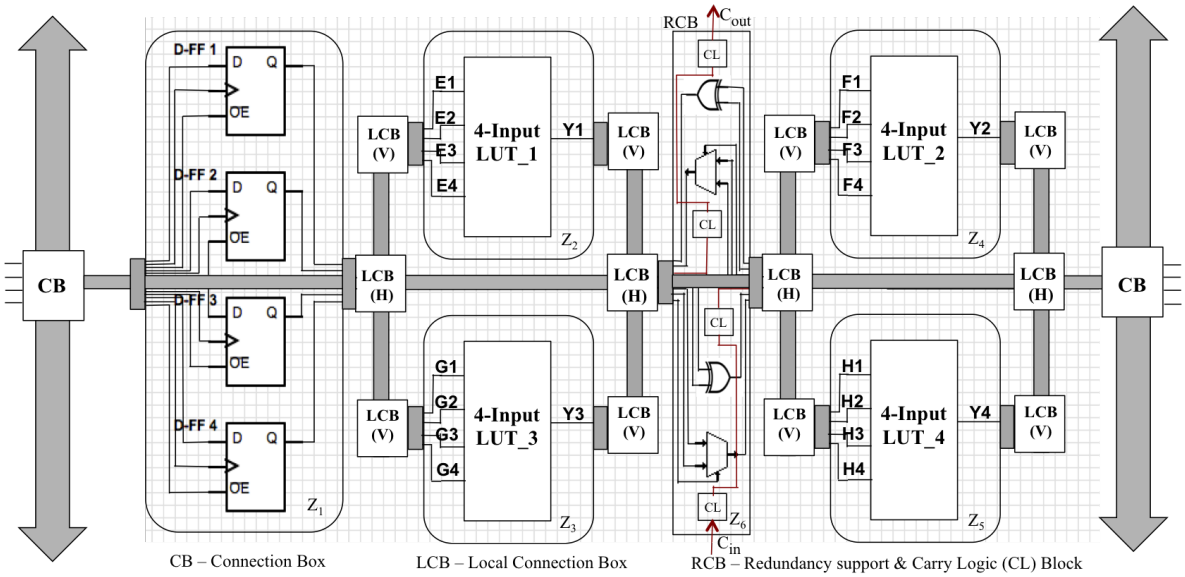


Figure 4.5 – Fault-aware CLB (FA-CLB)

The entire FA-CLB architecture is divided into 6 local zones (Z_1 to Z_6). All the sequential circuit elements (D-FFs) are arranged column-wise in the first local zone Z_1. Four combinational circuit elements (LUTs) are placed in the local zones Z_2 to Z_5. Each D-FF in Z_1 is associated with a corresponding LUT to form a logic element (LUT:D-FF pair). The local zone Z_6, called redundancy support and carry logic block (RCB) comprises resources to support redundancy scheme and carry logic. The internal connections among the local zones are made with the help of simple internal connection boxes: horizontal local connection box (LCB-H) and vertical local connection box (LCB-V).

The RCB is a dedicated reliability and carry logic support block built-in the proposed FA-CLB, as shown in Figure 4.5, connected through the local connection boxes and the routing lines. It includes the resources (XOR, for bit comparison) to implement the DMR scheme, multiplexers for different output configurations and for data selection between sequential data output and combinational data output (according to the application requirement), and carry chain logic circuit to support arithmetic operations. The carry chain is connected to other FA-CLBs through carry in (C_{in}) and carry out (C_{out}) lines. It helps in performing arithmetic operations on larger operands. A set of FA-CLBs connected with carry chain logic is used in implementing larger tasks.

CLBs can experience: i) temporary faults in the form of transient effects and/or user data upsets (in D-FFs), ii) uncertain operation due to upsets in configuration bits of the FA-CLB, and iii) permanent faults. In the ARDyT framework, configuration bitstream upsets (SBUs and MBUs) are handled by the newly proposed 3DH error correcting scheme which will be presented in Chapter 5. Permanent faults create lasting effects by damaging hardware resources which makes them non-recoverable, hence relocating the task to other healthy design areas is the only possible solution. The scheme managed to handle temporary faults in the configurable logic resource (FA-CLBs) is a 3-step process: **detect-freeze-re-execute** which consists of detecting the fault, freezing (holding) the inputs of the faulty circuit modules, and re-executing the task in the next operating cycle which possibly mitigates the effect of the temporary fault. In the re-execution process, when a transient error occurs and is detected during the operation, the system restores its input state to the previous input and re-computes. The challenge in performing re-execution lies in "how and where the previous input values of the task are kept to perform the operation?" Traditionally, *rollback* [108] and *roll-forward* [109] techniques are used to perform this task with the help of additional set of registers to keep the stable state values. To facilitate the task re-execution process in the ARDyT architecture, the conventional configurable logic block structure is modified in such a way that it does not require any additional registers to hold the previous set of stable inputs. The D-FFs grouped in the local zone Z_1 facilitate two operations: i) holding the input values of the LUT to reinforce and re-execute the task (in case of fault detection) and ii) storing the user data output from the computed task (as it is done conventionally).

As far as fault detection is concerned, there are two options through which built-in fault detection support can be provided in the proposed FA-CLB.

- **Case-I:** Faults in sequential circuit elements are detected by DMR. Combinational circuit elements are made fault-aware by the realizing them as self-checking circuits, as proposed in the previous section (provided that modifying transistor parameters is possible).
- **Case-II:** DMR scheme is applied to both combinational and sequential circuit elements (LUT:D-FF pairs) (when modifying transistor parameters is not possible).

The proposed FA-CLB architecture supports both cases. Arrangement of D-FFs in the first local zone makes them independent from LUTs. In Case-I, to detect faults in sequential circuit elements, the DMR scheme can be adapted easily without any complex internal routing and selection bits. Inputs (D_{in}) of redundant D-FFs can be directly taken from the connection box (cf. Section 4.3.1) and the corresponding outputs (D_{out}) can be compared at the RCB (Z_6), driven through local connection boxes. The designer can choose to utilize the logic resources with or without the DMR scheme. In Case-II, there are three modes of working configuration: i) without DMR, in which four D-FF and LUT pairs can be used for functional implementation; ii) with DMR, in which two D-FF and LUT pairs can be used for functional implementation with each pair having its duplicated resource;

the outputs of both are connected to the comparator; and iii) with partial DMR, in which selective resources are duplicated. The resource usage and reliability level trade-offs are summarized in Table 4.6.

Table 4.6 – Resource usage vs. reliability trade-off

FA-CLB Configuration	Functional Resource Utilization (%)	Reliability Level (%)
No DMR	100	0
Partial DMR	75	25
Full DMR	50	100

No DMR: 4 'D-Flip-flop - LUT Pair'

When a designer chooses to not apply DMR to a particular FA-CLB, all four 'D-FF and LUT' pairs can be used independently to implement task functionality. Each D-FF arranged in the first local zone of the FA-CLB is associated with its corresponding LUT in the local zones Z_2, Z_3, Z_4 and Z_5, i.e. $\{D-FF_1 : LUT_1\}$, $\{D-FF_2 : LUT_2\}$, $\{D-FF_3 : LUT_3\}$, and $\{D-FF_4 : LUT_4\}$. In such a case, the centralized manager R3M marks these FA-CLBs as unprotected and it does not keep record of its fault status. Such unprotected CLBs can be switched to protected ones, on the fly, by writing appropriate set of configuration bitstream using dynamic partial reconfiguration.

DMR : 2 'D-Flip-flop - LUT Pair'

Should the complete DMR scheme be used, the functional density of that FA-CLB is reduced by its half (i.e., 50% resources of a FA-CLB are used for functional implementation and 50% for redundancy is). Each 'D-FF and LUT' pair gets its duplicated resources: $\{D-FF_2 : LUT_2\}$ duplicates $\{D-FF_1 : LUT_1\}$ and $\{D-FF_4 : LUT_4\}$ duplicates $\{D-FF_3 : LUT_3\}$. During fault-free operation, D-FF_1 and D-FF_2 receive the same input and LUT_1 and LUT_2 are expected to produce the same output. The outputs of LUT_1 and LUT_2 are connected to the comparator which can detect a mismatch. Similar connections are configured for D-FF_3 and D-FF_4 in association with LUT_3 and LUT_4, respectively.

Partial DMR : 3 'D-Flip-flop - LUT Pair'

The proposed FA-CLB architecture also supports selective application of DMR, where some two 'D-FF and LUT' pairs duplicate each other with their outputs compared, whereas some other two 'D-FF and LUT' pairs are used for independent functional implementation; e.g. $\{D-FF_2 : LUT_2\}$ duplicates $\{D-FF_1 : LUT_1\}$, whereas $\{D-FF_3 : LUT_3\}$ and $\{D-FF_4 : LUT_4\}$ pairs are used independently. In this case, the functional capacity of the FA-CLB is 75%, as 3 out of 4 of its 'D-FF and LUT' pairs are used for functional implementation of the application task and one 'D-FF and LUT' pair is used for redundancy. In such a case, this particular FA-CLB becomes partially protected at the fine granular level.

4.3.1 Input Connection Configuration to Adapt DMR

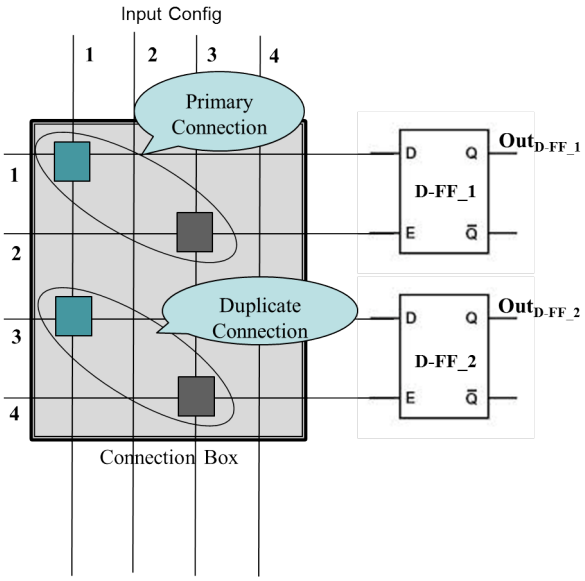


Figure 4.6 – DMR implementation of the input configuration cell

In a general case, to apply DMR-based fault detection scheme to any hardware resource, extra routing lines are required to make redundant paths. In the proposed architecture, the existing programmable routing lines in the connection box matrices support the DMR input configuration functional implementation. As shown in Figure 4.6, every line passing through connection box has four configurable connections, so the proposed FA-CLB structure can simply take advantage of this arrangement to apply DMR by driving appropriate configuration bits in the connection box.

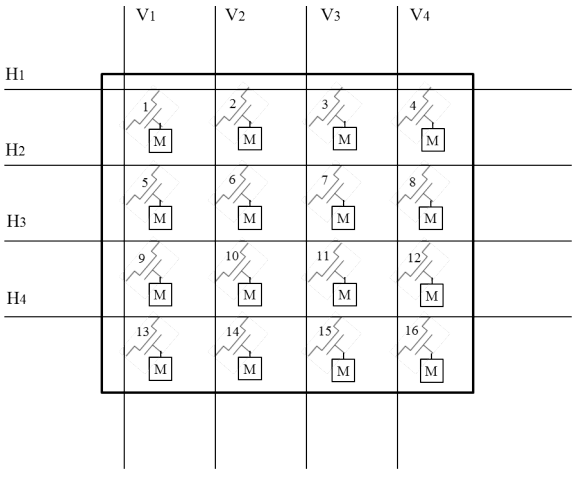


Figure 4.7 – Simplified connection box architecture

Figure 4.6 shows that the signals being carried by vertical lines 1 and 3 are served as inputs to both the D-FFs (D-FF_1 and D-FF_2) at their respective inputs, by making a primary and duplicate cross-point connections at the routing lines. In this case, duplication of input lines are free of extra overhead. Figure 4.7 illustrates the

internal architecture of proposed connection box (C.B). There are four horizontal and vertical channel lines forming a 4×4 cross point (matrix) switch. Each horizontal and vertical intersection is controlled by a pass transistor and a SRAM configuration bit. Writing appropriate values in the configuration bits of connection box, drives desired signals to the FA-CLB, to implement DMR'ed design. There are total 16 programmable connection points in the connection box. Lets consider, horizontal lines *H1* and *H3* are connected to inputs of a redundant logic and the desired input signal is available in the vertical channel line *V2*, then it is just sufficient to enable the switching transistors 2&10 through their SRAM bits.

4.3.2 Output Connection Configuration to Adapt DMR

While adapting DMR (fully or partially) inside the FA-CLB, the outputs of the 'D-FF and LUT' pairs have to be configured properly to raise the fault status signal efficiently. There are three possible ways of connecting the outputs of 'D-FF and LUT' pairs to the fault detection comparators, as shown in Figures 4.8, 4.9, and 4.10.

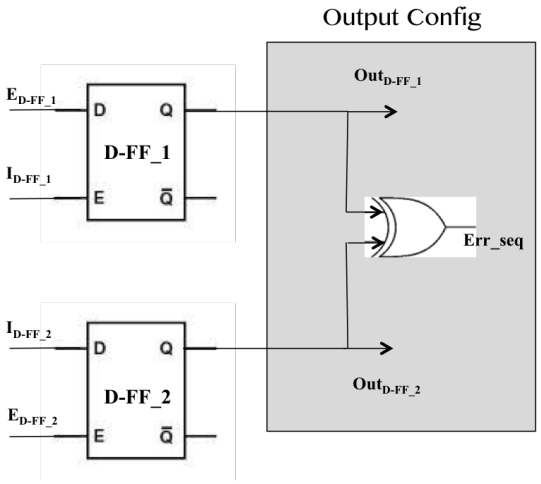


Figure 4.8 – Static DMR configuration

The first scheme of Figure 4.8 directly connects the outputs of the duplicated resources to the inputs of the comparator (XOR) and mismatch output is raised as the status signal. As this scheme uses a static hardwired comparator, there is no extra configuration bit used. In such a case, the comparator is always functional and error detection status signal output responds to every change in the comparator input. This scheme requires very little hardware (a XOR gate - 4T) compared to other two output configuration schemes, but it is not efficient in terms of power consumption.

The second scheme of Figure 4.9 also connects the outputs of the duplicated resources directly to the inputs of the comparator but the output mismatch is raised as the status signal by the output selection control through a multiplexer. The comparator output is controlled by the 'enable' signal of the multiplexer selection line. The output mismatch is notified to the R3M upon deciding and making control se-

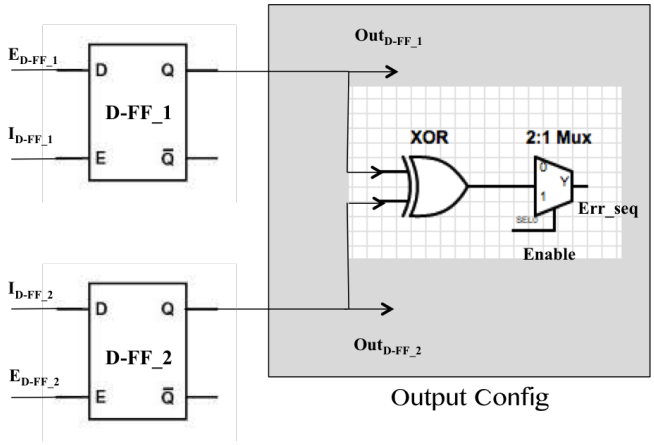


Figure 4.9 – DMR output configuration with output enabled

lection at this output multiplexer. This scheme requires one additional configuration bit (enable signal) and the 2 : 1 selection multiplexer.

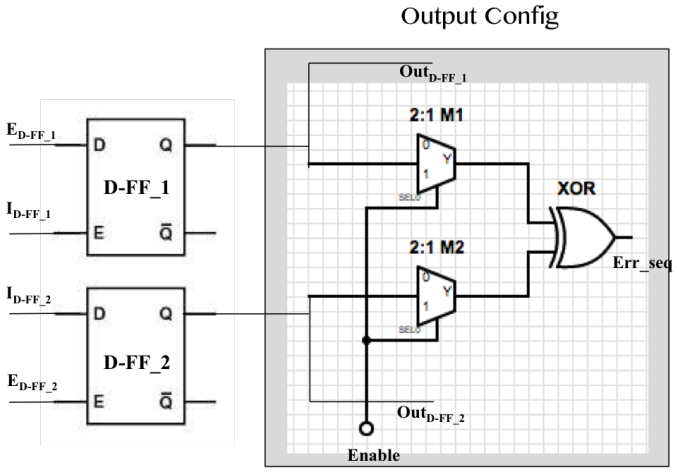


Figure 4.10 – Dynamic DMR output configuration with enabled input signal

The third scheme of Figure 4.10 separates the duplicated resources from the comparator input by a configurable input selection control. Two selection multiplexers connected at the input lines of the comparator enable or disable the inputs depending on the configuration provided. The advantage of this scheme is that the comparator is not always functional, because its inputs are configured through input multiplexers and enable lines.

Table 4.7 presents transistor count comparison of these three DMR output configuration schemes, each of which also has its own advantages and disadvantages in terms of hardware resources, power consumption and internal delay. The additional resources like XOR and extra multiplexers are integrated inside the proposed FA-CLB architecture (in the RCB), similarly as dedicated resources to support the DMR scheme. However, these built-in resources can be flexibly used according to the design approach. A tradeoff can be maintained between hardware overhead, power and propagation delay, while choosing appropriate output configuration scheme to

Table 4.7 – DMR output configuration: transistor count comparison

Output Config. 1 Figure 4.8	Output Config. 2 Figure 4.9	Output Config. 3 Figure 4.10
XOR - 4T	XOR - 4T 2:1 Mux - 6T SRAM Cell - 6T	XOR - 4T 2 × {2:1 Mux} - 12T SRAM Cell - 6T
Total - 4T	Total - 16T	Total - 22T

adapt DMR.

4.3.3 Architectural Support for Task Re-execution

Any transient error has a limited time duration. The FA-CLB architecture facilitates the process of task re-execution. A task could span over one or more fault-aware CLBs (FA-CLBs), depending on applications requirement to perform the task. Hence, the challenge would be to freeze the inputs of all FA-CLBs involved with the execution of a faulty task.

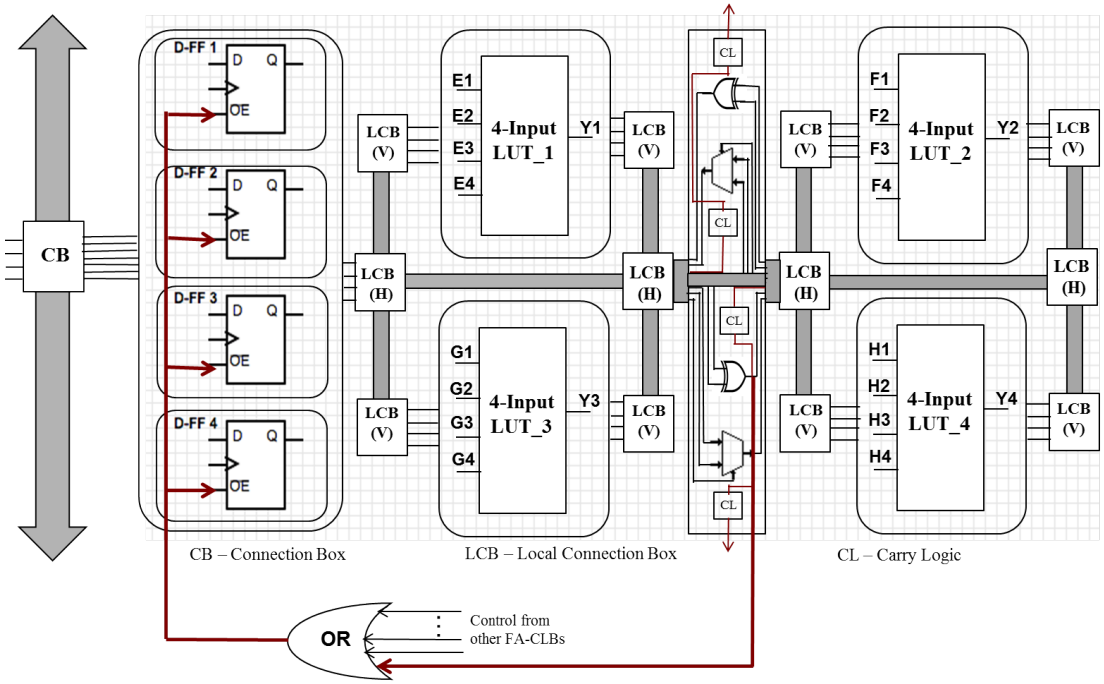


Figure 4.11 – Freezing the input: Re-execution of One-FA-CLB Task

In case of a small task which utilizes only one FA-CLB (1 FA-CLB = 4 CLE), the error status signals from combinational and sequential circuit elements could be OR-ed locally and routed to the enable (set) inputs of the D-FFs to freeze the inputs, as shown in Figure 4.11. It does not require any external connection box or switch box routing. The local error status signal from combinational and sequential circuit elements are routed through the local connection box (LCB) placed between the elements of the proposed FA-CLB. A dedicated OR gate provided inside the

FA-CLB sums-up the error status signal and feeds it to the enable inputs of the D-FFs to freeze the local inputs of the task.

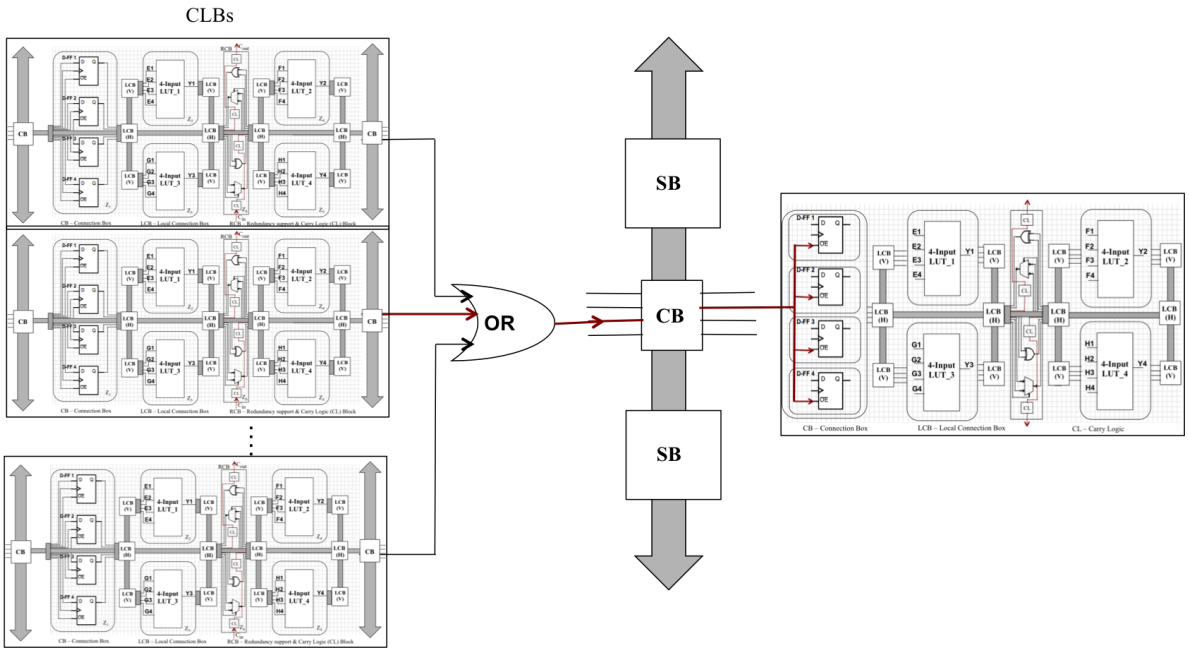


Figure 4.12 – Freezing the input: Re-execution of Multi-FA-CLB Task

On the other hand, larger tasks which utilize more than one FA-CLB would require routing lines through connection and switch boxes to route out the error status signal to freeze the inputs of other computational elements belonging to the same task, as depicted in Figure 4.12. Similar to local OR-ing of the error status signal in the case of one-cell task, different levels of hierarchical OR-ing are performed in multi-cell task, to route a single error status signal line to freeze all corresponding computational elements.

Additional OR gates are deployed in the architecture for the purpose OR-ing the fault status signals from the combinational and sequential circuit elements of different FA-CLBs related to the same task. The resulting (OR-ed) signal serves two purposes: i) to act as a control signal to freeze the corresponding input signals (as soon as the fault is detected), and ii) to update the fault status signal in the fault status register (FSR). At each GPRR level, there is a fault status register which has two dedicated bits to indicate the fault status values from combinational and sequential circuit elements.

4.4 Fault Mitigation Through the R3M

The R3M, the centralized reliability and reconfigurable resource manager, acts as a decision-making authority, whenever an uncertainty is detected in the implemented design. In the ARDyT architecture, the complete cycle of fault mitigation process in the FA-CLB is co-ordinated and handled by the R3M through *logic fault mitigation-application programming interface (LFM-API)*. The process includes sup-

porting mechanisms for regular monitoring, fault detection, fault notification, fault localization and fault mitigation.

The logic resources are being monitored regularly by the introspection plan running as a background process, without disturbing the application functionality. The introspection plan runs an interrogation protocol which reads dedicated 'reliability status registers' deployed in the architecture. The interrogation protocol and the details of those dedicated reliability status registers are explained in Chapter 3. At each GPRR level, there is a status register called FSR which holds the reliability status about different circuit elements available in that particular GPRR. The status bits CS and SS in the FSR represents fault occurrence in combinational and sequential circuit elements of the FA-CLB, respectively. The FA-CLB updates these 2 bits based on fault detection. The interrogation protocol reads these status bits and raises notification to the R3M whenever required. The fault status bits CS and SS are generated from 'Err_comb' and 'Err_seq' signals, respectively.

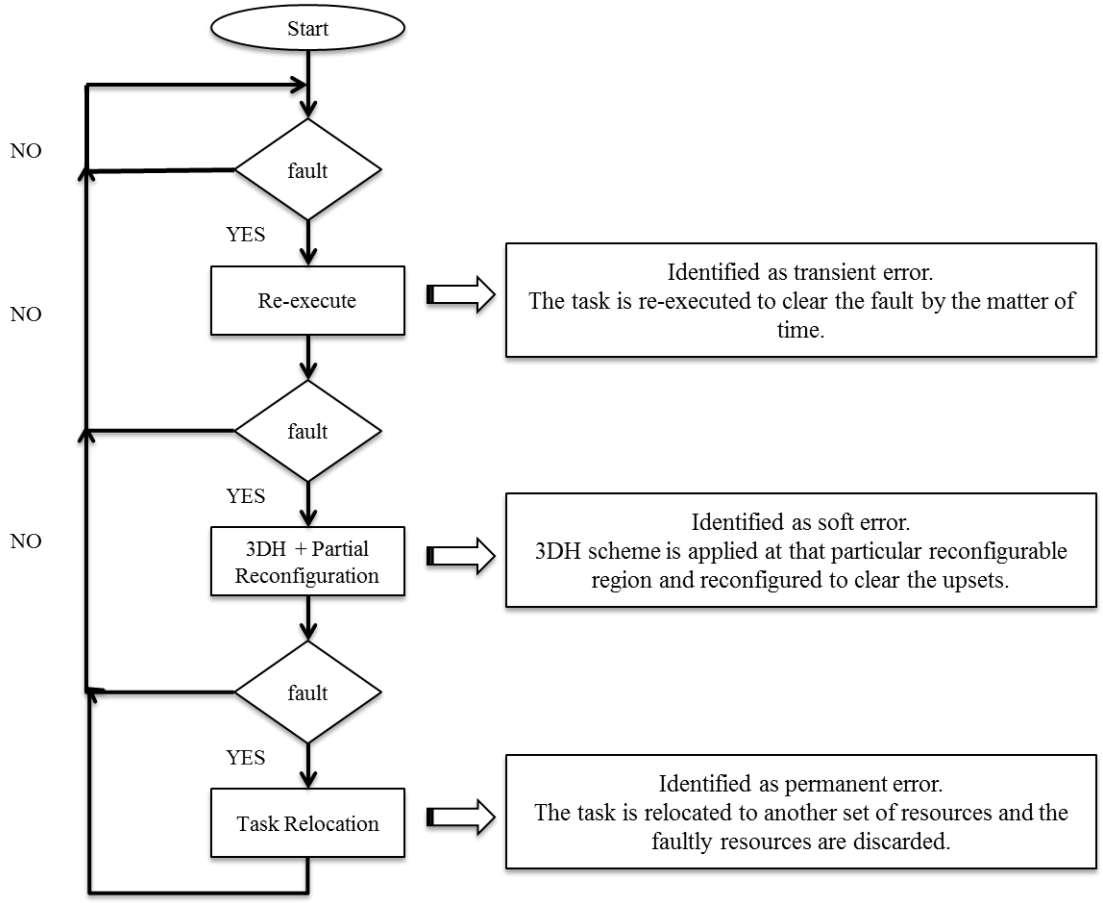


Figure 4.13 – Different stages of fault mitigation in the FA-CLB

The R3M handles faults in FA-CLBs according to the nature of the fault (recoverable and non-recoverable faults) indicated by the values of the CS and SS bits in the FSR. Two bits ($SSCS$) can assume four combinations of the following meanings: (00) – fault-free FA-CLB; (01) – a fault in the combinational circuit (e.g. transient); (10) – a fault in the sequential circuit element (user data memory); and (11) – an error in both combinational and sequential circuit elements. There are

three fault mitigation stages defined in the LFM-API, to handle the faults in FA-CLBs, as shown in Figure 4.13, including task re-execution, partial reconfiguration and task relocation.

4.4.1 Mitigating Transient Effects

As soon as the fault occurrence in FA-CLB is notified through CS and SS bits of the FSR, the fault status signal is routed to FTAL and then passed to the R3M through the introspection plan. At the same time, a control signal is generated by locally OR-ing the fault status signals at the GPRR level, to freeze the inputs of all the FA-CLBs involved in the execution of the task being handled by the faulty FA-CLB. Technically, this 'freeze' control signal is given to 'set' input lines of the corresponding D-FFs. Now these D-FFs hold the previous set of input values of which the generated output was faulty. Being the first stage of fault mitigation in LFM-API, the fault is assumed to be temporary. Hence, instructions to perform task re-execution are generated and that particular task is re-executed. As D-FFs of the associated FA-CLBs hold the previous inputs, now they would be able to compute for the same set of previous inputs. During this process, the effects of time-bounded (temporary) faults are cleared, at the cost of a simple cycle.

4.4.2 Mitigating Upsets in Configuration Bits

If the fault notification about FA-CLB is still active in the FSR, the fault mitigation process enters the second stage, in which the fault is assumed to be upset(s) of the programming bits of the FA-CLB. The complete configuration bitstream of the architecture is protected by the proposed 3DH error correcting scheme, which is running as a background process, performing error correction on the entire configuration bitstream, with the help of the ICAP. The R3M has a dedicated application programming interface (API), called *Configuration bitstream Fault Mitigation API (CFM-API)*, to handle this global configuration bitstream protection scheme in the ARDyT architecture. At this second stage of fault mitigation in the configurable logic resource, we require the 3DH configuration bitstream protection scheme to be performed on this particular set of configuration frames. This change of access control requires proper co-ordination and synchronization in managing frame addresses and the ICAP. The R3M manages the background read-back, configuration bitstream partitions and frame addresses, time-shared usage of the ICAP and co-ordinates different APIs to share the internal resources. Whenever LFM-API needs the control over the ICAP, the 3DH scheme and the related resources to perform error correction and then partial reconfiguration in a particular region of the architecture, it sends an interrupt to the R3M. The interrupt can be of different priority levels, depending on the sensitivity and urgency of the requirement. (Priority levels are based on the implemented application and the resource organization of the architecture). The interrupts are served on their priority level basis: either it is served immediately or it must wait till the running 3DH cycle terminates. Such an interrupt service requests are handled in a prioritized pipeline structure in the R3M. The interrupt service request by the LFM-API to get the control of the 3DH scheme

includes the frame address at which the configuration bitstream error correction scheme has to be launched. The corrected frames are written-back with the help of partial reconfiguration and the concerned task is re-executed.

4.4.3 Dealing with Hard Errors

If even after performing partial reconfiguration (with corrected configuration bits) and task re-execution, the fault status about FA-CLB persists in the FSR, it is declared as permanent hardware fault. It means that affected hardware resource is not recoverable from this kind of faults. In such a case, the faulty hardware is abandoned from the design and task implemented in that hardware is shifted/migrated to a healthy hardware resource. The task relocation requires proper synchronization of user states and local data, before and after the task migration. The LFM-API includes features such as inter-task communication, flag setup, and buffered internal states to achieving better synchronization of tasks during/after the task relocation.

Chapter 5

Built-in 3-Dimensional Hamming Multiple-Error Correcting Scheme

5.1 Proposed 3-Dimensional Hamming (3DH) Multiple-Error Correcting Code

In this section, the construction of the proposed 3-Dimensional Hamming code is described in detail. The basic idea is to arrange the information bits (to be protected) in a 3-dimensional format and applying simple, single bit error correcting code in all 3 axis of the 3-dimensional arrangement. The ultimate aim of constructing such code is to have an efficient, yet simple multiple-error correcting code. The analysis of the hardware complexity (logic gate count) and error correction performance (bit error rate (BER)) of various error correcting codes (ECCs), discussed in [121], [117], [110], [112], [124], [111], clearly shows that Hamming codes require the minimal hardware complexity compared to other ECCs. Hence, Hamming single error correction and double error detection (SEC/DED) is adapted as the primary error detection and correction scheme in the proposed 3DH code. The process of Hamming SEC/DED syndrome (parity/check bits) generation and error detection and correction is well explained in [123]. Hamming SEC/DED itself has no ability to correct more than a single bit error and detect more than a double bit error in a given data word. In other hand, the proposed 3DH scheme, which performs Hamming SEC/DED in 3 different co-ordinates, can detect and correct more than double bit errors, upto more complex multiple bit error patterns. The detailed algorithm and an example of the proposed error correcting code, very well explains the working flow of the proposed scheme.

The proposed 3DH code can be implemented to protect the configuration bit-

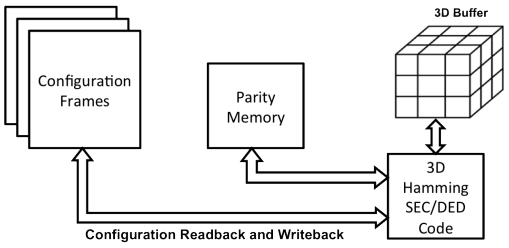


Figure 5.1 – General architecture that implements the 3DH error correcting scheme, proposed for SRAM-based FPGAs configuration memory protection.

stream of an FPGA, provided that the architecture contains a 3D memory whose buffer provides bit-wise access to memory bits in all three directions (X, Y, and Z). This is in contrast with a conventional 2D memory which can provide only one dimensional access to its data [113]. Should a similar solution be applied to the 3D memory, accessing configuration bits in the remaining two directions (Y and Z) would require multiple swapping of error correcting buffer contents which would significantly increase the access time. Because our work focuses on modeling a built-in configuration fault mitigation scheme, the design aspects of the 3D SRAM memory are omitted, as they can be found e.g. in recent works [114, 120], which discuss both the design of 3D SRAMs and their performance improvements. The general architecture that implements the proposed 3DH error correcting scheme to protect the configuration memory of SRAM-based FPGA is shown in Fig. 5.1.

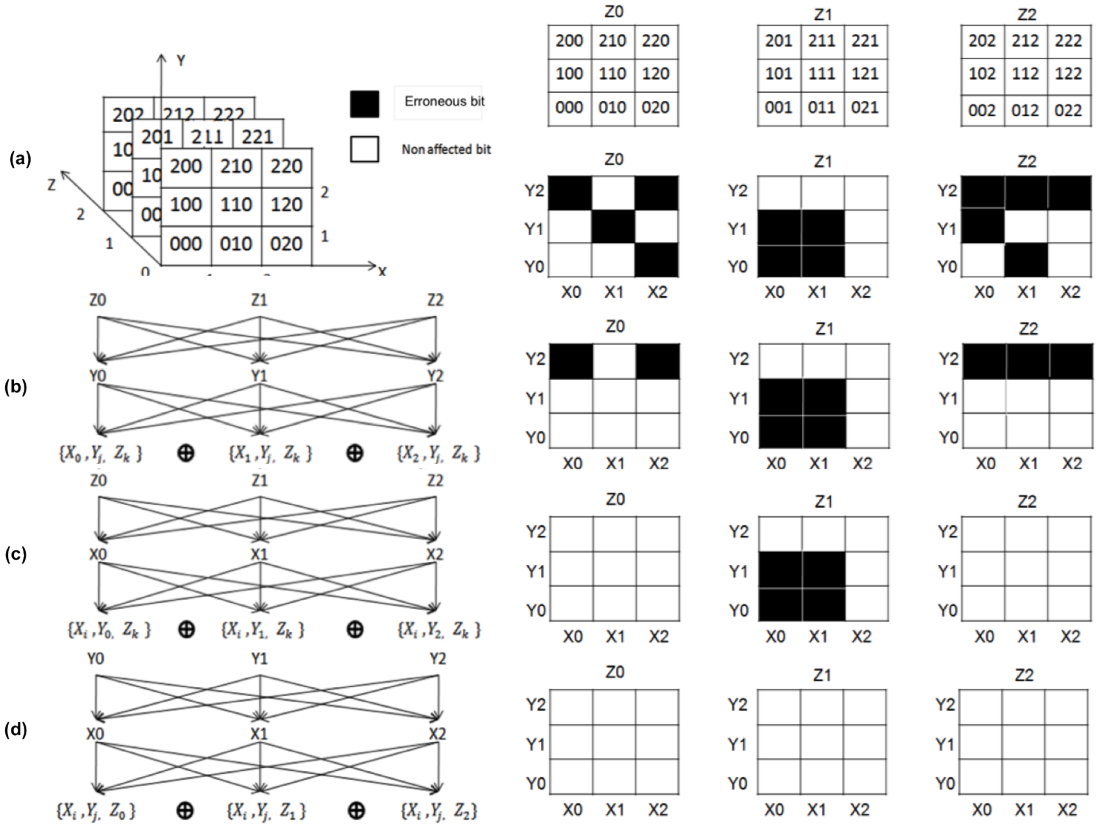


Figure 5.2 – Illustration of the 3D Hamming error correcting scheme: (a) Bit positions in terms of 3D co-ordinates (X, Y, and Z) and random errors introduced in data frames; (b) Error correction results after X-axis computation; (c) Error correction results after Y-axis computation; (d) Error correction results after Z-axis computation.

Bit Arrangements

The principles of error correction of the proposed scheme is illustrated in a simple example of the $3 \times 3 \times 3 = 27$ bit array, shown in Fig. 5.2. Fig. 5.2a shows the

arrangements of configuration bits in the 3D buffer, how bit positions are enumerated in terms of coordinate values (X, Y, and Z), and some randomly introduced multiple errors (shown as black boxes in Figure 5.2). For the total of 27 data bits, 13 bit errors are introduced which, despite they constitute almost 50% of all bits, still all can be corrected.

X-axis Hamming SEC/DED Computation

Z = 0		
Y = 0	Y = 1	Y= 2
SEC{{000}, {010}, {020}}	SEC{{100}, {110}, {120}}	SEC{{200}, {210}, {220}}
"0X0"	"1X0"	"2X0"

Z = 1		
Y = 0	Y = 1	Y= 2
SEC{{001}, {011}, {021}}	SEC{{101}, {111}, {121}}	SEC{{201}, {211}, {221}}
"0X1"	"1X1"	"2X1"

Z = 2		
Y = 0	Y = 1	Y= 2
SEC{{002}, {012}, {022}}	SEC{{102}, {112}, {122}}	SEC{{202}, {212}, {222}}
"0X2"	"1X2"	"2X2"

Figure 5.3 – Sequence of X-axis Hamming SEC/DED Computation

As defined by the proposed 3DH algorithm, at first, the Hamming SEC/DED codec, performs error correction along the X co-ordinate of the 3D buffer, word by word. The error correction phase involves, check bit syndrome generation, error detection, bit localization and single bit error correction (flipping the faulty bit). This process is done in all the Z frames, word by word, starting from frame Z0 till frame Z_{n-1}, where $'n'$ refers the value of the 3^{rd} co-ordinate (Z). In the given example, the 3^{rd} co-ordinate value, i.e, the depth of the 3D buffer is kept as 3. Hence, the hamming SEC/DED operation is performed from the word {{000},{010},{020}} in the frame Z0, till {{202},{212},{222}} in the frame Z2. The X-axis word accessing sequence is given in the illustration fig 5.3. The single bit errors located in the bit locations 020, 110, 012 and 102 belongs to the words in the X co-ordinate {{000},{010},{020}}, {{100},{110},{120}}, {{002},{012},{022}} and {{102},{112},{122}} respectively, are cleared during the process of X-axis hamming SEC/DED computations. Fig. 5.2b shows the results of the Hamming check bits generation along the X-axis, which allow to correct all single bit errors along the X-axis.

Y-axis Hamming SEC/DED Computation

Once all the single bit errors are corrected along the X-axis, the same operation is performed along the Y-axis word by word. It has to be noted that Y-axis words are column based bit arrangements, contradictory to the previous X-axis words, due to the nature of 3D arrangement and bit accessible memory buffer. The Y-axis hamming SEC/DED computation is performed word by word starting from

Z = 0		
X = 0	X = 1	X= 2
SEC{{000}, {100}, {200}}	SEC{{010}, {110}, {210}}	SEC{{020}, {120}, {220}}
"Y00"	"Y10"	"Y20"

Z = 1		
X = 0	X= 1	X= 2
SEC{{001}, {101}, {201}}	SEC{{011}, {111}, {211}}	SEC{{021}, {121}, {221}}
"Y01"	"Y11"	"Y21"

Z = 2		
X = 0	X = 1	X= 2
SEC{{002}, {102}, {202}}	SEC{{012}, {112}, {212}}	SEC{{022}, {122}, {222}}
"Y02"	"Y12"	"Y22"

Figure 5.4 – Sequence of Y-axis Hamming SEC/DED Computation

frame Z0 till frame Z_{n-1}. The single bit error correction is performed from the word {{000},{100},{200}} in the frame Z0, till {{022},{122},{222}} in the frame Z2. The Y-axis word accessing sequence is given in the illustration fig 5.4. Some of the non-correctable errors in the X-axis hamming SEC/DED computation are seen as correctable single bit errors in the column based (i.e, Y-axis) hamming SEC/DED process. For example, the error bits in the positions 200 and 220 belonged to the same word during X-axis hamming SEC/DED computation, where as, during the Y-axis hamming SEC/DED computation, bit positions 200 and 220 belongs to two different words, i.e., {{000},{100},{200}} and {{020},{120},{220}} respectively. This virtual separation helps in clearing the errors of more than one bit. In the similar way, the triple error in the bit positions 202, 212 and 222 belonged to the same word during the X-axis computation, where as, it is seen as clearly single bit errors during the Y-axis computation, hence it is cleared using the simple SEC/DED operation. The results of Y-axis computation for the given example is shown in fig 5.2c. It has to be noted that, even after the Y-axis computation, there are still some error patterns which stay uncorrected. To achieve better reliability, such complex patterns also has to be corrected. From the recent experimental results [21, 22, 26, 115, 116, 118, 122], it is evident that bunch of adjacent bits are getting affected more due to single radiation event. When such multiple bit upsets (MBUs) are arranged in local buffer for error correction, it is more obvious that the possibility of getting non-correctable error patterns are increased. This is a main reason to have 3^{rd} (Z) axis data arrangement and correction plans.

Z-axis Hamming SEC/DED Computation

The Z-axis data arrangement virtually breaks/separates complex error patterns, so that it can be treated as single bit errors in their respective words and corrected eventually. As shown in the given example, after X and Y-axis hamming SEC/DED computation, there is still a non-correctable 4-tuple error pattern, formed in the bit positions 101, 111, 001 and 011 of the frame Z1. These bit positions are left

Y = 0		
X = 0	X = 1	X= 2
SEC{{000}, {001}, {002}}	SEC{{010}, {011}, {012}}	SEC{{020}, {021}, {022}}
"00Z"	"01Z"	"02Z"

Y = 1		
X = 0	X= 1	X= 2
SEC{{100}, {101}, {102}}	SEC{{110}, {111}, {112}}	SEC{{120}, {121}, {122}}
"10Z"	"11Z"	"12Z"

Y = 2		
X = 0	X = 1	X= 2
SEC{{200}, {201}, {202}}	SEC{{210}, {211}, {212}}	SEC{{220}, {221}, {222}}
"20Z"	"21Z"	"22Z"

Figure 5.5 – Sequence of Z-axis Hamming SEC/DED Computation

uncorrected by the X and Y-axis computation, because they correspond to a so far non-correctable error pattern: multiple errors in adjacent bit positions in adjacent rows and columns. This is due to the natural fact of the hamming SEC/DED error correcting code, i.e., is the ability to correct only single bit error in any given word to be processed.

To deal with such errors, the same hamming SEC/DED operation is performed along the Z axis, word by word from {{000},{001},{002}} of frame Y0, to {{220},{221},{222}} of frame Y2. The Y co-ordinate value is considered as the height of the 3D buffer. The Z-axis word accessing sequence is given in the illustration fig 5.5. The result of Z-axis hamming SEC/DED computation is shown in fig 5.2d.

5.1.1 Dealing with Non-correctable Error Pattern

The seemingly non-correctable 4-tuple error pattern are handled, as explained in Fig. 5.6. Indeed, the same erroneous bits can be arranged in a correctable format, should this error pattern be viewed along the Y-axis (the latter can be seen as nothing else but virtual breaking of the group of non-correctable errors and dispersing them in Y-frames, so they could become correctable). The error bit 101, 111, 001 and 011 looks adjacent and in the same frame (Z1), when it is viewed from axis X as well as Y, hence it seems like a non-correctable 4-tuple error. When the same bits are viewed from a 3rd angle (Z-axis) in the 3D buffer, the bits 001 and 011 belongs to frame Y0 and 101 and 111 belongs to frame Y1. Hence applying hamming SEC/DED to words {{000},{001},{002}} , {{010},{011},{012}}, {{100},{101},{102}} and {{110},{111},{112}} will eventually clear the errors in the bit positions 001, 011, 101 and 111 respectively, considering them as single bit errors in their respective words. Then, these errors can eventually be corrected along the remaining 3rd axis Z.

In summary, the sample configuration data can be completely recovered even from such a large number of errors. In this example of the window of size $3 \times 3 \times 3$, the proposed scheme has recovered the 27 (3^3) information bits out of randomly

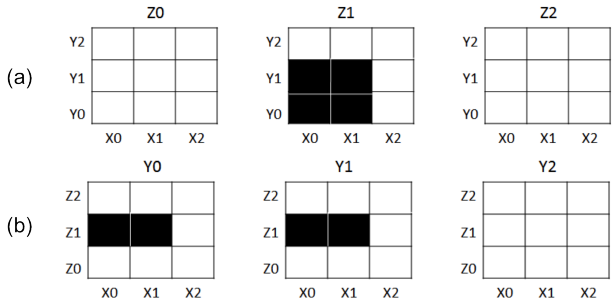


Figure 5.6 – Dealing with seemingly non-correctable error patterns: (a) Non-correctable 4-tuple error pattern in Z-frame; and (b) The same error pattern which is correctable in Y-frames

introduced 13 errors. More complex error patterns can be corrected by performing the 3DH correction iteratively and the actual number of iterations can be considered as the performance parameter. There are also some cases for which the proposed 3DH scheme fails to recover the data, but the occurrence percentage of such non-correctable error patterns is significantly smaller compared to other 1D and 2D Hamming error correcting schemes.

5.1.2 Optimizing the Number of Computation

The number of hamming SEC/DED computations performed in the 3DH scheme can be reduced/ optimized by involving an extra decision-making intelligence before the Z-axis computation. It is nothing but being informed about the non-correctable multiple errors detected during the Y-axis hamming SEC/DED computation.

Recording Y co-ordinate value (refer step 6 in the Section 5.1.3, 3DH algorithm), when multiple error detected in Y-axis computation helps us in reducing the number of computations in Z- axis. Let us consider an example, a error pattern after X and Y axes Hamming SEC/DED computation, in Z planes as shown in figure 5.7.

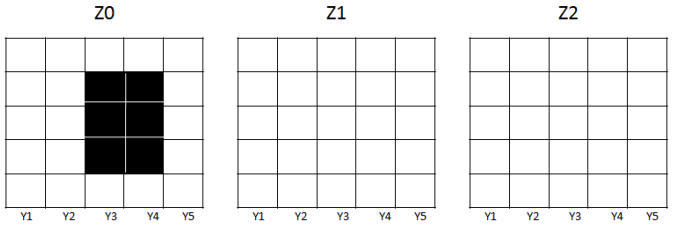


Figure 5.7 – Example Error Pattern- Reducing No. of Computation

If we continue to perform Z-axis computation blindly, it takes 25 (5×5 words in Z direction) Hamming SEC/DED computations to clear the errors. Instead of that, performing Z-axis computation only on the recorded Y values, needs only 10 computations. The recorded Y co-ordinate values can be kept in a local temporary register during the process of syndrome generation along the Y-axis and the recorded values can be fetched back by the control register, to notify the current address of the word to be processed in the Z-axis hamming SEC/DED computation. By this way, the total number of computations can be optimized and the overall performance of the proposed error correcting scheme can be improved.

5.1.3 Detailed Algorithm of 3DH Code

The proposed 3DH error correcting scheme of the FPGA's configuration bitstream is summarized as the following algorithm.

Algorithm 1

Step 1) Read the N-bit configuration data frame through the serial read-back bus along with corresponding parity bits stored in a separate parity memory array.

Step 2) Perform the Hamming SEC/DED coding on the N-bit data and store the resulting data in the 3D N-bit buffer. [Performing Hamming SEC/DED computation in the X axis words of the 3D buffer]

Step 3) Repeat Steps 1) and 2) until the buffer is full (for the 3D buffer formatted with suitable coordinate values see Fig. 5.2a).

Step 4) Read one by one the words of the 3D buffer along the Y axis and perform Hamming SEC/DED coding on them (all single bit errors are corrected during this operation).

Step 5) If no multiple bit error is detected, perform Step 4) until the 3D buffer is fully scanned along the Y axis.

Step 6) If multiple bit error is detected, record the current Y coordinate value and continue the same till the buffer is fully scanned along the Y axis.

Step 7) Take the recorded Y coordinate value and perform the Hamming SEC/DED coding along the Z axis for that particular Y value.

Step 8) Repeat Step 7) for all previously recorded Y values.

Step 9) Once a full cycle of scanning of the 3D buffer along all three axes is completed, proceed to Step 10) if additional iteration is required.

Step 10) Read the words one by one along the X axis and perform the Hamming SEC/DED coding on them.

Step 11) Continue performing Hamming SEC/DED coding along the X axis until the buffer is fully scanned in the X-direction, then proceed to Step 4).

Step 12) If any bit has been corrected, write back the corrected configuration data through the programming bus (thanks to partial reconfiguration).

Step 13) Restart Step 1), for the next frame.

5.2 Functional Implementation of Proposed 3DH Scheme

Implementing the proposed 3-dimensional hamming multiple-error correcting scheme in any FPGA architecture to protect the configuration bitstream, requires some additional considerations. The major point to be considered is the 3D buffer which is being used in the proposed 3DH code. The 3D buffer plays a vital in role in clearing complex multiple bit upset patterns and there by improving the reliability to a greater extend. However, today's conventional FPGA architectures are 2-dimensional by its physical fabrication nature. Hence, the hardware resources and the configuration bitstream plan in the architecture is usually spread in a 2-dimensional (X-Y) spatial plane. Adapting the proposed 3-dimensional hamming scheme in such conventional 2D FPGAs, require additional supportive mechanisms.

5.2.1 Block Diagram and Description

The detailed block diagram of the proposed 3 dimensional multiple-bit error correcting Hamming implementation scheme is shown in Fig. 5.8. The functional block and the error correction scheme in the architecture is a tightly attached cyclic process, exactly similar to the functional implementation of 'configuration read-back and verification process' installed in Xilinx architectures. However, the difference between both the configuration bitstream protection schemes comes in the form of handling multiple bit upsets and the usage of configuration bitstream golden copy.

Functional Description

Its two main blocks are the 2D and 3D address decoders. The 2D address decoder is a conventional one which is already available in most of the FPGA devices. Almost in all cases, it is used along with the Internal Configuration Access Port (ICAP) as in Xilinx FPGAs [125]. It is a serial access port used to read-back and verify the configuration bitstream without disturbing the functionality of the FPGA. The 'read-back' is a background process, meant for checking the integrity of the configuration bitstream. In recent FPGAs, the ICAP helps in performing 'CRC' and 'Frame ECC' (Frame Error Correction Coding). Here, in the proposed scheme, the 2D Address Decoder fetches the configuration data, frame by frame and arranges them in the 3D Buffer of predefined size. Conventional 'Frame ECC' schemes, as the one implemented in Xilinx virtex architectures, employ hamming SEC/DED to correct the bit errors in the configuration frames. The 'Frame ECC' process, performs the single bit error correction during the read-back phase of the configuration frames. Usually, frame is the smallest readable/ writable (configurable) unit in the configuration plan. It has to be noted that the 3D buffer's word size need not to be the same as of the configuration frame size. Hence, the hamming SEC/DED computation size various accordingly. Choosing the optimal size of the 3D buffer and SEC/DED scheme is discussed in the section 5.3.

The 3D Address Decoder is specifically designed to interact with the Hamming SEC/DED Codec, to perform the proposed multiple bit error correction scheme on the configuration data. It generates the address to access the words in all three (X, Y and Z) directions of the 3D Buffer. Here, a 'word' refers to the size of the

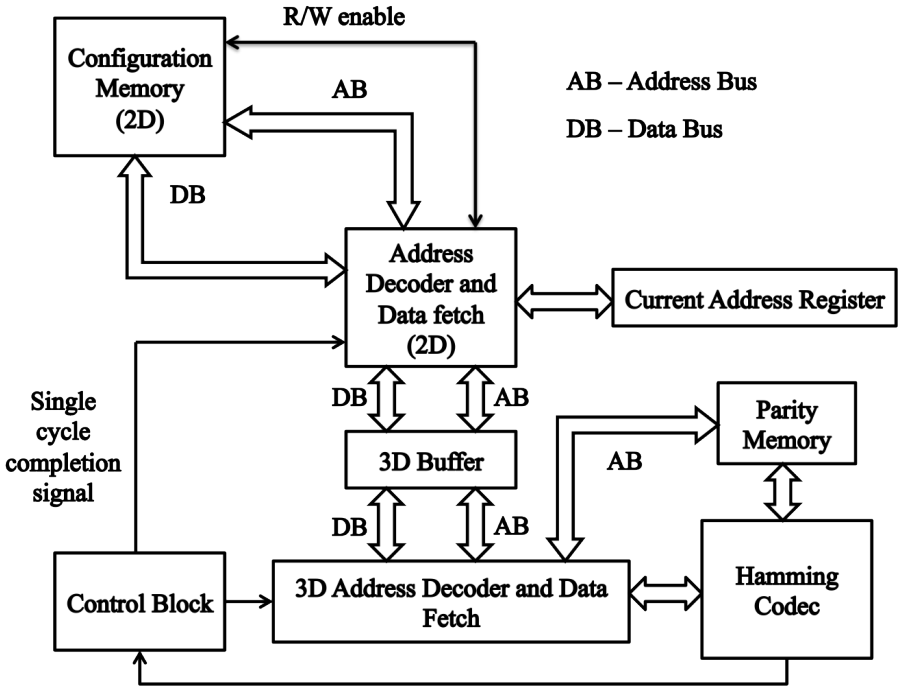


Figure 5.8 – Implementation block diagram of the 3D multiple bit error correcting Hamming scheme

configuration data arranged in the 3D Buffer, to which Hamming SEC/DED coding could be applied (the word sizes in X, Y, and Z axes are respectively n1, n2, and n3). The configuration bits are fetched out word by word from the 3D Buffer according to the address generated by the 3D Address Decoder. The fetched out words are provided to the Hamming SEC/DED Codec. The corresponding set of parity bits are provided to the Hamming SEC/DED Codec. The 3D Address Decoder synchronizes the configuration data fetching (from the 3D Buffer) with the parity data fetching (from the parity memory), to provide appropriate set of configuration and parity bits to the Hamming SEC/DED Codec.

Thereby, the Hamming SEC/DED Codec can generate the syndrome, detect the error, locate the faulty bit and perform the single bit error correction for each given word. The Hamming SEC/DED Codec performs error correction on each word fetched out from the 3D Buffer and generates a local control signal to indicate the completion of error correction on the current word of the 3D Buffer.

The control signal is routed to the Control Block, which in turn, generates two separate control signals to the 2D and 3D Address Decoders. The control signal to the 2D Address Decoder is generated once the error correction is done on all words of the 3D Buffer, which can then fetch the next set of configuration frames from the FPGA configuration memory (2D). The control signal to the 3D Address Decoder indicates the completion of error correction on the current word of the 3D Buffer, which can then fetch the next word from the 3D Buffer to perform error correction.

The Current Address Register keeps the address of a current set of FPGA configuration frames, which are being handled in the 3D Buffer. The Current Address Register helps in re-writing the corrected configuration frames to their original po-

sition in the FPGAs configuration plan, thanks to the partial reconfiguration and configuration write-back features commonly available in contemporary FPGA devices.

5.2.2 3D Virtualization of 2D Memory

As mentioned earlier, the 3D buffer plays an important and inevitable role in the proposed multiple-bit error correcting configuration bitstream protection scheme. To adapt this scheme, either the architecture itself should have the provision of having 3D buffer built-in, or, in the case of conventional 2D architectures, supportive mechanisms have to be added to facilitate the implementation of the proposed scheme. The provision of having dedicated built-in 3D buffers are possible in the case of custom made architectures as of ARDyT FPGA. The prime motive of ARDyT FPGA is to have a reliable architecture with some dedicated sources added to maintain the desired reliability level.

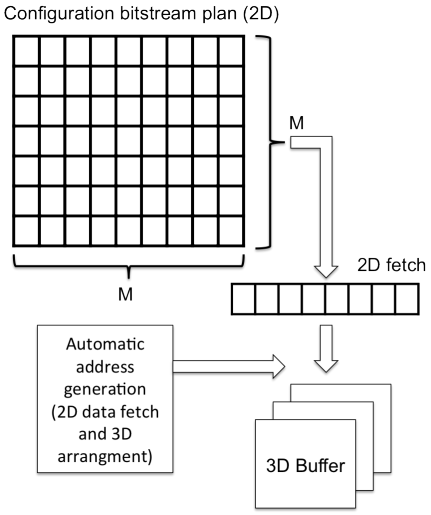


Figure 5.9 – 3D Virtualization of 2D Memory

However, this section describes about achieving such 3D buffer functionality in a 2D architecture itself, hence, making it feasible to apply the proposed 3DH scheme to commercial off the shelf (COTS) FPGAs as well. The figure 5.9 explains the scheme. Functionally, it shows, how the configuration frames arranged in 2D form can be fetched and re-arranged in a 3D format, with the help of appropriate address decoding. Be it the configuration bitstream memory or the 3D buffer, both are physically 2 dimensional memories. The arrangement and addressing of both, are going to make the difference. The smallest accessible unit of the configuration bitstream (frame), is read by the conventional read-back port and the its been re-arranged with the help of a address generator, in such a way that it virtually behaves like a 3D buffer.

For instance, lets consider a configuration bitstream of total 1331 bits to be re-arranged in such a format to form a virtual 3D-buffer, as shown in Figure 5.10 and 5.11.

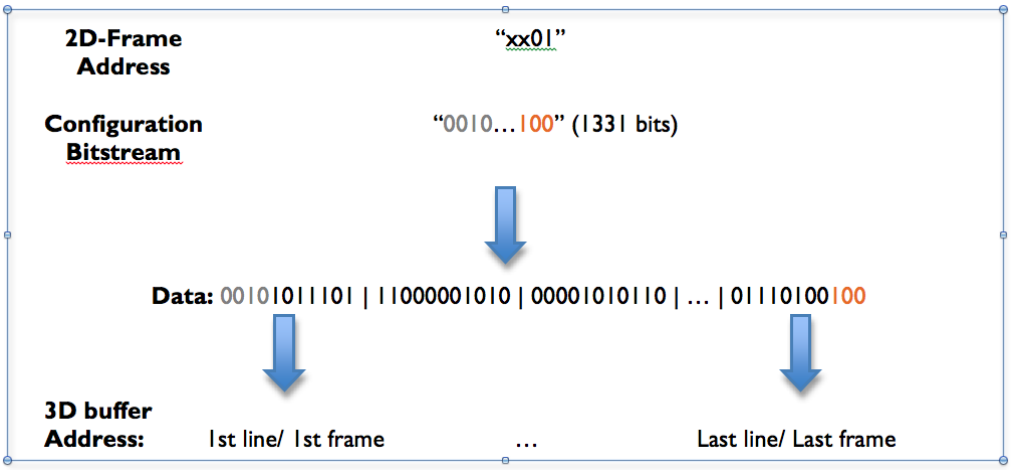


Figure 5.10 – 3D Virtualization Pattern

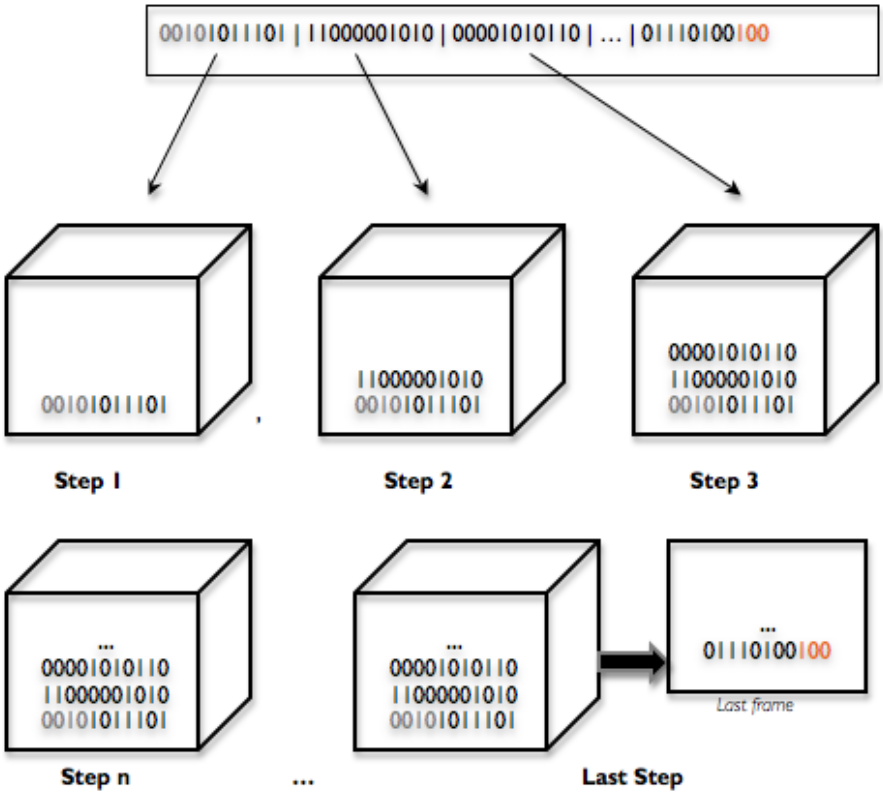


Figure 5.11 – 2D to 3D virtualization

The resultant 3D-buffer is $11 \times 11 \times 11$ in size, with same co-ordinates values for X, Y and Z. All-together, the 3D-buffer will have 121 X-axis words, 121 Y-axis words and 121 Z-axis words, of 11 bit each, where every bit in the 3D-buffer is a part of 3 words (one in each axis).

5.2.3 Calculation & Storage of Parity Bits

Parity/ check bits are one of the prime and very important part of this whole error correction scheme, without which, it is impossible to perform error detection/correction. Usually, parity bits are generated during the configuration bitstream generation process itself. There are some incorporated tool-set which generates corresponding parity bits along side the functional configuration bitstream. As the leading FPGA vendors like Altera, Xilinx have already incorporated hamming SEC/DED based error correcting schemes to protect the configuration bitstream, their CAD tools have built-in support to generate these parity bits, with some predefined design / input / functional constrains.

For the proposed 3DH scheme, the parity bits have to be generated for the words in all three co-ordinates (X, Y and Z). This is contradictory to the conventional parity generation, where the parity bits are generated only for one directional data. More precisely, the parity bits are generated for the frame sized configuration bitstream. Along with frame ECC, checksum values are also generated for each frame, to perform frame based CRC.

However, for the proposed scheme, apart from the configuration frame size, size of the 3D buffer and number of frames/buffer, also need to be determined before configuring the chip, to generate appropriate parity bits. Generally, parity bits are generated by estimating the hamming syndrome for the given string of bits. It has to be noted that, parity bits are used to protect the configuration bitstream, but how the reliability of those parity bits could be maintained?. Thanks to Hamming, by nature, the hamming code deals with the errors in information bits as well as the parity bits. Hence, there is no need for another mechanism to protect the parity bits.

The pre-calculated parity bits can be part of configuration data and it can be loaded into the device during the power on. The generated parity bits can be stored in a dedicated parity memory in the architecture or it can be kept in the internal block random access memories (BRAM). Generating and programming parity bits are not of a big concern, where as, the parity memory overhead would be. Hence, the optimization of parity memory overhead is discussed in the following section.

5.3 Optimal Size of the 3D Buffer and the Parity Memory Overhead

The size of the configuration data frame varies depending on the family of FPGA devices. Consequently, the number of check bits depends directly on the size of the configuration data frame and the error protection scheme used, like CRC and Frame ECC; for instance, in all Virtex 7 series FPGA devices: (i) all frames have a fixed, identical length of 3,232 bits and (ii) a 13-bit Hamming code and a 32-bit CRC with read-back are used for error detection and correction [74, 125].

Figure.5.12 shows three multi-dimensional organizations of data bits. Figure 5.12a represents a word of n1 bits, 5.12b represents a frame of n2 words, 5.12c represents a buffer (3D collection) of n3 frames. (Note: The 'frame' mentioned here is to describe the 2D arrangement of configuration bits, not be confused with the

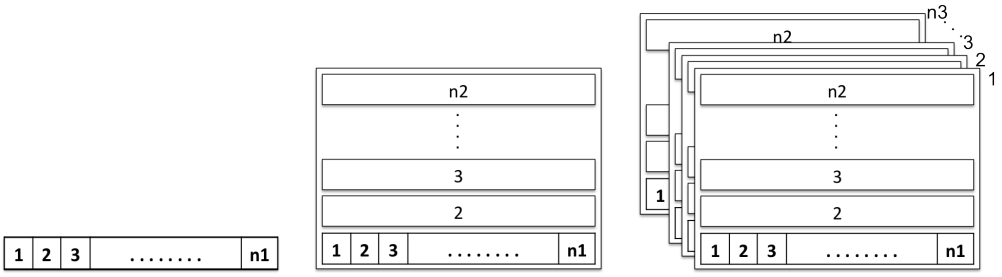


Figure 5.12 – Configuration organization: (a) 1D word; (b) 2D frame; and (c) 3D buffer

term configuration 'Frame' used in conventional configuration plan). Applying error correcting codes like hamming SEC/DED, only in the horizontal word format (1- directional word by word) requires the parity bits generated only for horizontal words. Such error correcting scheme is described as 1-directional hamming SEC/DED. For instance, the Xilinx virtex 7 series FPGAs employs this scheme; generating 13 parity bits for each frame of size 3232 bits to perform frame based hamming SEC/DED. The 2-directional hamming SEC/DED schemes, performs the syndrome generation in both horizontal as well as vertical axis of the 2D plane. That is, 2D schemes apply hamming SEC/DED on row-wise words and column-wise words, hence row-wise and column-wise parity bits also have to be generated. The parity memory overhead of such 2D schemes are calculated as described in [66]. The proposed 3-directional hamming scheme has the configuration bitstream arranged in 3-directional buffer and the hamming SEC/DED is performed in all three direction word by word. Apparently, the proposed 3DH scheme needs the parity bits to be generated, for the correspond words, in all three direction.

In the 3D buffer scheme proposed here, the overall parity memory overhead is given by

$$P_{3D} = (n1 \cdot k2 + n2 \cdot k1)n3 + n1 \cdot n2 \cdot k3, \tag{5.1}$$

where ki is the number of parity bits of the SEC/DED Hamming code for ni data bits, $i = 1, 2, 3$. The total parity memory overhead of the proposed scheme for a particular FPGA device is

$$Total_{POH} = P_{3D} \cdot \frac{\text{Total number of configuration frames in FPGA}}{\text{Number of frames per 3D buffer}} \tag{5.2}$$

Perfect SEC/DED Hamming code

This overhead depends on the choice of the co-ordinate values of the 3D buffer. Of particular interest is the so called *perfect SEC/DED Hamming code* for which the condition $n = 2^{k-1} - k$ holds, which protects the maximal number of data bits n for a given number of parity bits k. For instance, the minimum of $k = 6$ parity bits suffice to protect $n = 12$ data bits (which is the minimum for $k = 6$) although the same number of parity bits are also required to protect up to $n = 26$ data bits (the maximum for $k = 6$). To convey a reader with the importance of using perfect Hamming codes (or codes as much close as possible to them), we have

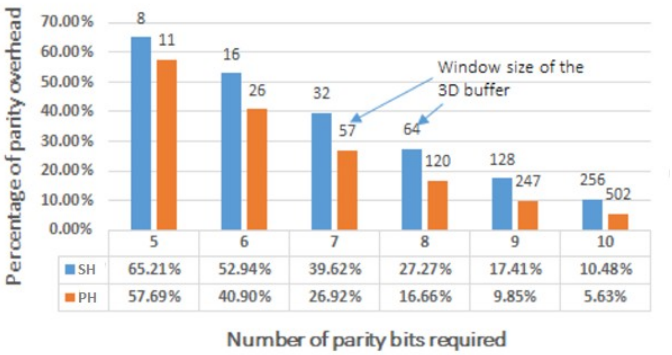


Figure 5.13 – Comparison of the parity memory overhead between 3D encoding using perfect Hamming codes (PH) and standard Hamming codes (SH) applied to $n = 2^j$ data bits.

considered two classes of 3DH schemes applied to 3D data cubes of size n (i.e. with $n1 = n2 = n3 = n$ and hence $k1 = k2 = k3 = k$): one for commonly used $n = 2^j$ and the other for n corresponding to perfect Hamming codes.

Fig. 5.13 shows the parity memory overhead for either class of codes for k ranging from 5 to 10. Clearly, choosing the sizes of the 3D buffer which meet the perfect Hamming code condition can lead to significant overhead reduction.

Unfortunately, choosing the 3D buffer size which meets exactly the perfect Hamming code condition is hardly feasible for existing devices. It greatly depends on the frame size of the FPGA's configuration plan. Nevertheless, it is desirable to choose the co-ordinates (n1, n2, and n3) of three axes (see Fig.5.12) such that each of three sizes of the 3D buffer match the perfect Hamming code condition as closely as possible.

```
                ============================================
                 Optimal Values of the 3D buffer co-ordinates
                ============================================
                     For Xilinx Virtex 7 series devices

Maximum number of frames/buffer = 64

Bits/frame = 3232

Enter the maximum limit for co-ordinates : n1, n2 and n3 = 128 128 128

The total configuration bits protected/buffer = 206848

The possible co-ordinate values(n1, n2 and n3) are,
           ---------------------------------------------------------
           Co-ordinate values of 3D Buffer       Total parity bits
           ---------------------------------------------------------
 Solution 1 =>  101      32      64              87488
 Solution 2 =>  128      101     16              108496
           ---------------------------------------------------------
Total number of optimal solution = 1
```

Figure 5.14 – Optimal values of the 3D buffer co-ordinates

To do this, a program has been developed which, upon providing the parameters 'Number of frames/buffer' and 'frame size (bits/frame)' of the FPGA, generates

the list of possible near optimal solutions, as shown in Fig. 5.14. where 'Number of frames/buffer' is a design parameter which can be decided by the developer, according to the application and the deploying environment. The time taken to complete a full error correction cycle of the whole configuration memory depends on the number of frames, bits/frame and the number of frames/buffer. Here, the solution resulting in the minimal parity memory overhead is called optimal.

As an example, the best sizes (n1, n2, n3) of the 3D buffer for the parameters of Virtex 7 series FPGAs [119] are estimated, by providing 'Number of frames/buffer' and 'frame size (bits/frame)' of the FPGA as the input parameters. Two best arrangements of 64 configuration frames of 3232 bits in the 3D format found were (32, 64, 101) and (16, 101, 128), where all permutations of (n1, n2, n3) would obviously result in the same overheads respectively 87488 and 108496 check bits. The first triple involves the minimum parity overhead equal to 42.2%, which is significantly larger than to implement error handling mechanisms used in the Virtex 7 series devices (the Frame ECC and CRC). Nevertheless, the latter error correcting scheme cannot handle multiple bit errors as effectively as the proposed scheme, because CRC must be supported by read-back requiring external circuitry to perform error correction.

5.3.1 Parity Memory Overhead Comparison

According to the theory of Hamming code,

$$2^C \geq N + C + 1 \tag{5.3}$$

where, C - is the number of required parity bits ; N - is the total number of data bits protected. The parity bits required for different data width (N) is shown in table 5.1. An additional memory space is required in the FPGA, to keep this parity bits. The memory overhead due to this additional parity bits decreases with the increased size of data width. This is due to the logarithmic relationship of the number of parity bits to the protected data as shown in table 5.1.

Table 5.1 – Parity bit estimation (based on equation 5.4)

Data width (N)	Parity bit for SEC	Parity bit for SEC/DED
1	2	3
2 - 4	3	4
5 - 11	4	5
12 - 26	5	6
27 - 57	6	7
58 - 120	7	8
121 - 247	8	9
248 - 502	9	10

SEC- Single Error Correction ; SEC/DED - Single Error Correction Double Error Detection

The actual memory overhead of the proposed 3DH scheme is listed in table 5.2 for different parity bit size. The parity memory overhead can be further optimized

by using the concept of *perfect hamming*. The condition of *perfect hamming* gives "the minimum number of parity bits required to protect the maximum number of data bits". According to Hamming theory, the *perfect hamming* condition is,

$$2^C = N + C + 1 \tag{5.4}$$

The table 5.2 and figure 5.15 shows, how the parity memory overhead is reduced by utilizing the condition of perfect hamming. The memory overhead of the proposed 3DH scheme is almost equal to the overhead of 2DH scheme, whereas, in-terms of error correcting efficiency, the 3DH scheme is better than the later one. Also, it should be noted in the Figure 5.15 that, the parity memory overhead of the 3DH is lesser than the 2DH scheme, when the computational data width increases for perfect hamming.

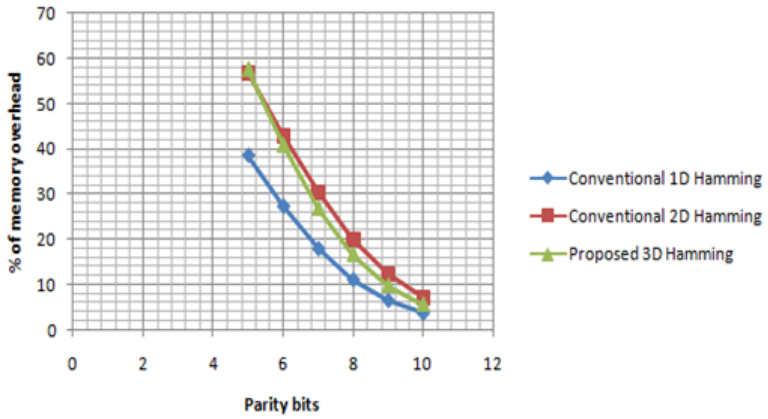


Figure 5.15 – Comparison of parity memory overhead

Table 5.2 – Parity memory overhead comparison

Parity bits per word	Total parity memory overhead			
	1DH Scheme	2DH Scheme	3DH Scheme	
			Binary weighted	Perfect hamming
5	38.46%	56.75%	65.21%	57.69%
6	27.27%	42.85%	52.94%	40.90%
7	17.94%	30.43%	39.62%	26.92%
8	11.11%	20.00%	27.27%	16.66%
9	06.56%	12.32%	17.41%	09.85%
10	03.75%	07.24%	10.48%	05.63%

Lets have an example case from the table 5.2. Considering the same co-ordinate values to X, Y and Z, the parity bits required to perform hamming SEC/DED computation on all the words in all three directions are same. Lets assume a parity bit size of 7, and hence, the parity memory overhead is 17.94% , 30.43% , 39.62% and 26.92% for 1D, 2D, binary weighted 3D and perfect hamming 3D respectively. It is evident that the proposed 3D scheme poses significantly less parity memory

overhead (i.e., 26.92%) when follows perfect hamming condition, compared to the 2D scheme (i.e., 30.43%). However, in a practical case, it is difficult to achieve exact perfect hamming condition while determining the 3D buffer size. Nevertheless, a nearest possible perfect hamming condition can be determined using the optimal 3D buffer size calculator as shown in figure 5.14.

5.3.2 Hardware Implementation Details

A test case evaluation is done by implementing the proposed 3DH configuration bitstream protection scheme for the optimal 3D buffer size of (X, Y, Z) co-ordinates {101, 32, 64 } for Xilinx Virtex 7 series device.

By adapting the proposed 3DH scheme to the Xilinx architecture, the already existing internal configuration access port (ICAP) is used for accessing the configuration frames. The configuration bits are fetched by the conventional read-back process frame by frame. The fetched configuration frames are mapped to a virtual 3D arrangement with the help of 3D-address decoder (designed for specifications of optimum 3D buffer size). The 3D-address decoder generates (n,k,p) address, each of $3 \times (4)$ bits wide. Here, 'n' is the column component (along X-Axis), 'k' is the line component (along Y-Axis) and 'p' is the frame component (along Z-Axis). The three parameters, (n,k,p) represents an address in the 3D-Buffer. There is always a parameter equal to null. The null parameter means that the word is along this axis and other parameters indicates his position in the 3D-Buffer (e.g. (n,k,p) = (1,0,3) means that the word is along the Y-Axis and it is positioned in column 1 of frame 3).

Slice logic utilization		Logic distribution (No. of LUT-FF pair used)			Specific feature utilization	
No. of slice Registers	No. of slice LUTs	No. with unused FF (73%)	No. with unused LUT (6%)	No. of fully used pairs	No. of Block RAM (2%)	No. of DSP
444 /400K	1500 /204K	1244 /1688	108 /1688	336 /1688	21 /750	5 /1128

Table 5.3 – Hardware resource utilization summary

The parity bits are generated at the time of configuration bitstream generation itself, with respect to the pre-calculated optimal co-ordinate values (X,Y,Z) of the 3D buffer. The generated parity bits are kept in Block RAMs (BRAMs) available in the architecture. The *Hamming* codec circuit is implemented to generate syndrome and to perform error detection and correction. The design summary report in Table 5.3 shows the hardware resource utilization to implement the proposed 3DH scheme in a commercial FPGA; virtex 7 series device - 7vx330tffg1157-3 (with 3D buffer coordinate size 101, 32 and 64 for X, Y and Z coordinates, respectively). The design simulations are done by manual fault injection in the configuration bitstream. According to timing summary, the maximum frequency of operation is 148.302 Mhz (equivalent to 6.743ns clock period). As the reconstruction of words in virtual 3D-buffer requires more clock edges, it evident that the timing performance is not so good. Better timing performance could be achieved by adapting some timing optimization techniques. In fact, the more time consuming task of "2D-to-3D"

and "3D-to-2D" word reconstruction can be avoided if the proposed 3DH scheme is incorporated with real 3D memory architectures.

5.4 Reliability Improvement

The reliability improvement of the proposed scheme is evaluated by comparing the percentages of non-correctable error patterns in 1D, 2D, and 3D Hamming code error correcting schemes. Some figures are listed in Table 5.4 for 3D data cubes with $n = 3$, 4, and 5, where the occurrence percentage of N_{ep} is the ratio of the total number of non-correctable error patterns to the entire sum of correctable and non-correctable error patterns for the given bit/word/window size of the data. All possible error patterns were generated and the N_{ep} values were obtained using MATLAB®. Unfortunately, estimating non-correctable patterns for $n \geq 6$ turned out computationally too complex. However, the trend of estimated 3 set of values help us in predicting the N_{ep} for the higher numbers.

Table 5.4 – Comparison: non correctable error patterns of 1D, 2D and 3D hamming scheme

	1DH scheme		2DH scheme		3DH Scheme	
n	T_b	$\%N_{ep}$	T_b	$\%N_{ep}$	T_b	$\%N_{ep}$
3	3	0.5	9	0.1718	27	0.0086
4	4	0.6875	16	0.2368	64	0.0060
5	5	0.8125	25	0.2384	125	0.0053

T_b - Total bits protected (Hamming size) ; $\%N_{ep}$ - % of non correctable error patterns

Table 5.4 reveals decreasing nature of the percentage of non-correctable error patterns with the increase of the 3D buffer size. Recall that the simple Hamming SEC/DED scheme is capable of correcting only single-bit errors in a word and that the percentage of non-correctable error patterns in it is very high. As for the 2D Hamming code, any multiple bit error in more than one adjacent row/column is uncorrectable [66]. The data listed in Table 5.4 show that the percentage of such uncorrectable error patterns is relatively high. On one hand, Table 5.4 reveals that the ratio of non-correctable error patterns in 1D and 2D schemes grows with the increase of word and window size, respectively. On the other hand, it shows that the ratio of non-correctable error patterns in the proposed 3D Hamming code is not only very small (less than 1%) but also, unlike the other two schemes, it tends to decrease with the increase of the window size. Clearly, the multiple bit error correcting efficiency of the proposed scheme is significantly higher than of the other schemes.

5.5 Fault Mitigation through R3M: Integrating the Proposed Configuration Protection Scheme in ARDyT FPGA

In the proposed 3-dimensional hamming based configuration bitstream protection scheme, detection and correction of errors happen as a continuous cyclic process, due to the fact that hamming error detection correcting codes, not just detect the error, but locates it's bit position as well in the (configuration) word being processed. The primary difference and advantage of the proposed scheme against the conventional methodology used in Xilinx architectures, is the technique of not using the external golden copy of the configuration bitstream as depicted in the Figure 5.16. The proposed 3DH scheme provides faster reconfiguration of frames affected by the errors/ upsets, because correction can be done using internal bus alone, unlike most known methods that rely on the external configuration backup and the I/O lines.

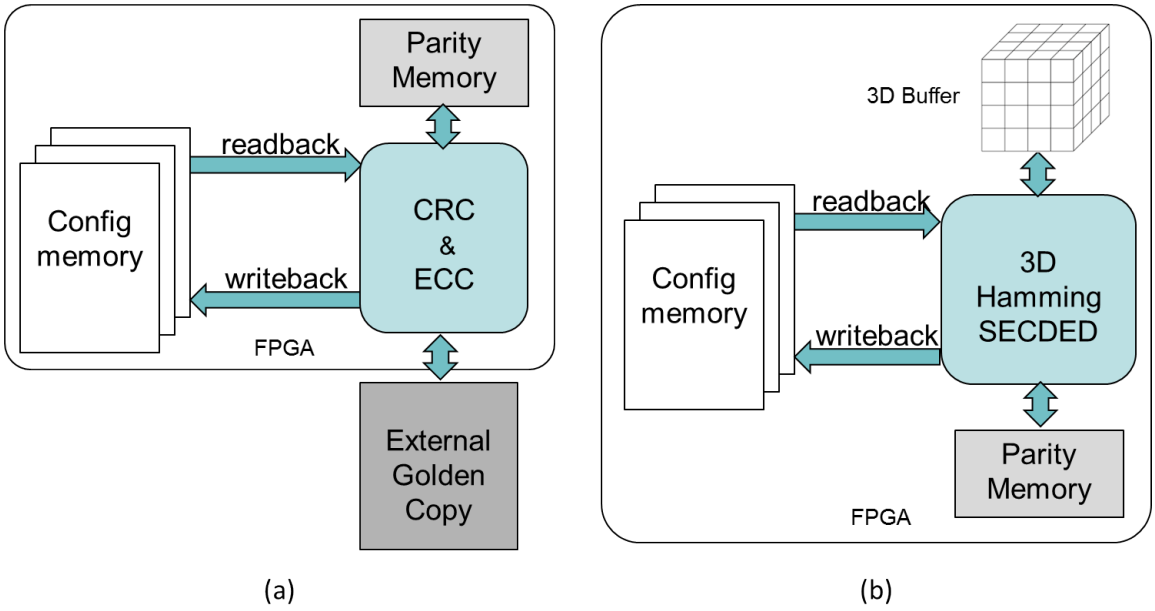


Figure 5.16 – (a) Conventional Xilinx CRC & ECC with external golden copy, (b) Proposed built-in scheme without a golden copy

In ARDyT architecture, the proposed 3DH scheme is incorporated with the internal configuration bitstream access resources (internal configuration access port (ICAP), background read-back and write-back etc.,). The proposed 3DH configuration bitstream protection scheme is locally operated at the architecture level but globally controlled by the centralized reliability manager (R3M) through the dedicated API - *configuration fault mitigation–application programming interface (CFM–API).*

R3M plays an important role in managing the background read-back, configuration bitstream verification, time-shared usage of internal configuration access ports and the synchronization of various partial bitstream configuration. The R3M has a detailed global view of the architecture and different partial bitstream. It also

keeps the track record of the events related to the configuration bitstream in the architecture, such as, tiling structure, dynamically reconfigurable partitions (DRPs) with their list of associated bitstream, relocated tasks due to permanent errors and present set of configuration frames being handled by the proposed 3DH scheme etc.,.

The proposed 3DH configuration bitstream protection scheme is a background process, during run-time. The CFM-API interacts with the introspection plan (detailed in Chapter 3), and co-ordinates the configuration bitstream protection process with the other processes managed by R3M. The internal configuration frame access resources used by the proposed configuration bitstream protection scheme are shared resources, among different fault mitigation strategies through their respective APIs (LFM-API, CFM-API and so on). Control over these shared resources to different APIs are provided by R3M.

Upon system initialization (or system (re)start), R3M initiates the background 3DH configuration bitstream protection process, introspection scheme and other related APIs. The CFM-API instructs the 3DH scheme to start the error correction from first frame address. The 3DH correcting code is performed to first set of frames (from the initial address) and continues the process to the next set of frames and so on, in a sequential manner. Whenever the logic fault mitigation strategy enters the second stage (i.e, error correction in configuration bitstream associated with a particular logic resource), the LFM-API requests the R3M to perform the 3DH error correction in that particular logic area (in the associated set of configuration frames). R3M then interrupts the CFM-API to gain the control over the background 3DH configuration protection scheme and provides a new set of configuration frame address on which the configuration bitstream protection scheme has to be performed. CFM-API handles the interrupt by saving the current frame address of the 3DH scheme and returns the control to R3M. As soon as the 3DH configuration protection scheme completes its operation in the given set of frame address, CFM-API retrieves the previous frame address from where it will resumes the next cycle of 3DH correction.

Similarly, whenever a permanent hardware fault is detected in any logic resource (stage 3 of logic fault mitigation), the LFM-API notifies R3M, to initiate task relocation. At this stage, the task relocation process requires the configuration bitstream access control and relevant resources (ICAP, read-back and write-back control) which are being used by the the 3DH configuration bitstream protection scheme. Hence, R3M interrupts CFM-API to gain the control over the configuration bitstream access resources, to perform task relocation. CFM-API handles the interrupt similar to previous case (stage 2 of LFM-API) and the control over the configuration bitstream access resources are provided to R3M, to perform task relocation in the logic fault mitigation strategy. The R3M communicates and co-ordinates between CFM-API and LFM-API for the time-shared resource usage.

Chapter 6

Conclusions & Further Works

6.1 Main Conclusions & Remarks

Constant technology scaling, aging and characteristic changes in high intense radiation environments increases multi-bit upset (MBU) fault models in the configuration memory of the reconfigurable architectures. Particle radiation is not restricted only to space environment. Even at ground level and high altitudes, natural and artificial particle radiation is been observed. Artificial ground level radiation sources include enriched radionuclides, accelerators, nuclear power plants and nuclear weapons. An example could be *Large Hadron Collider* at CERN. Hence, high reliable electronic systems gain their importance in ground level applications also.

As the market is expanding beyond high-cost space application, there is a need for developing a complete reliable architecture, targeting low-cost safety and mission critical applications. Fabrication process based rad-hard architectures pose high cost and overhead which can not be affordable in many applications. Design based fault tolerance in commercial-off-the-shelf (COTS) architectures pose extra design complexity and reduces design flexibility. Hence, designing new architecture models by adapting suitable reliability support mechanism at different levels (architecture, configuration, application and software) is necessary to achieve less overhead, less complex and flexible design implementations.

Radiation particle strike creates different fault models on different circuit elements which cause different consequences according to their circuit nature. Hence, fault-model-aware fault detection techniques and consequence-aware fault correction strategies are developed.

Recognized the need to develop adaptable fault-aware logic modules where the logic circuits are customized to support fault detection and correction strategies. This will greatly reduce the complexity in application development phase and time-to-market, as the designer does not have to focus on 'resource allocation & utilization' for the reliability aspects (as in the case of designing reliable applications in unreliable architectures).

A trade-off can be maintained between {Reliability level} and {Hardware Overhead, Latency, Energy Efficiency}. It can be collectively represented as *reliability efficiency evaluation factor (REEF)*. It is based on various aspects, including sensitivity of different tasks; monitoring and non-monitoring circuit area; and granularity

of fault detection and correction.

6.2 Summary of Contributions

In ARDyT research consortium, different project partners work at different level of the intended reliable architecture design. This dissertation is aimed at developing reliability mechanisms at the hardware and the configuration layers of the architecture; and linking the fault mitigation process to the dedicated reliability management layer (FTAL). Contributions through this research work to the ARDyT project is summarized here.

Conceptual view of ARDyT hardware architecture is developed. Basic building blocks, hardware hierarchy, reconfigurable partitions and their granularity are defined at the functional level. The architecture is made adaptable to support proposed fault mitigation strategies for configurable logic resource protection and configuration bitstream protection. Granularity of fault detection and notification is precised by defining grouped partial reconfigurable regions (GPRRs) in the architecture. Supportive mechanism to link the hardware modules with the dedicated reliability management layer (FTAL) is detailed. Fault-status reading registers (FSRs) are introduced at GPRR level. The fault occurrence (fault-status) notification is facilitated with the help of an interrogation protocol (introspection plan).

To deal with logic circuit faults and user data upsets, fault-aware configurable logic block (FA-CLB) architecture is proposed. Concept of fault-model-aware fault mitigation strategy is introduced. According to fault models and their consequences, different strategies are formulated separately for combinational and sequential circuit elements. The proposed fault-aware logic block architecture has lesser hardware overhead than DMR and TMR based designs.

To protect the configuration bitstream against single event upsets (SEU), three-directional Hamming-based multiple bit error correcting scheme is proposed which can effectively handle single and multiple bit upsets (SBU & MBU). The primary difference and advantage of the proposed scheme against the conventional methodology used in Xilinx architectures, is the technique of not using the external golden copy of the configuration bitstream. The proposed 3DH scheme is incorporated with the internal configuration bitstream access resources (ICAP, background read-back and write-back etc.,) and completely managed by the centralized reliability manager (R3M) in the FTAL. It provides faster reconfiguration of frames affected by the multiple errors/ upsets, because correction can be done using internal bus alone, unlike most known methods that rely on the external configuration backup and the I/O lines.

6.3 Proposed Topics for Future Research

6.3.1 3D-Hamming in 3D architectures

Feasibility study to adapt the proposed 3-directional hamming (3DH) based multiple-bit error handling technique to real 3D architectures. Non-volatile 3D memory

architectures are being experimented and developed by major players such as Intel's 3D XPoint^{TM} memory technology [126] and ReRAM (Resistive Random Access Memory) technology [127]. Integrating the 3DH-based MBU handling scheme and emerging 3D memory architectures will have great potential in terms of performance and reliability improvements.

6.3.2 Distinguishing Logic and Routing Bitstream

The concept of separately identifying configuration bitstream belongs to logic and routing resources is introduced in Chapter III, as a preliminary proposition. The main advantage of this proposition is the ability to differentiate *'logic circuit faults'* and *'routing faults'*, as the fault models and consequences are uniquely different for both logic circuit resources and routing resources. However the strategy to distinguish and re-group the configuration bitstream is not investigated, as it is not been covered in the scope of the project. However, it has the potential to be well developed as a prospective research topic.

One solution could be, the configuration bitstream can be physically separated in different layers; one layer containing configuration bits belong to logic resource programming and another layer containing configuration bits belong to routing resource programming. Possibly, this could also scale the architecture density. There are some research going on "multiple configuration layer" technology. For example, Dynamic Random Access Memory (DRAM) based 3D stacked memory is proposed as primary FPGA configuration data storage in [128], to store multiple sets of configuration data and to enable high-speed dynamic reconfiguration. Similarly, the authors in [129] propose stacked configuration memory based on nonvolatile Resistive RAM (ReRAM) technology which is compatible and scalable with CMOS process technology. Further research by co-relating this "multiple configuration layer" technology with the proposed "logic and routing configuration bitstream separation" would lead to interesting results.

6.3.3 Fault Mitigation in Routing Resources

Routing resources are one of the important and sensitive resources in reconfigurable architectures. It includes switch boxes, connection boxes, local connection boxes (in ARDyT architecture) and programmable routing lines. It is important to consider the reliability aspects of routing resources which are also prone to radiation-induced faults. There are various fault models that could occur in a routing network. Programmable routing resources can experience single event upsets (SEU), transients and bridging faults. The consequences of faults occurring in routing resources are apparently different from logic circuit faults and configuration bit upsets. Techniques has to be developed to mitigate the routing faults in reconfigurable architectures. A simple solution could be physical duplication of routing lines. However, "duplication with comparison (DWC)" can not be applied in all the cases, as it will drastically increase the overhead and routing complexity. A detailed study has to be carried out, to come up with feasible routing-fault mitigation strategies. Fault tolerant network-on-chip (NoC) architectures; intelligent routing algorithms; self-reliable switch box

and connection box architectures can be developed.

Bibliography

[1] Stephen Brown and Jonathan Rose, "Architecture of FPGAs and CPLDs: A Tutorial", IEEE Design and Test of Computers, Vol. 13, No. 2, 1996.

[2] Russell Tessier, Ian Kuon and Jonathan Rose, "FPGA Architecture: Survey and Challenges, Foundations and Trends in Electronic Design Automation, Vol. 2, No. 2, pp. 135–253, Feb. 2007.

[3] Microsemi Corporation, "Single-event effects in FPGAs: Ground Level and Atmospheric Background Radiation Effects in FPGAs", Technical Report 2007.

[4] Microsemi Corporation, "Understanding Soft and Firm Errors in Semiconductor Devices: Questions and Answers", Technical Report, 2002.

[5] U. Legat, A. Biasizzo, and F. Novak, "Self-reparable system on FPGA for single event upset recovery", Proceedings of 6th International IEEE Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), pp. 1–6, June 2011.

[6] Microsemi Corporation, "Microsemi Programming Antifuse Devices", Application Note AC225, Nov. 2011.

[7] Microsemi, "IGLOO nano Low Power Flash FPGAs with Flash*Freeze Technology", Revision 19, DSO110, Microsemi Corporation, Oct. 2015.

[8] Microsemi, "Military ProASIC3/EL Low Power Flash FPGAs with Flash*Freeze Technology", Revision 5, Microsemi Corporation, Sept. 2014.

[9] Altera Corporation, "Logic Array Blocks and Adaptive Logic Modules in Stratix III Devices", Stratix III devices handbook, Volume 1, May 2009.

[10] Altera Corporation, "MAX II device handbook", V1–3.3, June 2005. [Available Online]: http://www.altera.com/literature/hb/max2/max2 mii5v1.pdf.

[11] Lattice Semiconductor Corporation, "Lattice XP family data sheet", V3.1, Sept. 2005. [Available Online]: http://www.latticesemi.com/lit/docs/datasheets/fpga/xp_data_sheet.pdf

[12] Xilinx, "Spartan-3AN FPGA family data sheet", DS557, Feb. 2007. [Available Online]: http://direct.xilinx.com/bvdocs/publications/ds557.pdf.

[13] Xilinx, "Virtex-5 Family Overview", Product Specification, DS100 (v5.1) August 21, 2015. [Available Online]: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[14] Xilinx, "Virtex-6 Family Overview", Product Specification, DS150 (v2.5) August 20, 2015. [Available Online]: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.

[15] Xilinx, "Virtex-7 Family Overview", Product Specification, DS180 (v1.17) May 27, 2015. [Available Online]: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series _Overview.pdf.

[16] Altera, "Stratix V Device Overview", Product Specification, SV51001,Oct. 2015. [Available Online]: https://www.altera.com/literature/hb/stratix-v/stx5_51001.pdf.

[17] Angela Sutton, "No Room for Error: Creating Highly Reliable, High-Availability FPGA Designs", White Paper, Synopsys, Inc., April 2012. [Available Online]: https://www.synopsys.com/cgi-bin/proto/pdfdla/pdfr1.cgi?file=FPGAhigh-rel.pdf.

[18] Laprie et al., "Taxonomy showing relationship between Dependability & Security and Attributes, Threats and Means", June 2010. [Available Online]: https://en.wikipedia.org/wiki/Dependability#/media/File:Dep1.svg.

[19] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, pp. 11–33, Jan. 2004.

[20] Dhiraj K. Pradhan, "Fault-Tolerant Computer System Design", Prentice-Hall Publications, 1996.

[21] Quinn. H, Graham. P, Krone. J, Caffrey. M, and Rezgui. S, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," IEEE Transactions on Nuclear Science, vol. 52, no. 6, pp. 2455–2461, Dec. 2005.

[22] Quinn. H, Morgan. K, Graham. P, Krone. J, and Caffrey. M, "Static proton and heavy ion testing of the Xilinx Virtex-5 device", IEEE Workshop on Radiation Effects Data, vol. 0, pp. 177–184, July 2007.

[23] M J Wirthlina, H Takaib and A Hardinga, "Soft error rate estimations of the Kintex-7 FPGA within the ATLAS Liquid Argon (LAr) Calorimeter" Topical Workshop on Electronics and Particle Physics (TWEPP'13), Perugia, Italy, 23–27 Sept. 2013.

[24] Fernanda Lima Kastensmidt, Luigi Carro, and Ricardo Reis, "Fault-Tolerance Techniques for SRAM-Based FPGAs", vol. 32, edition 1, Springer, 2006.

[25] Heather Quinn, Paul Graham, Jim Krone, Michael Caffrey, Sana Rezgui, and Carl Carmichael, "Radiation-Induced Multi-Bit Upsets in Xilinx SRAM-Based FPGAs", IEEE Transactions on Nuclear Science, vol. 52, no. 6, pp. 2455-2461, June 2005.

[26] Cannon. E.H, Gordon. M.S, Heidel. D.F, Klein-Osowski. A.J, Oldiges. P, Rodbell. K.P, and Tang H, "Multi-bit upsets in 65nm SOI SRAMs", IEEE International Reliability Physics Symposium (IRPS'08), pp. 195–201, April 2008.

[27] Daniele Radaelli, Helmut Puchner, Skip Wong, and Sabbas Daniel, "Investigation of Multi-Bit Upsets in a 150 nm Technology SRAM Device", IEEE Transactions on Nuclear Science, vol. 52, pp. 2433–2437, no. 6, Dec. 2005.

[28] E. Maricau and G. Gielen, "Analog IC Reliability in Nanometer CMOS", Chapter 2–CMOS Reliability Overview, Analog Circuits and Signal Processing, Edition 1, Springer Science and Business Media, 2013.

[29] X.Y.Li, F. Wang, T. La, and Z.-M. Ling, "FPGA as process monitor–an effective method to characterize poly gate CD variation and its impact on product performance and yield", IEEE Transactions on Semiconductor Manufacturing, vol. 17, no. 3, pp. 267–272, Aug. 2004.

[30] M. Abramovici and C. E. Stroud, "BIST-based delay-fault testing in FPGAs", Journal of Electronic Testing: Theory and Applications, vol. 19, no. 5, pp. 549–558, Oct. 2003.

[31] P.Girard, O. Heron, S. Pravossoudovitch, and M. Renovell, "High quality TPG for delay faults in look-up tables of FPGAs", IEEE International Workshop on Electronic Design, Test and Applications, pp. 83–88, Jan. 2004.

[32] Minal Sawant, "Single Event Effects Complicate Military Avionics System Design", Journal of Military Electronics and Computing, Jan. 2012.

[33] John D. Gorbertt, Gary M. Swift, "Rad-hard SRAM FPGAs enable vast improvements in space exploration", Xilinx 408-559-7778, Xilinx Corporation, June 2012.

[34] Ramin Roosta, "A Comparison of Radiation-Hard and Radiation Tolerant FPGAs for Space Applications", NASA Electronic Parts and Packaging Program, JPL D-31228, Dec. 2004.

[35] O. Kononchuk, B.Y. Nguyen, "Silicon-On-Insulator (SOI) Technology: Manufacture and Applications", Edition 1, Woodhead Publishing, June 2014.

[36] Simoen, E., A. Mercha, C. Claeys and N. Lukyanchikova, "Low-frequency Noise in Silicon-on-Insulator Devices and Technologies", Solid State Electronics, vol. 51, no. 1, pp. 16–37, Jan. 2007.

[37] Mantelet G., Briet M., Rouxel G., Hachad S., Bancelin B., de Saint Roman D., "ATMEL ATF280E rad hard SRAM based reprogrammable FPGA SEE test

results," European Conference on Radiation and Its Effects on Components and Systems (RADECS'09), pp. 606–608, Sept. 2009.

[38] Narayana Murty Kodeti, "White Paper On Silicon On Insulator (SOI) Implementation", Infotech Enterprises Ltd, June 2009. [Available Online]: http://www.soiconsortium.org/pdf/SOI_Implementation_WhitePaper_Infotech_v2.pdf

[39] Xiaowei H., Lihua W., Yan Z., Yan L., Qianli Z., Liang C., and Jian W. , "A radiation-hardened SOI-based FPGA", Journal of Semiconductors, vol. 32, no. 7, pp. 075012-1–075012-6, July 2011.

[40] Yan Z., Lihua W., Xiaowei H., Yan L., Qianli Z., Liang C., and Jian, W., "An IO block array in a radiation-hardened SOI SRAM-based FPGA", Journal of Semiconductors, vol. 33, no. 1, pp. 015010-1–015010-7, Jan. 2012.

[41] S. E. Kerns, B. D. Shafer, L. R. Rockett, J. S. Pridmore, D. F. Berndt, N. van Vonno, F. E. Barber, "The design of radiation-hardened ICs for space: A compendium of approaches," Proceedings of the IEEE, vol. 76, no. 11, pp. 1470–1509, Nov. 1988.

[42] Kashfia Haque and Paul Beckett, "Radiation-Hard Field-Programmable Gate Arrays Configuration Technique Using Silicon on Sapphire," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no.2, pp. 232–241, Feb. 2014

[43] L. B. Freeman, "Critical Charge Calculations for a Bipolar SRAM Array," IBM Journal of Research and Development, vol. 40, no. 1, pp. 119–129, Jan. 1996.

[44] Yuan-yuan Wang, Zi-Ou Wang, Li-Jun Zhang, "A new 6-transistor SRAM cell for low power cache design", IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT'12), pp. 1–3, Oct/Nov. 2012.

[45] Lior Atias, Adam Teman and Alexander Fish, "Single Event Upset Mitigation in Low Power SRAM Design", IEEE 28th Convention of Electrical and Electronics Engineers in Israel, pp 1–5, Dec. 2014.

[46] 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, 1996.

[47] M. Shayan, V. Singh, A. D. Singh, and M. Fujita, "SEU tolerant robust memory cell design", IEEE 18th International On-Line Testing Symposium (IOLTS'12), pp. 13–18, June 2012.

[48] S. M. Jahinuzzaman, D. J. Rennie, and M. Sachdev, "A soft error tolerant 10T SRAM bit-cell with differential read capability", IEEE Transactions on Nuclear Science, vol. 56, no. 6, pp. 3768–3773, Dec. 2009.

[49] A. Pescovsky, O. Chertkow, L. Atias, and A. Fish, "SEU hardening: Incorporating an extreme low power bitcell design (SHIELD)", SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S'14), pp. 1-3, Oct. 2014.

[50] Lior Atias, Adam Temanand Alexander Fish, "A 13T Radiation Hardened SRAM Bitcell for Low-Voltage Operation", IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S'13), pp. 1–2, Oct. 2013.

[51] Michael A. Bajura, "Models and Algorithmic Limits for an ECC-Based Approach to Hardening Sub-100-nm SRAMs", IEEE Transactions on Nuclear Science, Vol. 50, No. 4, pp. 935–945, Aug. 2007.

[52] Fernanda Gusmao De Lima Kastensmidt, "Designing Single Event Upset Mitigation Techniques for Large SRAM-Based FPGA Components", PhD thesis, Universidade Federal Do Rio Grande Do Sul, 2003.

[53] Subhasish Mitra and Edward J. McCluskey, "Design of Redundant Systems Protected Against Common Mode Failure", IEEE Proceedings on VLSI Test Symposium (VTS'01), pp. 190–195, 2001.

[54] Xilinx Inc., "Xilinx TMRTool: Industry's First Triple Modular Redundancy Development Tool for Reconfigurable FPGAs", 2015.

[55] "Quadruple Modular Redundant Technology for Safety Systems", Honeywell Product Information, [Available Online]: https://www.honeywellprocess.com/library/marketing/notes/QMRTechnology PIN_May08.pdf.

[56] C. Poivey, D. Petrick, D. Espinosa, Austin Lesea, K. LaBel, M. Friendlich, H. Kim, Anthony Phan, "Effectiveness of Internal vs. External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis", 9th European Conference on Radiation and Its Effects on Components and Systems (RADECS'07), pp. 1-8, Sept. 2007.

[57] Heiner J., Collins N., Wirthlin M., "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing", IEEE Aerospace Conference, pp.1-10, March 2008.

[58] L. Jones, "Single Event Upset (SEU) detection and correction using Virtex-4 devices," Xilinx Inc., San Jose, CA, Application Note XAPP714, Jan. 2007.

[59] Alex Harding and Michael Wirthlin, "Improving the Reliability of Xilinx 7 Series FPGAs through Configuration Scrubbing", 20th Annual Fellowship Symposium–Utah NASA Space Grant Consortium, May 2014.

[60] M. Caffrey C. Carmichael and A. Salazar, "Correcting Single-Event Upsets (SEU) through Virtex partial configuration", Xilinx Technical report, XAPP216 (v1.0) June 2000.

[61] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs", 18th

International Symposium on Parallel and Distributed Processing, p. 145, April 2004.

[62] G. H. Asadi and M. B. Tahoori, "Soft error mitigation for SRAM-based FPGAs", 23rd IEEE VLSI Test Symposium (VTS'05), pp. 207–212, May 2005.

[63] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR", IEEE International Reliability Physics Symposium Proceedings, pp. 226–232, March 2006.

[64] He Wei,Wang Yueke, Xing Kefei, and Chen Li, "SEU readback interval strategy of SRAM-based FPGA for space application", IEEE International Conference on Computer Science and Automation Engineering (CSAE'11), vol. 4, pp. 238–241, June 2011.

[65] Dhiraj K. Pradhan and Costas Argyrides, "Multiple event upsets aware FPGAs using protected schemes", In book: Fault-Tolerant Distributed Algorithms on VLSI Chips, Dagstuhl Seminar Proceedings, pp. 1862-4405, 2009.

[66] Sang Phill Park, Dongsoo Lee, and K. Roy, "Soft-error-resilient FPGAs using built-in 2-d hamming product code", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, no. 2, pp. 248–256, Feb. 2012.

[67] Qian Zhao, Y. Ichinomiya, M. Amagasaki, M. Iida, and T. Sueyoshi, "A novel soft error detection and correction circuit for embedded reconfigurable systems", IEEE Embedded Systems Letters, vol. 3, no. 3, pp. 89–92, Sept. 2011.

[68] Frederic P. Miller, Agnes F. Vandome, John McBrewster, "Hamming Codes", Alphascript Publishing, 2009.

[69] Bernhard M.J. Leiner, "LDPC codes–a brief tutorial", April 8, 2005, [Available Online]: http://www.bernh.net/media/download/papers/ldpc.pdf

[70] Bernard Sklar, "Fundamentals of Turbo Codes", In book: Digital Communications: Fundamentals and Applications, Second Edition, Prentice-Hall, 2001.

[71] C.K.P. Clarke, "Reed-Solomon Error Correction", British Broadcasting Corporation R& D White Paper (WHP031), July 2002.

[72] Varsha P. Patil, D. G. Chougule, Radhika. R. Naik, "Viterbi Algorithm for Error Detection and Correction", IOSR Journal of Electronics and Communication Engineering (IOSR–JECE), pp. 60–65, 2013.

[73] Simon Tam, "Single Error Correction and Double Error Detection", Application Note: Virtex–II Pro, Virtex-4, and Virtex-5 Families, Xilinx Corporation, XAPP645 (v2.2) August 2006.

[74] Ken Chapman, "SEU Strategies for Virtex-5 Devices", Application Note: Virtex-5 Family, XAPP864 (v2.0), April 2010.

[75] C. Bolchini, A. Miele, and M.D. Santambrogio, "TMR and partial dynamic reconfiguration to mitigate SEU faults in FPGAs", IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'07), pp. 87–95, Sept. 2007.

[76] Yimao Cai, Yuanfu Zhao, and Lidong Lan, "Implementation of a reconfigurable computing system for space applications", International Conference on System Science, Engineering Design and Manufacturing Informatization (ICSEM'11), vol. 2, pp. 360–363, Oct. 2011.

[77] M.G. Gericota, L.F. Lemos, G.R. Alves, M.M. Barbosa, and J.M. Ferreira, "A frame-work for fault tolerant real time systems based on reconfigurable FPGAs", IEEE Conference on Emerging Technologies and Factory Automation, pp. 131–138, Sept. 2006.

[78] Y. Ichinomiya, S. Tanoue, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Improving the robustness of a softcore processor against SEUs by using TMR and partial reconfiguration", IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'10), pp. 47–54, May 2010.

[79] A. Ilias, K. Papadimitriou, and A. Dollas, "Combining duplication, partial reconfiguration and software for on-line error diagnosis and recovery in SRAM-based FPGAs", IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'10), pp. 73–76, May 2010.

[80] F. Lahrach, A. Doumar, and E. Chatelet, "Fault tolerance of multiple logic faults in SRAM-based FPGA systems", Euromicro Conference on Digital System Design (DSD'11), pp. 231–238, Sept. 2011.

[81] Hung-Manh Pham, S. Pillement, and D. Demigny, "Evaluation of fault-mitigation schemes for fault tolerant dynamic MPSoC", International Conference on Field Programmable Logic and Applications (FPL'10), pp. 159-162, Sept. 2010.

[82] Praveen Kumar Samudrala, Jeremy Ramos, and Srinivas Katkoori, "Selective Triple Modular Redundancy (STMR) for Single Event Upset (SEU) Tolerant for FPGAs",IEEE Transactions on Nuclear Science, Vol 51, No 5, pp. 2957-2969, October 2004.

[83] Qi Zhong Zhou, Jing Chen Nan, Yong Le Xie, and Shu Yan Jiang, "Fault-tolerance with dual module redundancy for dynamic reconfigurable FPGAs", International Conference on Applied Superconductivity and Electromagnetic Devices (ASEMD'11), pp. 1–4, Dec. 2011.

[84] 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, Dec. 2004.

[85] K.S. Morgan, D.L. McMurtrey, B.H. Pratt, and M.J. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs", IEEE Transactions on Nuclear Science, vol. 54, no. 6, pp. 2065–2072, Dec. 2007.

[86] B. Pratt, M. Caffrey, J.F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR", IEEE Transactions on Nuclear Science, vol. 55, no. 4, pp. 2274–2280, Aug. 2008.

[87] M. Niknahad, O. Sander, and J. Becker, "FGTMR– fine grain redundancy method for reconfigurable architectures under high failure rates", North–East Asia Symposium on Information Technology and Reliability (NASNIT'11), pp. 186–191, Oct. 2011.

[88] M. Niknahad, O. Sander, and J. Becker, "Fine grain fault tolerance– a key to high reliability for FPGAs in space", IEEE Aerospace Conference, pp. 1–10, March 2012.

[89] Sandi Habinc, "Suitability of Reprogrammable FPGAs in Space Applications: Feasibility Report", Technical report, European Space Agency, 2002. [Available Online]: http://microelectronics.esa.int/techno/fpga_002_01-0-4.pdf.

[90] Hung–Manh Pham, "Embedded computing architecture with dynamic hardware reconfiguration for intelligent automotive systems", PhD thesis, Institut de recherche en informatique et systemes aleatoires (IRISA)– CAIRN, Université de Rennes 1, France, 2010.

[91] Xilinx Inc, "Virtex-5 FPGA User Guide", UG190 (v5.4), March 2012. [Available Online]: www.xilinx.com/support/documentation/user_guides/ug190.pdf

[92] Xilinx Inc., "Radiation-Hardened, Space-Grade Virtex-5QV Family Overview", Product Specification, DS192 (v1.4), November 2014. [Available Online]: http://www.xilinx.com/support/documentation/data_sheets/ds192_V5QV_Device_Overview.pdf.

[93] Sourdis I., Strydis C., Bouganis C.S., Falsafi B., Gaydadjiev G.N., Malek A., Mariani R., Pnevmatikatos D., Pradhan D.K., Rauwerda G., Sunesen K., Tzilis S., "The DeSyRe Project: On–Demand System Reliability," Euromicro Conference on Digital System Design (DSD'12), pp. 335–342, 5–8 Sept. 2012.

[94] Walker J.A., Trefzer M.A., Bale S.J., Tyrrell A.M., "PAnDA: A Reconfigurable Architecture that Adapts to Physical Substrate Variations", IEEE Transactions on Computers, vol. 62, no. 8, pp. 1584–1596, Aug. 2013.

[95] Milos Krstic, Stefan Weidling, Vladimir Petrovic, and Egor S. Sogomonyan, "Enhanced architectures for soft error detection and correction in combinational and sequential circuits", ScienceDirect Journal on Microelectronics and Reliability, vol. 56, no. 1, pp. 212–220, Jan. 2016.

[96] Picard D. and Lagadec L., "Fast prototyping environment for embedded reconfigurable units", International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'11), pp. 1–8, June 2011.

[97] Sandi Habinc, Gaisler Research, "Functional Triple Modular Redundancy (FTMR): VHDL Design Methodology for Redundancy in Combinatorial and Sequential Logic", Design and Assessment Report, Version 2, Dec. 2002.

[98] Daniel Francisco Gómez Prado, "Tutorial on FPGA Routing", Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA, Technical Report No. 17, 2006.

[99] I. Polian, J. P. Hayes, S. M. Reddy, and B. Becker, "Modeling and Mitigating Transient Errors in Logic Circuits", IEEE Transactions on Dependable and Secure Computing, Vol. 8, No. 4, pp. 537–547, July/Aug. 2011.

[100] N. Miskov-Zivanov and D. Marculescu, "Multiple Transient Faults in Combinational and Sequential Circuits: A Systematic Approach," IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems, vol. 29, no. 10, pp. 1614–1627, Oct. 2010.

[101] X. Li, "Tolerating Radiation-Induced Transient Faults in Modern Processors", Ph.D. Thesis, University of California, Irvine, 2005. [Available Online]: http://newport.eecs.uci.edu/x̃iaobinl/dissertationTopChap.pdf.

[102] C. A. L. Lisboa, "Dealing with Radiation Induced Long Duration Transient Faults in Future Technologies", Ph.D. Thesis, Universidade Federal Do Rio Grande Do Sul, Instituto De Informatica, Porto Alegre, Brazil, June 2009.

[103] A. Matrosova, E. Loukovnikova, S. Ostanin, Zinchuck, and E. Nikolaeva, "Test Generation for Single and Multiple Stuck–at Faults of a Combinational Circuit Designed by Covering Shared ROBDD with CLBs", IEEE International Symposium on Defect and Fault–Tolerance in VLSI Systems, pp. 206–214, Sept. 2007.

[104] S. Pontarelli, G. C. Cardarilli, A. Leandri, M. Ottavi, and A. Salsano, "A self–checking cell logic block for fault tolerant FPGAs", IEEE International Symposium on Circuits and Systems (ISCAS'02), vol. 4, pp. 477–480, 2002.

[105] D. P. Vasudevan, P. K. Lala, and J. P. Parkerson, "Self-checking carryselect adder design based on two-rail encoding", IEEE Transactions on Circuits and Systems (ISCAS'07), vol. 54, no. 12, pp. 2696–2705, Dec. 2007.

[106] "Accelerator Series FPGAs – ACT 3 Family", Microsemi Corporation Application Note: Revision 3, Jan 2012. [Available Online]: http://www.microsemi.com/document-portal/doc_view/130668-accelerator-series-fpgas-act-3-family.

[107] A. Fariborz, B. Kerry, M. J. Hargrove, N.J. Rohrer, and S. Peter, "SOI CMOS dynamic circuits having threshold voltage control", US Patent No. 6433587 B1, Aug. 2002.

[108] M. Sonza-Reorda, M. Violante, C. Meinhardt, and R. Reis, "A low-cost SEE mitigation solution for soft-processors embedded in systems on programmable chips", Design, Automation & Test in Europe Conference Exhibition (DATE'09), pp. 352–357, April 2009.

[109] S.-Y. Yu, "Fault tolerance in adaptive real-time computing systems", Ph.D. Thesis, Dept. of Electrical Engineering, Stanford University, Stanford, CA, 2001.

[110] C. Benkeser, A. Burg, T. Cupaiuolo, and Qiuting Huang, "Design and optimization of an HSDPA turbo decoder ASIC", IEEE Journal on Solid-State Circuits, vol. 44, no. 1, pp. 98–106, Jan. 2009.

[111] P.J. Black and Teresa H.Y. Meng, "A 1–gb/s, four-state, sliding block viterbi decoder", IEEE Journal on Solid-State Circuits, vol. 32, no. 6, pp. 797–805, June 1997.

[112] Hsie–Chia Chang, C.B. Shung, and Chen–Yi Lee, "A reed–solomon product–code (RSPC) decoder chip for DVD applications", IEEE Journal of Solid–State Circuits, vol. 36, no. 2. pp. 229–238, Feb 2001.

[113] E. Grossar, M. Stucchi, K. Maex, and W. Dehaene, "Statistically aware SRAM memory array design", International Symposium on Quality Electronic Design (ISQED'06), pp. 25–30, Mar. 2006.

[114] Chun–Lung Hsu and Ching–FenWu, "High–performance 3D–SRAM architecture design", IEEE Asia Pacific Conference Circuits and Systems (APCCAS'10), pp. 907–910, Dec. 2010.

[115] Jameel Hussein and Gary Swif, "Mitigating single–event upsets (SEUs)", Technical report, Xilinx White Paper: 7 Series FPGAs, WP395 (v1.0), 2012.

[116] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron–induced soft error in SRAMs from a 250 nm to a 22 nm design rule", IEEE Transactions on Electron Devices, vol. 57, no. 7, pp.1527–1538, July 2010.

[117] Chih–Hao Liu, Shau–Wei Yen, Chih–Lung Chen, Hsie–Chia Chang, Chen–Yi Lee, Yar–Sun Hsu, and Shyh–Jye Jou, "An LDPC decoder chip based on self–routing network for IEEE 802.16e applications", IEEE Journal of Solid-State Circuits, vol. 43, no. 3, pp. 684–694, Mar. 2008.

[118] M. Maniatakos, M.K. Michael, and Y. Makris, " Vulnerability–based interleaving for multi-bit upset (MBU) protection in modern microprocessors", IEEE International Test Conference (ITC'12), pp. 1–8, Nov. 2012.

[119] Nick Mehta, "Xilinx 7 series FPGAs: The logical advantage", Xilinx White Paper: 7 Series FPGAs, WP405 (v1.0), Mar 6 2012.

[120] M. Pathak and Sung Kyu Lim, "Reliability and performance–aware 3D SRAM design", IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'11), pp. 1-4, Aug. 2011.

[121] Michael B. Pursley, "Introduction to Digital Communications", Upper Saddle River, NJ: Prentice Hall, 2005.

[122] P.M.B. Rao, M. Ebrahimi, R. Seyyedi, and M.B. Tahoori, "Protecting SRAM-based FPGAs against multiple bit upsets using erasure codes", ACM/EDAC/IEEE Design Automation Conference (DAC'14), pp. 1–6, Jun. 2014.

[123] Daniel J. Costello Shu Lin, "Error Control Coding: Fundamentals and Applications", Pearson–Prentice Hall, 2004.

[124] Leilei Song, Meng–Lin Yu, and M.S. Shaffer, "10- and 40–gb/s forward error correction devices for optical communications", IEEE Journal of Solid–State Circuits, vol. 37, no. 11, pp. 1565–1573, Nov. 2002.

[125] Xilinx, Inc., "7 Series FPGAs Configuration", User Guide, UG470 (v1.10), Jun. 2015.

[126] Jim Handy, Intel Corporation, "Understanding the Intel/Micron 3D XPoint^{TM} Memory", Storage Development Conference (SDC), Santa Clara, Sept. 2015.

[127] Kevin Gibb, TechInsights, "Resistive RAM (ReRAM) Memory is Finally Here", Design Lines Memory, EETimes, April 2015. [Available Online]: http://www.eetimes.com/author.asp?section_id=36doc_id=1326351

[128] Yangyang Pan and Tong Zhang, "DRAM–Based FPGA Enabled by Three–Dimensional (3D) Memory Stacking", ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2010.

[129] Young Yang Liauw, Zhiping Zhang, Wanki Kim, Gamal A.E., Wong S.S., "Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory," IEEE International on Solid-State Circuits Conference (ISSCC'12), Digest of Technical Papers, pp. 406–408, Feb. 2012.

*****

Abbreviations

3DH ..... **3** **D**irectional **H**amming

ALM ..... **A**daptive **L**ogic **M**odule

ALU ..... **A**rithmetic **L**ogic **U**nit

API ..... **A**pplication **P**rogramming **I**nterface

ARDyT ..... **A**rchitecture **R**econfigurable **Dy**namiquement **T**olérante aux fautes

ASIC ..... **A**pplication **S**pecific **I**ntegrated **C**ircuit

BB ..... **B**asic Building **B**lock

BER ..... **B**it **E**rror **R**ate

BIST ..... **B**uilt-**I**n **S**elf **T**est

BRAM ..... **B**lock **R**andom **A**ccess **M**emory

BTI ..... **B**ias **T**emperature **I**nstability

CAD ..... **C**omputer-**A**ided **D**esign

CCLB ..... **C**ustomized **C**onfigurable **L**ogic **B**lock

CED ..... **C**oncurrent **E**rror **D**etection

CLB ..... **C**onfigurable **L**ogic **B**lock

CLE ..... **C**onfigurable **L**ogic **E**lement

CMF ..... **C**ommon **M**ode **F**ailures

CMOS ..... **C**omplementary **M**etal-**O**xide **S**emiconductor

COTS ..... **C**ommercial **O**ff-**T**he-**S**helf

CPLD ..... **C**omplex **P**rogrammable **L**ogic **D**evice

CRAM ..... **C**onfiguration **R**andom-**A**ccess **M**emory

CRC ..... **C**yclic **R**edundancy **C**heck

D2D ..... **D**ie-to-**D**ie

DICE ..... **D**ual **I**nterlocked **S**torage **C**ell

DMR ..... **D**ual **M**odular **R**edundancy

DRAM ..... **D**ynamic **R**andom-**A**ccess **M**emory

DSP ..... **D**igital **S**ignal **P**rocessor

ECC ..... **E**rror-**C**orrecting **C**ode

EDAC ..... **E**rror **D**etection **A**nd **C**orrection

EDCA ..... **E**rror **D**etection and **C**orrection **A**nalysis

EM ..... **E**lectro-**M**igration

EMI ..... **E**lectro **M**agnetic **I**nterference

FA–CLB ..... **F**auilt-**A**ware **C**onfigurable **L**ogic **Block**

FEC ..... **F**orward **E**rror **C**orrection

FET ..... **F**ield-**E**ffect **T**ransistor

FGTMR ..... **F**ine **G**rain **T**riple **M**odular **R**edundancy

FIT ..... **F**ailure **I**n **T**ime

FPD ..... **F**ield-**P**rogrammable **D**evices

FPGA ..... **F**ield-**P**rogrammable **G**ate **A**rray

FSM ..... **F**inite-**S**tate **M**achine

FSR ..... **F**ault **S**tatus **R**egister

FT-DyMPSoC **F**ault **T**olerant-**D**ynamic **M**ulti-**P**rocessor **S**ystem-**o**n-**C**hip

FTAL ..... **F**ault **T**olerant **A**bstraction **L**ayer

GEO ..... **GE**ostationary **O**rbit

GPRR ..... **G**rouped **P**artial **R**econfigurable **R**egion

HCI ..... **H**ot **C**arrier **I**njection

ICAP ..... **I**nternal **C**onfiguration **A**ccess **P**ort

IP ..... **I**ntellectual **P**roperty

JTAG ..... **J**oint **T**est **A**ction **G**roup

LAB ..... **L**ogic **A**rray **B**lock

LDPC ..... **L**ow-**D**ensity **P**arity-**C**heck

LUT ..... **L**ook **U**p **T**able

MBU ..... **M**ulti **B**it **U**pset

MC ..... **M**atrix **C**odes

MCU ..... **M**ulti **C**ell **U**pset

MNU ..... **M**ulti-**N**ode **U**psets

NBTI ..... **N**egative-**B**ias **T**emperature **I**nstability

NMOS ..... **N**egative **M**etal **O**xide **S**emiconductor

PLD ..... **P**rogrammable **L**ogic **D**evice

PMOS ..... **P**ositive **M**etal **O**xide **S**emiconductor

PRM ..... **P**artially **R**econfigurable **M**odule

PRR ..... **P**artial **R**econfigurable **R**egion

PTMR ..... **P**artial **T**riple **M**odular **R**edundancy

QMR ..... **Q**uadruple **M**odular **R**edundancy

R3M ..... **R**un-rime **R**econfigurable **R**esource **M**anager

SBU ..... **S**ingle **B**it **U**pset

SEB ..... **S**ingle **E**vent **B**urnout

SEC ..... **S**ingle-**E**rror **C**orrecting

SEC/DED ... **S**ingle-**E**rror **C**orrecting and **D**ouble **E**rror **D**etecting

SEE ..... **S**ingle **E**vent **E**ffects

SEFI ..... **S**ingle-**E**vent **F**unctional **I**nterrupt

SEGR ..... **S**ingle-**E**vent **G**ate **R**upture

SEL ..... **S**ingle-**E**vent **L**atch-up

SET ..... **S**ingle-**E**vent **T**ransient

SEU ..... **S**ingle-**E**vent **U**pset

SHIELD ..... **S**EU **H**ardening: **I**ncorporating an **E**xtreme **L**ow–power **B**itcell **D**esign

SNHT ..... **S**uper-**N**ode **H**ash-**T**able

SNU ..... **S**ingle **N**ode **U**pset

SoC ..... **S**ystem-**o**n-**C**hip

SOI ..... **S**ilicon-**O**n-**I**nsulator

SOS ..... **S**ilicon-**O**n-**S**apphire

SRAM ..... **S**tatic **R**andom-**A**ccess **M**emory

STMR ..... **S**elective **T**riple **M**odular **R**edundancy

TDDB ..... **T**ime-**D**ependent **D**ielectric **B**reakdown

TID ..... **T**otal **I**onizing **D**ose

TMR ..... **T**riple **M**odular **R**edundancy

VHDL ..... **V**ery-High-Speed Integrated Circuit **H**ardware **D**escription **L**anguage

XTMR ..... **X**ilinx **T**riple **M**odular **R**edundancy