HAL
open science

Multiple fault mitigation in network-on-chip architectures through a bit-shuffling method

Romain Mercier

To cite this version:

Romain Mercier. Multiple fault mitigation in network-on-chip architectures through a bit-shuffling method. Hardware Architecture [cs.AR]. Université Rennes 1, 2021. English. NNT : 2021REN1S123 . tel-03716432v3

HAL Id: tel-03716432

https://theses.hal.science/tel-03716432v3

Submitted on 7 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Romain MERCIER

Multiple Fault Mitigation in Network-on-Chip Architectures Through a Bit-Shuffling Method

Thèse présentée et soutenue à Lannion, le 17 Décembre 2021
Unité de recherche : Inria Rennes - Bretagne Atlantique, IRISA - Équipe Taran

Rapporteurs avant soutenance :

Alberto BOSIO	Professeur, École Centrale de Lyon
Lorena ANGHEL	Professeur, Université de Grenoble-Alpes

Composition du Jury :

Président :	Patrick GIRARD	Directeur de recherche CNRS LIRMM, Montpellier
Examinateurs :	Alberto BOSIO	Professeur, École Centrale de Lyon
	Lorena ANGHEL	Professeur, Université de Grenoble-Alpes
Dir. de thèse :	Daniel CHILLET	Professeur, Université de Rennes 1
Encadrants :	Cédric KILLIAN	Maitre de conférences, Université de Rennes 1
	Angeliki KRITIKAKOU	Maitre de conférences, Université de Rennes 1

Invité :

Youri HELEN	Ingénieur de recherche, DGA MI

Acknowledgement

Mes premiers remerciements vont à mon directeur de thèse, Daniel Chillet, ainsi qu'à mes encadrants, Cédric Killian, Angeliki Kritikakou et Youri Helen. Ils ont su me guider tout au long de ces trois années tout en me laissant l'autonomie nécessaire à ma progression. Leurs conseils et leurs enseignements m'ont permis de progresser dans le domaine de la recherche et de l'enseignement tout au long de cette thèse. Je remercie également Olivier Sentieys pour m'avoir offert l'opportunité d'entreprendre cette aventure en m'acceptant au sein du master recherche à l'ENSSAT.

Je remercie les membres du jury, Alberto Bosio, Lorena Anghel et Patrick Girard, pour leur présence lors de la soutenance malgré les difficultés liées aux conditions sanitaires. Ma reconnaissance va particulièrement à Alberto et Lorena qui ont accepté de rapporter sur mon manuscrit malgré le manque de temps. Leurs retours précieux ont mis en lumière les points pertinents afin de les perfectionner lors de futur recherches.

Je remercie la Direction Générale de l'Armement (DGA) et l'Inria Bretagne Atlantique pour avoir financé cette thèse et m'avoir permis de vivre convenable durant cette période difficile.

Ces remerciements ne seraient pas complets sans mentionner les équipes TARAN et GRANIT. La bonne humeur et la convivialité qui émane de ce laboratoire m'a permis d'appréhender cette thèse dans un climat agréable et intellectuellement stimulant. Mention spéciale à Joël qui a seulement oublié 986 fois de faire le café. Je tiens aussi à citer Nadia et Emilie pour m'avoir accompagné lors des démarches administratives mais aussi tous les autres membres du laboratoire, que ce soit à Lannion ou à Rennes pour leur présence.

Je remercie également ma famille, mes parents, Claudine et Dominique, mais aussi mes frères et soeur, Anthony, Fabien et Estelle, pour m'avoir soutenue et encouragé durant toute mes études. Et également mes chères neveux et nièce, Lucas, Rafaël, Eliot et Manon pour m'avoir donné le sourire dans les moments difficiles.

Mes remerciements vont aussi à tous mes amis qui m'ont encouragé et soutenue durant cette entreprise. Mention particulière à Jordan, Lucie, Clément, Roxane et Adrien qui m'ont poussé à entreprendre ce long projet alors que je doutais de mes capacités. Je n'en serais pas là aujourd'hui sans eux !

Enfin, comme il est d'habitude de garder le meilleur pour la fin, je fini par remercier celle qui m'a soutenue, encouragé et motivé durant toute cette aventure à travers les bons comme les mauvais moments. Merci Marine.

Table of Contents

Résumé Étendu **1**

Introduction **9**

1 Network-on-Chip in Harsh Environments **13**

1.1 Network-on-Chip Communications 13

1.1.1 Network-on-Chip High-Level Description 14

1.1.2 High-Level Router Description 18

1.1.3 Conclusion 24

1.2 Origins and Sources of Faults 25

1.2.1 Faults Induced By Radiations 25

1.2.2 Faults Induced By Manufacturing and Aging Defects 27

1.2.3 Fault Modeling 29

1.2.4 Conclusion 30

1.3 Fault Impacts on Network-on-Chips 30

1.3.1 Fault Impacts on the Logic Part 30

1.3.2 Fault Impacts on the datapath 32

1.3.3 Fault Evolution Over the Time 32

1.4 Conclusion 32

2 State-of-the-Art of Fault-Tolerant NoC **35**

2.1 Overview of fault-tolerant NoCs 35

2.2 Detection and Correction Using Information Redundancy 36

2.2.1 ECC Theoretical Presentation 36

2.2.2 Conventional Error-Correcting Codes 37

2.2.3 Exotic ECCs for High Fault Coverage 40

2.2.4 ECC Distribution in NoC Architecture 42

2.3 Detection and Correction Based on NoC architectures 43

2.3.1 Fault-Tolerant Routing Algorithms 43

2.3.2 Spatial and Temporal Redundancies 45

2.3.3 Fault Tolerant Topologies 49
2.4 Other Detection Methods 50
2.4.1 Built-In Self-Test Methods 50
2.4.2 Monitoring Methods 52
2.4.3 Other State-of-the-Art Methods 52
2.5 Mitigation Using Approximate Communications 53
2.6 Examples of Fault-Tolerant NoC Architectures 54
2.7 Conclusion 55

3 Bit-Shuffling Technique for Error Mitigation in Network-on-Chip Architectures 57
3.1 BiSu-Principle Overview 57
3.1.1 Target Domain and Assumptions 57
3.1.2 Basic Concepts 59
3.1.3 Method Implementation 61
3.1.4 Header and Critical-Data Protection 64
3.1.5 Matching Data and Flit Sizes 66
3.2 BiSu-Efficiency Evaluation 68
3.2.1 Flit-Level Evaluations: Payload Error Mitigation 68
3.2.2 Flit-Level Evaluations: Header Protection 76
3.2.3 NoC-Level Evaluations 79
3.2.4 Application-Level Evaluation: Convolutional Neural Network 83
3.2.5 Case Studies: Image Processing and Data Mining Applications . . . 87
3.3 BiSu-Hardware Evaluation 95
3.3.1 Shuffler and De-shuffler Blocks 95
3.3.2 Merger and De-merger Blocks 98
3.3.3 Comparison of the Global Hardware Costs 100
3.3.4 Hardware Overheads of the Register Computing 102
3.4 Conclusion 103

4 Region-based Bit-Shuffling Approach: Trading Hardware Cost and Fault Efficiency 105
4.1 Region-Based Bit-Shuffling (R-BiSu) Principle 105
4.1.1 Definition of the NoC Regions 105
4.1.2 Region-based Bit-Shuffling approach (R-BiSu) 106

4.1.3 Region Error Mask (REM) Computation 110
4.2 R-BiSu Efficiency Evaluation 112
4.2.1 Experimental Setup 113
4.2.2 Efficiency Results at the NoC Scale 113
4.3 R-BiSu Hardware Evaluation 117
4.3.1 Experimental Setup 117
4.3.2 Shuffler/De-shuffler Block Number Comparison 118
4.3.3 R-BiSu Hardware Results 118
4.3.4 Hardware Overheads of the Register Updater 120
4.4 Efficiency Versus Hardware Cost Trade-off 121
4.4.1 Area Overhead Versus Efficiency 122
4.4.2 Power Overhead Versus Efficiency 123
4.5 Conclusion 124

Conclusion and Perspectives **125**

Publications and Communications **129**

Bibliography **131**

List of Figures **155**

List of Tables **159**

Acronyms **161**

Résumé Étendu

Réseaux sur Puce Tolérant aux Fautes

Depuis plusieurs décennies, les améliorations technologiques et la diminution de la taille des transistors ont permis d'atteindre une haute densité d'intégration au sein des puces suivant la loi de Moore [1], pour atteindre aujourd'hui des milliards de transistors par puce. Alors que les fréquences et la densité des puces rencontraient la limite de puissance [2], l'augmentation des performances a été atteinte en ajoutant plus de blocs dédiés (*Intellectual Properties* - IPs), c'est-à-dire de cœurs, de mémoires, d'accélérateurs matériels, etc., sur une seule puce, donnant naissance aux Systèmes sur Puce (*System-on-Chip* - SoC). De nos jours, les SoCs embarquent un grand nombre de cœurs, de mémoires et d'accélérateurs matériels. Par exemple, le processeur AMD Zen 2 [3] contient jusqu'à 64 cœurs. Cependant, l'augmentation du nombre de cœurs dans les puces induit un trafic de données de plus en plus important qui ne peut être géré par les moyens de communication classiques tels que les liaisons point-à-point ou les bus [4]. Pour lever ce verrou, les Réseaux sur Puce (*Network-on-Chips* - NoCs) [5, 6], sont apparus au cours des dernières décennies comme une solution alternative et évolutive pour fournir la large bande passante requise dans les SoCs. Cela est fait en gérant la communication entre plusieurs dizaines ou milliers de cœurs avec une Qualité de Service (*Quality of Service* - QoS) satisfaisante [7], comme dans le dispositif ACAP de Xilinx [8] et le processeur multi-cœur MPPA-256 Bostan [9] de Kalray.

Alors que la densité des transistors augmente, la réduction de la tension et la mise à l'échelle des technologies permettent d'accroître les performances des processeurs mais le taux de défaillance intrinsèque de l'électronique augmente [10]. Les transistors sont particulièrement touchés lorsque leur taille atteint 10 nm et moins [11]. Dans cette ère technologique, les NoCs sont devenus plus sensibles aux fautes, qui affectent leurs fonctionnalités. Les défauts peuvent être dus à des effets externes, par exemple les radiations [12], ou à des effets internes, par exemple les défauts de fabrication [13] et de vieillissement [14]. Les défauts induits peuvent affecter les NoCs de différentes manières sur des périodes plus ou moins longues selon le type de défaut. Alors que les défauts transitoires n'affectent le

composant que pendant quelques cycles d'horloge, les défauts permanents ne peuvent être éliminés et sont toujours présents. Les défauts permanents sont particulièrement critiques pour les NoCs car ils peuvent affecter leurs fonctionnalités sans possibilité de récupération.

Pour résoudre ces problèmes, des techniques de tolérance aux fautes sont couramment appliquées aux NoCs [15, 16, 17]. Ces techniques sont souvent réparties en quatre catégories : i) détection, ii) diagnostic, iii) correction et iv) atténuation. Les travaux présentés dans ce manuscrit se concentrent sur la troisième et la quatrième catégorie, en considérant des fautes permanentes. En effet, l'accumulation de ces fautes dans les systèmes n'est malheureusement pas bien traitée dans la littérature. Les techniques de tolérance aux fautes qui sont couramment appliquées sur les NoCs [16] pour y faire face sont généralement basées sur i) des algorithmes de routage [18], ii) la reconfiguration matérielle en utilisant des ressources de réserve ou un chemin de secours par défaut [16], iii) la réplication des circuits [13] et iv) la redondance d'informations [19]. Bien que les approches susmentionnées soient efficaces pour gérer des fautes permanentes solitaires, elles sont moins adaptées aux multiples fautes permanents. En effet, ces approches introduisent des coûts élevés, en termes de latence, de surface et de consommation d'énergie, tandis que leurs capacités d'atténuation sont limitées. Finalement, l'utilisation du calcul approximatif, déjà proposée dans les communications à haute performance et à faible consommation d'énergie [20], a récemment fait son apparition pour atténuer les fautes durant les communications.

Pour traiter efficacement les fautes permanentes multiples au sein des architectures NoCs, nous proposons, comme première contribution, une technique matérielle de brassage de bits (*Bit-Shuffling* - BiSu) [P1, P2] introduisant un faible surcoût matériel ainsi qu'un faible impact sur les performances. La technique proposée se concentre sur la réduction des impacts des fautes, au lieu de les corriger complètement. Des compromis entre l'efficacité et les coûts matériels de la méthode proposée peuvent être exploités. Pour cela, une approche de la technique BiSu basée sur l'utilisation de régions, appelée R-BiSu [P3], est proposée comme seconde contribution. Cette approche permet de réduire les surcharges matériel en relâchant l'efficacité de la technique BiSu standard.

Technique de Brassage de Bits pour l'Atténuation des Erreurs dans les Architectures Réseaux sur Puce

La méthode BiSu [P2, P1] est une technique matérielle de brassage de bits ayant de faible coûts en surface et en consommation d'énergie. De plus, notre technique a un faible

impact sur les performances des NoCs, ce qui entraîne un impact négligeable au niveau des applications. Comme mentionné précédemment, BiSu se concentre sur la réduction de l'impact des fautes, au lieu de les corriger complètement. Elle assure la protection des bits de poids forts (*Most Significant Bits* - MSBs), en transférant l'impact des défauts permanents sur les bits de poids faibles (*Least Significant Bits* - LSBs), tout en maintenant les MSBs corrects. La technique BiSu est efficace pour gérer des fautes multiples qui impactent les données transitant sur le NoC. Cependant, cette efficacité dépend de la précision requise par l'application considérée. C'est-à-dire que plus l'application requerra des données précises, moins de fautes pourront être gérées par la technique BiSu. Par conséquent, l'approche proposée est particulièrement adaptée aux domaines d'application où les données peuvent être approximées à la fois lors du calcul et de la communication [21], comme le traitement d'images, la classification de données, l'apprentissage automatique, etc. Cette méthode peut être mise en œuvre en tandem avec d'autres techniques telles que le *remapping* [22] et l'ordonnancement [23]. De plus, la méthode proposée peut être mise en œuvre avec n'importe quel protocole de transmission et n'importe quelle architecture NoC. Avec cette approche, la redondance spatiale n'est pas nécessaire, ce qui limite le surcoût surfacique. Par ailleurs, comme les chemins défectueux ne sont pas exclus pour le transfert de données approximables, les performances du NoC, telles que la latency et la bande passante, sont maintenues.

L'approche BiSu atténue les multiples défauts permanents qui peuvent survenir dans le Network-on-Chip (NoC), et plus particulièrement dans les chemins de données. Comme l'illustre la Figure A.2, les fautes peuvent être localisées dans i) les interconnexions entre les routeurs, ou ii) les mémoires tampons et la traversée au sein de chaque routeur. Comme ces éléments occupent la plus grande partie de la surface d'un routeur [24], ils ont une probabilité plus élevée d'accumuler des défauts dus aux effets des radiations, aux défauts de fabrication et de vieillissement, ou à d'autres défaillances intrinsèques. Pour les mêmes raisons, les interconnexions sont souvent affectées par des défauts permanents qui sont généralement modélisés par des collages à un ou zéro ou par des courts-circuits [25]. Elles sont donc traitées de manière similaire par la méthode BiSu.

Pour implémenter notre technique, nous considérons des messages classiques de S_{msg} bits routés par le NoC. La Figure A.1 illustre l'organisation d'un tel message en paquets et en flits. Un message est décomposé en N_P paquets de S_{pck} bits de charge utile, chaque paquet contient N_F flits de S_F bits de données et comprend un flit d'en-tête pour le contrôle du routage. Comme le montre la figure, nous décomposons en outre chaque flit

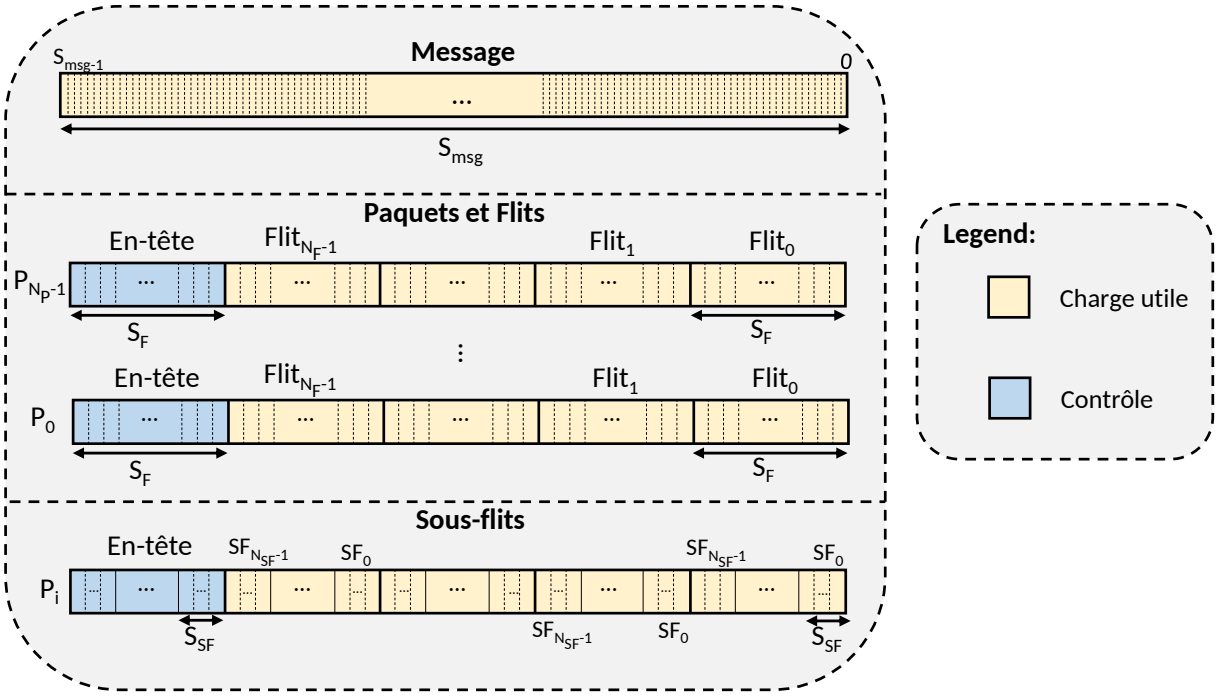


Figure A.1: Formatage des messages : Paquets, flits et sous-flits.

en N_{SF} sous-flits (SF) de taille binaire S_{SF} afin de permettre l'application de la technique de brassage de bits proposée.

La technique BiSu applique des fonctions de brassage et de dé-brassage qui permutent, au moment de l'exécution, deux ou plusieurs sous-flits au sein du même flit, afin de transférer l'impact des fautes sur les LSBs. La Figure A.2 illustre, à travers un exemple, les principes de notre approche. Considérons des flits traversant un routeur défectueux du nord au sud, comme le montre la flèche violette de la Figure A.2-(a). Pour des raisons de simplification, on ne considère qu'une simple mémoire tampon, mais BiSu est également applicable avec des canaux virtuels. Comme le montre la Figure A.2-(b), nous considérons $S_F = 8$ bits et une taille de sous-flit égale à $S_{SF} = 2$ bits. Par conséquent, le nombre de sous-flits dans un flit est égal à $N_{SF} = 4$ (SF_0 à SF_3). Lorsqu'aucune faute ne se produit, les fonctions de brassage et de dé-brassage sont désactivées et les flits traversent le routeur sans modification. Considérons maintenant que deux défauts permanents se produisent dans la traversée du routeur en affectant les MSBs, c'est-à-dire les bits 7 et 6 de tous les flits entrants. La partie droite de la Figure A.2-(b) illustre le croisement des paquets sans aucune modification dans le flit. Les bits 6 et 7 des deux flits de charge utile sont affectés, ce qui entraîne des erreurs dans la plage $\{0, \pm 64, \pm 128, \pm 192\}$, selon la valeur initiale des

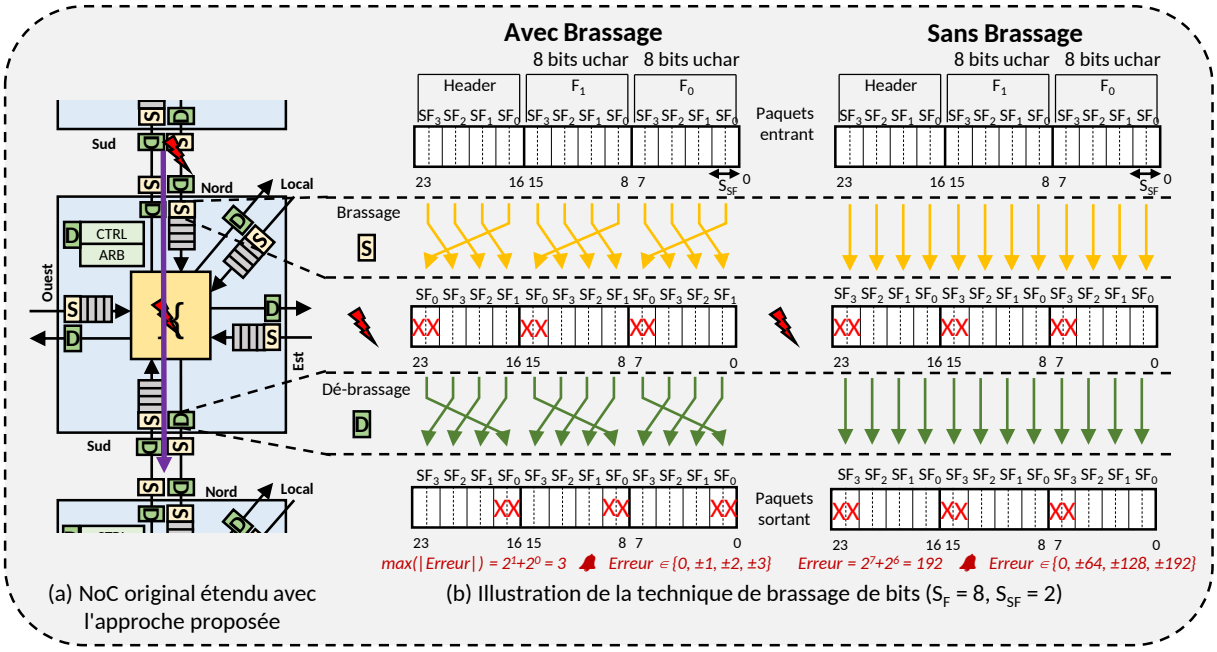


Figure A.2: NoC classique étendu avec la technique BiSu.

bits affectés. La partie gauche de la Figure A.2-(b) illustre la méthode de brassage de bits proposée. Cette technique est activée dans les ports d'entrée du routeur. Ensuite, avant de traverser le chemin défectueux, les sous-flits sont réorganisés en échangeant les LSBs et les MSBs des flits de charge utile afin d'allouer les MSBs au chemin matériel non défectueux. Ainsi, l'impact des fautes est réduit à la plage $\{0, \pm1, \pm2, \pm3\}$, en fonction des valeurs des LSBs. Avant que le flit ne quitte le routeur, les sous-flits sont ramenés à leur position initiale, et le flit est envoyé au port de sortie.

Les résultats obtenus avec notre méthode ont démontré que la technique BiSu est efficace pour atténuer les fortes densités de fautes pendant les communications sur puce avec un surcoûts matériel limité par rapport à des techniques complexes, telles que les ECCs. Nous avons vu par l'expérimentation que la précision des applications résistantes aux fautes peut être maintenue avec BiSu permettant ainsi d'obtenir des résultats acceptables. D'autre part, nous avons démontré que la méthode BiSu a des coûts matériels raisonnables qui sont similaires à ceux d'un code de Hamming étendu et qui varient en fonction de la taille des sous-flits. En particulier, la consommation électrique de la méthode proposée reste particulièrement faible. En confrontant les résultats concernant l'efficacité et les coûts matériels de la technique BiSu, nous mettons en évidence l'existence d'un compromis qui est géré par la taille des sous-flits.

Brassage de Bits Basé sur des Régions

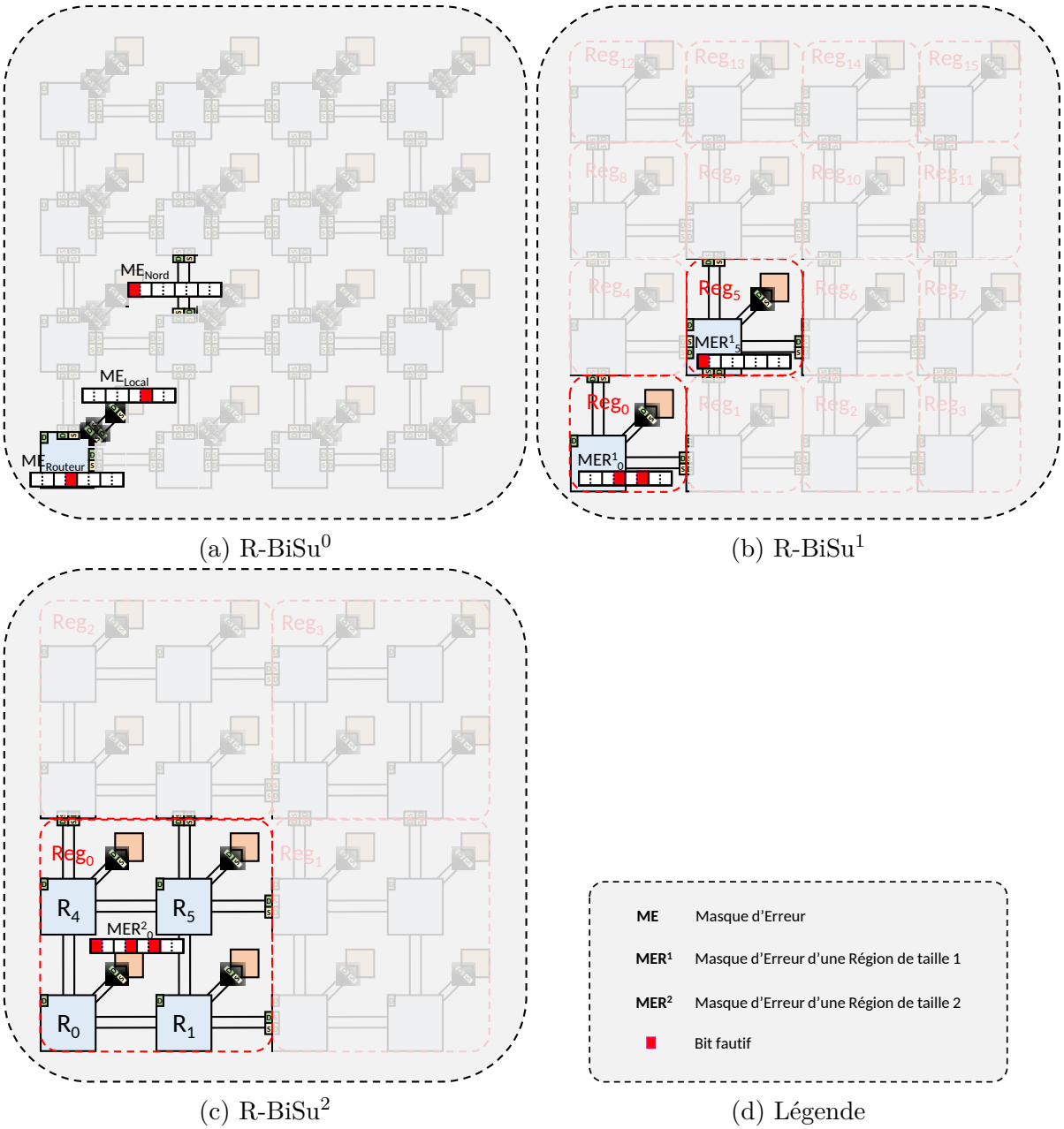
Comme nous l'avons vue précédemment, des compromis entre l'efficacité et les coûts matériels de la technique BiSu peuvent être exploités afin d'optimiser l'efficacité de cette dernière. Ainsi, notre seconde contribution consiste à explorer ces compromis en explorant une implémentation de BiSu basée sur des régions (*Region-based Bit-Shuffling* - R-BiSu) [P3] dans le but de réduire le surcoût matériel en relâchant l'efficacité de la technique BiSu standard. L'idée de base de R-BiSu est de diviser le NoC en régions de routeurs qui sont globalement protégées par la méthode BiSu. Dans nos travaux nous considérons des régions carrées régulières mais la méthode est applicable à d'autres tailles et formes de régions.

La Figure A.3 illustre l'implémentation de l'approche R-BiSu dans un NoC de taille 4×4, où les régions sont délimitées par les carrés pointillés rouges. Pour comparer visuellement l'impact de l'approche Region-based Bit-Shuffling (R-BiSu) sur la densité des blocs de brassage et de dé-brassage, nous affichons sur la Figure A.3a l'implémentation de la méthode BiSu où chaque routeur et chaque interconnexion sont protégés par un couple de blocs de brassage et de dé-brassage. Par souci de clarté, nous définissons R-BiSu^0 la mise en œuvre de la technique BiSu et R-BiSu^n l'implémentation de la technique R-BiSu en utilisant une région de taille n. Les figures A.3b et A.3c représentent respectivement les configurations R-BiSu^1 et R-BiSu^2. Sur ces figures, nous observons que R-BiSu^0 nécessite 304 blocs de brassage et de dé-brassage, alors que les configurations R-BiSu^1 et R-BiSu^2 ne nécessitent respectivement que 144 et 80 de blocs, ce qui réduit l'impact sur les surcoûts matériels.

L'exploration de l'approche R-BiSu a permis de mettre en évidence les compromis existants entre l'efficacité et les coûts matériels. Nous avons vu que ces compromis sont influencés par la taille des sous-flits et régions considérés. De plus, nous avons constaté que l'augmentation de la taille de la région de 0 (BiSu standard) à 2 (R-BiSu^2), en considérant des régions carrées régulières, permet de réduire drastiquement les coûts matériels avec un faible impact sur l'efficacité de la méthode.

Le reste du manuscrit est organisé de la façon suivante :

Chapitre 1: Ce chapitre présente les bases nécessaires et le contexte pour comprendre les contributions présentées dans ce manuscrit. Tout d'abord, une description de l'architecture des NoCs est donnée. Ensuite, les sources de fautes



(a) R-BiSu^0 (b) R-BiSu^1

(c) R-BiSu^2 (d) Légende

Figure A.3: Valeurs des masques d'erreur dans un NoC défectueux avec la technique R-BiSu en considérant différentes tailles de régions.

qui peuvent détériorer les NoCs, et en général les circuits intégrés, sont présentées. Enfin, les impacts de ces fautes sur les NoCs sont expliqués.

Chapitre 2: L'état actuel de l'art concernant le domaine de la tolérance aux fautes dans les NoCs est présenté dans ce chapitre. Les différentes méthodes utilisées pour détecter, diagnostiquer, corriger ou atténuer les fautes dans les NoCs sont explorées. Le contexte de notre travail est également décrit dans ce chapitre.

Chapitre 3: Ce chapitre présente notre première contribution. La méthode de brassage de bits utilisée pour atténuer les fautes permanentes multiples dans les chemins de données des NoCs est présentée en détail. La technique BiSu est évaluée à différentes échelles et comparée à un code de Hamming étendu afin de quantifier son efficacité et ses coûts matériels.

Chapitre 4: Dans ce chapitre, une implémentation de la méthode BiSu basée sur les régions est proposée afin de réduire les coûts matériels au prix d'une efficacité réduite en matière de tolérance aux fautes. Les compromis entre l'efficacité et les coûts matériels de la technique R-BiSu sont explorés afin de déterminer les implémentations optimisées de la méthode.

Introduction

Since several decades, technology improvements and transistor shrinking enabled high transistor density per chip according to the Moore's Law [1], reaching today billions of transistors per chip. While frequencies and chip density met the power limit [2], performance increase has been reached by adding more Intellectual Properties (IPs), i.e. cores, memories, hardware accelerators, etc., on a single chip giving birth to System-on-Chip (SoC). Nowadays, SoC includes a large number of cores, memories and hardware accelerators. For example, the AMD Zen 2 processor [3] contains up to 64 cores. However, the increase of the core number in chips induces more and more data traffic which cannot be managed by conventional communication means such as point-to-point links or busses [4]. To address this gap, NoCs [5, 6], appeared in the last decades as a scalable solution to provide the high bandwidth required in SoCs, by managing communication between several tens or thousands of cores with a satisfying Quality of Service (QoS) [7], such as in ACAP Xilinx devices [8] and Kalray MPPA-256 Bostan many-core processor [9].

While transistor density increases and voltage reduction and technology scaling enable increasing computer performances, the intrinsic failure rate of electronics is increased [10]. Transistors are particularly impacted while their size reaches 10 nm and below [11]. In this technology era, NoCs became more sensitive to faults, which affect their functionality. Faults can occur due to external effects, i.e. radiations [12], or internal effects, i.e. manufacturing [13] and aging [14] defects. The induced faults can affect NoCs by different ways over shorter or longer periods of time according to the fault type. While transient faults affect the component only during few clock cycles, permanent faults cannot be removed and are always present. Permanent faults are particularly critical for NoCs since they can affect their functionality with no possibility of recovery.

To address these problems, fault tolerant techniques are commonly applied on NoCs [15, 16, 17]. These techniques are often split into four categories: i) detection, ii) diagnosis, iii) correction and iv) mitigation. Unfortunately, the accumulation of permanent faults is not well addressed in the literature. Furthermore, when these techniques are implemented, the hardware costs, i.e. area cost and power consumption, are very high. To address this gap, we proposed the Bit-Shuffling (BiSu) method [P2, P1], a bit-shuffling hardware technique

with low area and power consumption. Moreover, our technique has a low impact on NoC performances leading to negligible impact at application level. The proposed technique focuses on reducing the impact of faults, instead of fully correcting them. It ensures the protection of Most Significant Bits (MSBs), by transferring the impact of the permanent faults to the Least Significant Bits (LSBs), keeping the MSBs correct. This is achieved by dividing a flit into several blocks of bits, named Subflits (SFs), and by exchanging (shuffling) the position of the SFs on each flit, at run-time. The BiSu technique is efficient for multiple faults, depending on the accuracy needed by the application executed on the NoC-based architecture. Therefore, the proposed approach is especially suitable for error-resilient applications, i.e. domains where approximate data are tolerated both for computation and communication [21], such as image processing, data mining, machine learning, etc. Trade-offs between efficiency and hardware costs can be exploited through the implementation of the proposed method. For that, a region-based approach of the BiSu technique, named R-BiSu [P3], is proposed, and allows to reduce the hardware overheads by relaxing the efficiency of the standard BiSu technique.

The rest of the manuscript is organized as follows:

Chapter 1: This chapter presents the necessary baselines and the context to understand the contributions presented in this manuscript. First, a description of NoC architecture is given. Then, fault sources which can deteriorate NoCs, and in general the integrated circuits, are presented. Finally, the impacts of these faults on NoCs are explained.

Chapter 2: The current state-of-the-art concerning NoC fault-tolerant field is presented in this chapter. Different methods used to detect, diagnose, correct or mitigate faults in NoC architectures are explored. The context of our work is also described in this chapter.

Chapter 3: This chapter presents our first contribution. The bit-shuffling method used to mitigate the multiple permanent faults in the NoC datapaths is presented in details. The BiSu technique is evaluated at different scales and compared to an extended Hamming code in order to quantify its efficiency and its hardware costs.

Chapter 4: In this chapter, a region-based implementation of the BiSu method is proposed in order to decrease the hardware overheads at the cost of reduced fault-tolerant efficiency. Trade-offs between the efficiency and the hardware

costs of the R-BiSu technique are explored to determine optimized implementations of the method.

Chapter 1

Network-on-Chip in Harsh Environments

Network-on-Chip (NoC) architectures have been largely explored through the last decades to solve the high-bandwidth request in on-chip communications. These architectures are characterized by several parameters which define how the NoC is constituted and how it operates to transmit the information. Nowadays, the NoCs are widely used to manage the multiprocessor System-on-Chips (SoCs) communications in many fields such as avionic and space fields. Unfortunately, these architectures are susceptible to be impacted by failures affecting their performances or data which transit on them. These failures can occur due to internal or external effects which induce faults during the lifetime of the NoC. In this chapter, we present the necessary baselines required to understand the proposed contributions developed in this manuscript. For that, a general description of NoC architectures is given in Section 1.1. Then, the sources of internal and external faults which can cause failures in the NoC are presented in Section 1.2. Finally, Section 1.3 details how the aforementioned faults can impact the NoC functioning before to conclude the chapter in Section 1.4.

1.1 Network-on-Chip Communications

NoC field is studied since many decades to increase NoC performances providing efficient on-chip communications for multi-processor architectures. For that, several NoC architectures have been proposed such as in the ACAP Xilinx devices [8] and in the Kalray MPPA-256 Bostan many-core processor [9]. In this section, we describe the basic concepts of NoC architecture to provide the necessary baselines for the reading of this manuscript.

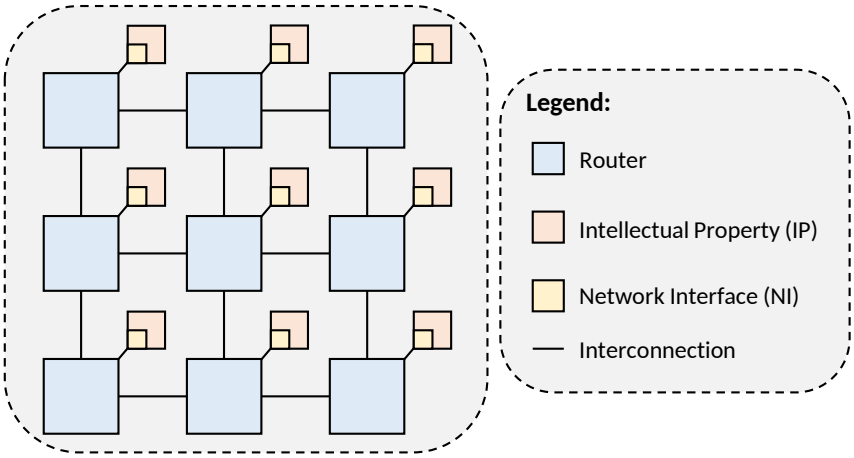


Figure 1.1: Description of the main elements present in network-on-chips.

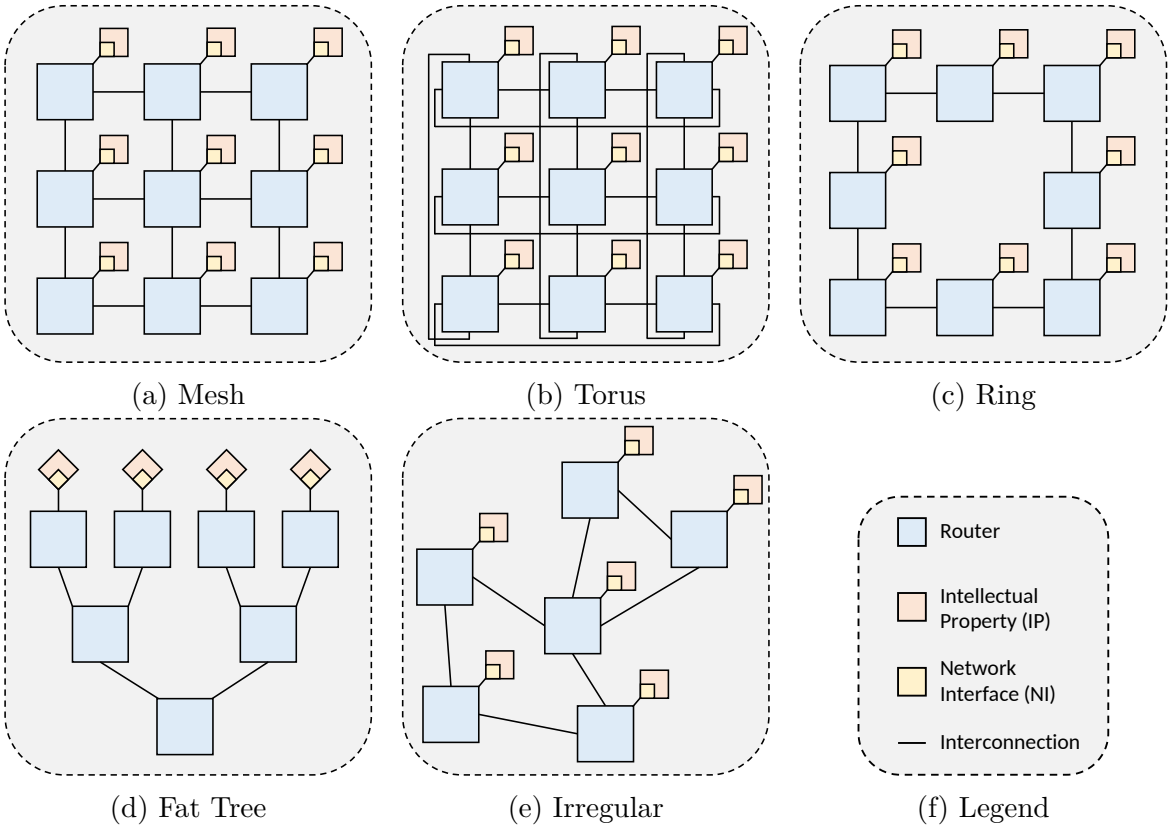
1.1.1 Network-on-Chip High-Level Description

In this part, we propose a high-level description of NoC architecture. For that, we first present the different elements which compose the NoC to describe how they are constituted and how they are organized. Then, we detail how the messages are formatted for on-chip communications and the different communication modes which can be used to transmit the messages.

1.1.1.1 Network-on-Chip Components

NoCs are composed of three main elements which are i) routers, ii) interconnections and iii) Network Interfaces (NIs). As displayed in Figure 1.1, the routers are connected together by the interconnections, which are parallel busses, and each router is connected to an Intellectual Property (IP), i.e. cores, memories, hardware accelerators and so on, using a NI.

When a source IP needs to push a message in the NoC, the associated NI formats the message to send it through the NoC, where routers transmit it, through interconnections towards a neighboring router. When the message reaches the destination, the router forwards the message towards the NI of the destination which decodes the message to send it to the associated destination IP.



(a) Mesh (b) Torus (c) Ring

(d) Fat Tree (e) Irregular (f) Legend

Figure 1.2: Several network-on-chip topologies.

1.1.1.2 Network-on-Chip Topology

The NoC topology determines how routers are interconnected together. In this manuscript, we limit the exploration to 2-D architectures. Several examples of these topologies are displayed in Figure 1.2. The Mesh topology, shown in Figure 1.2a, is the most used topology in the literature where routers form a matrix and are connected to their neighbors. In this way, corner, border and central routers have respectively two, three and four neighbor routers. However, as performances of the mesh NoC are proportional to its size, this topology reaches its limits when large NoCs are considered, i.e. the latency for communications between opposite corners or edges can exceed the acceptable limit to respect the required Quality of Service (QoS). To solve this problem, the Torus topology connects the opposite border routers as displayed in Figure 1.2b. This configuration allows for the latency to be largely reduced for edge to edge communications. Moreover, additional interconnections increase the NoC bandwidth but also increase the critical path due to the long wire length required to connect the opposite NoC edges.

Other topologies may be found in the literature such as Ring and Fat Tree topologies which are respectively displayed in Figures 1.2c and 1.2d. In the Ring topology, each router is only connected to two routers to form a circle, hence reducing communication complexity. However, this topology has a reduced bandwidth due to less interconnection density. In the Fat Tree topology all messages have to transit through the top router of the architecture leading to significant impacts on the bandwidth with high traffic density. These topologies can be enhanced to increase the bandwidth of the NoC. For example, in [26], the bandwidth of 2-D mesh NoC is amplified by adding supplementary long interconnections between routers of the same column or row. In [27], regular topologies are enhanced using long-range interconnections which are specific to the considered application.

As topologies presented above respect a logical implementation of the design, they are classified as regular topologies. Others, called irregular topologies, as shown in Figure 1.2e, can be found in the literature. They are often used for low-size NoC increasing the message transmission complexity to adapt the NoC in specific architectures. Many other topologies can be found in the literature but they are less used in practical cases, so they will not be detailed in this document.

1.1.1.3 Message Formatting for on-Chip Communications

In NoC communications, messages need to be split into smaller units to be forwarded through routers as shown in Figure 1.3. Messages are split into smaller fixed-length Flow controL uniTS (FLITS) where each flit has the same size as the interconnections in the NoC. Flits made of data are called payloads and are forwarded through the NoC in order to reach the destination IP. An additional flit called header flit is added to the payloads. The header contains necessary routing information decoded by each router to route the message toward the destination. The header and the payloads form a packet where the total number of flits depends on the architecture. A message can be divided into several packets to be forwarded through the NoC. For example, in Figure 1.3 the message is split into N_p packets and each of these packets is split into N_{flit} flits plus the header. The message formatting is done at the NI of the source, while the NI of the receiving IP extracts the message from the incoming packets. These operations are called packetization and de-packetization in this report.

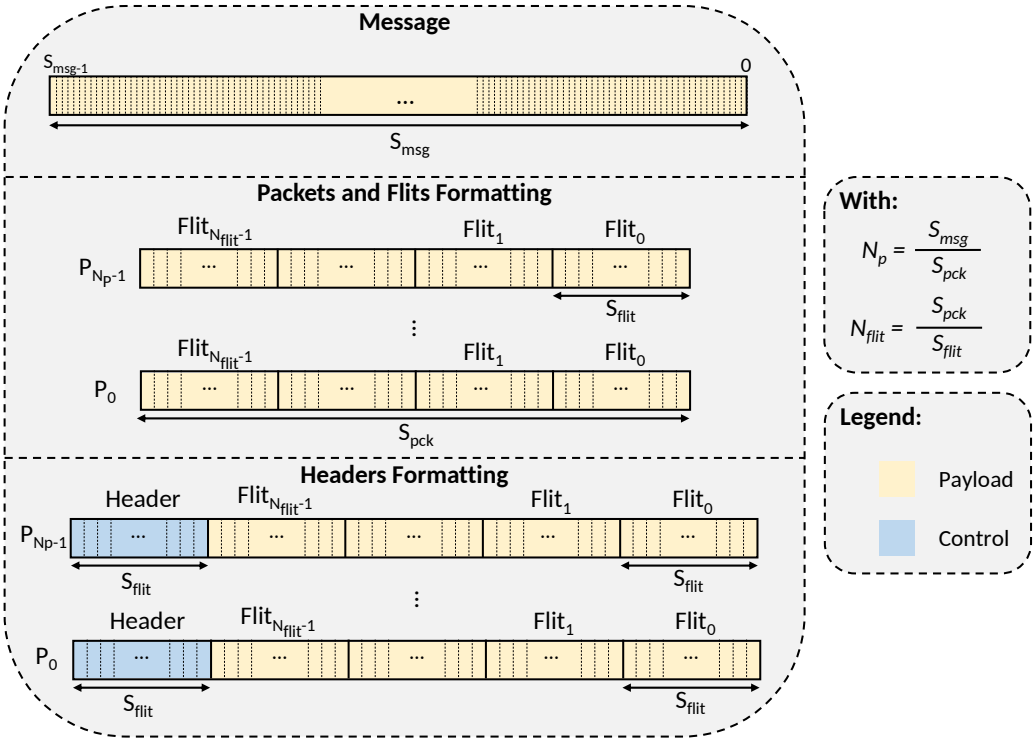


Figure 1.3: Flit-based message formatting for NoC communications.

1.1.1.4 Communication Modes

The communication modes [28, 14], also called switching modes, indicate how the messages are transmitted through the NoC. These modes are generally split into two categories which are the circuit-switching mode and the packet-switching mode.

Circuit-Switching Mode: In this mode, a path is reserved from source to destination in order to ensure the message transmission. For that, a first packet, called allocation packet, is sent through the NoC to reserve the path from the source until the destination. Then, the destination IP replies by sending an acknowledge packet to confirm that the path is reserved for the message transmission. When the acknowledge packet is received by the source IP, the communication is performed through the reserved path. Finally, once the message reception is acknowledged, the source IP sends a last packet to the destination IP in order to release the reserved path. The allocation packet can be blocked if it needs to cross a router which is already reserved by an another transmission. In this case, the allocation packet needs to wait the release of the router to pursue the path allocation until its destination. This mode ensures a constant latency for data transfer when the path is

established. However, the bandwidth is limited due to the path reservation which obstructs the other communications. This effect is particularly true for large NoC architecture.

Packet-Switching Mode: In this mode, each packet is forwarded through the NoC without path allocation since the priority is done locally in each router increasing the bandwidth. Consequently, the latency may vary each flit according to NoC traffic. Three ways to transfer the flits through the NoC can be found in the literature. The most used is the wormhole switching mode where the flits can be spread in several routers. In this way, the latency for packet transmission is reduced and the need of buffers in the architecture is low, limiting the hardware costs in terms of area and power. However, the risk of deadlock, i.e. packets block themselves or each other, is increased. The second technique is called Store and Forward (SaF) packet switching mode. In this mode packets are entirely stored in the buffer before sent in the next router. Hardware costs and latency are drastically increased according to the packet size since it cannot be spread in several routers. The last technique is the Virtual Cut Through (VCT) which is a mix of the two aforementioned modes. In this mode, a packet may be spread on several router such as in wormhole switching mode, allowing low latency. However, the packet can also be entirely stored in the buffers of a given router in case of NoC congestion. Thus, this method needs as many buffers as SaF mode, but flits do not need to wait for the entire packet to be stored before being sent to the next router or the destination IP.

In the rest of this report, we consider NoC architectures based on the packet-switching mode.

1.1.2 High-Level Router Description

In this part, we propose a high-level description of the routers which compose the NoCs. Figure 1.4 displays a common router representation which can be found in the literature. In this figure, we can note that the router is composed of five input and output ports which are used to communicate with the neighboring routers, i.e. the north, east, south and west ports, and with the associated IP through the local port.

As shown in Figure 1.4, the router is generally split into four blocks which are the buffers, the arbitration controller, the routing controller and the crossbar. As depicted in Figure 1.5, the execution of these different blocks is generally dispatched into 4 stages which are i) the buffering, ii) the arbitration, iii) the routing computation and iv) the message forwarding with verification through the flow control. In this figure, we can note

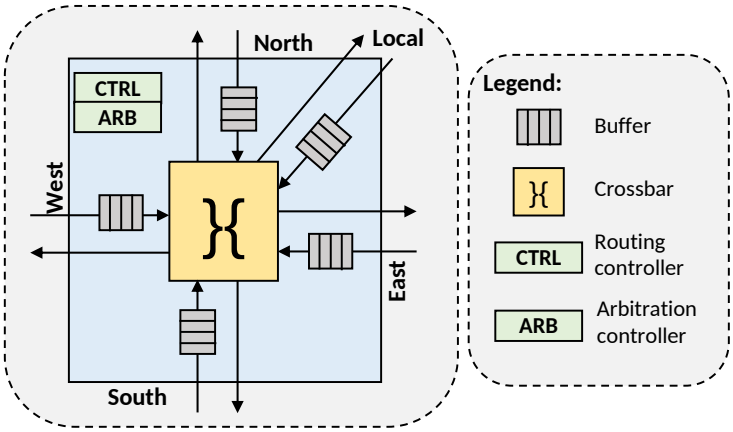


Figure 1.4: Classic router architecture with input buffering.

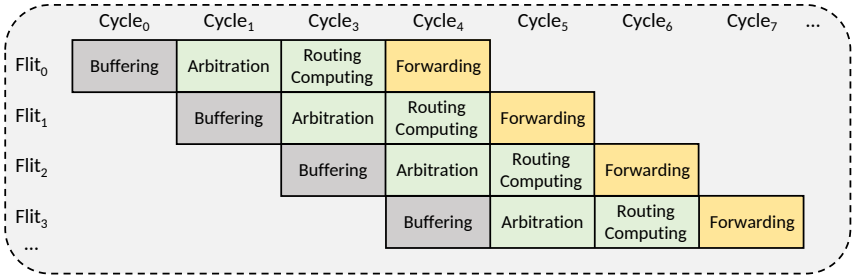


Figure 1.5: Pipeline of a classic router architecture with input buffering.

that the different stages are pipelined to enhance the NoC performances. Of course, the number of stages can vary from an architecture to another. For example, in [29], the proposed pipeline is divided into 6 stages. In the rest of this part, the roles of the different blocks which compose the router pipeline are explained.

1.1.2.1 Message Buffering

In NoC architectures, buffers are used to store the information, i.e. the flits, in the routers. Buffers can be placed at the router inputs, router outputs or both of them. They are useful in the presence of high packet densities to avoid congestion which is present when the flits need to wait a high number of cycles before being forwarded out of the router. This usually happens when the buffers of the next router are full which makes it impossible to send the current packet, impacting drastically the QoS and the performances of the NoC. In addition, other packet requests in the congested router need to wait until the release of the congestion. This phenomenon is called bottleneck in the NoC field. Congestion and bottleneck phenomena are generally managed by stalling the

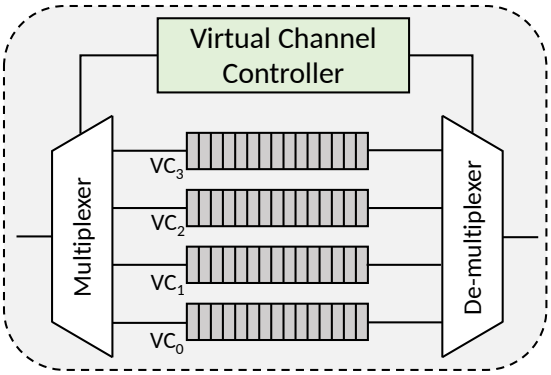


Figure 1.6: Architecture of virtual channels in a NoC router.

packet forward or dropping the packets which wait since too many clock cycles and send them again from the source IP.

The use of virtual channels [30] allows to reduce the congestion by using several parallel buffers as depicted in Figure 1.6 where a virtual channel of size 4 is illustrated. With virtual channels, several packets can be stored at the same time in the router inputs and any other packet requests can be accepted by the router arbitration. For this purpose, the priority and the arbitration of the virtual channels are managed by the virtual channel controller. Thus, in case a packet cannot be sent due to congestion in the next router, other packets can be sent towards the congestion-free output ports avoiding the deadlock phenomenon and reducing the congestion on the routers. However, the virtual channels induce large hardware costs since the buffers are the most expensive elements of the router, in terms of area and power consumption [31]. Despite the high induced hardware costs, increasing the buffer size, and the buffer number in the case of virtual channels, permits to enhance the bandwidth and the QoS of the NoCs.

Several router architectures have been proposed in the literature to reduce the hardware costs by removing the buffers [32, 33, 34]. In these architectures, packets are directly forwarded toward the next router. If latter cannot accept the packet request, then the packet is deflected toward another router. In the case where none of the neighboring routers can accept the packet request, it is dropped and re-sent from the source IP. This type of architecture ensures low hardware costs and low latency when the packet density is low. Otherwise, the global latency is drastically increased due to the dropped packets which can be critical in real-time applications.

1.1.2.2 Priority Arbitration

During system operation, a router may have to handle multiple packet requests, i.e. packets need to cross it toward the next router from different inputs of the router at the same time. In this case, an arbitration [35] must be used to manage the priorities between input requests. To be efficient, an arbitration must be fair [25] in the attribution of the priorities among the different input requests avoiding congestion, bottleneck and starvation. Starvation can occur when the requests from one input have not the same probability to be accepted than the requests of the other inputs. To tackle this problem, various arbiters are available in the literature [25, 35]. In the following paragraph we describe the most used arbiters in NoC architectures.

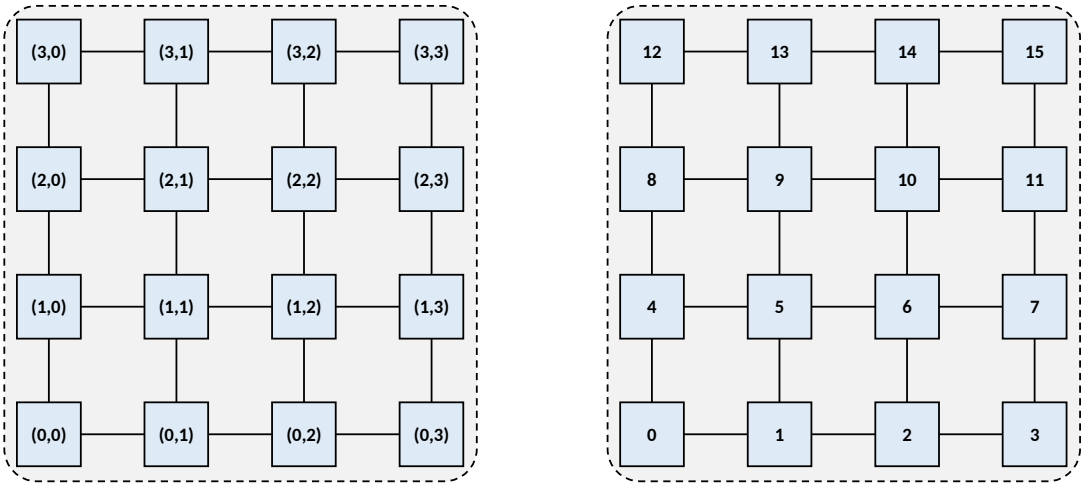
Round-Robin Arbiter: Each input has a priority level with a deterministic order. The request sent from the highest priority port triggers the routing of its associated packet toward the appropriate output port. On the next round of arbitration, the priority levels turn so that the second high-priority port becomes the high-priority port and so on allowing a strong fairness between the different inputs of the router.

Queuing Arbiter: Also called first coming arbitration, the requests are processed in the order of their arrival in the input ports. For that, each request is characterized by a time stamp which provides the arrival time of the packet in the router. The first incoming packet is determined from a tree of comparators and it is processed to be forwarded toward the appropriate output. This arbiter is claimed to have a high system-level fairness.

Matrix Arbiter: Also called least recently served arbitration, the priority is given to the input which has the longest elapsed time since its last accepted request. This arbitration is known for its strong fairness and its low cost for a low number of inputs.

1.1.2.3 Network-on-chip Path Determination

The routing algorithm determines the path taken by the packets to cross the NoC from the source IP towards the destination IP [36, 25, 28]. For that, a coordinate system is required to identify each router in the NoC. The positions of the routers in the NoC are generally defined by their localization in a two-dimensional Cartesian coordinate system, noted (x, y), as represented in Figure 1.7a or by a digital system, as represented in Figure 1.7b. The routing algorithm is locally performed in the router by the controller,



(a) Cartesian system. (b) Digital system.

Figure 1.7: Definition of the router position in a mesh NoC.

as depicted in Figure 1.4. For that, the routing controller uses information provided by the header of each packet. The routing algorithms are generally split into four categories which are i) deterministic, ii) semi-adaptive, iii) adaptive and iv) stochastic. These different types of routing algorithm are described in the following paragraphs.

Deterministic Routing Algorithms: They are often used in NoC architectures due to low costs and simple implementation in regular topologies. With these algorithms, the path followed by the packets between a couple of IPs is always the same. Then, the minimum latency is easier to compute but this type of routing algorithm cannot manage any alteration in the NoC architecture or traffic which can appear during the travel of packets. The most famous deterministic routing algorithm is the XY routing algorithm [37] which consists in forwarding the packet, first, on the horizontal axis, i.e. X axis in the Cartesian referential, then on the vertical axis, i.e. Y axis in the Cartesian referential. Some examples of packet forwarding with the XY routing algorithm are illustrated in Figure 1.8 where the paths of transiting packets are illustrated from the sources S toward the destinations D.

Semi-Adaptive Routing Algorithms: In the semi-adaptive routing algorithms, multiple paths can be taken by packets for a given couple of IPs. However, these algorithms do not consider the state of the NoC, i.e. congestion or malfunction, to determine the path. With this type of routing algorithm, the packet takes one of the shortest paths

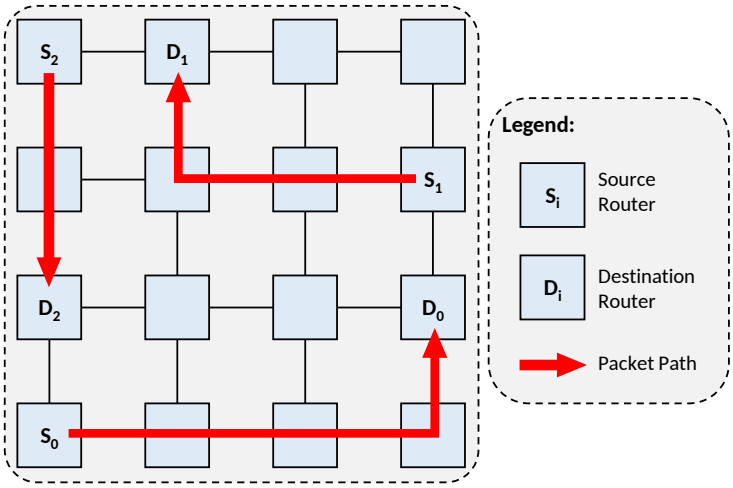


Figure 1.8: XY routing algorithm in a 4×4 mesh NoC.

to reach its destination. The most famous adaptive algorithms are based on the Turn model algorithm [38] which forbids some turns avoiding deadlocks and livelocks during the packet transmissions. The most used Turn-model-based routing algorithms are the West-First, the North-last and the Negative-first routing.

Adaptive Routing Algorithms: Contrary to the deterministic routing algorithms, the adaptive routing algorithms take into account the state of the NoC, i.e. traffic and architecture, to send the packets avoiding congestion and malfunctions. Then, the latency cannot be known before packet sending since the path between a couple of IPs is not necessarily one of the shortest paths. This type of algorithm can be logic-based or table-based inducing higher hardware costs in the NoC, particularly for large NoC. One of the most famous adaptive routing algorithm is the Odd-even algorithm which is based on the Turn model [39]. As this type of routing algorithms guarantees a certain level of fault tolerance by avoiding malfunctioning links and routers, they will be more detailed in Chapter 2.

1.1.2.4 On-chip Flow Control

The flow control [25, 14, 40, 41] determines how resources, i.e. buffer capacity, channel bandwidth or control state, are allocated during the packet transmission on the NoC. This control can be done between the source and the destination, i.e. end-to-end, or between two consecutive routers, i.e. switch-to-switch. The flow control needs to allocate

the resources of the NoC to nearly reach the theoretical bandwidth with a predictable and low latency. In the following paragraph, we present several flow control techniques which can be found in the literature.

Credit-Based Flow Control: Each router output has a counter which indicates the number of free spaces, i.e. credits, in the buffer of the next router. When a flit is sent toward the router, a credit is consumed while one credit is released when the next router extracts a flit from the considered buffer.

Handshaking Flow Control: A signal is sent by the next router each time it receives a flit from the upstream router. This signal allows for the upstream router to know if the flit which it sent earlier is correctly received by the next router.

Ack/Nack Flow Control: It is similar to handshaking flow, excepting that a copy of the flit is kept in the upstream router. If the next router correctly receives the flit, then it sends an Acknowledgement (Ack) signal allowing to delete the copy from the previous router. In the case where the flit is not correctly received, the flit can be sent again by the previous router.

STALL/GO Flow Control: This flow control necessitates two wires to know the state of the next buffer. If this one can store a flit, then the signal GO is raised, otherwise, the signal STALL is raised to prevent flits from being sent.

1.1.3 Conclusion

As mentioned in this section, the NoC paradigm offers high performances and high scalability for multiprocessor-based on-chip communications. NoCs can be set up in many ways through the different parameters exposed in this part offering high adaptability for desired applications. However, despite the high performances provided by the NoCs, i.e. low packet latency and high bandwidth, they can be victims of internal and external effects that disrupt their functionalities. These effects are described in the next part of this chapter.

1.2 Origins and Sources of Faults

In this section, we present the internal and external effects which can affect the correct NoC behavior and, in general, the integrated circuits [15, 42]. These effects are generally split into three main categories which are i) radiations [43], ii) aging effects [15] and iii) manufacturing defects [44]. These three main effects are detailed in this section to understand where the faults, which affect the integrated circuits, come from. Then, the fault models used to characterize the effects of the induced faults are presented.

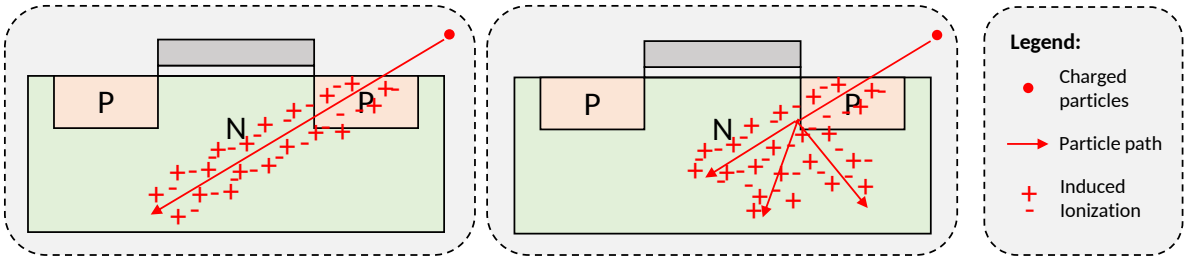
1.2.1 Faults Induced By Radiations

Radiations are well known to be the most common external effects which degrade the integrated circuits [12]. They are well studied since several decades to understand how they interact with electronic systems at the transistor-scale. Radiations may originate from four sources [45] which are i) coronal mass ejections, ii) solar winds, iii) solar flares and iv) cosmic rays. These radiation sources project particles, i.e. protons, neutrons, electrons, heavy ions, alpha particles, and muons, which interact with the transistors degrading their performances and functionalities. These interactions are particularly present in space and high altitude [46], i.e. harsh environments, where the particles are imprisoned in the Van Allen belts due to the magnetic fields. However, radiations can also impact integrated circuits at the terrestrial-level [47] due to particles which interact with the atmosphere. This phenomenon is mainly due to proton particles and it is also called proton shower. The radiation impacts are split into two categories [48] which are the cumulative effects and the Single Event Effects (SEEs).

1.2.1.1 Cumulative Effects

The long-term exposition to radiations in harsh environments can degrade the integrated circuits by changing the characteristics of the transistors which compose them [49]. These cumulative effects are generally classified into two categories which are i) the Total Ionizing Dose (TID) effects [50] and the Total Non-Ionizing Dose (TNID) effects [51], also called the displacement damage effects.

TID effects [50] degrade the integrated circuits by transferring ionizing energy to the transistors. The impacts on the electronic systems can be multiple such as threshold voltage shifts, leakage currents and timing skews [52]. Moreover, TID effects can make the



(a) Primary interactions (b) Secondary interactions (c) Legend

Figure 1.9: PMOS transistor ionization due to a particle strike.

transistors more sensible to other effects, such as the SEEs which are detailed below, aggravating the impacts on the transistors [53]. Contrary to TID effects, TNID effects [51] transfer non-ionizing energy through the particle strikes on the transistors of the integrated circuits. These strikes have the effect of moving the atoms in the semiconductors leading to potential damages or defects in the semiconductors. However, as TNID effects have low impact on Application-Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) architectures [48], they are often not considered.

1.2.1.2 Single Event Effects

In addition to the cumulative effects, charged particles which strike the components of the integrated circuits can induce SEEs by deposing ionizing energy [48, 12]. Figure 1.9 presents the two interaction types which can occur when a particle strikes a CMOS transistor. First, the ionizing energy can be deposited by particles which strike the transistors, i.e. primary interactions, as represented in Figure 1.9a. In addition, the charged particles can interact with the atoms of the semiconductors causing nuclear reactions that will create other charged particles. These additional particles can deposit ionizing energy in the semiconductors, i.e. secondary interactions, as shown in Figure 1.9b. As a result, the semiconductors can be affected by non-destructive effects, i.e. transient faults, or destructive effects, i.e. permanent faults.

Non-destructive effects: The most common non-destructive effects is the Single Event Transient (SET) [54]. The SET is a temporary voltage spike released by the transistor which is stricken by a charged particle. The SETs can spread as well in the digital circuits as in the analogue circuits. The capture of one SET by a storage element, such as flip-flop, latch or SRAM cell, can lead to a bit-flip of the memory value called Single Event Upset

(SEU) [55]. Nowadays, the SEUs are becoming more common due to the technology scaling [56]. Indeed, with the reduction of the transistor size and of the gap between them, one charged particle which strikes the circuit can impact several transistors at the same time causing multiple SEUs, called Multiple Cell Upset (MCU) or Multiple Bit Upset (MBU) when the impacted cells belong to the same word [48]. In this way, the share of MCUs and MBUs increases reaching today up to 20% of the SEUs which occur in space applications [57]. These effects are transient faults which can be removed by re-writing memory values. However, they can have critical impacts on complex devices causing component malfunctioning until reset of the entire system, called Single-Event Functional Interrupt (SEFI) [58].

Destructive effects: In addition to the transient effects, charged particles can have destructive effects on the transistors, i.e. permanent or hard faults. These permanent damages can be due to different phenomena [48, 59]. The most common is the Single-Event Latch-up (SEL) [60, 61] which is characterized by a high current flow which can destroy the transistor through thermal effects if the power supply is maintained. A similar effect, but less common, is the Single-Event Snap-Back (SESB) [62] which damages the circuit due to the current effects. The strikes of the heavy ions can also destroy the transistors causing Single-Event Gate Ruptures (SEGRs) [63, 64], also called Single-Event Dielectric Ruptures (SEDRs) or Single-Event Hard Errors (SEHEs) when memory cells are impacted. Other destructive effects, such as the Single-Event Burnouts (SEBs) [63] can be observed but they are not frequent in ASIC and FPGA applications.

1.2.2 Faults Induced By Manufacturing and Aging Defects

The manufacturing and aging defects [42, 15] are the second well known source of faults in the electronic devices. While radiations are especially known to induce soft faults, the manufacturing and aging defects induce hard faults which cannot be removed. The occurrence rate of these defect types is often represented with the help of the bathtub curve [65, 66] which is depicted in Figure 1.10. In this figure, we can observe that the devices are more prone to faults during the start of their life, i.e. manufacturing defects and during the end of their life, i.e. aging defects.

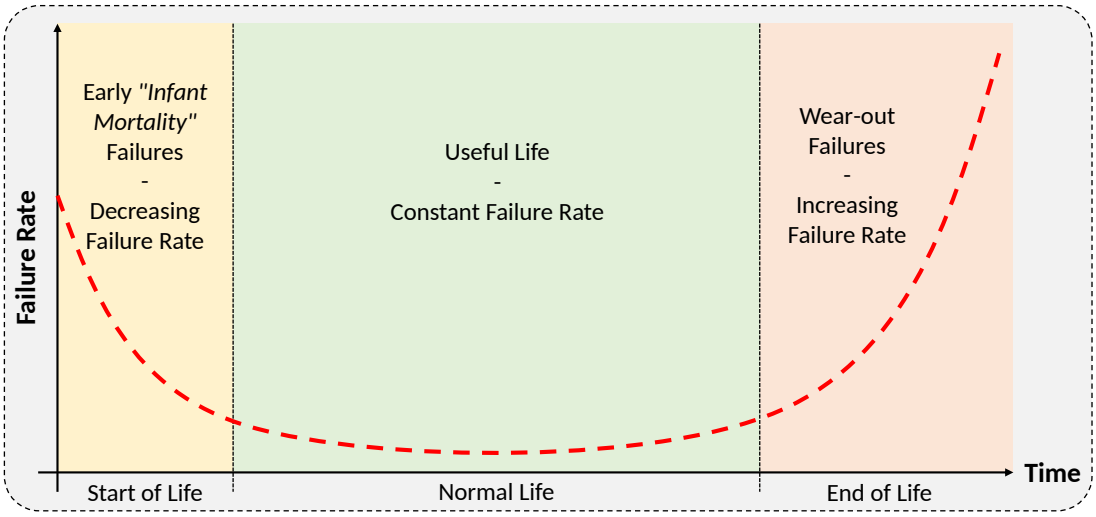


Figure 1.10: Failure rate during the system lifetime.

1.2.2.1 Aging Defects

Aging defects are due to physical effects which impact the devices along their lifetime, such as electromigration [67], Negative Bias Temperature Instability (NBTI) [68], Hot Carrier Injection (HCI) [69] or Time Dependent Dielectric Breakdown (TDDB) [70]. Accumulation of these effects leads to permanent faults which occur more frequently at the end of the device life, as shown in Figure 1.10. Moreover, the occurrence of these defects increases with the technology scaling [10], impacting the correct behavior of the entire system [14, 15]. Some of these physical effects are briefly described in the following paragraphs to understand where the faults induced by the aging defects come from.

Electromigration: The first source of aging defects is the electromigration [67, 71]. This effect occurs in the wires where the metal atoms are transported over the time under the current effect aggravating the variations of the wire thickness. These variations can impact the propagation delay inducing timing faults in the system. But, wire thickness variations are more known to induce permanent damages called open or short faults. Open faults are due to the cutting of the wires where its thickness was too thin. Short faults, also called bridge faults, result from the unwanted connection of two or more wires which occurs when the thickness of one wire becomes too large connecting it to the adjacent wires.

Negative Bias Temperature Instability (NBTI): The NBTI [72] affects the components by increasing the voltage threshold along the time. The induced damages can be partially reversed by restarting the device but they are often permanent. This phenomenon is accentuated by temperature increases and by low negative voltages. While the NBTI impacts the PMOS transistors, the NMOS transistors can be affected by a similar phenomenon called Positive Bias Temperature Instability (PBTI) [73]. However, as the PBTI impacts on the PMOS transistors are very small compare to the NBTI impact on the NMOS transistors, they are not already mentioned in the literature [15].

Hot Carrier Injection (HCI): The HCIs can permanently damage transistors by changing their characteristics [74], such as the threshold voltage. This effect is due to the electrons and holes, i.e. fast carriers, which are accelerated too fast by electric fields.

Time Dependent Dielectric Breakdown (TDDB): The TDDBs [75] occur when high electric fields cross the transistors causing permanent damages. Low electric fields can also induce TDDBs along the time due to the charge accumulations inside the transistors. Moreover, the occurrence of TDDBs is accentuated with the voltage scaling in recent technologies.

1.2.2.2 Manufacturing Defects

In addition to the aging defects, the devices are also prone to faults at the beginning of their life, as shown in Figure 1.10. These faults are due to the variability of the manufacturing processes which can affect the behavior of the components [44]. These defects are generally detected and corrected during the conception phases but some transistors can become faulty after several operation hours. To address this issue, burn-in processes [76] are applied to the devices to ageing them until the normal operation phase, called useful life in Figure 1.10. For that, the devices are turned on during several hours during which process variations, temperature variations and mechanical constraints are applied. However, some manufacturing defects can stay undetectable and induce permanent faults during the useful life phase.

1.2.3 Fault Modeling

The fault impacts on the circuits are often characterized using fault models [77, 25]. The transient and intermittent faults are generally modeled with the bit-flip fault model

which consists in switching the state of one or several bits in the word, i.e. a logic one (zero) is turned into a logic zero (one). Concerning the permanent faults, several fault models can be found in the literature. The most used stays the stuck-at fault model which consists in setting the affected bit at one, i.e. stuck-at-one fault model, or at zero, i.e. stuck-at-zero fault model. Other fault models are available in the literature for the permanent faults such as the short or bridge fault model. In this latter, the unexpected connection between two wires distorts the state of the first wire due to the logic value of the second wire.

1.2.4 Conclusion

As mentioned in this section, electronic devices can be impacted by external and internal effects causing temporary or permanent damages to the components. We presented the main fault sources which can affect the devices but many others, such as electromagnetic interferences, electrostatic discharges, process variability and dynamic temperature variations, can also induce transient or permanent faults in systems. As we have seen through this section, there are many names for faults. For sake of simplicity, in the rest of this manuscript we decide to name Single Hard Errors (SHEs) the faults which impact only one bit and Multiple Hard Errors (MHEs) the faults which impact several adjacent bits. Finally, in this section we described the fault impacts at transistor level. However, these faults can differently affect NoCs at application level as detailed in the next section.

1.3 Fault Impacts on Network-on-Chips

In this section, we present how the faults can impact the NoC behavior. As the induced faults have not the same impacts according to the part of the NoC that is affected, we discriminate the exploration into two distinct parts, i.e. the logic part and the datapath. Then, the variations in the duration of faults are presented.

1.3.1 Fault Impacts on the Logic Part

Faults induced in the NoC logic parts, i.e. arbitration, routing controller, flow control, etc., directly impact the performances leading to power over-consumption, as explained below. In this part, we describe the impacts that a fault can have on the different NoC logic parts.

1.3.1.1 Fault in Arbitration

The arbitration defines the priority of the input packets in routers. As seen in Section 1.1.2.2, an arbiter needs to be fair to be efficient. However, faults impacting the router arbiter can affect its fairness leading to starvation phenomenon. This latter can induce congestion in NoC and, in the worst cases, it can lead to bottlenecks in router inputs which never obtain the priority. In this case, packets need to be dropped and re-transmitted which drastically impacts NoC performances and power consumption.

1.3.1.2 Fault in Routing Controller

As mentioned in Section 1.1.2.3, routing controllers are used in each router to determine the path taken by the packets to cross the NoC according to the routing algorithm. Faults impacting controller blocks can lead to wrong routing computations resulting in packet miss-routing. These miss-routings can lead to several complications in NoC such as congestion, bottlenecks, deadlocks or livelocks which cause packets dropping and re-transmission impacting drastically NoC performances and power consumption.

1.3.1.3 Fault in Flow Control

As seen in Section 1.1.2.4, flow control is used to confirm the correct packet transmissions and to indicate the buffer states. Faults impacting NoC can have various effects on this logic part. They can lead to false acknowledgements or false non-acknowledgements causing packet re-transmissions. Moreover, for false acknowledgements, re-transmission requests are made only after a certain amount of time which drastically impacts the latency. In addition, faults impacting buffer states can lead to packet dropping or overwriting. Therefore, packets need to be re-transmitted which increases the packet density in NoC causing performance degradation and power over-consumption.

1.3.1.4 Fault in Virtual-Channel Controller

As presented in Section 1.1.2.1, virtual channels are used in NoC architectures to decrease the congestion increasing the bandwidth. Faults impacting virtual-channel controllers can affect the priority between the different channels leading to packet retention, i.e. starvation phenomenon. This latter induces congestion and can lead in the worst cases to packet re-transmissions impacting drastically NoC performances and power consumption.

1.3.2 Fault Impacts on the datapath

As the datapath, i.e. crossbar, buffers, and interconnections, is the biggest part of the NoC in terms of area [24], it has higher probability to be impacted by faults. These faults impact packets transiting on the NoC. As mentioned in Section 1.1.1.3, the header flit contains information to route the packets through the NoC and to acknowledge the transmission. Then, faults impacting headers can lead to routing errors, that can cause congestion, bottlenecks, deadlocks or livelocks, hence, packets can be dropped and re-transmitted impacting drastically the NoC performances. Furthermore, payloads contain the necessary data for the execution of the application. Faults impacting the data can compromise the execution of the application affecting the results. Considering error-sensible applications, packets affected by faults are re-transmitted at the cost of NoC performances due to congestion increasing. In the worst case, faults can cause critical failure of the application which can be dramatic in some fields such as aeronautic and aerospace.

1.3.3 Fault Evolution Over the Time

In Section 1.2.1, we saw that the faults induced by radiations can be transient or permanent according to the time they affect the integrated circuits. On the same base, the faults impacting NoC may have different duration times. While a fault which impacts NoC only during few cycles is called transient or soft fault, it is called permanent or hard fault when it has an impact with no limit of duration. A third category of faults, called intermittent faults, can be found in the literature. This kind of faults affects the NoC during tens or hundreds cycles before disappearing [78].

1.4 Conclusion

In this chapter, we saw that the faults due to external or internal effects can dramatically impact the performances and the NoC behavior, especially in terms of latency, QoS and power consumption. To tackle this, fault tolerant methods need to be included in the NoC architecture to detect and correct the potential faults which occur at-run time allowing to maintain NoC performances. Especially, these methods can avoid critical failure and extend the NoC lifetime face to the accumulation of faults, and especially of permanent faults. Indeed, the accumulation of permanent faults is particularly critical for NoC architectures since they can compromise the communications leading to a critical

failure of the devices. In this case, the NoC is no longer usable.

Chapter 2

State-of-the-Art of Fault-Tolerant NoC

In this chapter, we present the current state-of-the-art of the fault tolerant field applied to Network-on-Chips (NoCs). The aim of this field is to enhance the reliability and the lifetime of the NoCs in presence of faults which can appear when systems operate in harsh environments. For that, existing methods are applied to NoCs to manage fault occurrences during device operating time. The chapter is organized as follows: First, a general description of the fault-tolerant NoC field is made in Section 2.1. Then, Section 2.2 presents the methods based on the information redundancy which can be used to detect and correct faults occurring in the NoCs. Other methods based on reconfiguration of NoC architectures are mentioned in Section 2.3. Section 2.5 presents available methods to detect and diagnose the faults in NoCs. Recent proposed approaches based on approximate communications and approximate computations are provided in Section 2.4. Finally, several examples of fault-tolerant NoC architectures which group several techniques presented in this chapter are detailed in Section 2.6 before to conclude the chapter in Section 2.7.

2.1 Overview of fault-tolerant NoCs

In this section, we present an overview of the fault-tolerant field applying to NoC architectures. Despite the fact that we address the mitigation of permanent faults in this manuscript, this chapter presents a global survey of fault-tolerant NoC-based methods. These methods are generally classified into four categories [17, 79, 13] which are:

— **Fault Detection:** These techniques consist to detect the fault occurrences in the NoC to report them without knowing precisely their location.

— **Fault Diagnosis:** Diagnosis methods allow to detect and precisely locate faults in

NoC to manage them thanks to recovery techniques.

— **Fault Correction:** Allow to correct or remove the faults from the NoC ensuring accurate transmissions.

— **Fault Mitigation:** Contrary to the correction techniques, mitigation methods are used to reduce the impact of faults on the NoC instead of correcting them. Then, the faults are always present but their impacts can be accepted up to a certain limit.

Fault tolerant methods can be used at different levels [80], i.e. software or hardware levels, to manage transient, intermittent, and permanent faults [15]. While techniques used at hardware level are often based on information, spatial, or temporal redundancies, techniques applied at software level use NoC logics, i.e. routing algorithm, arbitration, flow control, etc., to handle faults [16]. In the rest of this chapter, we present more in details a set of representative approaches from the literature.

2.2 Detection and Correction Using Information Redundancy

In this section, we present the methods based on the information redundancy, also called Error-Detecting Codes (EDCs) and Error-Correcting Codes (ECCs) [13]. For sake of simplicity, we will use the designation ECC to define both of the EDCs and ECCs since in most cases these codes are able to make both of fault detection and correction. These methods are well used in the literature to detect, diagnose and correct faults which occur in the NoCs and are based on redundant bits [81, 82]. First, we propose a theoretical presentation of the characteristics of the ECCs. Then, we present some conventional ECCs which are well known in the fault tolerant field. Exotic ECCs able to manage large number of faults in the NoC are also summarized. Finally, the ECC implementations in NoC architectures are described.

2.2.1 ECC Theoretical Presentation

The ECCs are generally characterized by the length of the non-encoded data k, the length of the encoded data n and the minimum Hamming distance d_{min}. Using this latter, the number of detectable faults F_d and correctable faults F_c can be calculated with

Equations 2.1.

$$F_d = d_{min} - 1 \qquad F_c = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \tag{2.1}$$

As the ECCs use redundant bits to protect data on the communication media, they have an impact on the transmission performances, i.e. on the bandwidth. This impact can be quantified by calculating the information rate of the code which is given by Equation 2.2. However, the information rate gives no information about the number of detectable and correctable faults of the considered ECC. Then, both of these metrics need to be used to evaluate the code efficiency.

$$R = \frac{k}{n} \tag{2.2}$$

Finally, the hardware costs, i.e. area cost and power consumption, need to be taken into account in the ECC characterization since they are related to the ECC efficiency. For instance, the flit size can be increased to avoid the impact of the redundant bits on the NoC bandwidth, and so on, increase the efficiency of the ECC. However, increasing flit size drastically increases hardware costs induced by the ECC. Then, the code efficiency must be related to its costs to be fairly quantified and compared.

2.2.2 Conventional Error-Correcting Codes

Conventional ECCs, such as parity bit, Hamming code, etc., are the most used codes in practical cases for on-chip or embedded architectures due to their good trade-off between efficiency and hardware costs. However, these codes can generally correct few faults. In the following parts, we present ECCs which are used in the NoC architectures.

2.2.2.1 Parity Bit

The parity bit [13] is the simplest ECC which can be found in the literature. It is used to detect one fault in the flits using one redundant bit. This bit is computed by XOR operations ($\oplus$) between all bits which compose the flit. In other word, it is equal to 1 if the flit contains an odd number of 1 and equal to 0 if the flit contains an even number of 1. As the minimal Hamming distance of this code is equal to 2, it can detect a fault number equal to $F_d = 2 - 1 = 1$. However, as the computation of the correctable fault number gives $F_c = \left\lfloor \frac{2-1}{2} \right\rfloor = 0$, the parity bit cannot correct a fault.

2.2.2.2 Cycle Redundancy Check (CRC) Code

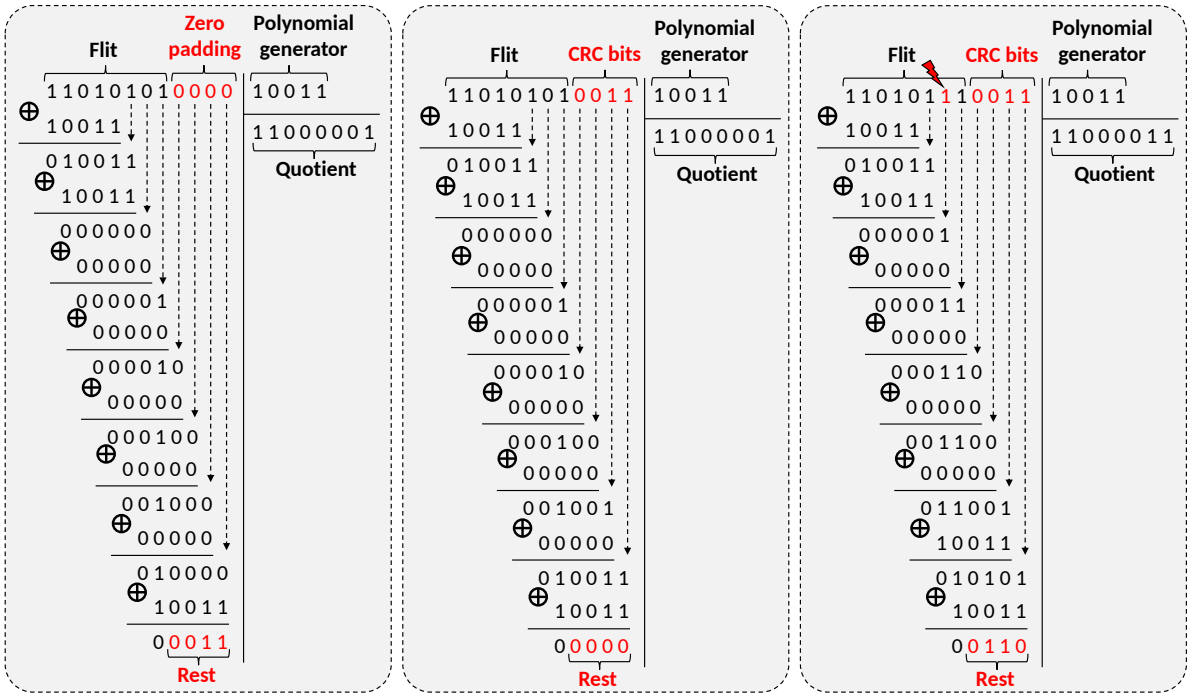
The Cycle Redundancy Check (CRC) codes [83] are used to detect faults in flits. For that, n redundant bits (CRC-n) are added to each flit. These bit values are determined from the rest of the binary Euclidean division between the flit which needs to be encoded and a polynomial generator of degree $n+1$. Moreover, a zero padding of n zeros is applied to the flit. The decoding of the flit allows These faults impact packets transiting the NoCto detect faults through the rest of the binary Euclidean division between the encoded flit and the polynomial generator. If the rest of the division is equal to zero, then the encoded flit is not impacted by faults. Else, at least one fault has been detected. However, this ECC cannot diagnose the fault position in the flits. Then, it is often used to detect the presence of faults and activate other fault tolerant methods to manage the faulty flits.

For example, we consider the encoding of the message $m_2 = 11010101$ with a CRC-4 using the polynomial generator of degree 5 $G_2 = 10011$. As displayed in Figure 2.1a, the four redundant bits are determined from the rest of the binary Euclidean division between the flit and the polynomial generator. Then, we obtain the encoded flit $F_2 = 110101010011$ where the four red bits are the four redundant bits of the CRC-4. As displayed in Figure 2.1b, the flit decoding is performed by computing the binary Euclidean division between the encoded flit and the polynomial generator. In this case, we can note that the rest of the division is equal to 0000 which means that the flit is not impacted by faults. However, if we consider a faulty bit in the encoded flit, as illustrated in Figure 2.1c, we note that the rest of the division is not equal to 0000 which means that the flit is impacted by faults, i.e. one fault in our example.

The CRC codes are mainly used in the literature to detect the faults which occur during the NoC communications [84, 85]. For example, in [86], four CRC codes are used in each router to detect faults. In [87], CRC codes are used to detect the faulty flits at the reception. In most cases, the packet re-transmission is used to manage the faults.

2.2.2.3 Hamming Code

The Hamming code [88] is the most used ECC to protect on-chip communications due to its good trade-off between its hardware costs and its efficiency. This code allows to detect and correct one fault [19]. However, the number of detectable faults can be increased to two bits with the extended version of the Hamming code [89] which consists of applying a parity bit to the encoded flit. The redundant bits of the Hamming code are



(a) CRC-4 encoding (b) CRC-4 correct decoding (c) CRC-4 faulty decoding

Figure 2.1: CRC-4 encoding and decoding.

computed using XOR operations ($\oplus$) between several specific bits of the flit, as displayed in Figure 2.2a for an 8-bit flit. The encoded flit is then composed of the flit and the redundant bits. During the decoding phase, the redundant bits are re-computed and compared to the transmitted redundant bits using XOR operations, as displayed in Figure 2.2b. The resulted bits $s_3s_2s_1s_0$ form a binary number which indicates the position of the fault or which is equal to zero in the fault-free cases.

The Hamming code is largely used in NoCs to detect and correct transient or permanent faults. In most cases, the Hamming code encodes the entire flit [90, 91, 92]. However, the number of detectable and correctable faults per flit can be increased by splitting the flit into several parts which are encoded with smaller Hamming codes [93, 94]. Moreover, this approach enhances the latency compare to the standard Hamming code which encodes the entire flit. This ECC can be also used to only detect the faults reducing the hardware costs and the impact on the NoC performances. In this case, other fault tolerant mechanisms are activated to manage the faults such as re-transmission [95] or reconfiguration using spare resources [96].

Flit:

$d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1\ d_0$

Redundancy bits computing:

$r_0 = d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_6$

$r_1 = d_0 \oplus d_2 \oplus d_5 \oplus d_6$

$r_2 = d_1 \oplus d_2 \oplus d_3 \oplus d_7$

$r_3 = d_4 \oplus d_5 \oplus d_6 \oplus d_7$

Encoded flit:

$d_7\ d_6\ d_5\ d_4\ r_3\ d_3\ d_2\ d_1\ r_2\ d_0\ r_1\ r_0$

(a) Hamming encoding

Encoded flit:

$d'_7\ d'_6\ d'_5\ d'_4\ r'_3\ d'_3\ d'_2\ d'_1\ r'_2\ d'_0\ r'_1\ r'_0$

Redundancy bit computing:

$s_0 = r'_0 \oplus d'_0 \oplus d'_1 \oplus d'_3 \oplus d'_4 \oplus d'_6$

$s_1 = r'_1 \oplus d'_0 \oplus d'_2 \oplus d'_5 \oplus d'_6$

$s_2 = r'_2 \oplus d'_1 \oplus d'_2 \oplus d'_3 \oplus d'_7$

$s_3 = r'_3 \oplus d'_4 \oplus d'_5 \oplus d'_6 \oplus d'_7$

Syndrom:

$s_3\ s_2\ s_1\ s_0$

(b) Hamming Decoding

Figure 2.2: Encoding and decoding of a 8-bit flit with the Hamming code.

2.2.2.4 Other Codes

Numerous other conventional ECCs can be found in the literature to detect and correct more faults such as the Bose–Chaudhuri–Hocquenghem (BCH) code [97, 98, 99], the Hsiao code [81, 100, 101], the Reed-Solomon (RS) and Low-Density Parity-Check (LDPC) codes [102] or the turbo codes [103]. However, these codes are rarely used in the NoC architectures due to their high hardware costs in terms of area and power consumption [104].

2.2.3 Exotic ECCs for High Fault Coverage

Exotic ECCs can be found in the literature allowing the detection and the correction of a high fault number. However, these codes often have high hardware costs and impact the NoC performances in terms of latency. For example, in [105], an ECC is proposed to detect and correct up to 14 and 7 faults respectively. Some other approaches based on the use of ECCs are presented in the following parts.

2.2.3.1 ECC Coupling

Exotic ECCs can be the result of a mix between several conventional ECCs. For example, in [106], a Hamming code and a Hsiao code are merged allowing, respectively, the

detection and the correction of 6 and 3 adjacent faults. In [101], an interleaved combination of CRC and Hsiao codes allows for up to 2 and 4 adjacent faults to be respectively detected and corrected. In [107], Hamming code and parity bit sharing are used to correct up to 24 faults at the price of very high hardware costs, i.e. around ×3-4 the baseline architecture.

The number of detectable and correctable faults can be increased by coupling the ECCs with other methods in the literature. In [108], hamming code is coupled with duplication method allowing 4 error detections and 3 error corrections and up to 8 error detections and 7 error corrections when the bits are interleaved. In [109], the flits are encoded with Hamming code, interleaved and a forbidden pattern code is finally applied allowing the correction of 14 adjacent faults in a 32-bit flit. In [87], a machine learning technique is applied to predict the faults during the transmissions. While a CRC code is used to detect the fault, other ECCs are activated in the router where the faults are susceptible to occur according to the prediction of the machine learning algorithm. In [110], CRC and Hamming code are used with a reinforcement learning mechanism allowing to enable and disable fault tolerant technique according to the probability of fault occurrence in the NoC.

2.2.3.2 Two-Dimensional ECC

The 2-D ECCs are proposed for this purpose to enhance the detection and correction capabilities of the codes. For this purpose, flits are generally split into sub-groups, called subflit in this manuscript, to form a matrix. Then, this matrix is vertically and horizontally encoded with ECCs. For example, in [111], the flits are encoded using a decimal matrix code which uses logic operations to horizontally encode the flits and parity bits to vertically encode the flits. In [112], the two dimensions of the matrix are encoded with parity bits using interleaved data encoding. The number of parity bits can be set independently for the vertical and the horizontal encoding allowing configurable detection and correction capabilities. For example, to detect and correct 4 faults in a 64-bit flit split into 8 subflits of 8 bits, 2 parity bits are necessary for each dimension increasing the number of redundant bits to 32. Another approach is proposed in [113] where an extended Hamming code is used to horizontally encode the flits. The redundant bits induced by the extended Hamming code, except the parity bits, are then vertically encoded using parity bits. In this way, the redundant bits of the Hamming codes used to compute the vertically parity bits are not transmitted on the NoC reducing the hardware costs. For example, to encode a 32-bit flit split into 8 subflits of 4 bits, 20 redundant bits are necessary to encode the flits allowing

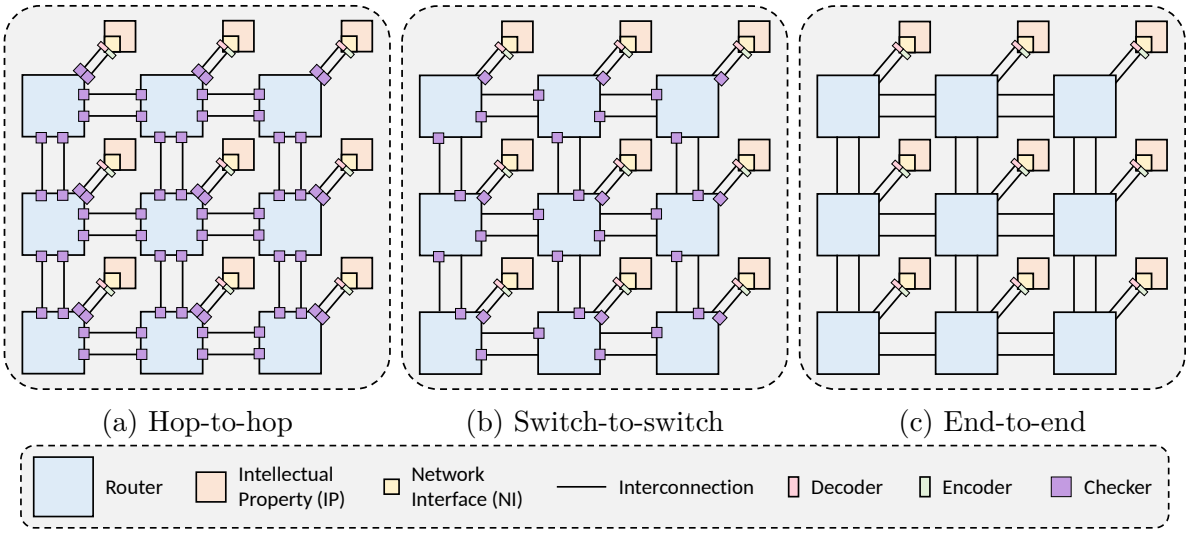
the detection and the correction of 6 and 5 faults. In addition, the author proposes a duplication of each bit to increase the number of correctable faults up to 12.

2.2.4 ECC Distribution in NoC Architecture

The distribution of the blocks of the ECCs, i.e. the density and the positions of encoders and decoders in the NoC, can be explored to reduce the hardware costs, in terms of area and power consumption, relaxing the efficiency of the codes. For example, in [114], the NoC is split into regions and the ECCs are used to detect the faults only at the region borders.

Many ECC distributions are proposed in the literature [115]. The most well know are represented in Figure 2.3. In these configurations, encoders and decoders are located in the Network Interfaces (NIs) while inter-decoders [116], also called checkers in this manuscript, are placed in routers and interconnections of the NoC. We can note that, whatever the distribution used, only the checker locations are changed since encoders and decoders are always located in NIs. In Figure 2.3a, checkers are implemented at each router input and at each router output. This distribution, also called hop-to-hop, offers a higher detection and correction capabilities at the price of high hardware costs. Figure 2.3b proposes a distribution called switch-to-switch where checkers are placed only at the router inputs or at the router outputs. It offers a good trade-off between the efficiency and the hardware costs according to the ECC used. Finally, in Figure 2.3c, a lightweight distribution, called end-to-end, is proposed where checkers are not used in the architecture. Only encoders and decoders are present in the NIs to drastically reduce the hardware costs sacrificing the ECC efficiency. Of course, many other ECC distributions can be found in the literature [117] to optimize the trade-off between the efficiency and the hardware costs.

Furthermore, some approaches can be found in the literature to reduce the hardware costs of the ECCs maintaining their performances. For example, in [118], the header flits are encoded with several small Hamming codes and only the detector part of the checkers is implemented in the routers. In the case where one or several faults are detected in the header flit, then the packet is deflected towards the associated NI to be decoded and corrected. Once the faults are corrected, the packet is re-encoded and re-send in the NoC to reach its final destination.



(a) Hop-to-hop (b) Switch-to-switch (c) End-to-end

(d) Legend

Figure 2.3: ECC distributions in a 3×3 NoC.

2.3 Detection and Correction Based on NoC architectures

This section presents fault-tolerant methods which use the NoC architecture to detect and correct the faults. First, methods based on routing algorithms to manage faults are presented, then, we explore methods exploiting spatial and temporal redundancies. Finally, fault tolerant NoC topologies are summarized.

2.3.1 Fault-Tolerant Routing Algorithms

Fault-tolerant routing algorithms are used in the NoCs to circumvent faults [16]. These algorithms allow to maintain the packet correctness, hence extend the lifetime of NoCs [119, 120], i.e. they can operate for a longer period of time. First, we present some fault-tolerant routing algorithms published in the literature. Then, we explore the approaches of the literature which enhance NoC performances when fault-tolerant routing algorithms are applied. Finally, NoC architectures covered by these methods are described.

2.3.1.1 Overview

In general, routing algorithms need to be adaptive to manage faults occurring in NoCs. These algorithms can be table-based [121, 122, 123] or logic-based [124, 125, 126].

While the logic-based routing algorithms are able to manage several faults with acceptable hardware costs, the table-based routing algorithms can manage a high fault number but the hardware costs increase drastically with the NoC size.

However, deterministic and semi-adaptive routing algorithms can be enhanced to avoid faulty paths in the network. For instance, the deterministic XY routing algorithm can be associated with other routing algorithms, i.e. YX [127], turn model [128] and others [129], to provide an algorithm able to circumvent faulty links and faulty routers. These routing algorithms are well known to have reduced hardware costs but they can only manage few faults in the NoC.

Most of the proposed fault-tolerant algorithms proposed in the literature are based on the turn model routing algorithm [130, 131, 128]. These algorithms permit to maintain NoC performances through the management of high fault number. However, the induced hardware costs are significantly increased.

Finally, another type of routing algorithms was proposed in the literature to manage a high fault number. The stochastic routing algorithms [15, 132, 133], also called flooding algorithms, propagate the packets in all possible directions at each router of the NoC allowing for at least one fault-free copy of the packet to reach the destination if it is possible. However, these algorithms are less studied in recent years due to their high impact on the NoC performances, as it saturates the NoC from several packets for each transmission.

2.3.1.2 Performances Enhancement

As mentioned previously, the fault-tolerant routing algorithms can drastically impact the NoC performances. Many works have been proposed in the literature to reduce these impacts.

Some approaches propose to split the NoC into regions [134, 18, 41]. When a region is impacted by one or several faults, it is considered as faulty and can be circumvented by the packets. This solution presents large advantages when several faults are present in a same region since the number of restrictions is reduced, limiting the complexity of the routing algorithm. However, the deflection of the packets around the faulty region can drastically impact the NoC performances due to the induced congestion. Segment-based approaches [135, 136] are also proposed to divided the NoC into segments where the path restrictions can be set with a low granularity, reducing the impact on the NoC performances.

Other approaches analyse all the paths to forward the packets toward the destination Intellectual Property (IP). In these approaches, a fault-free path is selected to limit the NoC congestion [137, 138, 139]. For example, in [140], the chosen path is determined using a tree-based organization of the possible paths.

2.3.1.3 Covered NoC Architectures

The fault-tolerant routing algorithms proposed in the literature enhance many NoC architectures against fault occurrences, such as deflection-based NoCs [105, 141], multicast architectures [142, 143]. In addition, the routing controller unit can be enhanced to manage faults which impact it [144] avoiding miss-routing, packet dropping and congestion in the NoC. However, these errors can be detected at-run time by detecting the miss-routing [145, 41] or by comparing the destination and the source of the packets [146].

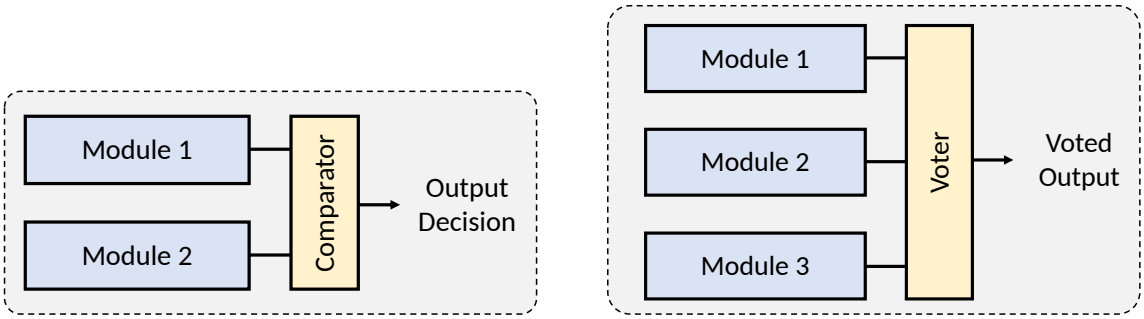
2.3.2 Spatial and Temporal Redundancies

Spatial and temporal redundancies [16, 15, 147] are often used in the literature to manage faults in NoC architectures. While techniques based on spatial redundancy are able to manage both transient and permanent faults, the methods based on temporal redundancy usually only manage transient faults. In the rest of this part, we present the works based on these redundancies.

2.3.2.1 Circuit Replications

Circuit replication, also called N-Modular Redundancy (NMR) [13, 148], consists to fully or partially replicate N times the architecture to protect. The replicated parts of the architecture are called modules. Most popular approaches are the Double Modular Redundancy (DMR) [13] and the Triple Modular Redundancy (TMR) [149], which are respectively composed of two and three modules, as depicted in Figure 2.4.

As depicted in Figure 2.4a, the DMR compares the outputs of the duplicated modules [13] to detect the faults. This approach can detect a fault when only one module is faulty, or if both modules are faulty and provide different outputs. Moreover, the fault positions cannot be diagnosed and the faulty module cannot be determined. Due to this lack of information, the DMR is marginally used in the literature to protect the NoC architectures.



(a) Double modular redundancy. (b) Triple modular redundancy.

Figure 2.4: DMR and TMR principles.

As shown in Figure 2.4b, the TMR computes the output based on a vote over triplicated modules outputs [13]. TMR can then mask a fault of one faulty module. Similar to DMR, the faults cannot be diagnosed in the faulty modules. However, the voter can determine which module is impacted by the faults. This approach is often used in the NoC architectures [150] to enhance specific parts despite the hardware overheads which require more than three times the baseline area. However, the impact on the performances is low compared to other methods, such as ECCs [151], since only the voter is added in the critical path. Based on this, some works are proposed in the literature to reduce the hardware overheads induced by TMR allowing the use of this method for lightweight NoC architectures. In [152], a lightweight version of the TMR is proposed. In this approach, only two modules are activated to detect the occurrence of faults using a comparator. When a fault is detected, the third module is activated to manage the faulty module using a voter instead of the comparator. This approach provides an interesting power consumption reduction. In [153], redundant logic is added in the isolated combinatorial circuits to avoid the replication of the entire modules reducing the hardware costs of the standard TMR. In [154], the last logic-gate level of the triplicated modules is designed with a triple transistor logic which acts as a voter circuit, and thus, replaces the voter reducing the hardware costs in terms of area and power consumption.

Other approaches based on the circuit replication are proposed in the literature at different scales. For instance, in [155], circuit replication at the transistor-level using quadded transistors is proposed. However, the hardware overheads induced by this approach are drastically increased, i.e. around eight times for the area, four times for the power consumption and two times for the delay.

2.3.2.2 Reconfiguration

The reconfiguration [156] is mainly used in the NoCs to manage the permanent-fault occurrences in routers and interconnections [157]. Reconfiguration approaches can be split into several categories based on i) spare resources, ii) default back-up path, and iii) reconfiguration in degraded mode.

Spare Resources: Methods using spare resources are well known in the NoC field to replace faulty elements impacted by permanent faults. These techniques increase the NoC lifetime. Many works of the literature propose to use spare interconnections or wires [158] to manage permanent faults. In [159], the faulty routers are managed by adding redundant interconnection between the associated IP and one of the neighboring router. In [160], interconnections are segmented into several parts which are enhanced with one redundant wire to replace the faulty one. Other methods propose to use redundant routers as spare resources. In [161], one spare router is added per 4-router square sub-networks to replace one faulty router among the four which compose the sub-network. In [162], each router of the network is duplicated and connected by multiplexers to increase the reliability and the lifetime of the network. In [163], one spare router is added at each column of the mesh NoC to replace one faulty router of this column. For that, IPs are connected to the two vertically close routers of its associated router using multiplexers. However, spare resources induce significant hardware costs in terms of area and power consumption. To reduce these costs, fine-grain insertion of spare resources is proposed in the literature. For example, in [164], two spare registers are used to replace the faulty status fields which control the virtual channels of the input buffers. In [84], spare buffers are added at the router inputs and outputs to provide additional datapaths. Finally, another approaches to reduce the hardware costs consist to use polymorphic spare resources which can replace more than one functionality in the network [165, 166].

Default Back-up Paths: These approaches consist to by-pass the faulty routers or faulty interconnections in the NoC. For that, the multiplexers which manage the default back-up paths need to be reconfigured to by-pass the faulty elements maintaining the connectivity between the IPs. In [167], a unidirectional cycle back-up path is used to by-pass any faulty router in the network. In [168], back-up paths are used to horizontally or vertically by-pass faulty routers. In [169], extra modules, called control path back-up mechanism, are added at the router inputs and outputs. These extra modules are

connected between them with back-up paths and are used to by-pass the router when it is declared as faulty. In [170], on-demand by-pass links are used to manage faulty crossbars. In [171], a back-up network layer is used to by-pass faulty routers of the main layer maintaining the network bandwidth. Although default-backup paths have low area and power consumption, the latency drastically increases under multiple faults, due to the routing complexity. In addition, congestion, bottlenecks and other network issues can occur up to having several inaccessible IPs.

Degraded Mode: NoCs may also be reconfigured in a degraded mode using only the remaining healthy resources [122]. With this approach, the correct transmission of the data is ensured since the faulty part of the architecture is deactivated [172]. For example, faulty interconnections or routers can be deleted from the NoC or 32-bit buffers can be re-arranged into 16-bit buffers removing the faulty part. However, while the hardware costs of this approach are low due to the absence of spare resources or default back-up paths, the network performances are significantly degraded in presence of multiple permanent faults. In the worst cases, the NoC can be inoperative due to isolated IPs.

2.3.2.3 Temporal Redundancy

Most of the temporal-redundancy methods use the packet re-transmissions to manage the faults [95, 105, 173]. However, as saw in Section 1.3, the packet re-transmissions has negative impacts on the NoC since it increases the congestion leading to reduced NoC performances. To reduce these negative impacts, the re-transmission can be done between two consecutive routers [174], instead of between the source and destination IPs.

Re-send the packets provides transient fault management, however this approach can be limited with intermittent faults and even be useless in presence of permanent faults since the re-transmitted packets always take the same path on the NoC [13]. To remedy this, the approach presented in [175] re-sends the packet after several cycles, providing sufficient time for intermittent faults to be vanished. Concerning the permanent faults, many works are proposed in the literature to manage them using temporal redundancy. In [176, 177], the interconnections are divided into several sections which can be deactivated if they are impacted by permanent faults. In this case, flits are serialized and sent using the fault-free sections of the interconnection. However, this approach impacts the latency, hence decreasing the NoC performances. In [178], the interconnections are divided into two parts, i.e. the most significant part and the least significant part, which are each

controlled by a dedicated flow control. When a half flit is impacted by permanent faults, the associated flow control sends a Non-Aknowledgement (NAck) signal to re-send the faulty half flit. Nevertheless, this latter is duplicated on the other half that has been sent without errors to ensure the correct transmission. Then, the flit is restored at the next-router input using the two fault-free parts of the flit. However, this technique degrades the latency affecting the NoC performances.

Other approaches are proposed in the literature to manage a higher fault number. For example, in [98], the flits are triplicated. While the first flit is not modified, the bits of the second flit are reversed and the two halves of the third flit are reversed. The receiver compares the three triplicated flits to detect and correct the fault impacts. However, this approach drastically degrades the NoC performances since each packet is triplicated, i.e. performances are divided by around three.

The methods using temporal redundancy can also target the logic part of the router. For instance, in [179], the routing computation stage and the virtual-channel allocation stage are computed several times during several cycles allowing detection of soft faults.

2.3.3 Fault Tolerant Topologies

Faults occurring in the NoC interconnections can be managed with the help of fault-tolerant topology [16]. For that, redundant interconnections are added to over-connect the routers between them, hence increasing the network connectivity [180]. Then, increasing the number of router ports, also called the radix degree, allow to avoid IP isolation when permanent faults occur in one or several interconnections. However, as the network complexity increases with the radix degree, more complex routing algorithms are necessary to forward the packets in the NoC increasing the hardware costs. In response, topology generator [181] targeting specific application are proposed in the literature. Many fault-tolerant topologies can be found in the literature, such as the hexagonal topology [131] or the incomplete ternary n-cube [182]. However, as mentioned previously, they often need specific routing algorithms which increase the network complexity, and therefore, the hardware costs. In addition, the NoC performances can be drastically impacted when multiple permanent faults are present in the network due to the necessary detours to reach the destination IP.

2.4 Other Detection Methods

In this section, we present methods of the literature which can be used to detect and diagnose faults in the NoCs [79, 40, 183]. These methods can operate at run time, i.e. online methods, or need to stop the system, i.e. offline methods. While the online methods can detect both transient and permanent faults, offline methods can only detect permanent defects which cannot be removed by restarting the system. First, we present the principle of the well-known Built-In Self-Test (BIST) methods. Then, monitoring-based methods are described. Finally, other methods which can be found in the literature are summarized.

2.4.1 Built-In Self-Test Methods

BIST methods are well-known in the fault-tolerant field to detect and diagnose the permanent faults. These methods can be used in NoC architectures to locate faults in routers [184, 185, 186], interconnections [187, 188, 189] or both of them [190, 142, 191]. They can be divided into two sub-categories which are online [192] and offline [188] methods. Some works of the literature mix offline and online detection techniques where only the tested regions [41] or the idle paths [193] are put offline while the rest of the NoC continues to operate. These approaches implement BIST methods at fine-grain resolution increasing the fault coverage while maintaining the NoC performances.

To operate, BIST methods use test flits which are sent through the circuit under test, i.e. routers or interconnections. For this purpose, two block types are necessary, the Test Pattern Generators (TPGs) and the Output Response Analyzers (ORAs). While the TPGs send test flits, also called test patterns, the ORAs analyze them once they cross the circuit under test. The comparison with a golden reference allows to detect and locate the faults inside the circuit. An example of one BIST implementation is displayed in Figure 2.5 where the TPGs and the ORAs are respectively located at each output and input of the routers. When a router needs to be tested, i.e. middle router in Figure 2.5, the TPGs of the neighboring routers send test patterns towards each other neighboring routers and towards the associated IP of the router under test. The red paths in Figure 2.5 represent the paths taken by the test flits sent by the TPG of the west router to test the middle router. When the test flits reach the ORAs of the destinations, the received flits are analyzed to determine if the router under test is impacted by faults or not.

BIST methods can be used to diagnose a wide range of fault models using adapted test

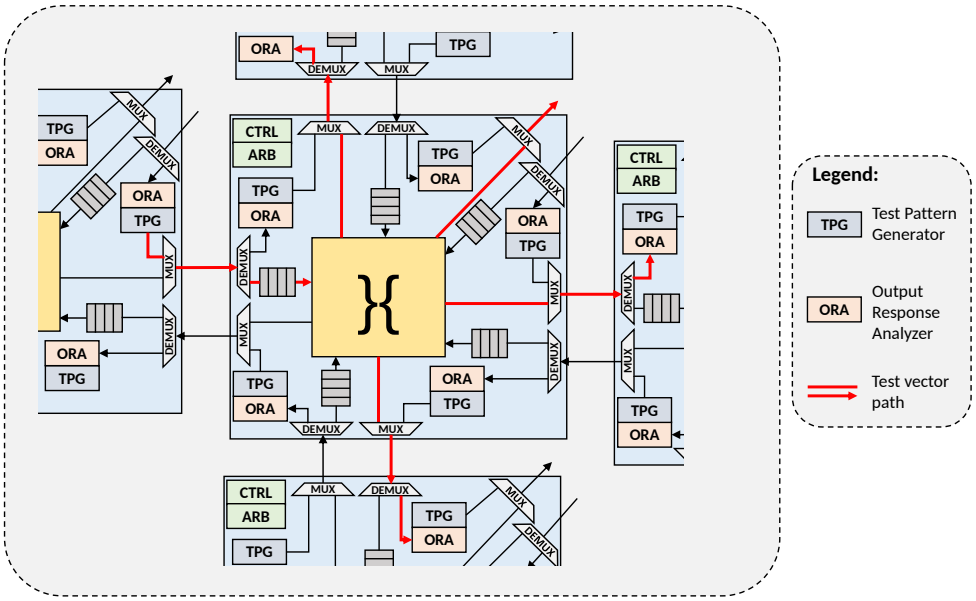


Figure 2.5: Router architecture implementing BIST technique.

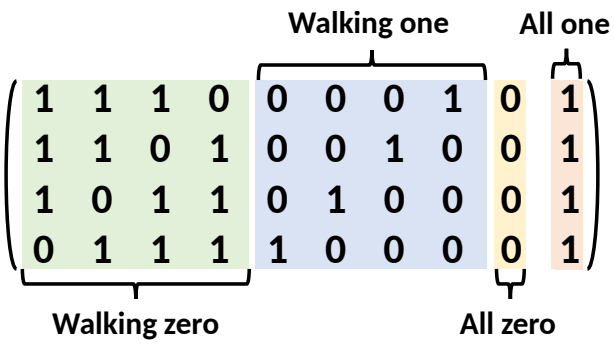


Figure 2.6: Test vectors used in the BIST methods.

patterns. Figure 2.6 presents the four most used test patterns in the BIST techniques [194] which are i) all one, ii) all zero, iii) walking one and iv) walking zero test patterns. While all one and all zero test patterns are used to target the stuck-at and open faults, walking one and walking zero test patterns target short and bridge fault types. These test patterns mainly allow to test the datapaths, however the control logic can also be diagnosed [142] by detecting the miss-routing and dropped test flits when they cross the router under test.

2.4.2 Monitoring Methods

The faults which affect the flit during the packet transmissions can be detected by monitoring the NoC traffic to identify disruptions [195, 196] due to the fault occurrences. For that, NoC assertions [197], extra module assertions [198] or router invariances [199] can be used to detect disruptions generated by the faults. Based on this, the monitoring methods can provide the state of the NoC by evaluating its deterioration [200] but also help to mitigate deterioration induced by external or internal effects such as Negative Bias Temperature Instability (NBTI) effects [201].

2.4.3 Other State-of-the-Art Methods

The literature provides some other methods to detects faults affecting the NoCs. Flow control can be used to detect [202] and locate [203] faults. In [204], a software-based self-test which uses bounded model checking is proposed to detect NoC errors by applying test patterns at-run time. In [205], an online method is proposed to detect miss-routing in the mesh architectures using XY routing algorithm. For that, the source and destination addresses are compared to the current-router address to determine if the packet takes unexpected path. In this case, a fault is present in the routing-computation stage of the upstream router. The fault occurrences can also be predicted by machine learning algorithms [87]. In this case, the fault-tolerant methods used to manage the faults can be adapted at-run time according to the predictions [110]. Of course, detection methods can be combined at different layers [206] to enhance the trade-off between the fault coverage and the NoC performances.

2.5 Mitigation Using Approximate Communications

In recent years, approximate computing [207, 208] appears as an energy-efficient opportunity to reduce the hardware costs relaxing the accuracy. This paradigm can be applied in several application fields [209, 210, 211], such as machine learning, image processing, data mining and others, which require lower precision. Since few years, approximate computing has given birth to approximate communications which can be applied to the NoC architectures offering energy-efficient and high-performance communications [20]. For example, in [21], a NoC architecture, called AxNoC, is presented to reduce the power consumption of the NoC using a dual-voltage power management to transfer accurate flits with high voltage and save energy by sending data at low voltage at the cost of errors.

Instead of aiming power-efficient and high-performance communications, approximate communications can be used to provide fault tolerant communications in NoCs [212]. However, despite the fact that this field offers many perspectives for the fault-tolerant field, it stays few studied in the literature. In [213], the APPROX-NOC framework is proposed for high throughput communication compressing the data transiting on the NoC. For that, data are first approximated to match with a compressible reference data pattern, then, the data pattern is compressed before to be sent through the NoC. In [214, 215], an approach statically changes the assignment of lines in datapath busses, by placing the Most Significant Bits (MSBs) on the borders of the bus to attenuate the electromagnetic influences between neighboring lines. However, the line assignments cannot be modified during execution, and thus, external and internal effects cannot be addressed by this technique. In [216], an online quality management framework for approximate communications in NoC architecture is proposed. This framework allows to reduce the time needed to compute the approximation level. This approach ensures low latency and low hardware overheads taking into account the considered approximate application. In [217], a multi-plane NoC is proposed to increase performances using a second bufferless network. While approximate packets can be dropped during the message forwarding to avoid congestion, the main network ensures a 100% accuracy in packet transmissions. However, this method necessitates two parallel networks to operate which drastically increases the hardware costs. Another similar approach is proposed in [218] where stochastic communications are used. However, this approach drastically impacts the NoC performances because of the large flit number required to encode the message with the generated random pattern inducing high congestion in the NoC.

2.6 Examples of Fault-Tolerant NoC Architectures

In the previous sections we surveyed the different method types which can be used to enhance the NoC architectures against the transient and the permanent faults. However, these techniques can be combined to increase the fault tolerance of the NoCs. In this section, we present several NoC architectures proposed in the recent years where several of the above-mentioned techniques are combined.

The VICIS router architecture is presented in [24] to handle permanent faults. For that, diagnosis, i.e. BIST, and reconfiguration techniques are used to contain the failures within the routers. In this way, the router architecture is enhanced to manage the occurrence of permanent failures. ECCs are used to protect the datapath elements and a bus is added to the crossbar to by-pass the permanent faults. In addition, the router buffers can be reconfigured in a degraded mode to use only the fault-free space and the input ports can be re-organized by swapping them to manage the faulty links. Finally, a table-based fault-tolerant routing algorithm is used avoiding link deactivation.

In [219], the proposed architecture offers on-demand fault tolerance by proposing multiple traffic classes which depend of the flit priority level. While the transient faults are managed with the help of a Hamming code, the permanent faults are tackled using spare wires in the interconnections. According to the priority level of the packet, the on-demand fault-tolerance manager adapts the path taken by the packet to improve power-efficient fault tolerance.

The uDIREC framework is proposed in [220] for permanent-fault diagnosis and reconfiguration of the NoC architectures. This framework uses a low-cost diagnosis method which can locate the faults at the unidirectional link granularity. The fault diagnostics are used to update a proposed routing algorithm, called MOUNT, to maximize the utilization of the remaining resources. In this way, the NoC can be used in degraded mode.

In [221], a fault-resilient and self-healing NoC architecture, called FASHION, is presented. In this architecture, three units, i.e. self-monitoring, self-re-configuring and BIST units, are added to manage the components failures in the NoC. While the self-monitoring unit is used to capture events in the NoC and initiate periodic runs of the BIST unit, the self-re-configuring unit is used to disable the faulty components and keep the NoC in a degraded mode when permanent faults are detected in the NoC.

In [222], a reliable NoC router is proposed to tackle permanent faults. For that, the five pipeline stages are enhanced with fault tolerant methods. The virtual channel buffers

are enhanced with an ECC which can correct one faulty bit per flit. In the case where a permanent fault is detected, the impacted virtual channel is disabled. The routing computation is doubled using one redundant cycle. In the case where permanent faults affect the virtual allocation stage, a default back-up path is used to send the packet in a default direction. A run-time arbiter selection strategy is implemented to manage the faults occurring in the switch allocation stage. Finally, a double by-pass is used to tackle the permanent faults which occur in the crossbar.

2.7 Conclusion

In this chapter, we surveyed the methods proposed to detect, diagnose, correct, or mitigate transient and permanent faults. Through this chapter, we can note that, contrary to the transient faults, the correction or mitigation of the permanent faults suffers from a lak of studies despite the fact that they are more susceptible to occur with power and technology scaling as mentioned in Section 1.2.

The state-of-the-art methods to manage permanent faults in NoC architecture are often based on i) mitigation through routing algorithms, ii) hardware reconfiguration through spare resources or default backup path, iii) correction through circuit replication and iv) information redundancy. However, we saw in this chapter that these methods often induce high hardware overheads in terms of area and power consumption and that they can drastically decrease the NoC performances. Moreover, these methods are limited in presence of multiple permanent faults reducing the lifetime of the NoC.

To tackle these problems, we propose Bit-Shuffling (BiSu) technique [P2, P1], a bit-shuffling hardware technique with low area and performance overheads to efficiently deal with multiple permanent faults in the NoC architectures. This technique is based on the emerging approximate-communication field which appears since few years, as mentioned in Section 2.5.

Chapter 3

Bit-Shuffling Technique for Error Mitigation in Network-on-Chip Architectures

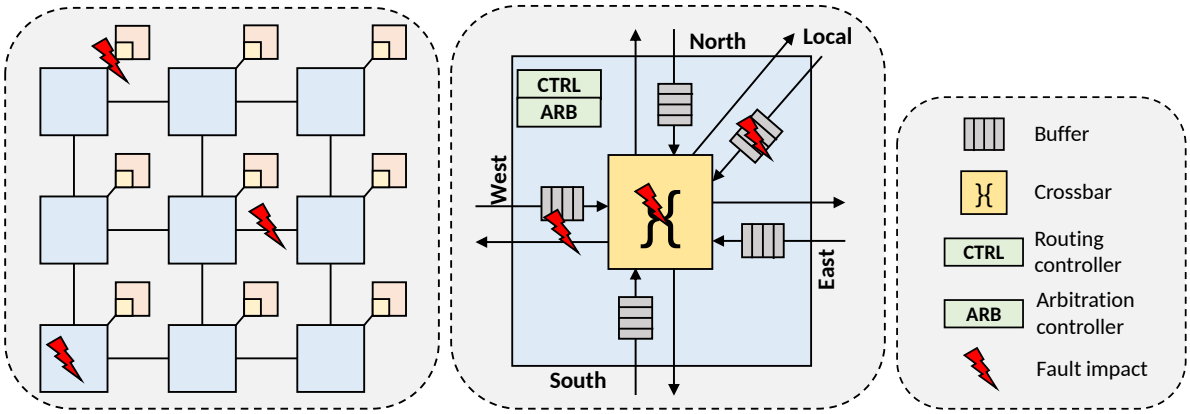
In this chapter, we present the first contribution of this manuscript, named the Bit-Shuffling (BiSu) [P2, P1] technique which targets the mitigation of permanent faults in Network-on-Chip (NoC) architectures providing low area, power and performance overheads. For that, we first present in Section 3.1 the principle of the BiSu method applied in the NoC architectures. Then, an evaluation of the BiSu technique efficiency is shown in Section 3.2. Finally, the hardware costs induced by the proposed method are studied in Section 3.3 before concluding in Section 3.4.

3.1 BiSu-Principle Overview

This section presents the principle of the BiSu technique which is used to achieve mitigation of multiple permanent faults by reducing their impacts, instead of fully correcting them. First, the target domain and the assumptions are presented, then, the principle of the BiSu is described, i.e., the re-organization of bits in flits to place the Most Significant Bits (MSBs) on a non-faulty path, before detailing its implementation. Afterwards, an extension of the BiSu technique for flits containing sensible data, such as header or instructions, is explored. Finally, the BiSu technique is extended to manage data sizes that are different from the subflit size.

3.1.1 Target Domain and Assumptions

The BiSu approach targets the mitigation of multiple permanent faults which can occur in the NoC datapath, i.e. interconnections, buffers, crossbars, and wires within the



(a) NoC level (b) Router level (c) Legend

Figure 3.1: Fault impacts in the NoC architecture.

routers, as illustrated by the red flashes in Figure 3.1. As already mentioned in Chapter 1, devices and components become more susceptible to multiple permanent faults [48] due to nano-scale technologies and power scaling. On the other hand, as the datapath elements are the biggest components of NoC, they have a higher probability of accumulating permanent faults due to radiations, manufacturing and aging defects, as already mentioned in Section 1.3. The proposed method can be used to mitigate permanent faults defined by stuck-at, short or bridge models [25] which impact the NoC datapath. In fact, the BiSu technique considers only the state of the datapath at the wire scale, i.e. faulty or not, regardless of the used fault model since this latter will be only considered for the detection and the diagnosis of the faults. For this work, we assume that the fault positions are provided by detection methods such as Built-In Self-Test (BIST) techniques [164, 187]. As explained in Section 2.4, these techniques diagnose the faults in the NoC datapath at the wire granularity allowing to know which wire is faulty or not.

As the objective of the proposed approach is to reduce the impact of multiple permanent faults, instead of correcting them, the targeted domains concern error resilient applications used in the approximate computing fields, i.e. applications which can tolerate errors until a certain level, such as image processing, data mining, machine learning and others [21]. In this work, the error resilience of the applications is used to tolerate approximate communications due to the faults which affect data transiting on the NoC. The proposed BiSu technique can be implemented in tandem with other techniques, such as application remapping [22] and scheduling [23], which operate at different scales to enhance the NoC fault tolerance increasing the lifetime of the systems. For instance, routers

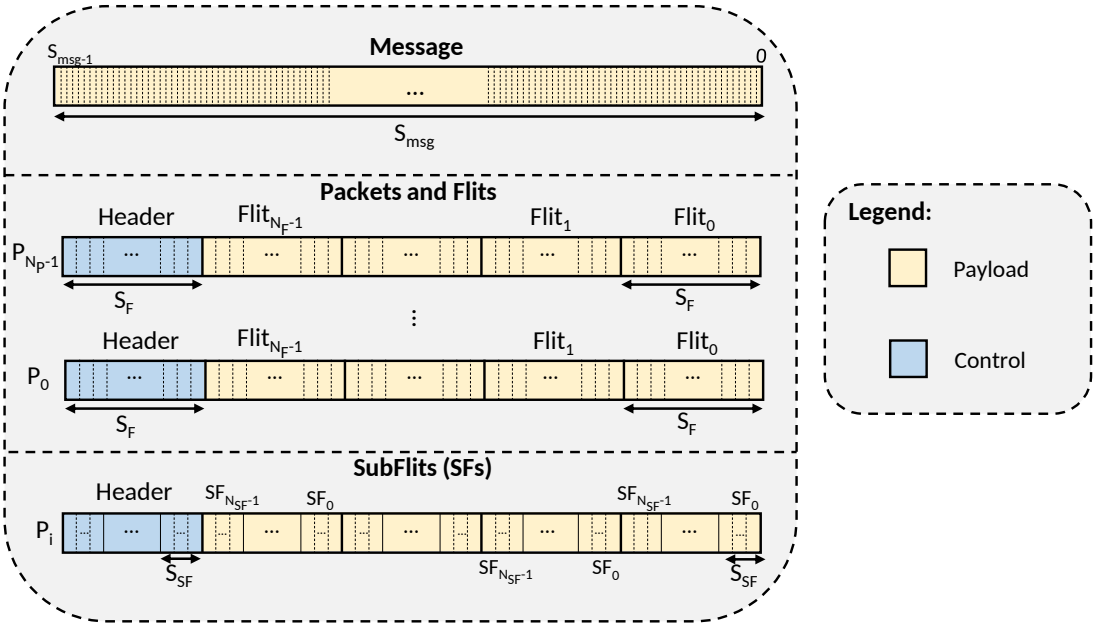


Figure 3.2: Message formatting: Packets, flits and subflits.

where the bit-shuffling method is not efficient anymore can be excluded and by-passed or circumvented thanks to adaptive routing algorithms. Finally, the proposed method can be implemented with any transmission protocol and NoC architecture.

3.1.2 Basic Concepts

As mentioned previously, the BiSu technique focuses on reducing the impact of permanent faults which occur in NoC datapath, instead of fully correcting them. It ensures the protection of MSBs, by transferring the impact of the permanent faults on the Least Significant Bits (LSBs), keeping the MSBs correct. To implement this approach, we consider the classic message organization described in Section 1.1.1.3 where we add a supplementary sub-division, as depicted in the Figure 3.2. We further decompose each flit into N_{SF} Subflits (SFs) of S_{SF} bit size to apply the proposed bit-shuffling technique. Table 3.1 summarizes the notation used for the classic NoC routing messages and for the SF decomposition.

The BiSu technique applies shuffling and de-shuffling functions that switch, at runtime, two or more SFs within the same flit, in order to transfer the impact of faults on LSBs. Figure 3.3 illustrates, through an example, the principle of our approach. Here we consider flits crossing a faulty router from north to south, as shown by the purple arrow of

Symbol	Definition	Symbol	Definition
S_{msg}	Message size	$N_P = \frac{S_{msg}}{S_{pck}}$	Number of packet
S_{pck}	Payload size	$N_F = \frac{S_{pck}}{S_F}$	Number of flit
S_F	Flit size	$N_{SF} = \frac{S_F}{S_{SF}}$	Number of subflit
S_{SF}	Subflit size		

Table 3.1: Notation summary.

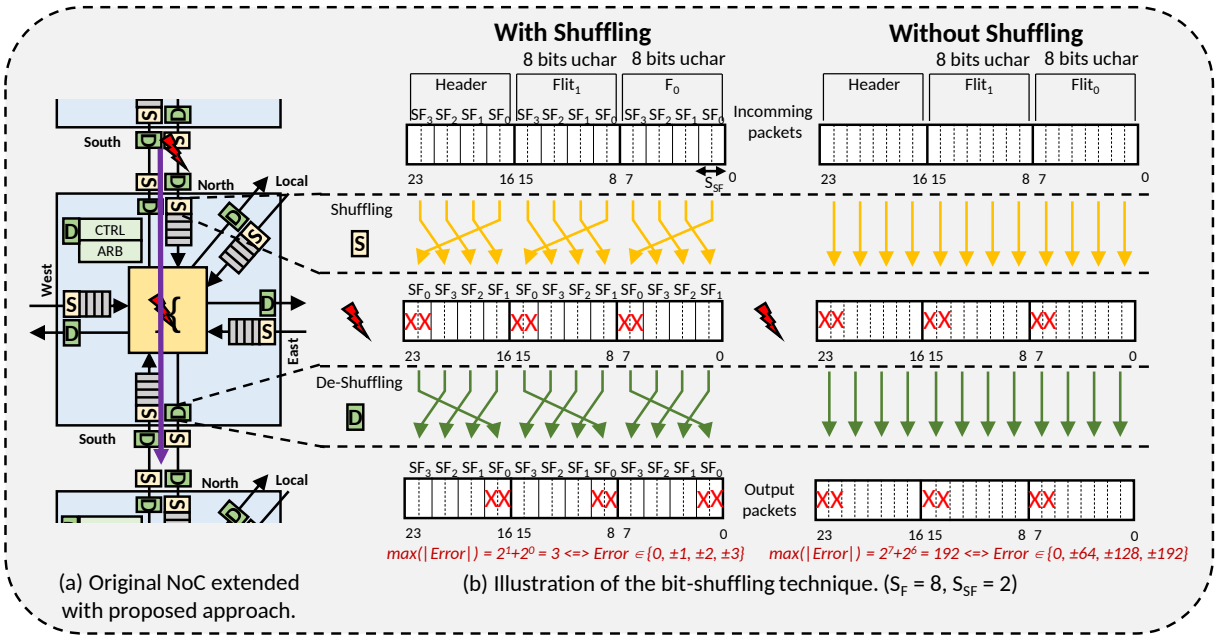


Figure 3.3: Classic NoC extended with the BiSu technique.

Figure 3.3a. For simplification reasons, the illustration example considers a single buffer channel, however BiSu technique is also applicable with virtual channels. The example focuses on payload flits, while header flits are discussed in Section 3.1.4. As depicted in Figure 3.3b, we consider a flit size equal to $S_F = 8$ bits and a SF size equal to $S_{SF} = 2$ bits. Therefore, the number of SFs (N_{SF}) in a flit is equal to $N_{SF} = 4$ (SF_0 to SF_3). When no fault occurs, shuffling and de-shuffling functions are disabled and flits cross the NoC router without any modification.

We consider now that two permanent faults occur in the input buffer, affecting the MSBs, i.e., bits 7 and 6, of all incoming flits. The right part of Figure 3.3b illustrates the crossing of packets without any modification in the flit. The bits 6 and 7 of the two payload flits are affected, leading to errors within the range $\{0, \pm 64, \pm 128, \pm 192\}$, depending on

the initial value of the affected bits. For example, if we consider that the bits 6 and 7 are initially equal to 1, then applying two stuck-at-0 fault gives an error of -192 compared to the initial flit value. The left part of Figure 3.3b illustrates the proposed bit-shuffling method. The bit-shuffling technique is enabled in the input ports of the router. Then, before crossing the faulty path, the SFs are re-organized by swapping LSBs and MSBs of data bits to allocate MSBs on non-faulty hardware path, i.e. SF_0 and SF_3 are swapped inside each flit. Hence, the impact of the faults is reduced to the range $\{0, \pm1, \pm2, \pm3\}$, depending on the values of the LSBs. Finally, the SFs are brought to their initial position before the flit leaves the router and it is sent to the output port.

In the above-mentioned example, we consider a subflit size equal to 2 bits, however, this parameter can be configured, affecting the efficiency and the hardware costs of the method. Indeed, when the subflit size is decreased, it permits to improve the efficiency, nevertheless it leads to increase the hardware costs.

3.1.3 Method Implementation

In this part, we present the details concerning the implementation of the BiSu approach. For that, we first present the NoC architecture enhanced with the proposed method, i.e. with the shuffler and de-shuffler blocks. Then, the computation of the registers which manage the SF re-organizations in the shuffler and de-shuffler blocks is detailed.

3.1.3.1 Hardware Architecture

To implement the proposed BiSu technique, the NoC routers are extended with extra hardware blocks, i.e., shuffler and de-shuffler blocks. While the shuffler block re-organizes the SFs with the objective of minimizing the fault impact, the de-shuffler block brings back the initial order of the SFs. To deal with the targeted faults, BiSu technique is applied i) between two routers, to mitigate errors on the interconnection bus, and ii) between the input and output ports, to mitigate errors inside the router. To achieve that, the aforementioned paths integrate shuffler and de-shuffler hardware blocks, as depicted in Figure 3.3a. As the flits are shuffled when they cross the router, the routing controller (CTRL) needs to re-organize the header of the current packet in order to read the routing information and propagate the packet toward the expected neighboring router or the associated Intellectual Property (IP). Then, one extra de-shuffler block is added in the routing controller.

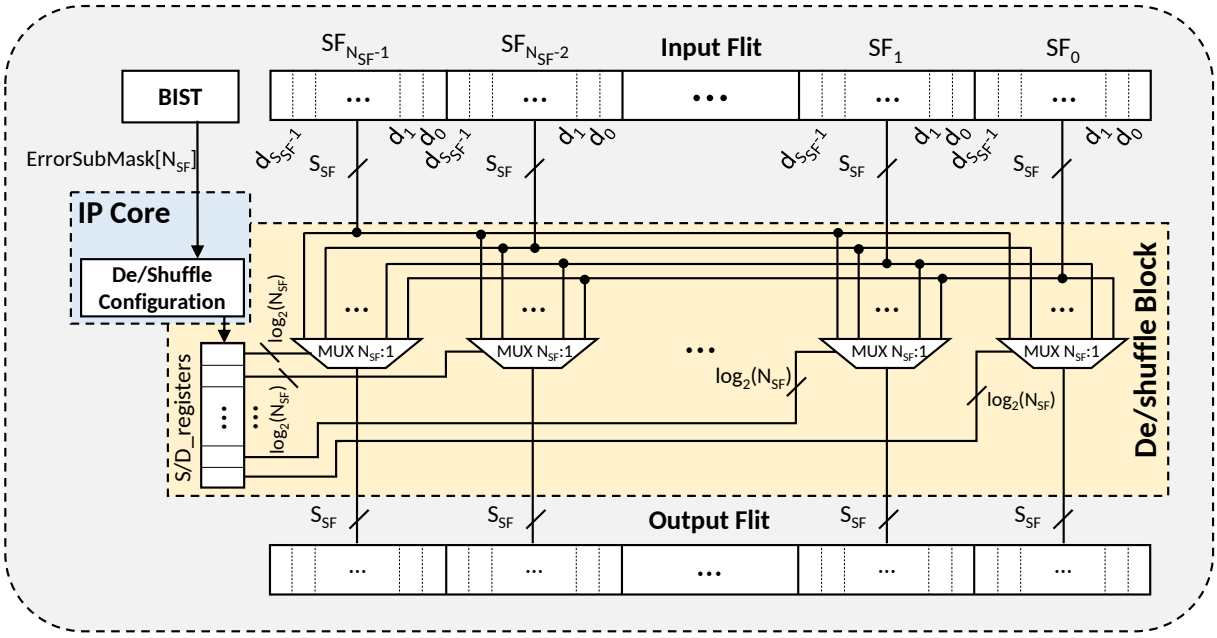


Figure 3.4: Shuffler and de-shuffler block architecture.

The shuffler and de-shuffler blocks have similar hardware architecture. As presented in Figure 3.4, it is composed of N_{SF} simple multiplexers of N_{SF} S_{SF}-bit inputs to one S_{SF}-bit output and N_{SF} registers which contain the configuration of the multiplexers. The only difference between shuffler and de-shuffler blocks is the value of the registers which are respectively named S_{regs} and D_{regs} for the S and D blocks. Following the proposed approach, spatial redundancy is not required, thus the area overhead is limited. Furthermore, faulty paths are not excluded, maintaining the NoC performances.

3.1.3.2 Registers Computation

Algorithm 1 describes the computation of the register values of the shuffler and de-shuffler blocks. The computation is based on Bubble sort [223] and provides the bit-shuffling configurations of the multiplexers that minimize the fault impacts. The algorithm takes as input the mask of the fault positions (E_{Mask}), provided by the BIST method. This mask has the same size than the data-bus of the NoC. Each bit of the mask gives the state of corresponding wire in the datapath. While a '0' means that the wire is fault-free, an '1' means that the wire is faulty. The algorithm can be executed in two different ways. First, the dedicated IP cores of the routers can be used to execute the algorithm. Otherwise, it can be executed with dedicated circuits, called register updater

Algorithm 1 Shuffler and de-shuffler registers updating.

Input: $E_{Mask}[S_F]$
Output: $S_{regs}[N_{SF}]$, $D_{regs}[N_{SF}]$

1: // Variable Initializations
2: **for** $(i = 0$ to $N_{SF} - 1)$ **do**
3: $\quad SubMask[i] \leftarrow \sum_{j=0}^{N_{SF}-1}(2^j \times E_{Mask}[i \times N_{SF} + j])$
4: **end for**
5: $S_{regs} \leftarrow i \qquad \forall i \in [0, N_{SF}[$
6: $D_{regs} \leftarrow i \qquad \forall i \in [0, N_{SF}[$
7: $inversion \leftarrow TRUE$
8: // Deshuffling Register Computation
9: **for** $((i = 0$ to $N_{SF} - 2)$ && $(inversion))$ **do**
10: $\quad inversion \leftarrow FALSE$
11: $\quad$ **for** $(j = 0$ to $N_{SF} - 2 - i)$ **do**
12: $\quad\quad$ **if** $(SubMask[j] < SubMask[j+1])$ **then**
13: $\quad\quad\quad swap(SubMask[j], SubMask[j+1])$
14: $\quad\quad\quad swap(D_{regs}[j], D_{regs}[j+1])$
15: $\quad\quad\quad inversion \leftarrow TRUE$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end for**
19: // Shuffling Register Computation
20: **for** $(i = 0$ to $N_{SF} - 1)$ **do**
21: $\quad S_{regs}[D_{regs}[i]] \leftarrow i$
22: **end for**
23: **return** $S_{regs}[N_{SF}]$, $D_{regs}[N_{SF}]$

block, which are added at each couple of shuffler and de-shuffler blocks. While the first solution increases the pressure on the dedicated IP cores with no hardware overheads, the second solution induces hardware overheads in terms of area and power consumption. Of course, one dedicated circuit can be used to compute the registers of several couples of shuffler and de-shuffler blocks limiting the induced hardware overheads. However, this approach increases drastically the complexity and requires controller blocks to serve the computed registers at the right pair of shuffler and de-shuffler blocks.

For clarity, lines 2−4 of Algorithm 1 organize the bits of the error-position mask in N_{SF} groups of S_{SF} bits, which are named $SubMask$. Each $SubMask$ informs about the faults in the associated subflit. For example, in the shuffling illustration presented in Figure 3.3, we consider flits of 8 bits divided into 4 subflits of 2 bits. As the bits 6 and 7 of the flits are affected by permanent faults, we have the error mask $E_{mask} = [1100 \quad 0000]$ that gives

$SubMask[0] = 00$ and $SubMask[1] = 00$, $SubMask[2] = 00$ and $SubMask[3] = 11$. As second example, considering a 16-bit datapath with 4-bit subflits, where bits 6, 7 and 13 are faulty, the mask of the error positions is $E_{mask} = [0010 \quad 0000 \quad 1100 \quad 0000]$ that gives $SubMask[0] = 0000$, $SubMask[1] = 1100$, $SubMask[2] = 0000$, and $SubMask[3] = 0010$.

In lines $5 - 7$, the variables and registers are initialized. Each register is initialized to ensure the flit transfers without any shuffling ($S_{regs} = D_{regs} = [3, 2, 1, 0]$).

In lines $9-18$, the bubble sort algorithm computes the values of the de-shuffler register according to the input $SubMask[N_{SF}]$. For that, the algorithm orders the $SubMask$ values in a decreasing order and the same ordering is applied over array D_{regs}. For example, if $SubMask[1]$ is inferior to $SubMask[2]$, the two values are swapped, and therefore, the values $D_{regs}[1]$ and $D_{regs}[2]$ are also swapped. When the computation is over, the de-shuffler register contains the multiplexer configuration $D_{regs}[N_{SF}]$ for the architecture presented in Figure 3.4. In this way, the $i - th$ value of the register indicates which input subflit is sent to the $i - th$ output subflit. For example, if $D_{regs}[2] = 1$, the subflit SF_1 will be forwarded towards the output subflit SF_2. Finally, as the hardware architectures of the shuffler and de-shuffler blocks are similar, the shuffler registers are computed from the de-shuffler registers, as shown in lines 18 to 20.

Based on this, for the example presented in Figure 3.3, we obtain the registers $S_{regs} = [0, 3, 2, 1]$ and $D_{regs} = [2, 1, 0, 3]$ which corresponds to the paths taken by the subflits in this figure during shuffling and de-shuffling operations. And more, the shuffling registers of the above-mentioned 16-bit example are $S_{regs} = [1, 3, 0, 2]$ and $D_{regs} = [2, 0, 3, 1]$.

3.1.4 Header and Critical-Data Protection

As header flits contain control information, in particular for packet routing, and they cannot tolerate errors. For example, for an architecture with 256 cores, i.e. 16×16 mesh where packets are composed of 32 flits, the header contains 8 bits for source core, 8 bits for destination core, and 4 bits for packet size. As other information such as packet identification can be added in the header, we can consider that around half of the header flits is unused, as represented in Figure 3.5a. Thus, to handle faults affecting headers, the BiSu technique is extended as follows: For NoCs using large data buses (i.e., 64 bits), header flits usually include several unused bits which are placed on the LSBs, as depicted in Figure 3.5a. Based on this, when faults occur on MSBs, the method transfers the faults on the unused SFs, removing any errors.

However, header flits with small data buses (i.e., 32 bits) do not usually include enough

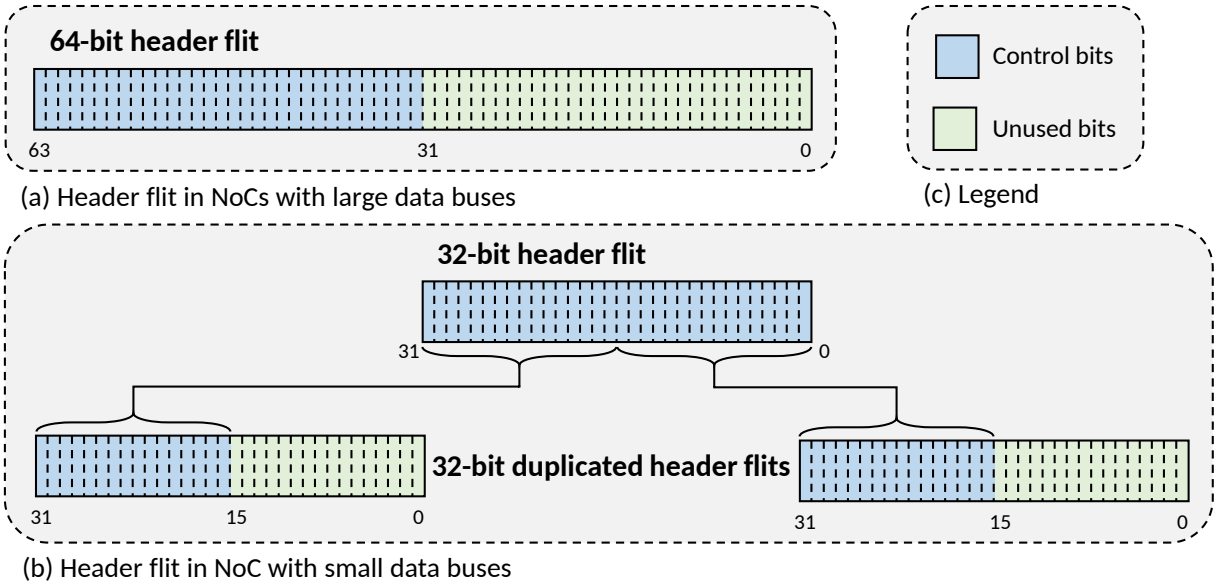


Figure 3.5: Protection of headers and sensible flits with the BiSu approach.

unused bits. To address this, we distribute the information of the header flit into 2 flits, as shown in Figure 3.5b, in order to artificially insert unused bits. Hence, half of the two new header flits can be used to tolerate errors, with a small impact on the NoC latency, i.e., adding a single flit in a packet. The same approach can be used for sensible data, for example for instructions, with a reduced overhead since it is applied in packetization and de-packetization blocks which are localized in Network Interfaces (NIs).

Notice that, today's NoCs are typically based on large buses. Hence, the distribution on two header flits is a solution that requires to be applied only when the shuffling technique cannot guarantee protection of the header bits. As flit size and subflit size are not changed in both of these propositions, Algorithm 1 is applied without any modifications. Moreover, as routing information can be inserted on the first half header, the router architecture does not need modifications.

A second approach can be used to protect header flits which transit in NoCs. As illustrated in Figure 3.6, this approach consists in distributing the information contained in header flit on the MSBs of each flit which composes the packet. In this way, control bits are ensured to be protected by being placed on the MSBs of flits. However, this approach suffers from several weaknesses. First, the control bits which are used to forward the packet through the NoC, such as the destination address, need to be placed in the first flit to avoid any architecture modifications. Then, as the flit MSBs are used to protect the header control bits, the BiSu efficiency is relaxed reducing the accuracy on the transmitted data in

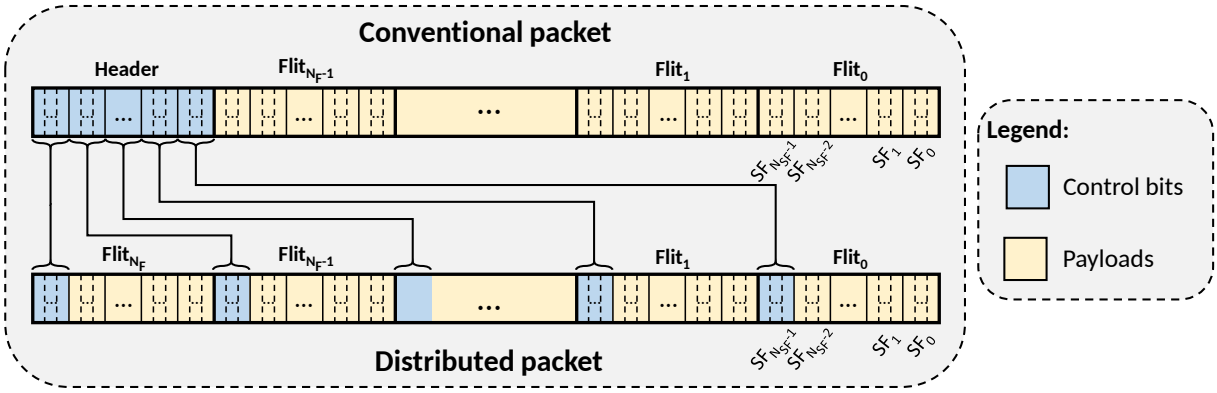


Figure 3.6: Second approach for header protection with the BiSu technique.

presence of faults. Finally, this approach can be only used to protect sensible information of headers and not to ensure accurate transmissions of the payloads containing critical data such as instructions.

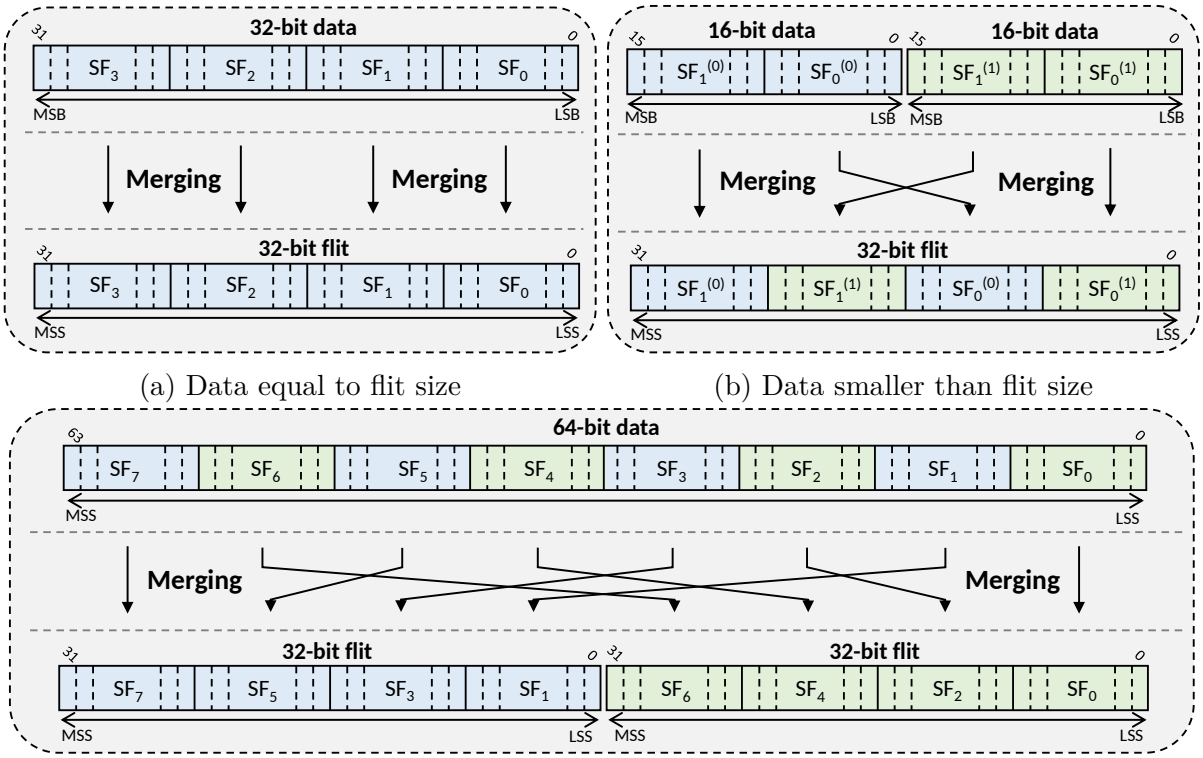
Based on this, we will consider only the first approach in the rest of this manuscript since it ensures both of the headers and sensible payloads protection with small impact on the NoC performances.

3.1.5 Matching Data and Flit Sizes

To efficiently protect the communications with the BiSu technique, packet organization must be considered. The implementation of the BiSu technique must take into account the data size (S_{data}) and the flit size (S_F) to organize flits inside the NIs. Indeed, to propose efficient method, the MSBs and the LSBs of the flit must contain respectively the MSBs and the LSBs of data to avoid the corruption of the significant data during shuffling operations. For sake of clarity, we define: i) Most Significant Subflit (MSS) as the SF including the MSBs of the flit, and ii) Least Significant Subflit (LSS) as the SF including the LSBs of the flit. Up to now we have considered only one data size type per flit and databus width equal to a power of two. However, when considering different data sizes, three cases can occur, as illustrated in Figure 3.7:

a) $S_{data} = S_F$, this is the straightforward case. The data are placed inside the flits without any re-organization, as show Figure 3.7a. The LSBs and the MSBs of the data are respectively placed at the LSSs and at the MSS.

b) $S_{data} < S_F$, more than one data is sent in one flit. Hence, the data are interleaved within the flit, as shown in Figure 3.7b. The MSBs and the LSBs of the data are



(a) Data equal to flit size (b) Data smaller than flit size

(c) Data larger than flit size

Figure 3.7: Organization of 32-bit flits for different data sizes considering 8-bit subflits.

respectively grouped inside the MSSs and inside the LSSs.

c) $S_{data} > S_F$, a data is sent on several flits. The LSBs and MSBs of the data are equally distributed on. the flits, as illustrated in Figure 3.7c. In this way, all produced flits contain a part of the MSBs and LSBs of the data.

With these re-organizations, the MSSs always hold the important data, compared to the LSSs, making efficient the bit-shuffling method, even when the datapath is impacted by multiple permanent faults. Moreover, packet organization is always included in classic NoC with the help of packetization and de-packetization blocks through the NI. Therefore, we only need to insert two additional blocks, called merger and de-merger blocks, that re-organize the data inside flits. As a result, the proposed method does not require high extra hardware and has small impact on the performances since the adding blocks include only combinatorial logic in respect of the critical path. However, the BiSu efficiency is valid only when the subflit size is lower than the data size.

3.2 BiSu-Efficiency Evaluation

In this section, we present a multi-level efficiency evaluation of the BiSu technique through several experiments made on payloads and headers. As the BiSu efficiency mainly depends of the subflit size (cf. Section 3.1.2), we study the impact of the subflit size on the BiSu robustness. We present the results as generic as possible by performing a wide exploration where each possible fault position is considered. Moreover, to avoid any masking of a stuck-at fault due to specific data values, we always perform a bit-flip on the data to consider the worst case. The obtained results are compared with an extended Hamming code. First, a flit-level evaluation of the payload mitigation and the header protection with the BiSu approach is proposed. Then, a NoC-level study is presented. After that, the obtained results with a Convolutional Neural Network (CNN) application where the input are protected with the BiSu technique are detailed. Finally, we explore case studies through image processing and data mining applications, and we show that our method reduces the fault impact on the data transiting on the NoC.

3.2.1 Flit-Level Evaluations: Payload Error Mitigation

In this part, we propose to study the efficiency of the BiSu technique and the impact of the subflit size at the flit-level considering data payloads. For that, we first use a mathematical model to calculate the Maximum Absolute Error (MAE) and the Mean Square Error (MSE) induced by one Multiple Hard Error (MHE). Then, the obtained theoretical results are compared with experimental results to validate the mathematical model and to evaluate the BiSu efficiency compare to the extended Hamming code. Finally, the proposed method is evaluated under multiple Single Hard Errors (SHEs). Experimental results are generated using C++ simulations on a Fedora 28 Linux distribution executed on an 8-cores Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz.

3.2.1.1 Mathematical Model

Let us consider a fault defined by its size (F_{size}) in number of bits and the position of the first faulty bit f_0. For example, if a fault impacts the bits 8, 9 and 10 of a flit, then we have $F_{size} = 3$ and $f_0 = 8$. Note that, with this definition a SHE is a MHE with $F_{size} = 1$. Based on the above definitions, the fault impact on unprotected flit can be computed with Equation 3.1 which gives the MAE since we consider that all the faulty bits are impacted

by the fault to avoid masking effects.

$$MAE_{unprotected} = \sum_{i=f_0}^{f_0+F_{size}-1} (2^i) \tag{3.1}$$

Now, let's compute the fault impact when the BiSu method is applied. First, we need to determine the number of faulty subflits (N_{FSF}), i.e., the number of subflits impacted by the fault in the forwarded flit. Indeed, when two bits or more are impacted by MHE, in this case these faulty bits can cover more than one subflit, as depicted in Figure 3.8. For that, we use Equation 3.2 to compute the position of the first faulty bit f_0 excluding the fault-free LSSs. Then, the size of the fault F_{size} is added and the obtained value is divided by the subflit size. By rounding up the result, we obtain the number of impacted subflits.

$$N_{FSF} = \left\lceil \frac{(f_0 \% S_{SF}) + F_{size}}{S_{SF}} \right\rceil \tag{3.2}$$

Then, the faulty subflits need to be discriminated to know how they will be re-organized with the BiSu technique. We define Least Significant Faulty Subflit (LSFS) (respectively Most Significant Faulty Subflit (MSFS)) as the subflit where the first (respectively last) faulty bit is localized. All other intermediate subflits are defined as Intermediary Significant Faulty Subflits (ISFSs). The ISFSs are always placed by BiSu technique in the LSS, since they are completely faulty. As shown in Figure 3.8, faulty subflits can be re-organized into two different ways, depending on whether the MSFS is completely or partially impacted. When the MSFS is partially impacted by the fault, as shown in Figure 3.8a, the BiSu technique places it after the ISFSs and the LSFS to minimize the fault impact. Otherwise, when the MSFS is completely faulty, as in Figure 3.8b, the BiSu technique places it after the ISFSs and before the LSFS. In this figure, we notice that the re-organization does not keep the same order between the LSSs which are placed on the faulty path. This complexity is due to the fact that we reduce the fault impact as much as possible by sacrificing in priority the LSBs.

In Figure 3.8, we observe that the original burst fault is split into two burst faults, when the LSFS is partially impacted, due to the re-organization performed by BiSu technique. For clarity reasons, we consider, in the rest of this proof, that the burst fault is always divided into two burst faults which are one next to the other when the LSFS is fully impacted. Based in this, we use Equation 3.3 to distinguish whether the MSFS is partially

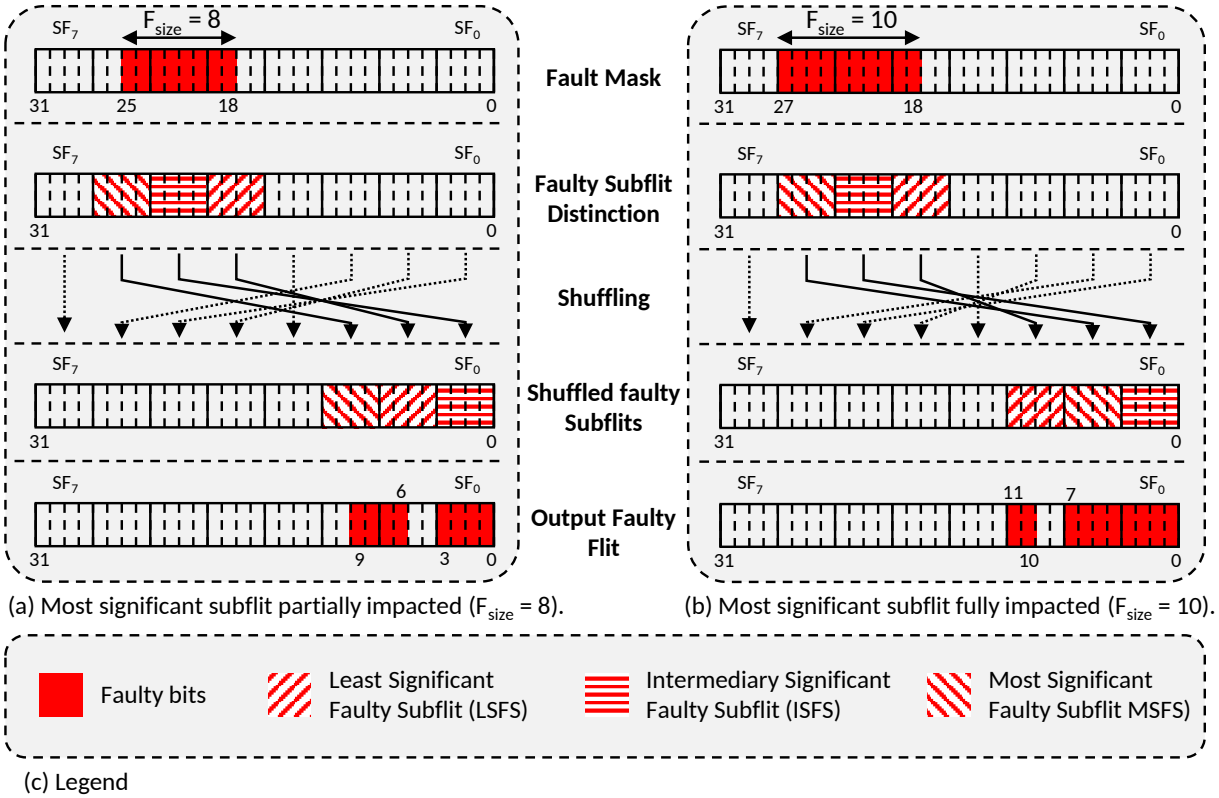


Figure 3.8: Mathematical computation of the fault impact on a 32-bit flit with 4-bit subflits.

or completely impacted by the fault. If α is equal to 0, the MSFS is completely impacted by the fault. Otherwise, it is not.

$$\alpha = (f_0 + F_{size})\%S_{SF} \tag{3.3}$$

According to the α value, we compute the number of completely faulty subflits (N_{FFSF}) without taking into account the LSFS:

— If $\alpha = 0$:

$$\begin{cases} N_{FFSF} = N_{FSF} - 1 & if \quad N_{FSF} - 1 > 0 \\ N_{FFSF} = 0 & else \end{cases}$$

— Else:

$$\begin{cases} N_{FFSF} = N_{FSF} - 2 & if \quad N_{FSF} - 2 > 0 \\ N_{FFSF} = 0 & else \end{cases}$$

The N_{FFSF} value can be used to compute the fault impact of the burst fault. The first half of the burst fault, i.e., the ISFS and the MSFS if it is completely impacted, is

computed using Equation 3.4. The second half of the burst fault, i.e. the LSFS and the MSFS if it is partially impacted, is computed using Equation 3.5.

$$MAE^{(1)}_{BiSu} = \sum_{i=0}^{N_{FFSF} \times S_{SF} - 1} (2^i) \tag{3.4}$$

$$MAE^{(2)}_{BiSu} = \sum_{i=N_{FFSF} \times S_{SF}}^{F_{size}-1} (2^{(f_0 \% S_{SF})+i}) \tag{3.5}$$

The total MAE induced by the original burst fault, is given by Equation 3.6 which is obtained by adding Equations 3.4 and 3.5.

$$MAE_{BiSu_temp} = \sum_{i=0}^{N_{FFSF} \times S_{SF} - 1} (2^i) + \sum_{i=N_{FFSF} \times S_{SF}}^{F_{size}-1} (2^{(f_0 \% S_{SF})+i}) \tag{3.6}$$

However, Equation 3.6 needs to be extended in order to take into account corner cases, i.e., when the value N_{FFSF} is equal to zero. Thus, we obtain Equation 3.7.

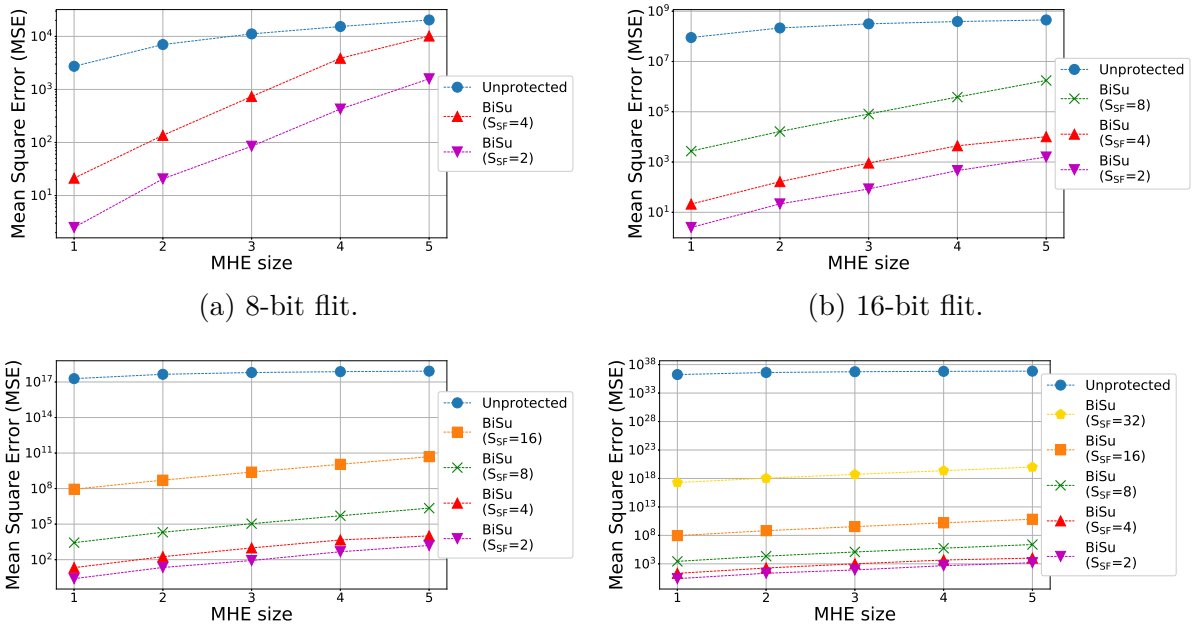
$$\begin{aligned} MAE_{BiSu} = & \sum_{i=-1}^{N_{FFSF} \times S_{SF} - 1} (2^i) - 2^{-1} \\ & + \sum_{i=0}^{F_{size} - N_{FFSF} \times S_{SF} - 1} (2^{(f_0 \% S_{SF}) + N_{FFSF} \times S_{SF} + i}) \end{aligned} \tag{3.7}$$

Finally, the MSE of all possible fault positions can be computed with Equation 3.8 using the MAE of the unprotected and the protected cases respectively given by Equations 3.1 and 3.7.

$$MSE = \frac{1}{S_F - F_{size} + 1} \sum_{f_0=0}^{S_F - F_{size}} (MAE^2) \tag{3.8}$$

3.2.1.2 Mathematical-Model Evaluation

In this part, we confront the mathematical model to experimental results by the evaluation of the BiSu robustness. For that, we consider random payload flits of size 8, 16, 32 and 64 bits which contain unsigned integer data protected by a couple of shuffler and de-shuffler blocks. As we focus on approximate applications, payload flits can tolerate data approximation up to a certain level. Therefore, we use the MSE metric to quantify



(a) 8-bit flit.

(b) 16-bit flit.

(c) 32-bit flit.

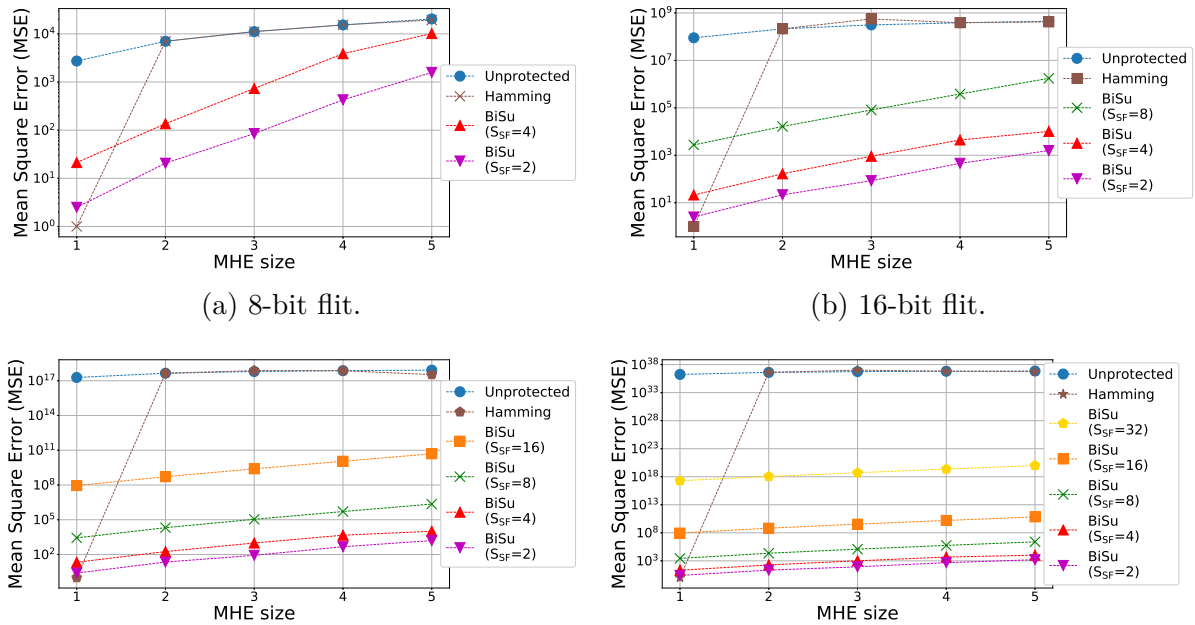
(d) 64-bit flit.

Figure 3.9: Theoretical payload flit accuracy for one MHE.

the impact of one MHE with size of 1 up to 5 faulty bits using subflit size from 2, 4, 8, 16 and 32 according to the flit size, i.e. the subflit size is necessarily lower than the flit size. Theoretical results are computed using Equation 3.8.

Figure 3.9 shows the theoretical MSE for protected and unprotected cases. In this figure, we first observe that the BiSu technique significantly reduces the MHE impact compare to the unprotected case. In addition, we can note that the subflit size impacts the fault mitigation, i.e. reducing the subflit size offers more flexibility to adapt the shuffling to the fault position. In this way, the fault impact is reduced as much as possible according to the chosen subflit size. For example, considering 32-bit flit segmented into 4-bit subflit impacted by one MHE of size 2 (cf. red points in Figure 3.9c), the MSE is reduced from 4.46×10^{17} to 1.79×10^{2} when the BiSu technique is applied to protect the flit.

Figure 3.10 displays the obtained MSE for the protected, unprotected and Hamming cases. We can observe that the experimental results match with the theoretical results of Figure 3.9 in terms of MSE values validating the mathematical model. Moreover, the same observation than the one made previously can be made, i.e. reducing the subflit size offers more flexibility to adapt the shuffling to the fault position. Finally, the comparison with the results obtained with an extended Hamming code demonstrates that for more



(a) 8-bit flit.

(b) 16-bit flit.

(c) 32-bit flit.

(d) 64-bit flit.

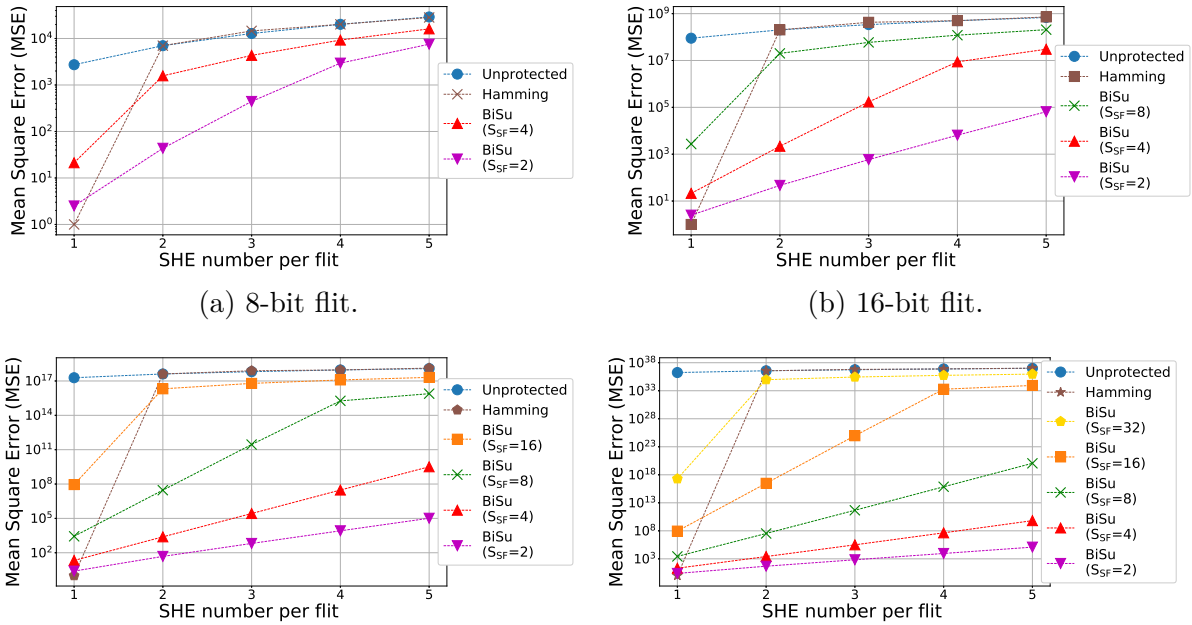
Figure 3.10: Experimental payload flit accuracy for one MHE.

than one faulty bit the BiSu technique maintains the accuracy compare to Hamming code. Indeed, this latter is efficient to correct one fault, but face to several faulty bits it is inefficient and it can even make false correction degrading the results comparing to the unprotected case. For example, considering 16-bit flit segmented into 4-bit subflit impacted by one MHE of size 3 (cf. red points in Figure 3.10b), the MSE is increase from 3.13×10^8 to 5.59×10^8.

3.2.1.3 Mitigation of Multiple SHEs

In this part, we study the impact of multiple SHEs on data flits protected with one couple of shuffler and de-shuffler blocks to evaluate our method robustness. For that, we perform an exhaustive evaluation where the MSE and the Bit-Error Rate (BER) are computed considering each possible SHE positions in the flit. We consider random payload flits of size 8, 16, 32 and 64 bits divided into subflits of size 2, 4, 8, 16 and 32 according to the flit size. Figures 3.11 and 3.12 show respectively the obtained MSE and BER considering 1 to 5 SHEs for unprotected, protected and Hamming cases.

Figure 3.11 shows that the subflit size directly impacts the efficiency of BiSu technique. Reducing the subflit size increases the efficiency of the proposed method, i.e., fault



(a) 8-bit flit.

(b) 16-bit flit.

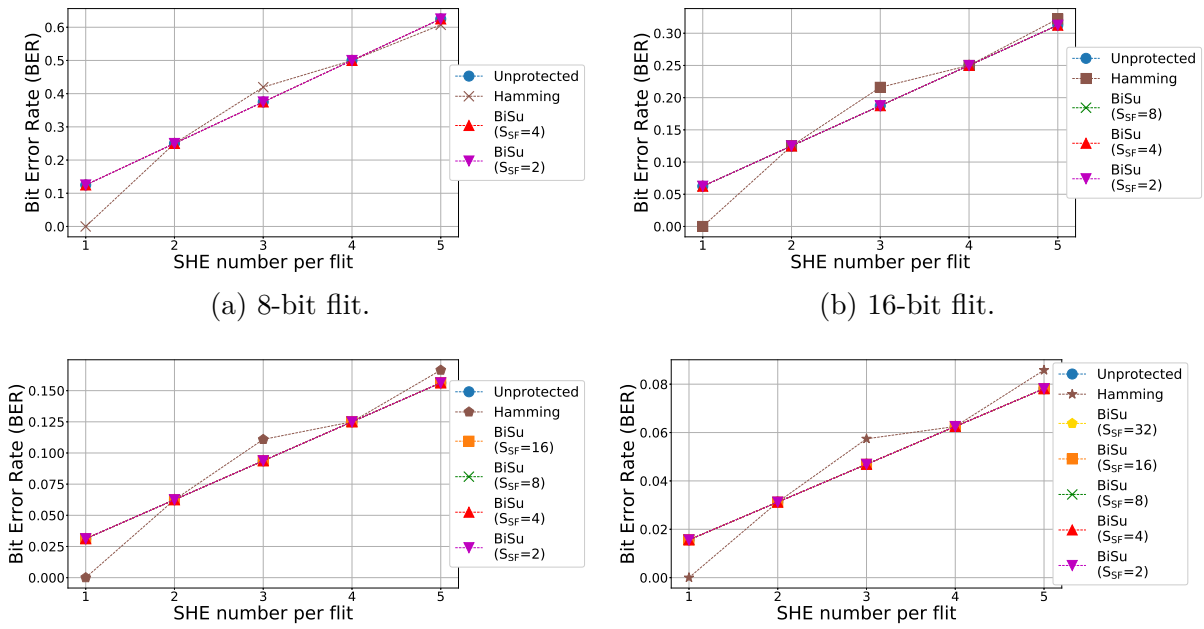
(c) 32-bit flit.

(d) 64-bit flit.

Figure 3.11: Experimental payload flit accuracy for several SHEs using MSE metric.

mitigation is improved. Moreover, we can note that whatever the subflit size used to mitigate the impact of the SHEs, the obtained MSE with the BiSu technique is largely reduced compared to the unprotected case. For example, considering 32-bit flit with 4-bit subflits impacted by 3 SHEs (cf. red points in Figure 3.11c), the MSE is decreased from 6.51×10^{17} to 2.67×10^{5} compared to the unprotected case, and from 7.78×10^{17} to 2.67×10^{5} compared to the Hamming case. Through this example, we can note once again that even if the Hamming code is better to manage one SHE, it becomes less appropriate face to multiple faults. Indeed, it is not able to correct more than one faulty bit. And more, we can see in the previous example that the Hamming code can even provide false correction in presence of more than two permanent faults increasing the impact of the SHEs on the flit. Then, we can conclude that the BiSu technique is especially suitable for the mitigation of multiple permanent faults. Finally, we can observe in Figure 3.11 that the proposed method reaches its limit when the number of faults is equal or superior to the number of subflits. This limit represents the worst case where each subflit is impacted by at least one fault. Of course, this limit can be lowered according to the accuracy level required by the considered application.

In Figure 3.12, we see that the subflit size has no influence on the BER induced by



(a) 8-bit flit. (b) 16-bit flit.

(c) 32-bit flit. (d) 64-bit flit.

Figure 3.12: Experimental payload flit accuracy for several SHEs using BER metric.

the SHEs. Moreover, we note that the used of the BiSu method has no impact on the BER. This is due to the fact that the permanent faults are deferred on the LSBs instead of being corrected. Then, we have the same number of faulty bits than the unprotected case. However, the faults impact only the LSBs instead of the MSBs reducing drastically their impacts. Concerning the Hamming case, we observe that, in presence of one fault, it corrects the single fault leading to a BER equal to 0. However, in presence of more than one fault, it is not able to correct them which leads to a BER equal or superior than the unprotected and protected cases. Moreover, the BER metric highlights the false corrections of the Hamming code which seem to occur only for an odd fault number, i.e. when the parity bit of the extended Hamming detect a fault.

Based on the above observations, we can conclude that a trade-off exists between the BiSu efficiency and the subflit size. Of course, as the proposed method focuses the mitigation of faults in approximate applications, this trade-off must be chosen in order to respect the required application accuracy.

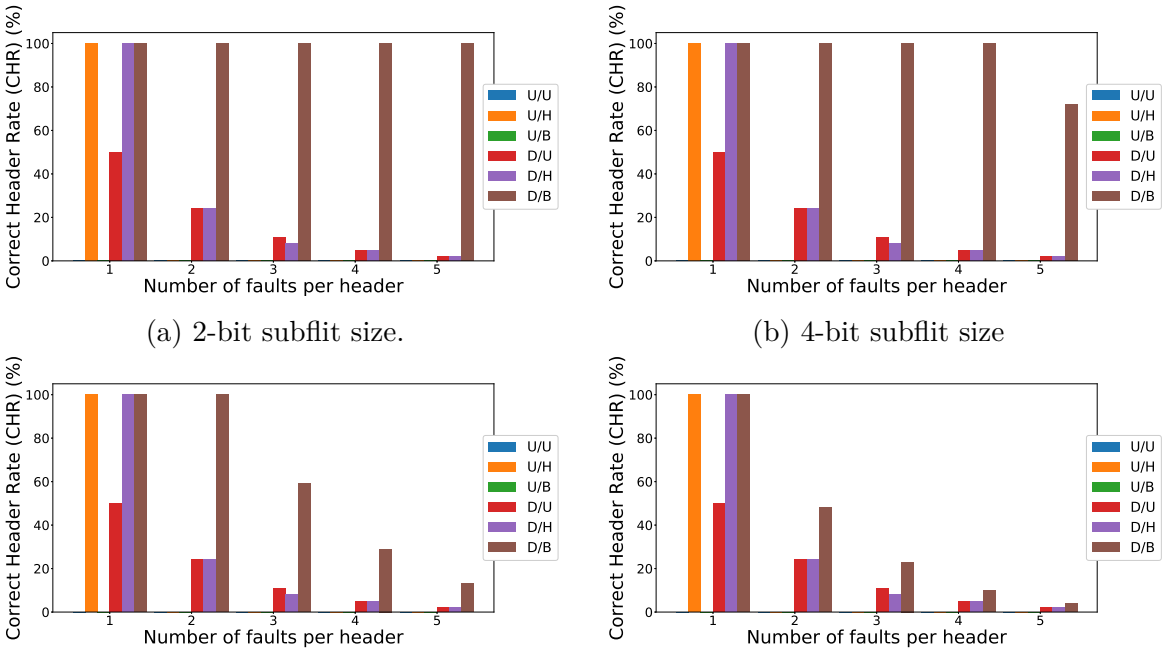
3.2.2 Flit-Level Evaluations: Header Protection

As mentioned in Section 3.1.4, transmissions of headers or flits containing sensible data, i.e. instructions, cannot tolerate any corruption. In this part, we evaluate the approach which consists in distributing header information into two header flits to protect the critical data during transmission. For that, we compute the percentage of correct header transmissions, i.e. percentage of headers which reach their destination without error on their control bits, under 1 up to 5 SHEs considering all possible fault positions for the following methods:

1. Unduplicated and Unprotected (U/U).
2. Unduplicated with Hamming code (U/H).
3. Unduplicated with BiSu technique (U/B).
4. Duplicated and Unprotected (D/U).
5. Duplicated with Hamming code (D/H).
6. Duplicated with BiSu technique (D/B).

Simulations are performed for flit of size 32 and 64 bits with subflit of size 2, 4, 8, 16 and 32 bits according to the flit size. We use C++ simulations on a Fedora 28 Linux distribution executed on an 8-cores Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz. For both of the flit sizes, we consider that the header flits contain 32 control bits which cannot tolerate faults. The BiSu technique is applied using one couple of shuffler and de-shuffler blocks.

Figure 3.13 shows the results for a 32-bit header flit. As we consider that the header flits contain 32 control bits, there is no unused bits. Through the different plots displayed in this figure, we can note that the unduplicated cases cannot ensure correct header transmissions in presence of multiple permanent faults. Only the Hamming code can manage one fault. So, as the number of unused bits is insufficient to tackle the permanent faults with the BiSu technique, header distribution on two flits is required to achieve 100% correct header transmissions under multiple permanent faults. Considering the duplicate cases, we can observe that the unprotected headers stay too sensible in presence of permanent faults since they can affect the MSBs which contain the control bits. Then, the headers cannot be correctly transmitted even under at least one fault. For its part, the Hamming code is able to manage only a single fault per header, i.e. only one permanent fault in the datapath. Then, it is inefficient in presence of more than one fault affecting headers.



(a) 2-bit subflit size.

(b) 4-bit subflit size

(c) 8-bit subflit size.

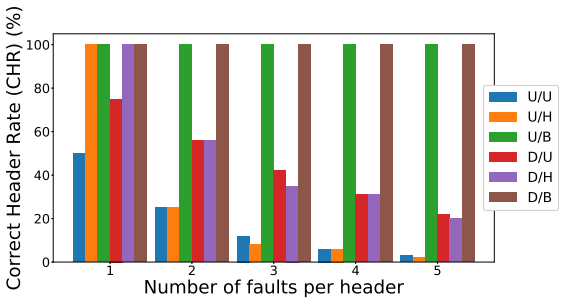
(d) 16-bit subflit size

Figure 3.13: Influence of the method on the header accuracy considering 32-bit flit.

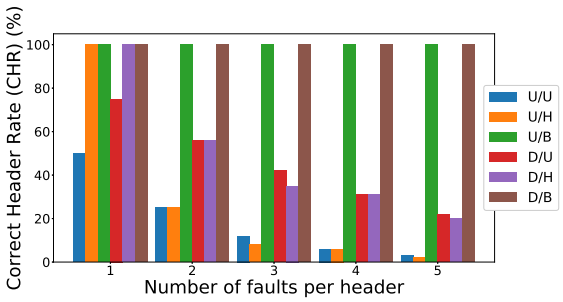
However, we note that the BiSu technique applied with the header distribution on two flits is able to correct header flits transmitting in presence of multiple permanent faults. For example, until 4 permanent faults can be managed with a subflit size equal to 4 bits. Finally, we note that the subflit size influences the fault number which can be managed during transmissions. This points will be detailed below.

Figure 3.14 presents the percentage of correct transmissions for 64-bit header flits containing 32 control bits, i.e. there are 32 unused bits. We note that the unprotected and Hamming cases are not efficient to transmit headers without fault impacts on the control bits. Moreover, the header distribution on two flits is useless on these two cases since the fault can always impact the MSB. However, we observe that applying the BiSu method provides header protection even when the header distribution method is not applied. For example, considering a subflit size equal to 8 bits (cf. Figure 3.14c), the BiSu technique used on unduplicated header flits is able to manage 4 faulty bits with 100% of correct header transmissions. Of course, the subflit size impacts the BiSu efficiency with or without the header distribution on two flits.

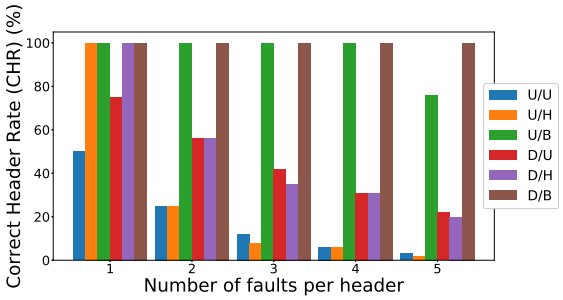
In order to study the influence of the subflit size on the BiSu efficiency concerning the protection of header flits, we display in Figure 3.15 the percentage of correct header



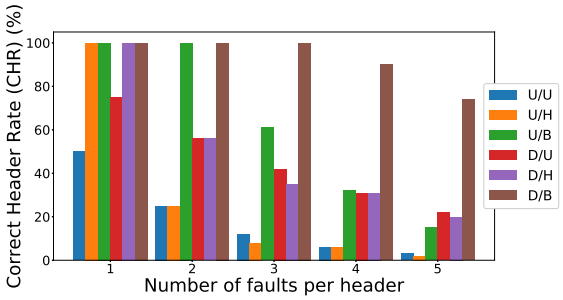
(a) 2-bit subflit size.



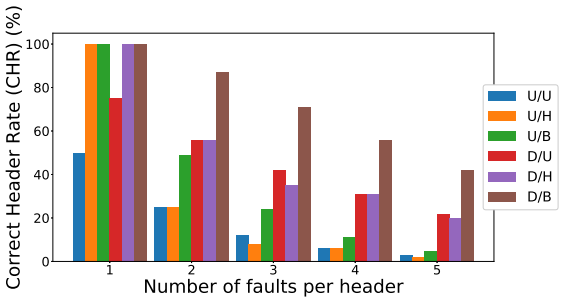
(b) 4-bit subflit size



(c) 8-bit subflit size.

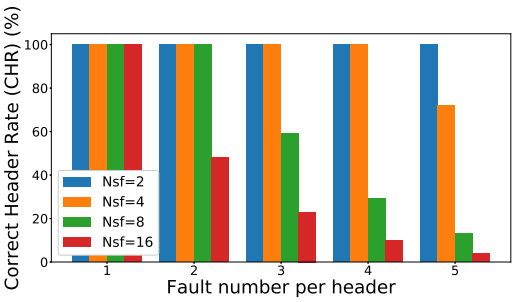


(d) 16-bit subflit size

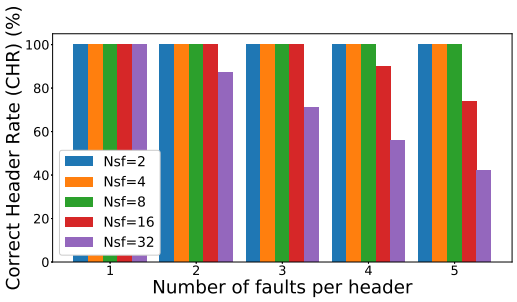


(e) 32-bit subflit size.

Figure 3.14: Influence of the method on the header accuracy considering 64-bit flit.



(a) 32-bit flit size.



(b) 64-bit flit size

Figure 3.15: Influence of the subflit size on the header flit accuracy.

transmissions considering flit size equal to 32 and 64 bits under different subflit sizes. In this figure, we focus on the results obtained using the header distribution. Through these results, we note that decreasing the subflit size allows to manage higher permanent faults with the BiSu method. We can extrapolate that the number of permanent faults which can be managed with the proposed method is equal to the subflit number containing only unused bits. For example, considering 32-bit flits with 4-bit subflits (cf. orange bar in Figure 3.15a), we have 4 subflits which contain only unused bit in each header, i.e. 16 unused bits per header flit with the header distribution on two flits. Then, until 4 permanent faults can be managed keeping 100% of correct header transmissions.

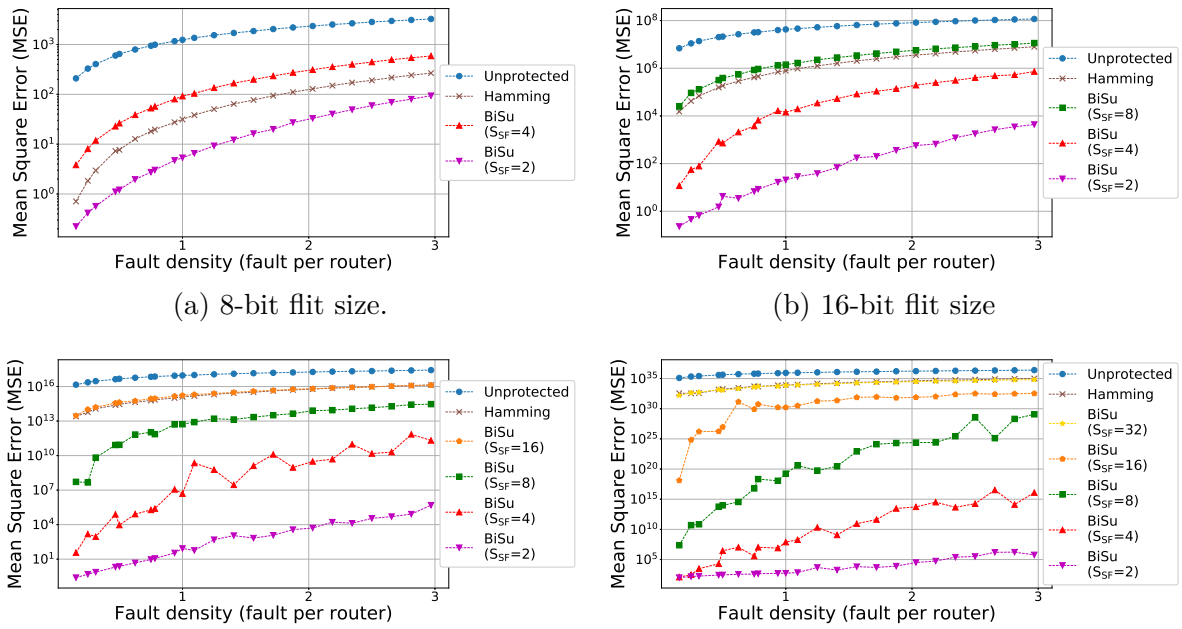
Based on these results, we can conclude that for large databus, the header distribution is not required to manage permanent faults with the BiSu approach. However, when a small data bus is used, the header distribution on two flits is useful, as shown in Figure 3.13. Moreover, we highlight the fact that the number of faults which can be managed with the proposed method mainly depends on the number of unused subflits. This value can be controlled by defining the subflit size according to the number of unused bits in headers or payloads containing critical data. Based on the results depicted in Figure 3.15, we can argue that the proposed BiSu approach can manage in the worst case a number of faults equal to the number of unused subflits and in the best case equal to the number of unused bits. While the worst case means that each unused subflit tackles one fault, the better case means that each unused bit is used to tackle one fault.

3.2.3 NoC-Level Evaluations

In this part, we evaluate the BiSu technique at the NoC-level. For that, we first present the experimental setup. Then we study the behavior of the BiSu technique at the NoC level considering data payloads which can tolerate faults during on-chip transmissions. Finally, the behaviors of the proposed method is studied considering headers which cannot tolerate any errors during the NoC communications.

3.2.3.1 Experimental Setup

In this experiment, we consider a 8×8 NoC crossed by packets of 16 payload flits and 1 or 2 header flits depending on whether the header distribution technique, presented in Section 3.1.4, is applied. The behavior of the BiSu technique is compared to an extended Hamming code and the unprotected case. We consider $10,000$ fault-injection patterns,



(a) 8-bit flit size.

(b) 16-bit flit size

(c) 32-bit flit size.

(d) 64-bit flit size

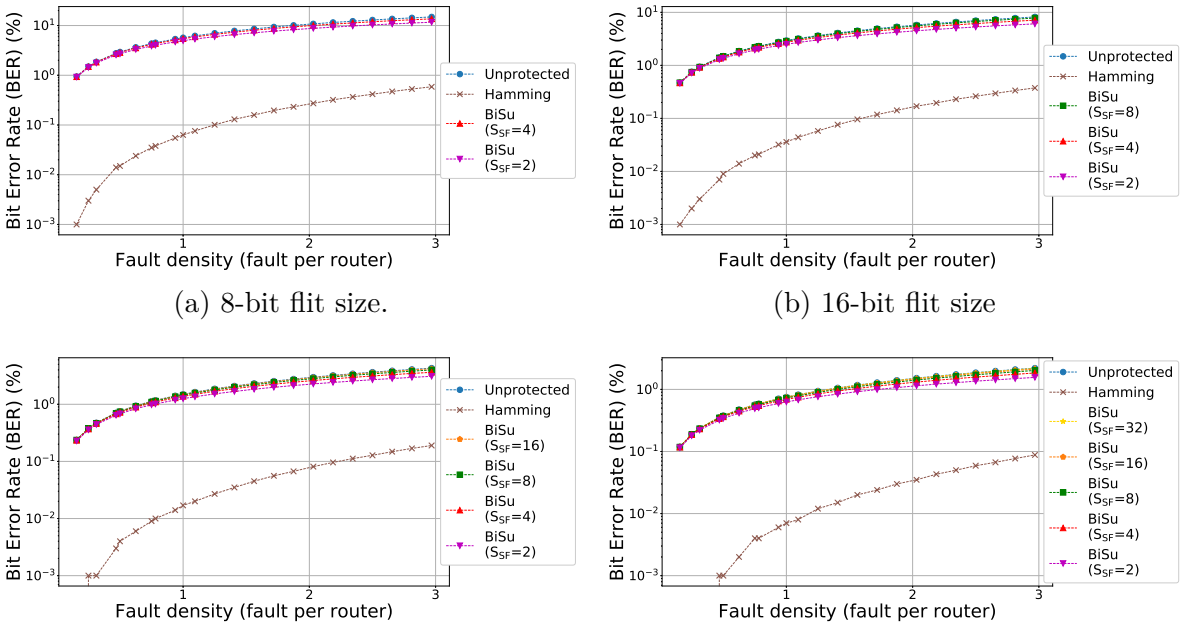
Figure 3.16: BiSu efficiency for payload protection at the NoC-level using the MSE metric.

where the permanent faults are modeled with the stuck-at fault model. As in the previous experiments, we consider that a fault always impacts the flit, and thus, a bit-flip is applied on the faulty bits to avoid the masking effects due to the bit values. The SHEs are randomly injected in the NoC impacting the datapath of the NoC, i.e. the buffers, crossbars and interconnections. Flits are injected according to the TORNADO injection pattern, where each IP sends one packet to each other. The C++ simulations are performed on an Ubuntu 18.04.5 LTS Linux distribution executed on a 48-cores Intel© Xeon(R) Silver 4214 CPU @ 2.20GHz with a Quadro RTX 5000/PCIe/SSE2 graphic card.

3.2.3.2 Payload-Mitigation Experimental Results

In this part, we study the efficiency of the BiSu technique at the NoC level considering data payloads. For that, we compute the MSE and the BER, respectively depicted in Figures 3.16 and 3.17, to quantify the fault impact on the payloads. The results obtained for flit sizes equal to 8, 16, 32 and 64 bits considering different subflit sizes are compared to the unprotected and Hamming cases.

Figure 3.16 displays the MSE according to the fault density. For sake of clarity, we express the fault density in terms of average number of faults per router, i.e. number



(a) 8-bit flit size.

(b) 16-bit flit size

(c) 32-bit flit size.

(d) 64-bit flit size

Figure 3.17: BiSu efficiency for payload protection at the NoC-level using the BER metric.

of faults divided by the number of routers of the NoC. Thus, a density of 1.00 fault per router does not mean that each router is impacted by one fault, but that the fault average per router in the NoC is equal to 1.00. For example, if we consider that 16 faults impact a NoC of 64 routers, then the fault density is equal to 0.50 fault per router. In this figure, we observe that our method significantly reduces the MSE of the flits which cross the faulty NoC compared to the unprotected case, even under a high fault density. Furthermore, we note that the BiSu efficiency increases when the subflit size is reduced. Moreover, we observe that the results obtained with the extended Hamming code are approximately equal to the BiSu technique with a subflit size equal to half the size of the flit, i.e. the largest possible size. Then, we can argue that the BiSu method is more efficient than the extended Hamming code to manage the fault impacts at the NoC level. For example, if we consider a fault density equal to 1.00 fault per router for 32-bit flits with 4-bit subflits (cf. red points in Figure 3.16c), then the BiSu technique reduces the MSE from 9.22×10^{16} to 4.94×10^{6}, while on the contrary, extended Hamming code can only reduce the MSE up to 1.29×10^{15}.

Figure 3.17 displays the BER according to the fault density. First, we observe in this figure that the extended Hamming code presents better results than the BiSu technique.

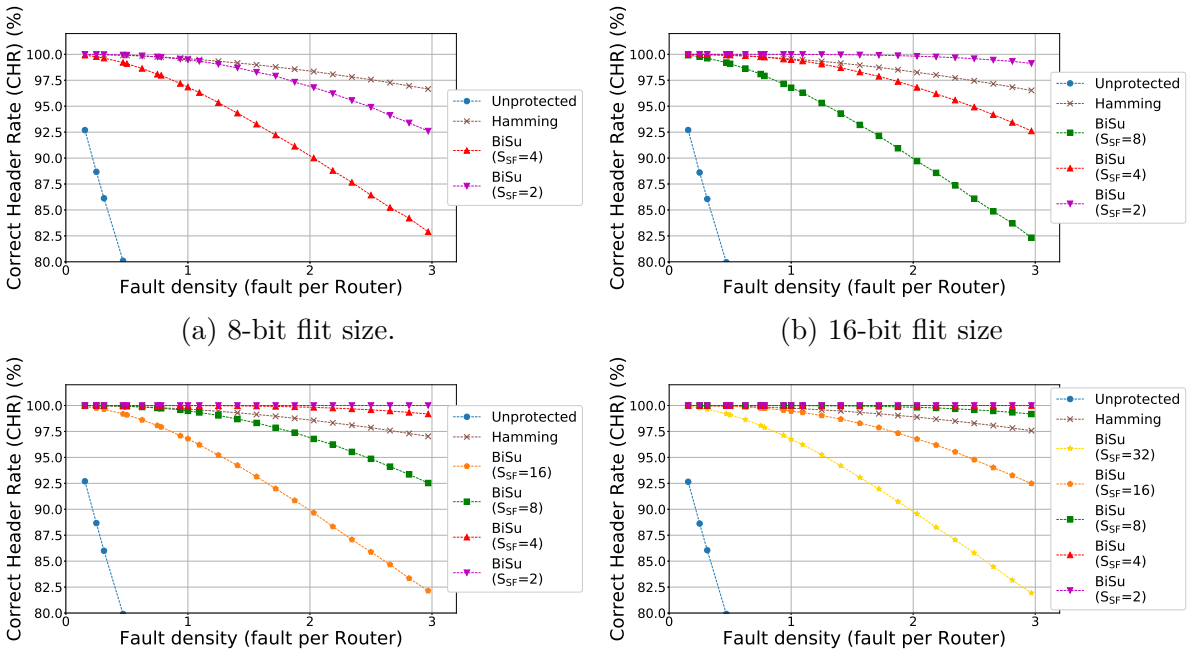
This is due to the fact that the extended Hamming code is able to correct all single SHE in the flits while the BiSu technique can only defer the fault impacts on the LSBs. Thus, the extended Hamming code reduces the number of faulty bits contrary to the BiSu technique. For example, considering 32-bit flits with 4-bit subflits (cf. red points in Figure 3.17c), using the extended Hamming code decreases the BER from 1.50×10^{-2} to 1.70×10^{-4} for a fault density equal to 1.00 fault per router. However, as mentioned previously, the extended Hamming code is not able to correct more than one faulty bit in the same flit at the same time. Then, it cannot protect the MSBs against multiple faults contrary to the BiSu technique which protects the MSBs by transferring the fault impacts on the LSBs. Thus, the BiSu method stays more efficient than the extended Hamming code.

Second, we observe in Figure 3.17 that the subflit size has an impact on the BER. Indeed, decreasing the subflit size leads to BER reduction. As the fault impacts are reported on the LSBs, these latter can be used to tackle several faults during several successive shuffling operations, i.e. faults are tackled using bits which are already faulty. Thus, reducing the subflit size increases the probability to defer several fault impacts on the same bits. For example, decreasing the subflit size from 8 to 4 bits considering 32-bit flits (cf. green and red points in Figure 3.17c) and a fault density equal to 1.00 fault per router leads to BER reduction from 1.45×10^{-2} to 1.38×10^{-2}.

3.2.3.3 Header-Mitigation Experimental Results

In this part, we evaluate the efficiency of the BiSu technique at the NoC level considering control flits, i.e. headers. In this purpose, we quantify the fault impact by computing the Correct Header Transmission Rate (CHTR), i.e. the percentage of header flits reaching their destination without any error on their control bits. The results obtained for flit sizes equal to 8, 16, 32 and 64 bits considering different subflit sizes are compared to Hamming and unprotected cases.

Figure 3.18 depicts the CHTR according to the fault density. In this figure, we first note that the Hamming case and the BiSu method ensure correct transmissions of headers when the NoC is faulty. However, in presence of a high density of faults, i.e. it cannot preserve the control bits of the headers. Indeed, contrary to the Hamming code, the BiSu technique can protect the headers against a high density of faults using adequate flit and subflit sizes. For example, considering 32-bit flit, the Hamming code can only guarantee a CHTR greater than 99% for a fault densities inferior to approximately 1.6 faults per router. In other hand, the BiSu method can maintain the correctness of the header transmissions



(a) 8-bit flit size.

(b) 16-bit flit size

(c) 32-bit flit size.

(d) 64-bit flit size

Figure 3.18: BiSu efficiency for header protection at the NoC-level using the Correct Header Transmission Rate (CHTR) metric.

above 99% for fault densities smaller than 1.25 faults per router considering subflits of size 8 and for a fault density greater than 3.00 faults per router when the subflit size is equal to 2 bits.

3.2.4 Application-Level Evaluation: Convolutional Neural Network

In this part, we evaluate the BiSu technique at the application level. For that, we will study the image transfer from memory to the IP, which supports the CNN execution. We use a CNN defined with the Keras API [224] for image recognizing using the Cifar10 [225] database. This latter is composed of $60,000$ 32×32 colored images which are split into 10 classes. While $50,000$ of these images are used to train the CNN during the learning phase, the remaining $10,000$ images are used to test the CNN and quantify the accuracy of the classification. Note that the term *epoch* refers to how many times the training base is used during the learning phase. Theoretically, the validation accuracy is up to 75% for 25 epochs and up to 79% for 50 epochs.

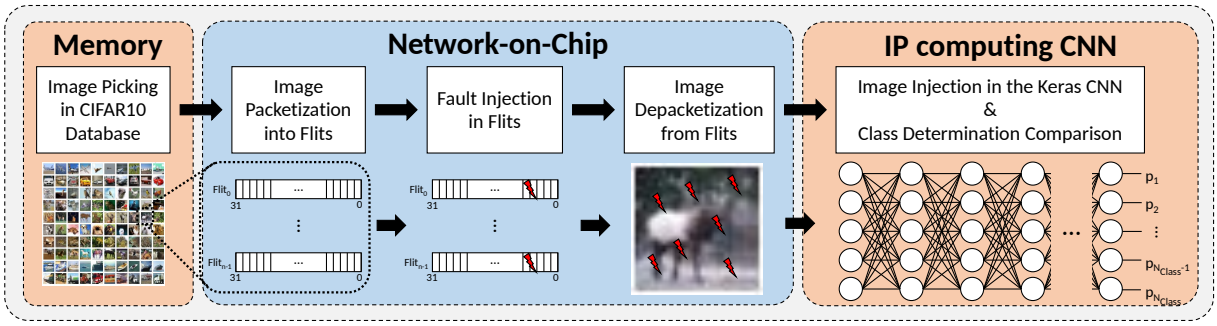


Figure 3.19: Fault injection in the Keras Convolutional Neural Network (CNN) benchmark.

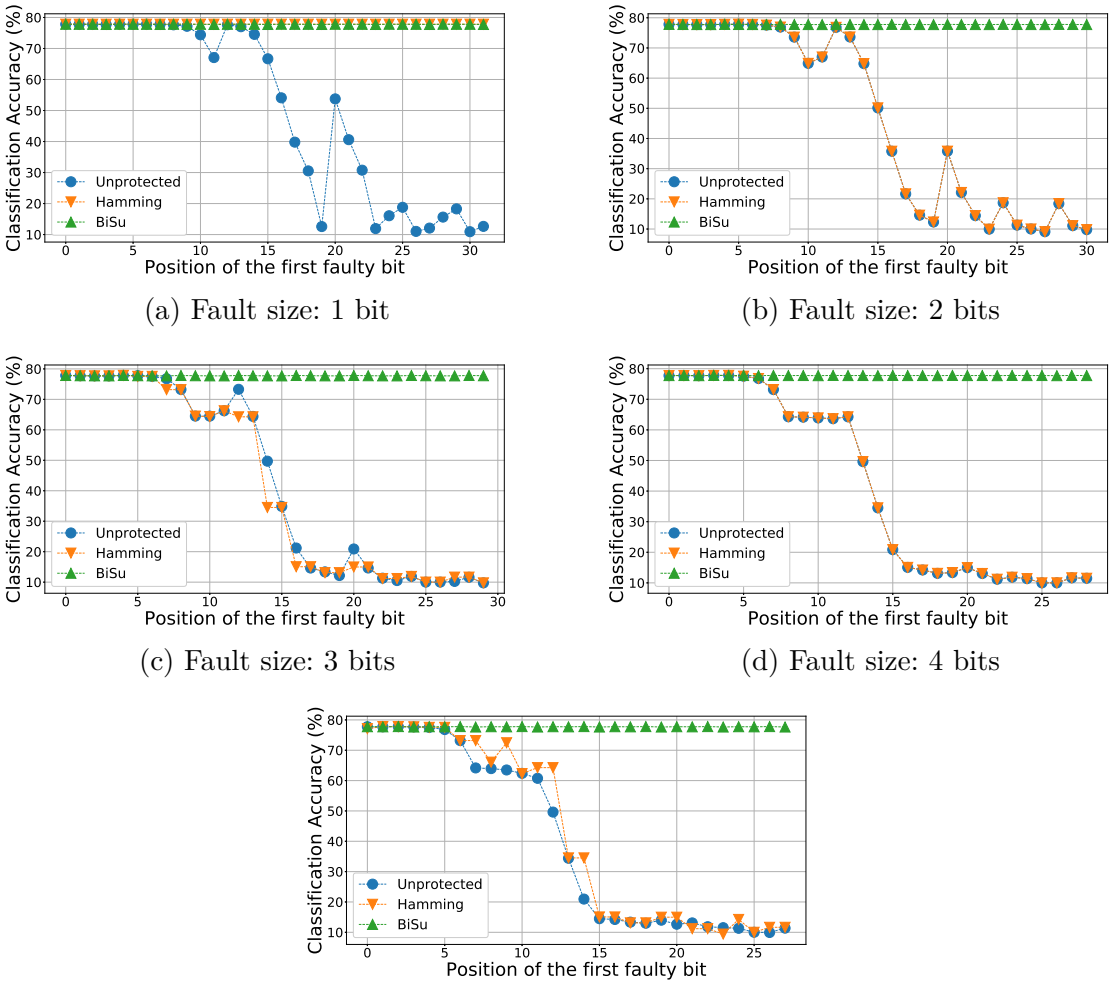
In this part, we evaluate the BiSu technique at the application level.

3.2.4.1 Experimental Setup

For our experiment, we consider a fault-free training phase of 50 epochs to obtain a theoretical classification accuracy of 79% and we focus on the inference phase, which is supposed to be implemented on NoC architectures. As shown in Figure 3.19, the data of the images which are used to test the CNN accuracy are normalized into 16-bit floating-point format and packetized into 32-bit flits, following the approach presented in Section 3.1.5. Fault injections are performed on those flits to simulate the faulty NoC communications. The faulty images are pushed in the CNN to compute the approximated classification accuracy. In this experiment, we perform an exhaustive exploration of all possible fault positions considering one MHE. As for the flit-level evaluations and the NoC-level evaluations, we consider that the fault always changes the value of the impacted bits to avoid masking effects due to the data values. The results are computed for MHE sizes from 1 to 5 bits. The classification accuracy obtained with the BiSu technique considering subflit of size 4 bits is compared with the classification accuracy obtained with the extended Hamming code and with no protection. Simulations are performed on a Fedora 28 Linux distribution executed on an 8-cores Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz by using Python 3.6.8 with the help of the Keras and TensorFlow APIs.

3.2.4.2 Experimental Results

Figure 3.20 depicts the classification accuracy, i.e., the percentage of correct classification, according to the position of the first faulty bit of the MHE considering a fault size from 1 to 5 bits.



(a) Fault size: 1 bit

(b) Fault size: 2 bits

(c) Fault size: 3 bits

(d) Fault size: 4 bits

(e) Fault size: 5 bits

Figure 3.20: Classification accuracy of the CNN using the CIFAR10 database under different sizes of MHE.

In Figure 3.20a, we consider a fault of 1 bit. We observe that when the flits are unprotected, the classification accuracy is affected when bits with a position higher than 8 are impacted by the fault. In particular, it can drop to 10%, when MSBs are affected. However, when the BiSu technique is applied, the class determination accuracy stays at around 79%, which is the maximum theoretical value that we can obtain with this trained CNN. Regarding the Hamming code, the results are similar to BiSu technique, since only one bit is faulty, and thus, it can be corrected, providing the maximum theoretical accuracy.

In Figure 3.20b, the fault has a size of 2 consecutive bits. Similar to the previous experiment, the accuracy starts to decrease when the bits with positions higher than 8 are affected by the fault. Indeed, since the MHE has a size of 2 bits, the first faulty bit is the bit number 8. In this case, the classification accuracy for unprotected flits and flits protected with Hamming code has similar behavior and drops to 10% when the bits with a position greater than 8 are affected. On the contrary, when the flits are protected with the BiSu technique, the accuracy remains at 79% regardless the position of the fault. These observations can be extended to the case where the MHE size is equal to 4 bits, as displayed in Figure 3.20d.

Figure 3.20c shows the results when the fault has a size of 3 bits. The behavior is similar to Figure 3.20b, except that the accuracy starts to degrade when the first faulty bit is in position 7 due to the fault size equal to 3 bits. As a 3-bit fault is considered, the Hamming code is unable to detect the fault. In addition, as the parity bit detect 1 fault due to the odd number of faulty bits, the Hamming code can perform wrong corrections. These false corrections explain why the classification accuracy with Hamming code is sometimes better or worse than the accuracy obtained with the unprotected flits. In other hand, BiSu technique offers a maximum theoretical accuracy even under a 3-bit fault. These observations can be extended to the case where the size of the MHE is equal to 5 bits, as displayed in Figure 3.20e.

To conclude, we saw through the previous experiments that the BiSu technique is more efficient than the Hamming code when more than 1 bit are affected in the incoming flits. Indeed, our method is able to maintain the classification accuracy at the maximum theoretical value when conventional methods, such as Hamming code, are not able to preserve the performances of the CNN. In addition, we saw through the cases presented in Figure 3.20 that the class-determination accuracy is degraded when the bits of positions higher than 8, i.e. the position of the first critical bit, are affected by the fault. Thus, we

can extrapolate that the BiSu technique can theoretically, in the best case, handle up to 9 faulty bits, i.e. one MHE of size 9 or nine SHEs of size 1, by positioning the faults to bits at positions 0 up to 8. In addition, we can compute the minimal number of faulty bits which can be managed in the worst case with the BiSu technique by using Equation 3.9. In this case, each subflit containing bit positions smaller than the position of the first critical bit is affected by only one SHE. Thus, for the presented experiments, the BiSu approach can maintain the classification accuracy at the maximum theoretical value under 2 SHEs.

$$N_{min_faulty_bit} = \left\lfloor \frac{First_critical_bit_position}{S_{SF}} \right\rfloor \tag{3.9}$$

3.2.5 Case Studies: Image Processing and Data Mining Applications

In this part, we propose two case studies by using image processing and data mining applications. For that, we first present the experimental setup of the two case studies to detail how the data which transit on the considered NoC are affected by the permanent faults. Then, we present the results obtained for the computation of a Sobel filter and a K-means clustering algorithm to evaluate the efficiency of the BiSu method concerning the data protection for these applications which can support approximate results.

3.2.5.1 Experimental Setup

During these case studies, we consider data exchanges between a main memory and a core, located at a distance of 3 routers in a $N \times N$ mesh NoC, as illustrated in Figure 3.21. The XY routing algorithm is used to transmit data through the NoC, using a flit size of 32 bits. While the purple arrow depicts the routing path to load data from memory to core, i.e. the loading path, the cyan arrow represents the routing path to store data from core to memory, i.e. the storing path. The proposed BiSu technique is implemented through the shuffler and de-shuffler blocks which mitigate the permanent faults inside each router and each interconnection. All simulations are performed on a Fedora 28 Linux distribution executed on an 8-cores Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz. The obtained results are compared to the unprotected and Hamming cases. The red lightning bolts of Figure 3.21 represents the injected faults on the datapath, impacting flits. Thus, for the four faulty interconnections and routers, we consider:

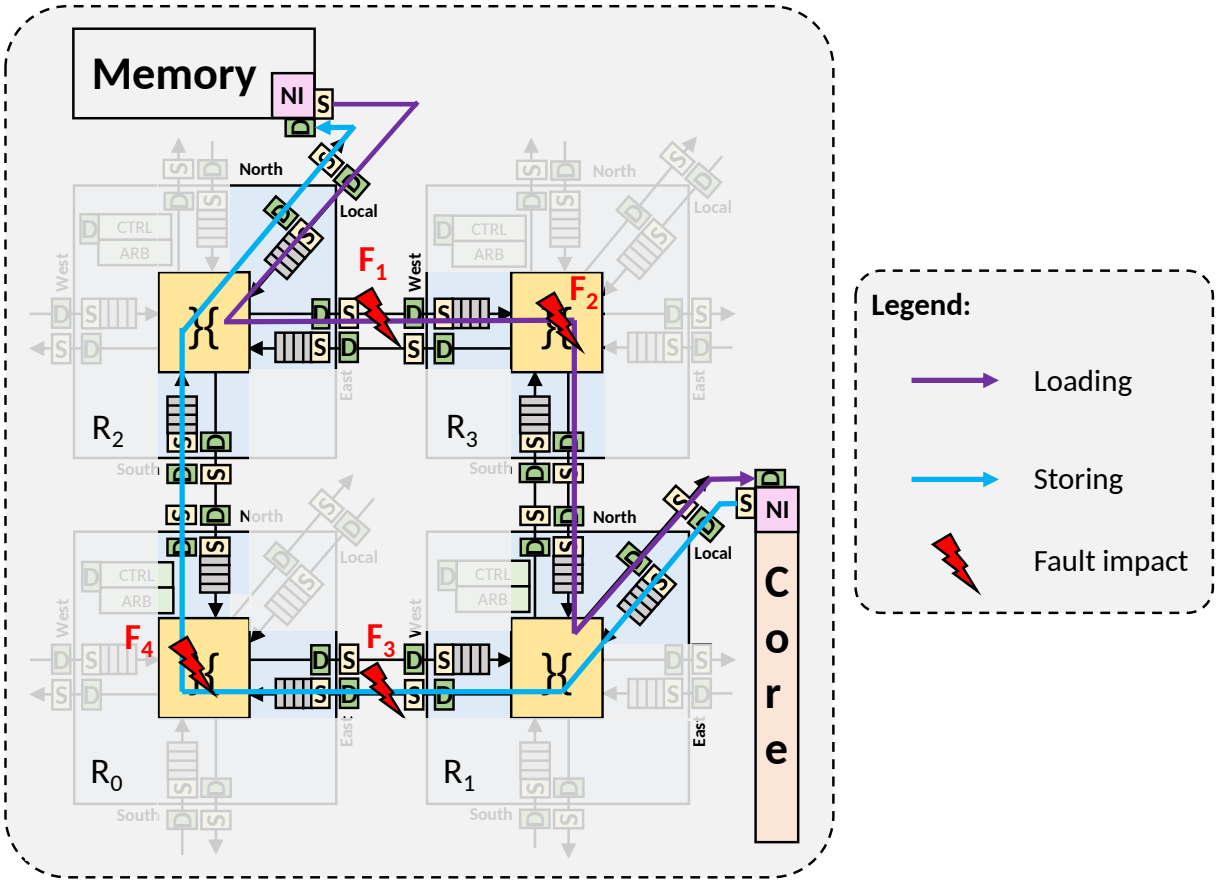


Figure 3.21: Fault localization on the load and the store paths on a part of mesh NoC.

F1: One SHE in the interconnection between routers R_2 and R_3, i.e. loading path, which affects the bit number 13 according to the stuck-at-1 fault model.

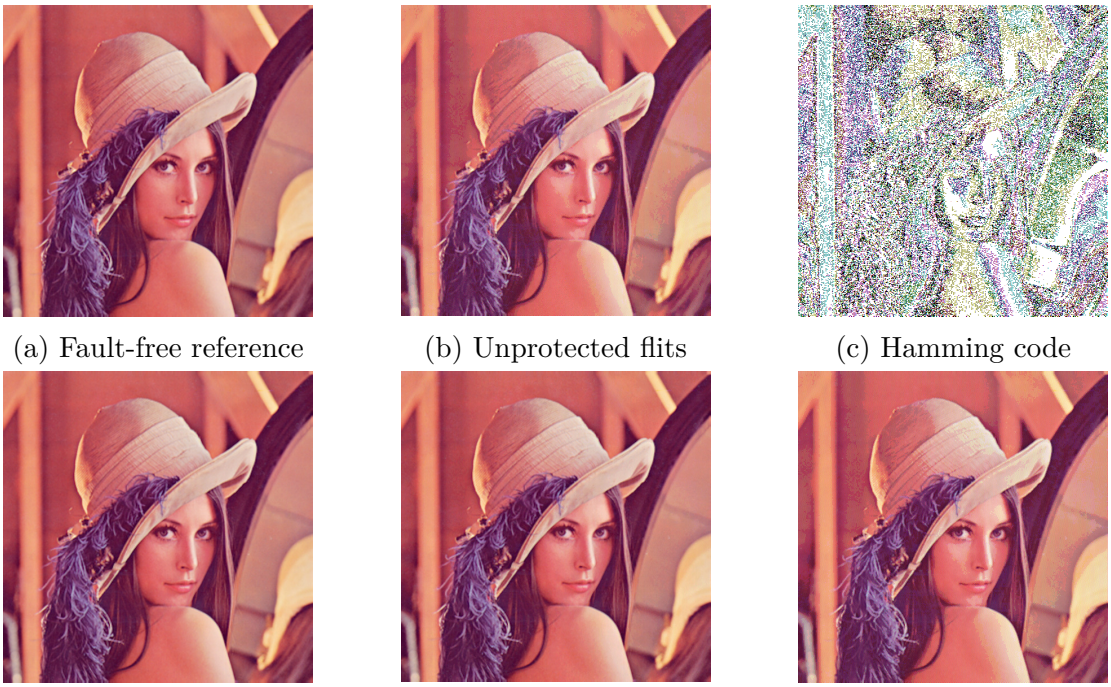
F2: One 2-bit MHE in the router R_3, i.e. loading path, which affects the bit number 27 and 28 according to the stuck-at-1 fault model.

F3: Two SHEs in the interconnection between routers R_1 and R_2, i.e. storing path, which affect the bit number 5 and 29 according to the stuck-at-0 fault model.

F4: One 3-bit MHE in the router R_0, i.e. storing path, which affects the bit number 8, 9 and 10 according to the stuck-at-0 fault model.

3.2.5.2 Sobel Filter

For the first experiment, we simulate the execution of a Sobel filter [226], used for edge detection in image processing fields, with the help of the Matlab R2017b software. As



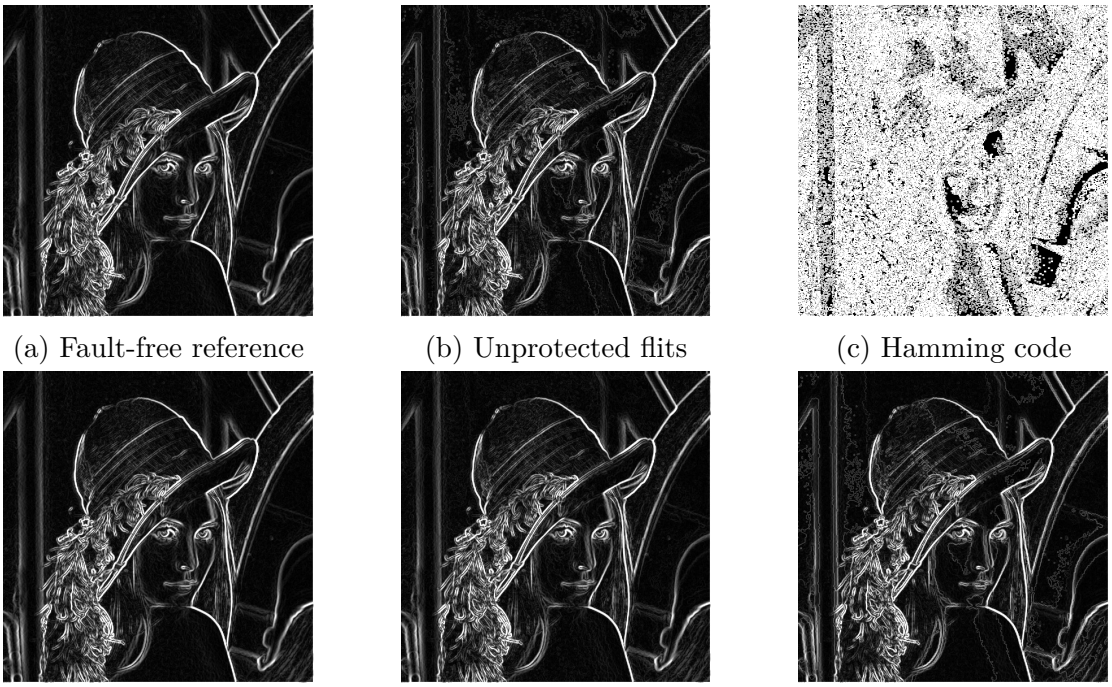
(a) Fault-free reference (b) Unprotected flits (c) Hamming code

(d) BiSu ($S_{SF} = 2$) (e) BiSu ($S_{SF} = 4$) (f) BiSu ($S_{SF} = 8$)

Figure 3.22: State of the initial image in the core after data loading from memory to core via the loading path.

shown at the top of Figure 3.21, initial and output images are stored in the memory while the Sobel filter is computed in the core. During loading operations, 8-bit data (unsigned integer) transit the NoC through the purple path. During storing operations, 64-bit data (double precision) pass through the NoC via the blue path. As we consider a NoC with flit size equal to 32 bits, data are organized within flits as described in Section 3.1.5 using subflit sizes equal to 2, 4 and 8 bits. To evaluate our approach, the Peak Signal-to-Noise Ratio (PSNR) is computed based on the fault-free reference. In the rest of this part, we analyze the results for the i) loaded image (cf. Figure 3.22), ii) the computed Sobel filter (cf. Figure 3.23) and iii) the stored Sobel filter (cf. Figure 3.24). Table 3.2 groups the PSNR values for each of the studied cases.

Figure 3.22 presents the state of the initial image when it is loaded from the memory to the core using the purple path of Figure 3.21. As this path is impacted by the permanent faults described previously, the data which compose the initial image are affected and the loaded image is deteriorated. Figures 3.22a and 3.22b display respectively the fault-free reference and the unprotected case. By comparing these two pictures, we note that the unprotected image is degraded due to the permanent faults present on the path even if



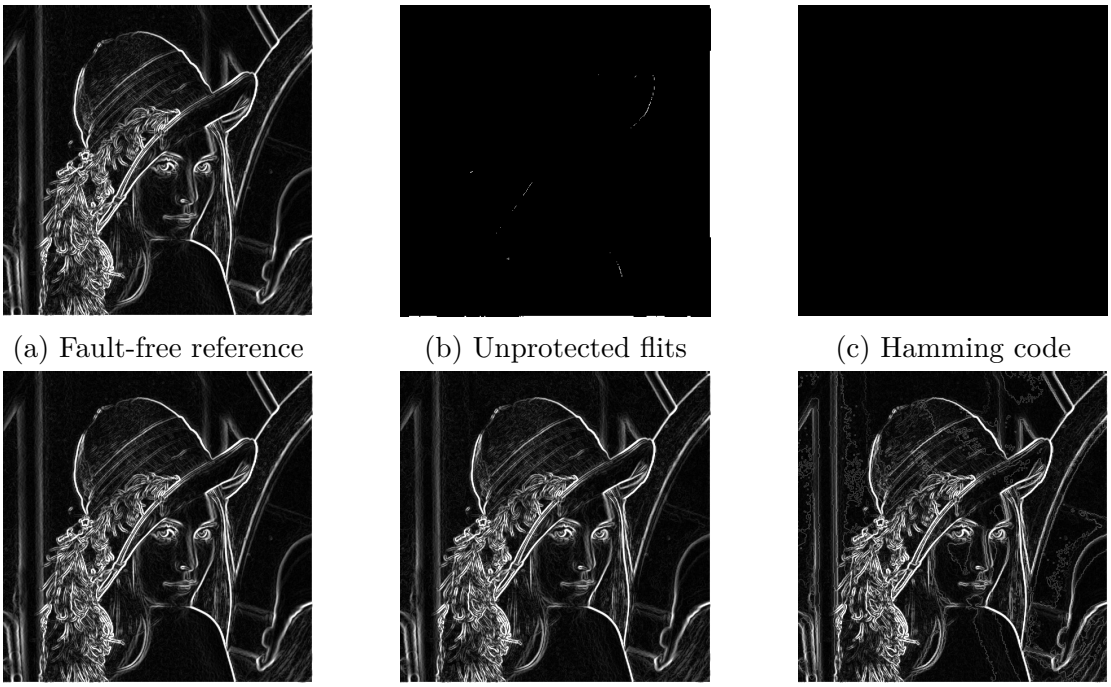
(a) Fault-free reference (b) Unprotected flits (c) Hamming code

(d) BiSu ($S_{SF} = 2$) (e) BiSu ($S_{SF} = 4$) (f) BiSu ($S_{SF} = 8$)

Figure 3.23: Results of the Sobel-filter computing in the core using the loaded images via the loading path.

the impact is difficult to analyze visually. Moreover, we observe in Figure 3.22c that the use of an Hamming code is useless in this case since the loading path is impacted by multiple permanent faults giving a totally different image compare to the reference. On other hand, the loaded images using the BiSu technique with subflit sizes of 2, 4 and 8 bits are respectively shown in Figures 3.22d, 3.22e and 3.22f. While the subflit sizes 2 and 4 produce similar images compare to fault-free reference, the use of 8-bit subflit induces slight degradations, as for the unprotected case. Indeed, for this case, the inefficiency is linked to the transmitted data size which is equal to 8 bits. Therefore, by using 8-bit subflit, we shuffle entire data instead of the least or the most significant part leading to an inoperative shuffling. Thus, the BiSu method reaches its limits when the subflit size is equal to the data size since it cannot mitigate the faults which impact the data.

As the Sobel filter is applied on the loaded image, the computing quality depends of the data deterioration. Thus, the same observations than for the loaded images can be made for the different cases presented in Figure 3.23. We note that the unprotected case, depicted in Figure 3.23b is corrupted by the presence of grains on the picture which are due to the deterioration of the loaded image. Figure 3.23c shows that the use of an



(a) Fault-free reference (b) Unprotected flits (c) Hamming code

(d) BiSu ($S_{SF} = 2$) (e) BiSu ($S_{SF} = 4$) (f) BiSu ($S_{SF} = 8$)

Figure 3.24: Results of the Sobel-filter computing after data storing in the memory using the storing path.

Hamming code is totally useless in this application considering so many faults since the results of the Sobel filter computation is unrecognizable. Finally, while the BiSu technique using subflit sizes of 2 (cf. Figure 3.23d) and 4 bits (cf. Figure 3.23e) gives results very close to the reference (cf. Figure 3.23a), using 8-bit subflits (cf. Figure 3.23f) gives the same result than the unprotected case for the same reasons than latter exposed previously.

The last results, displayed in Figure 3.24, present the pictures once they are stored in the memory. Indeed, as the data which compose the Sobel pictures transit through the NoC via the storing path (cf. Figure 3.21), they are impacted by the permanent faults $F2$ and $F3$. Figure 3.24a displays the fault-free reference. We observe that the unprotected case, depicted in Figure 3.24b, is particularly corrupted by the stuck-at-0 faults. As this type of fault sets the impacted bit at 0, the pixel values are decreased and decoded as black pixels when the picture is restored. The same effect can be observed in Figure 3.24c where the data are encoded with an Hamming code. In other hand, we note that the use of the BiSu technique with subflit size equal to 2 bits (cf. Figure 3.24d) and 4 bits (cf. Figure 3.24e) keeps the results close to the reference. Moreover, Figure 3.24f offers the possibility to manage the faulty storing path contrary to the fault impacts on the loading

Location	**Unprotected**	**Hamming Code**	**BiSu ($S_{SF} = 2$)**	**BiSu ($S_{SF} = 4$)**	**BiSu ($S_{SF} = 8$)**
Core Image	25.29	5.45	50.17	37.97	23.51
Core Sobel	23.67	−5.07	43.29	34.48	23.29
Memory Sobel	10.89	NaN	43.29	34.48	23.29

Table 3.2: PSNR results for the computation of the Sobel filter.

path. This is due to the fact that the Sobel images are composed of 64-bit data while the initial image is composed of 8-bit data. As the subflit size is lower than the data size, the BiSu method is efficient in managing permanent faults. Based on this, we note that the result with a subflit size equal to 8 bits is only degraded due to the fault impacts on the loading path on the initial image.

As the image processing applications have a high resilience, the fault impacts are hard to see on the picture because of the imperfection of the human eye. In consequence, we compute the PSNR to precisely evaluate the fault impacts on the results according to the fault-free reference. The obtained PSNR values are displayed in Table 3.2. Through these values, we see that the Hamming code is definitely so useless in presence of multiple permanent faults that the PSNR value cannot be computed for the stored picture. In other hand, the PSNR values obtained with the BiSu method show that the image quality is largely maintained in presence of faults when subflit sizes of 2 bits and 4 bits are considered. However, the results obtained for a subflit size equal to 8 bits show that the method reaches its limits when the subflit size is equal to the data size. Finally, the decrease of the PSNR value when the subflit increases shows that this latter directly impacts the BiSu efficiency.

Based on these results, we can argue that the BiSu approach is more efficient than other methods, such as the extended Hamming code, to protect data during on-chip communication. The technique is then suitable for application needing high accuracy.

Metric	**Unprotected**	**Hamming Code**	**BiSu ($S_{SF} = 2$)**	**BiSu ($S_{SF} = 4$)**	**BiSu ($S_{SF} = 8$)**	**BiSu ($S_{SF} = 16$)**
MSE	9.7×10^{-3}	1.1×10^{-2}	5.1×10^{-8}	6.5×10^{-8}	3.6×10^{-7}	1.1×10^{-3}
CER (%)	90.49	92.95	0.13	0.07	0.34	8.85

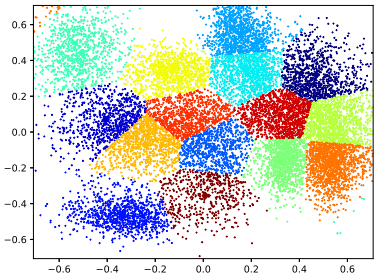
Table 3.3: Results of the K-means clustering algorithm using the MSE and the CER metrics.

3.2.5.3 K-Means Clustering Algorithm

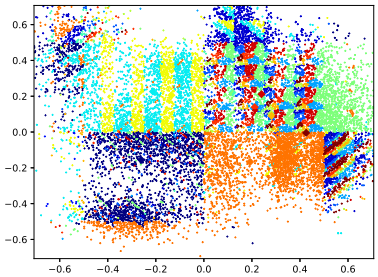
For the second studying case, we consider a K-means clustering algorithm [227], typically used in signal processing and data mining, i.e. image classification and voice identification. This algorithm is used to cluster a set of random input data by minimizing the accumulated square distance between centroids, i.e. cluster centers, and their associated data through an iterative process. During these experiments, we compute the MSE of the mean centroid positions and the Clustering-Error Rate (CER) using the fault-free case as reference. Simulations are performed with a C++11 testbench, using 32-bit signed fixed-point data with 1 bit for the integer part. We use 20 data sets composed of 15 centroids and $1,000$ sample data are generated by centroids. Each data set is performed using at most 150 iterations.

Figure 3.25 depicts the results for the first data set. The fault-free reference is given in Figure 3.25a. Through these results, we can note that the algorithm cannot perform clusters in the unprotected and Hamming cases, respectively displayed in Figures 3.25b and 3.25c. This is due to the square distance computation which is totally distorted by the presence of the multiple permanent faults on the loading and storing paths. On the contrary, the BiSu technique enables a correct clustering which is visually very close to the reference, as shown in Figures 3.25d, 3.25e, 3.25f and 3.25g which respectively display the results considering subflit sizes of 2, 4, 8 and 16 bits.

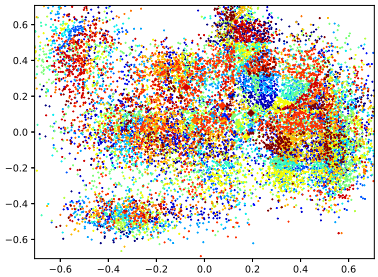
To further evaluate the BiSu approach, we compare in Table 3.3 the MSE and CER considering all 20 data sets. These results show that the BiSu technique largely reduces the two metrics considered compare to the unprotected and Hamming cases. For example, the BiSu method considering 4-bit subflit size (cf. Figure 3.25e) reduces the MSE of the mean centroid positions from 9.7×10^{-3} to 6.5×10^{-8} and decreases the CER from 90.49% to 0.07% compare to the unprotected case. In comparison, we note that the Hamming code has low impact on the MSE and the CER. Finally, we deduce from these results



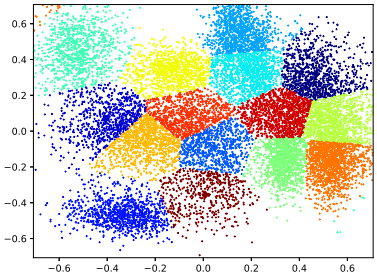
(a) Fault-free reference



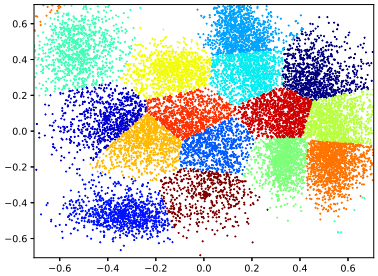
(b) Unprotected flits



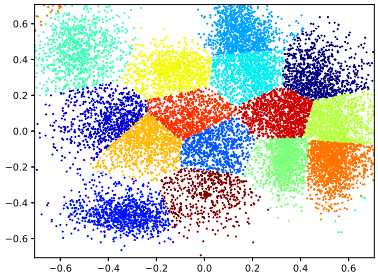
(c) Hamming code



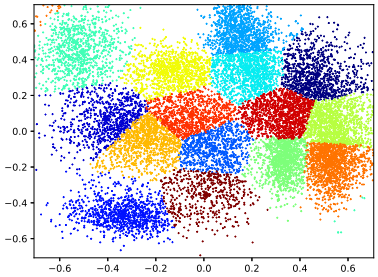
(d) BiSu ($S_{SF} = 2$)



(e) BiSu ($S_{SF} = 4$)



(f) BiSu ($S_{SF} = 8$)



(g) BiSu ($S_{SF} = 16$)

Figure 3.25: K-means clustering results for the first data set.

that the subflit size has an impact on the efficiency of the method since the values of the evaluation metrics are better when the subflit size is decreased.

Based on these results, we can argue that the BiSu approach is efficient to protect data during on-chip communication and that it is suitable for application needing high accuracy.

3.3 BiSu-Hardware Evaluation

In this section, we evaluate the hardware costs of the BiSu technique in terms of area, power consumption and critical path. For this purpose, we synthesize the shuffler, de-shuffler, merger, and de-merger blocks using the High-Level Synthesis (HLS) tools of Mentor Graphic. Synthesis are performed based on 28 nm FDSOI technology targeting a clock frequency of 1 GHz. As comparison, we also synthesize the encoder, checker and decoder blocks of an extended Hamming code, which is typically used inside NoC routers. For each studying case, we consider that each interconnection and each router of the NoC are protected. First, we study the hardware costs of the shuffler and de-shuffler blocks which are used to re-organize the flits at-run time. Then, the overheads brough by the merger and de-merger blocks, which are used to manage the flit organization in the NIs, is explored. Thereafter, we make a global comparison of the BiSu hardware costs at the NoC level. Finally, we present the impacts on the hardware costs of the implementation of the register updater block at-run time instead of implementing a software version of this algorithm on IP cores.

3.3.1 Shuffler and De-shuffler Blocks

In this part, we study the implementation costs of the BiSu technique in the NoC, i.e. routers and interconnections. While Table 3.4 shows the area, the power consumption and the critical path of the BiSu technique for flit sizes of 32 and 64 bits considering different subflit sizes, Table 3.5 focuses on the hardware costs of the extended Hamming code for flit sizes of 32 and 64 bits. For this code, we perform the evaluation on the encoder and decoder blocks which are located between the NIs and the router, and on the checker blocks which are located within the routers.

For a fair comparison, as the placement of the Hamming blocks is not homogeneous in a NoC, we perform the comparison between the two techniques on a 8×8 NoC. The

NoC Parameters									
S_F	32				64				
S_{SF}	2	4	8	16	2	4	8	16	32
Bus Size	32				64				
Shuffler/De-shuffler Block									
Area (µm²)	502	291	191	142	1,751	940	545	364	278
Power (mW)	0.31	0.23	0.23	0.19	0.79	0.57	0.46	0.44	0.39
CP (ns)	0.17	0.29	0.22	0.21	0.25	0.25	0.30	0.23	0.37
8×8 **NoC CONNECT [228]**									
Area (µm²)	2,140,223				3,667,347				
Power (mW)	1,862				3,209				
BiSu Overhead for a 8×8 **NoC**									
Area (%)	31.5	18.3	12.0	8.9	64.2	34.4	20.0	13.3	10.2
Power (%)	22.2	16.8	16.2	13.7	33.3	23.9	19.1	18.5	16.1

Table 3.4: Hardware implementation costs for the proposed BiSu method.

NoC Parameters						
S_F	32			64		
Bus Size	39			72		
Hamming	**Encoder**	**Checker**	**Decoder**	**Encoder**	**Checker**	**Decoder**
Area (µm²)	205	519	459	393	1,318	971
Power (mW)	0.26	0.66	0.58	0.50	1.70	1.20
CP (ns)	0.32	0.69	0.58	0.32	0.85	0.76
8×8 **NoC CONNECT [228]**						
Area (µm²)	2,470,372			4,061,930		
Power (mW)	2,154			3,547		
Hamming Overhead for a 8×8 **NoC**						
Area (%)	31.4			33.8		
Power (%)	39.1			44.3		

Table 3.5: Hardware implementation costs for Hamming technique.

use of the Hamming code requires the integration of 64 encoders, 64 decoders, and finally 576 checkers bringing the main overheads. For this NoC-scale evaluation, we consider the state-of-the-art CONNECT router [228] based on 5-ports router, four virtual channels of 8-flit depth and a round-robin arbitration. Table 3.4 provides the area cost and the power consumption of the considered CONNECT NoC. For example, considering a bus size equal to 32 bits, the NoC requires an area of $2,140,223\ \mu m^2$ and consumes $1,862$ mW. However, it has to be noticed that the Hamming code increases the size of the bus as indicated in Table 3.5. For instance, considering flit of 32 bits, the Hamming code requires 7 extra bits, hence increasing the size of the bus and buffers to 39 bits. It results in the area and the power up respectively to $2,470,372\ \mu m^2$ and $2,154$ mW.

Regarding the costs of the techniques, we observe that the shuffler and de-shuffler blocks have generally lower area and power consumption than the Hamming blocks, even if the Hamming encoder has a low cost compare to the checker and the decoder. For example, for 32-bit flits with 4-bit subflits, the area of one shuffler or de-shuffler block is only 291 μm^2 and it only consumes 0.23 mW. In comparison, one Hamming checker block requires an area of 519 μm^2 and consumes 0.66 mW. Regarding the BiSu technique, we observe that higher area and power consumption are required for smaller subflits, due to the higher number of multiplexers. Indeed, a smaller subflit size means a higher subflit number. In this way, the BiSu hardware costs can become higher than the Hamming hardware costs when the subflit size is drastically decreased. For instance, for 32-bit flits with 2-bit subflits, the shuffler block requires 502 μm^2. However, the power consumption remains below the one of the Hamming blocks.

At the NoC-scale, we observe through Tables 3.4 and 3.5 that the BiSu technique has lower overheads in the 8×8 NoC architecture. For example, considering 32-bit flits with 8-bit subflits, the area and the power overheads are respectively decreased from 31.4% to 12.0% and from to 39.1% to 16.2% by using the BiSu method instead of the extended Hamming code. However, when the subflit size is drastically decreased the hardware overheads become equivalent and even greater than those induced by the Hamming code. For instance, considering 32-bit flits, the area and power overheads are respectively up to 18.3% and 16.8% with a subflit size of 4 bits and to 31.5% and 22.2% with a subflit size of 2 bits. Nevertheless, these configurations offer a fine-grain error mitigation which is not often required by the applications as they are able to mitigate a high number of faults (cf. Section 3.2). Finally, we observe that the critical path of BiSu technique is lower than the critical path of the Hamming checker and decoder, regardless the subflit size. For

example, if we consider a 32-bit flit with a 4-bit subflit, the critical path is only of 0.29 ns for the shuffler and de-shuffler blocks against 0.69 ns for the Hamming checker, hence our proposal can support higher NoC frequency than the one supported with Hamming.

Based on these results, we can argue that the BiSu method induces lower hardware overheads than other state-of-the-art methods, such as an extended Hamming code. However, the added blocks in the NIs need to be taken into account to fairly compare the two methods.

3.3.2 Merger and De-merger Blocks

In this part, we explore the impacts on the hardware costs of the merging and de-merging blocks which are respectively added to the packetization and de-packetization blocks of the NIs. As explained in Section 3.1.5, these blocks ensure the management of data when their sizes are different to the flit size. The hardware overheads induced by the use of the header distribution on two flits to mitigate faults impacting the header (cf. Section 3.1.4) are also studied.

Tables 3.6a and 3.6b show respectively the area, power and critical path overheads for the packetization and the de-packetization blocks. For instance, considering 32-bit flit size, the area and the power consumption of the packetization block are respectively of $2,386\ \mu m^2$ and 2.61 mW, and of $2,478\ \mu m^2$ and 2.73 mW for the de-packetization block. When a merger block is used to organize the subflits in the flits (M-Packetization), we observe an overhead of 43.5% for the area and of 35.3% for the power consumption considering a subflit size of 4 bits. For the de-packatization part, we observe an overhead of 91.4% for the area, and of 58.6% for the power consumption. When the header distribution on two flits is applied (M/D-packetization), the overheads are respectively upped to 46.7% and 38.8% for the area and the power and to 96.3% and 59.4% for the de-packetization part (M/D-de-packetization).

Through these results, we note that the merger and de-merger blocks have a high impact on the area and the power consumption of the packetization and de-packetization blocks. However, the size of these blocks stay small compared to the entire NoC. In addition, we observe that the implementation of the header distribution on two flits has few impacts on the hardware costs. As this approach needs only one supplementary flit per packet, it has a slight impact on the latency which is proportional to the distance between the source and the destination and to the packet size. Finally, the critical path remains close to 1 ns in all cases.

NoC Parameters									
S_F	32				64				
S_{SF}	2	4	8	16	2	4	8	16	32
Packetization Hardware Costs									
Area (µm²)	2,386				4,401				
Power (mW)	2.61				4.90				
CP (ns)	1.00				1.00				
M-Packetization Overhead									
Area (%)	69.3	43.5	33.2	25.5	84.2	48.6	30.9	22.9	19.3
Power (%)	58.6	35.3	21.9	19.4	73.5	42.0	24.3	15.7	13.5
CP (%)	0.0	−1.0	0.0	0	−1.0	−1.0	−1.0	−1.0	−2.0
M/D-Packetization Overhead									
Area (%)	72.7	46.7	34.8	28.8	84.5	50.1	35.1	25.3	23.1
Power (%)	63.8	38.8	25.0	23.4	77.2	44.1	29.4	19.5	17.9
CP (%)	0.0	−1.0	−2.0	0.0	0.0	−1.0	−4.0	−6.0	−13.0

(a) Packetization block.

NoC Parameters									
S_F	32				64				
S_{SF}	2	4	8	16	2	4	8	16	32
De-packetization Hardware Costs									
Area (µm²)	2,478				4,495				
Power (mW)	2.73				5.01				
CP (ns)	0.91				0.96				
M-De-packetization Overhead									
Area (%)	126.9	91.4	70.5	52.8	214.4	149.2	113.2	86.1	36.8
Power (%)	104.9	58.6	38.6	28.6	191.8	99.7	56.1	35.3	16.5
CP (%)	9.9	9.9	9.9	9.9	4.2	4.2	4.2	4.2	4.2
M/D-De-packetization Overhead									
Area (%)	129.0	96.3	74.6	54.5	216.9	152.0	115.5	88.0	38.9
Power (%)	105.6	59.4	38.6	28.8	192.6	100.2	55.8	34.5	16.3
CP (%)	9.9	9.9	9.9	9.9	4.2	4.2	4.2	4.2	4.2

(b) De-packetization block.

Table 3.6: Hardware implementation overhead for packetization/de-packetization blocks extended with merger/de-merger blocks and with 2-flit header distribution capabilities.

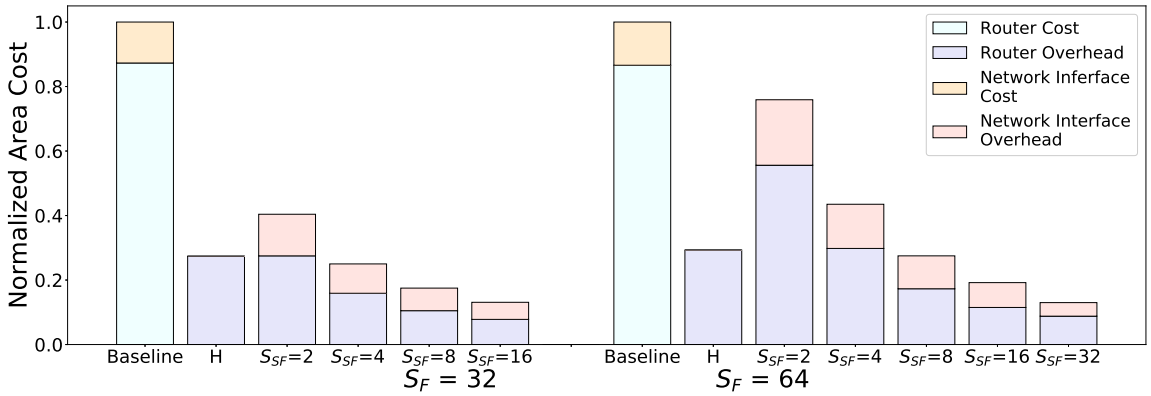
Based on this, we can conclude that the BiSu method has high impacts at the NI-scale. However, these impacts must be put into perspective with the total costs of the proposed method by taking into account the overhead induced in the routers and in the interconnections. Moreover, it is important to note that these results are the worst case as all possible data sizes among 8, 16, 32 and 64 bits are taken into account. Thus, the costs at the NI-scale can be largely reduced by limiting the possible data sizes which need to be managed by the merger and de-merger blocks when specific application is considered.

3.3.3 Comparison of the Global Hardware Costs

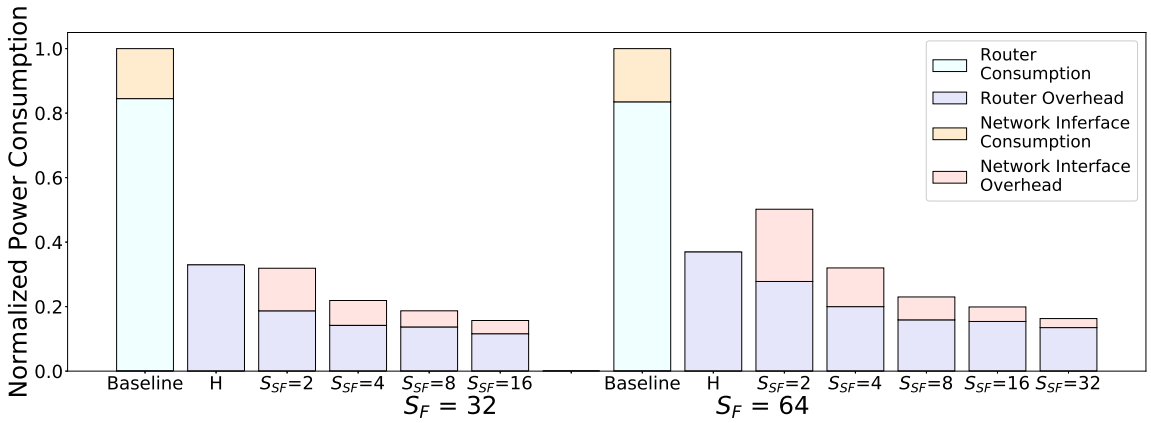
In this part, we compare the global hardware costs induced by the BiSu method. For that, we compute the area and power overheads induced in the entire NoC, i.e. routers, interconnections and IPs, taking as baseline the hardware costs of the unprotected 8×8 NoC. The obtained results are compared to the hardware overheads induced by the extended Hamming code.

Figure 3.26 shows the global overheads for the BiSu technique and the extended Hamming code (H). The results are displayed for flit sizes of 32 and 64 bits. Different subflit sizes (S_{SF}) are explored for the BiSu method. In this study, we consider that the BiSu technique is enhanced with the header distribution on two flits. For each case, we discriminate the hardware overheads induced by the NIs and those induced by the routers and interconnections. For example, we can observe that in the unprotected architecture the NIs represent 12.7% of the global area and 15.5% of the global power consumption.

Through the results displayed in Figure 3.26, we first observe that the global overheads of the proposed method mainly depend of the subflit size, i.e. decreasing the subflit size increases the area and power overheads. For example, considering 32-bit flit size, the area and power overheads are respectively decreased from 25.1% to 17.5% and from 21.9% to 18.7% when the subflit size is increased from 4 to 8 bits. However, we note that the flit size has a small influence on the global hardware overheads since the hardware costs of the baseline increase with the flit size. Indeed, we saw in this figure that the area and power overheads increase slightly when the flit size increases from 32 to 64 bits. For instance, considering 4-bit subflits, the area and power overheads are respectively increased from 25.1% to 43.5% and from 21.9% to 31.9%. By comparing these results with the Hamming overheads, we observe that the BiSu technique can have higher, equal or lower hardware overheads according to the considered subflit size. For example, considering 64-bit flits with 4-bit subflits, the overheads of the BiSu technique are of 43.5% for the area overhead



(a) Area overhead.



(b) Power overhead.

Figure 3.26: Overhead comparison between the BiSu technique with header distribution on two flits and the Hamming code considering a 8×8 NoC.

and 31.9% for the power overhead, against 29.3% and 37.0% for the Hamming code. However, when the subflit size is increased to 8 bits, the area and the power overheads are respectively down to 27.5% and 23.0%. Finally, we note that the power consumption is particularly reduced by using the BiSu technique.

In Sections 3.3.1 and 3.3.2, we shown that the NoC performances are not impacted by the combinatorial blocks used to shuffle and merge the flits. Indeed, the critical path of the NoC with the BiSu technique stays approximately the same as that of the routing logic, contrary to the Hamming implementation. Moreover, the BiSu approach has a limited impact on the frequency since few additional cycles are required for its implementation. Concerning the computation of the registers, the algorithm is optimized to complete when no inversion is performed during the Bubble sort. Moreover, this algorithm is executed

NoC Parameters									
S_F	32				64				
S_{SF}	2	4	8	16	2	4	8	16	32
N_{SF}	16	8	4	2	32	16	8	4	2
Bus Size	32				64				
Register Updater Block									
Area (µm²)	3,234	1,481	770	285	8,115	3,541	1,798	1,109	469
Power (mW)	3.20	1.49	0.78	0.30	7.57	3.51	1.82	1.13	0.49
CP (ns)	0.84	0.79	0.81	0.60	0.98	0.96	0.80	0.99	0.97
Latency (ns)	370	120	44	17	1250	370	120	44	17
Register Updater Overheads for a 8×8 NoC CONNECT [228]									
Area (%)	98.9	45.3	23.6	8.7	248.2	108.3	55.0	33.9	14.3
Power (%)	112.1	52.3	27.4	10.6	265.6	123.3	63.7	39.7	17.3

Table 3.7: Hardware implementation costs for the register updater block.

only when a fault is detected limiting the impact on NoC performances. Finally, it has to be noticed that the lightweight configurations of the BiSu technique outperform the Hamming method on the data protection, while fine-grained mitigation configurations consume more hardware costs for a much higher efficiency.

3.3.4 Hardware Overheads of the Register Computing

In this part, we explore the hardware costs of the register updater block. This block is used to compute the shuffling and de-shuffling registers, as detailed in Section 3.1.3. For that, we consider that one register updater block is added at each couple of shuffler and de-shuffler blocks in the 8×8 NoC.

Table 3.7 shows the area cost, the power consumption, the critical path and the latency of the register updater block for flit sizes of 32 and 64 bits considering different subflit sizes. The additional overheads induced by this block are computed using the unprotected 8×8 NoC as baseline. Through these results, we observe that the implementation of the register updater blocks drastically increases the area and power overheads in the NoC. For instance, considering 32-bit flits with 4-bit subflits, the area and power overheads are respectively up to 83.3% and 86.2% against only 38.0% and 33.9% when the shuffling and de-shuffling registers are computed using the dedicated cores of the IPs. In this table, we can note that the hardware costs of the register updater block depend of the subflit size, i.e. decreasing the subflit size increases the hardware costs. For example, for 64-bit flits,

increasing the subflit size from 4 bits to 8 bits leads to overhead reduction for both area and power overheads from 108.3% to 55.0% and from 123.3% to 63.7%.

The times required for the execution of the algorithm, which computes the register values according to the error mask, are shown in Table 3.7. Based on these results, we can determine that the necessary latency to compute the register values depends on the number of subflits present inside the flits. For example, if the flit is composed of 16 subflits, then the necessary latency to update the registers is equal to 370 ns and to 120 ns when the flit is composed of 8 subflits.

3.4 Conclusion

In this chapter, we introduced the BiSu method which provides permanent-fault mitigation by transferring MSBs to LSBs to keep them safety during on-chip communications. This method can be adapted to any NoC architecture and can handle the difference between the data size and the NoC bus size. Moreover, fault impacts on both of the payloads and the headers are managed.

Along this chapter, we demonstrated that the BiSu technique is efficient to mitigate high fault densities during on-chip communications with limited hardware overheads compared to state-of-the-art techniques, such as Error-Correcting Codes (ECCs). We saw through experimentation that the accuracy of fault resilient applications can be maintained with the proposed approach allowing for acceptable results. In other hand, we demonstrated that the BiSu method has reasonable hardware costs, which are similar to an extended Hamming code according to the subflit size. In particular, the power consumption of the proposed method stays particularly low. By confronting the results concerning the efficiency and the hardware costs of the BiSu technique, we highlight the existence of a trade-off which is managed by the subflit size.

Finally, we saw that the registers which configure the shuffling and de-shuffling operations can be computed with dedicated hardware. We explored in this section a dense integration of the proposed mitigation technique by placing the shuffler and de-shuffler blocks in each input and output of routers. In the next chapter, we explore other distribution strategy to reduce even more the BiSu hardware costs by relaxing its efficiency. Moreover, as this approach reduces the number of shuffler and de-shuffler couples, it enables the computing of the shuffler and de-shuffler registers with the dedicated circuits.

Chapter 4

Region-based Bit-Shuffling Approach: Trading Hardware Cost and Fault Efficiency

In this chapter, we propose a region-based approach of the Bit-Shuffling (BiSu) technique, called Region-based Bit-Shuffling (R-BiSu) [P3]. This approach reduces hardware costs in terms of area and power consumption by relaxing the mitigation efficiency of the BiSu technique. First, the principle of the R-BiSu method is presented in Section 4.1. Then, an exhaustive evaluation of this approach is given in Section 4.2. Section 4.3 explores the hardware costs of the R-BiSu approach. After that, Section 4.4 highlights the existence of a Pareto front between the efficiency and the hardware costs of the proposed approach. Finally, Section 4.5 concludes this chapter.

4.1 Region-Based Bit-Shuffling (R-BiSu) Principle

In this section, we present the principle of the R-BiSu approach which reduces the hardware overheads by relaxing the BiSu efficiency. As the target domains and the assumptions are the same than those of the BiSu technique, which are presented in Section 3.1.1, they are not recalled in this chapter. Thus, first, we detail the concept of the proposed region-based approach by describing how the Network-on-Chip (NoC) regions are constructed. Then, we present how the concept of region can be applied for the BiSu technique. Finally, the computing of the error masks for the entire NoC regions is presented.

4.1.1 Definition of the NoC Regions

To apply the R-BiSu approach, the NoC needs to be split into regions. To make that, we define a reproducible pattern which can be used to split NoCs whatever their sizes. As

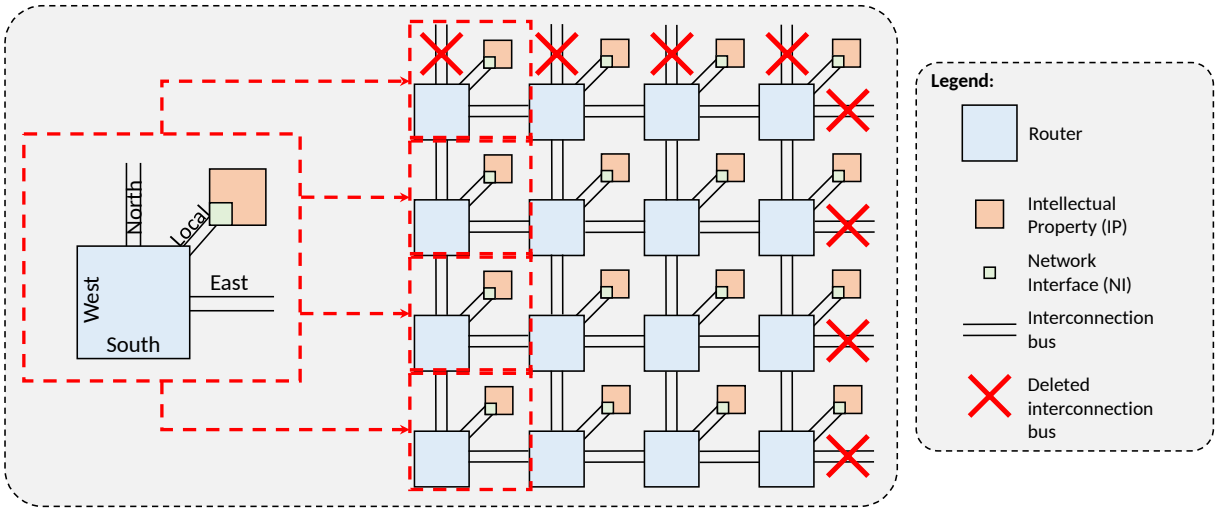


Figure 4.1: NoC decomposition into regular regions.

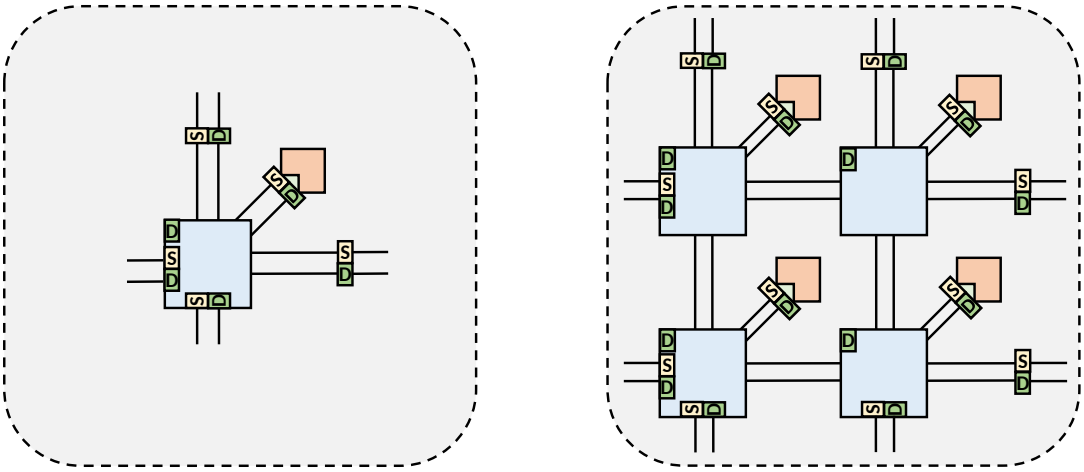
depicted in Figure 4.1, we define as reproducible pattern the router with its local, north and east interconnections. In this figure, we can note that NoCs can be easily constituted with this pattern by deleting the unused north and east interconnections of the routers which constitute the north and east NoC edges since they are not used.

Figure 4.2 illustrates regions of size 1 and 2. In this figure, we can observe that the regions are globally protected with the standard BiSu method, i.e. only region inputs and outputs are respectively enhanced with shuffler and de-shuffler blocks. In this work, we consider only regular square regions. However, the method can be extended for other region sizes and shapes. Thus, for sake of clarity, we name the regions according to their size. For instance, a region of size $S_{reg} \times S_{reg}$ is named region of size S_{reg} in this manuscript. We note that the region of size 1, displayed in Figure 4.2a corresponds to the reproducible pattern that we defined above since it is composed of only one router. Moreover, regions of larger sizes are simply constituted by assembling this pattern as for the NoC construction.

4.1.2 Region-based Bit-Shuffling approach (R-BiSu)

Contrary to the standard BiSu method, the R-BiSu approach focuses on coarse-grain mitigation. For that, the standard BiSu technique is applied at region scale instead of at each router and interconnection. In this way, the R-BiSu approach reduces the hardware costs by relaxing the efficiency of the BiSu technique since it is applied on a larger area.

Figure 4.3 illustrates the implementation of the R-BiSu approach in a 4×4 NoC, where the regions are defined by the red dotted squares. To visually compare the impact of the

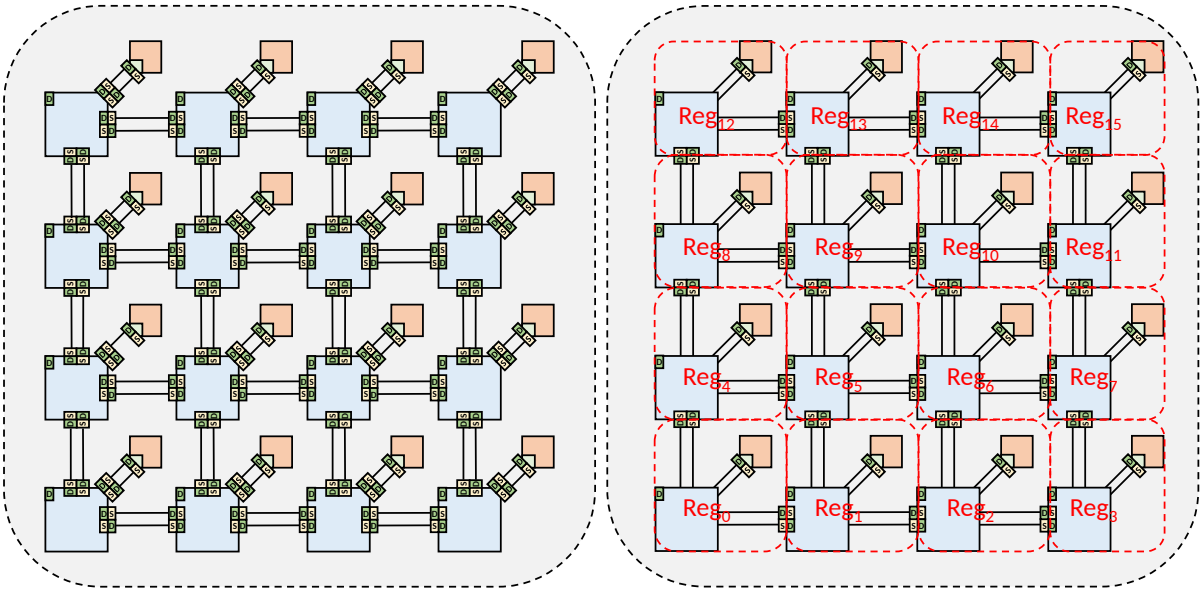


(a) Size 1 (b) Size 2

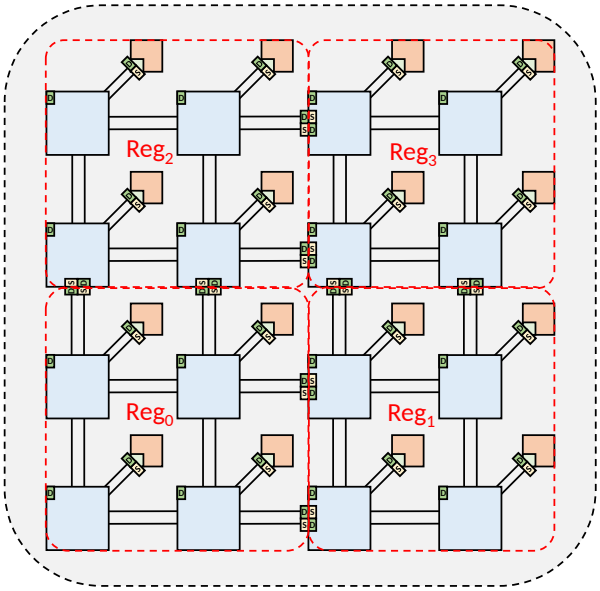
Figure 4.2: Enhanced NoC regions with the R-BiSu technique considering different region sizes.

R-BiSu approach on the shuffler and de-shuffler block density, we display in Figure 4.3a the basic BiSu method, where each router and each interconnection are protected with one couple of shuffler and de-shuffler blocks. For sake of clarity, we define the implementation of the basic BiSu technique as the size 0. For sake of simplicity, we define R-BiSu^n the implementation of the R-BiSu approach using region of size n. Figures 4.3b and 4.3c depict respectively R-BiSu^1 and R-BiSu^2 configurations. In these figures, we observe that, while the R-BiSu^0 requires 304 shuffler and de-shuffler blocks, the R-BiSu^1 and R-BiSu^2 require respectively only 144 and 80 blocks reducing the impact on the hardware overheads induced in the NoC. Note that, here, we count the number of shuffler and de-shuffler blocks and not the number of shuffler and de-shuffler block pairs.

To compute the required number of shuffler and de-shuffler blocks, named N_{exotic_region}, applying the R-BiSu in NoCs, we discriminate the different region types which are corner, X-border, Y-border and middle regions. The number of required blocks for each of these regions are respectively named N_{cor}, N_{x_bor}, N_{y_bor} and N_{mid} and are respectively given by Equations 4.1, 4.2, 4.3 and 4.4, where S_{noc_x} and S_{noc_y} represent respectively the X size and the Y size of the NoC, and where S_{reg_x} and S_{reg_y} defined respectively the X size and the Y size of the regions. Note that, in these equations, we do not consider the shuffler and de-shuffler blocks which are contained in Intellectual Properties (IPs) and Routing Controller (RC) since they are independent from the region size. So, they will be counted



(a) R-BiSu^0 (b) R-BiSu^1



(c) R-BiSu^2

Figure 4.3: NoC enhanced with the R-BiSu approach considering different region sizes.

later.

$$N_{cor} = 2 \times (S_{reg_x} + S_{reg_y}) \tag{4.1}$$

$$N_{x_bor} = 2 \times (S_{reg_x} + 2S_{reg_y}) \tag{4.2}$$

$$N_{y_bor} = 2 \times (2S_{reg_x} + S_{reg_y}) \tag{4.3}$$

$$N_{mid} = 4 \times (S_{reg_x} + S_{reg_y}) \tag{4.4}$$

Therefore, the number of required shuffler and de-shuffler blocks to apply the R-BiSu^n approach is given by Equation 4.5 where the shuffler and de-shuffler blocks which are contained in IPs and Routing Controller (RC) are counted.

$$\begin{aligned} N_{exotic_region} &= 4 \times N_{cor} \\ &+ 2 \times \frac{S_{noc_x} - 2S_{reg_x}}{S_{reg_x}} \times N_{x_bor} \\ &+ 2 \times \frac{S_{noc_y} - 2S_{reg_y}}{S_{reg_y}} \times N_{y_bor} \\ &+ \frac{S_{noc_x} - 2S_{reg_x}}{S_{reg_x}} \times \frac{S_{noc_y} - 2S_{reg_y}}{S_{reg_y}} \times N_{mid} \\ &+ 3 \times S_{noc_x} \times S_{noc_y} \end{aligned} \tag{4.5}$$

By computing the number of blocks for each of the region type, we obtain after simplification Equation 4.6. Through this equation, as expected, we observe that the number of added blocks, i.e. shuffler and de-shuffler blocks, which are necessary to apply the R-BiSu approach, decreases when the region size is increased.

$$N_{exotic_region} = 4 \times S_{noc_x} S_{noc_y} (\frac{1}{S_{reg_x}} + \frac{1}{S_{reg_y}}) + 3 \times S_{noc_x} S_{noc_y} - 4 \times (S_{noc_x} + S_{noc_y}) \tag{4.6}$$

In our case, the equations can be simplified since we consider only regular square regions. Thus, we have $S_{noc_x} = S_{noc_y} = S_{noc}$ and $S_{reg_x} = S_{reg_y} = S_{reg}$. By applying these simplifications, we obtain Equation 4.7. As previously, we observe through this equation

that the required block number decreases when the region size is increased.

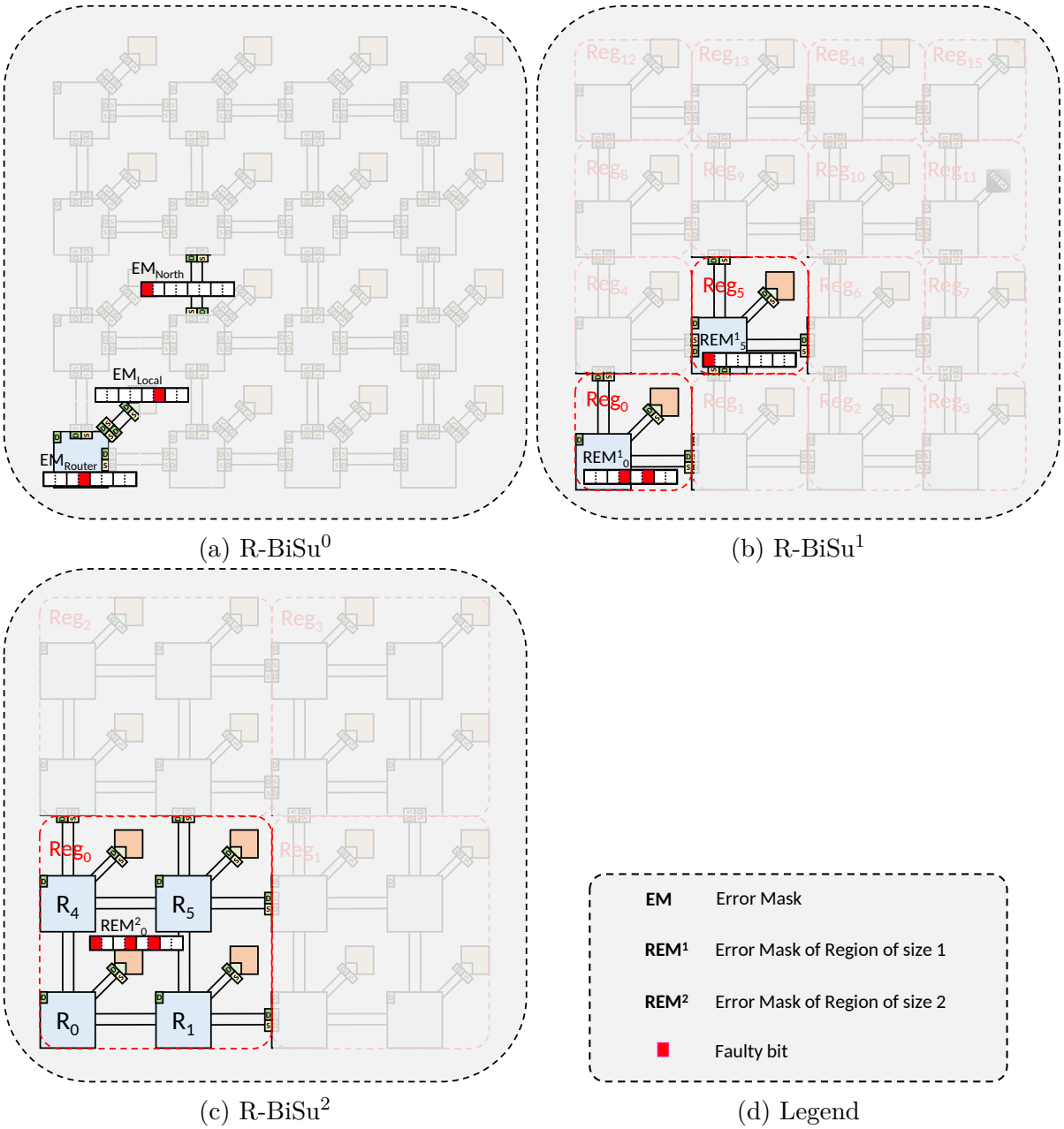
$$N_{square_region} = \frac{8 \times S_{noc}^2}{S_{reg}} + 3timesS_{noc}^2 - 8 \times S_{noc} \tag{4.7}$$

4.1.3 Region Error Mask (REM) Computation

To implement the R-BiSu technique, the registers of the shuffler and de-shuffler blocks need to be computed. All faults which are present inside a considered region need to be taken into account as they can be accumulated during the transmission of the packets. As mentioned in Chapter 2, the error masks can be provided for entire region with the help of detection methods available in the literature. However, these methods offer a lower fault coverage. Therefore, we propose a method to compute the Region Error Masks (REMs) by considering that the used detection method provides the error masks of each router and each interconnection of the NoC.

The R-BiSu method uses information regarding the faulty state of the region, given by the REM, to reduce as much as possible the impact of faults. This is achieved through a hierarchical method, which computes the error masks of $N \times N$ regions based on error masks of $(N-1) \times (N-1)$ regions. Figure 4.4 shows an example where a 4×4 NoC with 8-bit flit size and 2-bit subflit size is affected by three faults. These faults affect i) the bit number 4 of the router R_0, ii) the bit number 2 of the router R_0 local interconnection and iii) the bit number 7 of the router R_5 north interconnection. The associated error masks indicating faults (highlighted by red color) for the different region sizes are displayed in this figure.

Figure 4.5 depicts how the REMs are computed for regions of sizes 1 and 2 by using the available error masks of the size 0. First, as depicted by the blue squares, the REMs of size 1 (REM^1) are computed using OR operations between the error mask of the router and the error masks of the local, north and east interconnections. South and west interconnections are not considered since they are not included in the regular pattern defined in Section 4.1.1. As depicted in Figure 4.4b, the resulted error masks indicate the faulty bits of all the 1-size regions. For example, the REM_0^1 indicates that the bit number 2 and 4 are faulty, which is consistent with the error masks of the 0-size regions. As depicted by the green square in Figure 4.5, to compute the REMs of size 2 (REM^2), the same operations are performed based on the REMs of the 1-size regions (REM^1). As displayed in this figure, the obtained error mask indicates the faulty bits for all the



(a) R-BiSu^0 (b) R-BiSu^1

(c) R-BiSu^2 (d) Legend

Figure 4.4: Error mask values in a faulty NoC with the R-BiSu technique considering different region sizes.

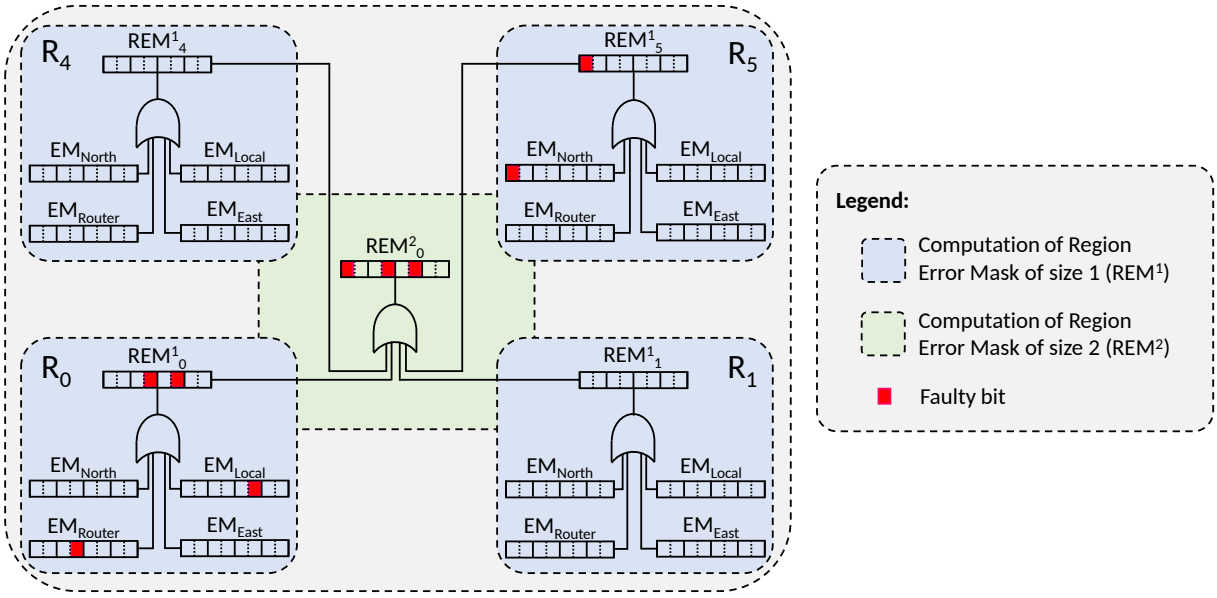


Figure 4.5: Region error mask computation.

considered 2-size regions.

By comparing the error masks of Figure 4.4, we note that the faults are individually addressed when the region size is low, which guarantee a high efficiency of the R-BiSu approach since the Least Significant Subflit (LSS) is able to tackle several faults. For instance, in Figure 4.5, if we consider region of size 0, only one subflit is faulty at the same time during one shuffling operation since the faults are scattered in several routers and interconnections throughout the region. In this case, the impact of the faults on the data is reduced as possible. However, when the region size is increased, the scattered faults are managed by a single shuffling operation, which impacts the efficiency of the R-BiSu approach since the faults can impact several subflits. Moreover, packets crossing large faulty regions can be shuffled and de-shuffling while they do not necessarily encounter faults. For example in Figure 4.4c, if the IP of the router R_1 sends packets toward the region Reg_1 or Reg_3 or toward the IP of the router R_5, they do not encounter the faults of the region Reg_0 but they are nevertheless shuffled and de-shuffled.

4.2 R-BiSu Efficiency Evaluation

In this section, we evaluate the impact of the region size on the efficiency of the R-BiSu approach when packets travel on a faulty NoC. For that, we first present the experimental

setup. Then, the impact of the region size on the R-BiSu efficiency is explored at the NoC scale.

4.2.1 Experimental Setup

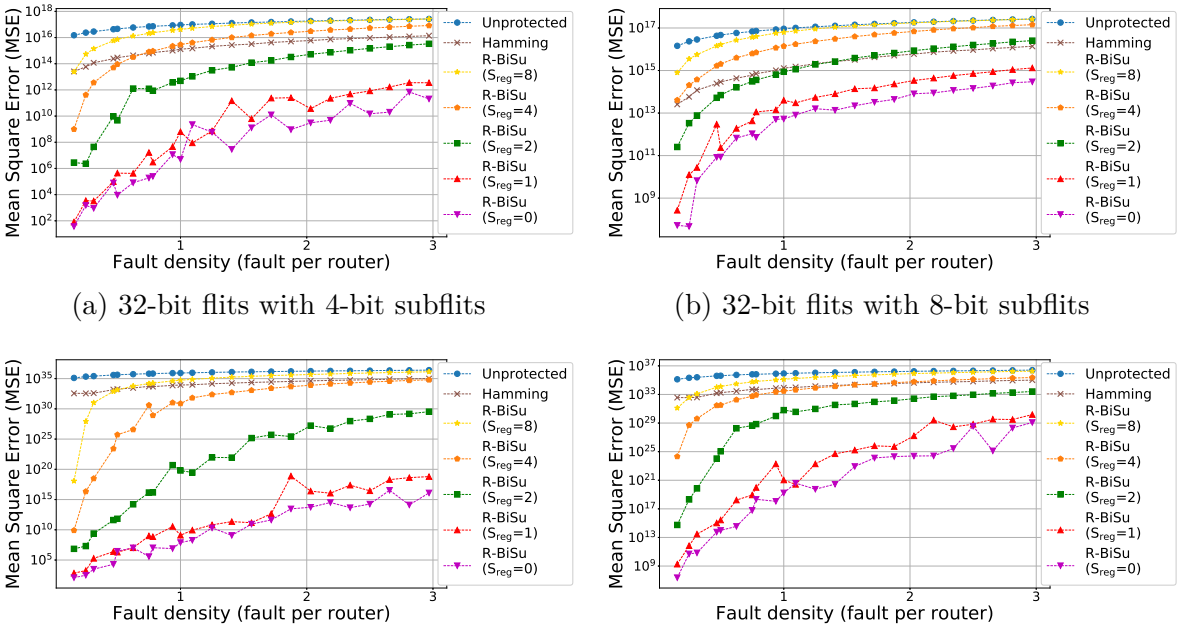
For the experiments, we consider a 8×8 NoC using the XY routing algorithm where the R-BiSu approach is implemented into square regions. Packets of 16 flits are injected according to the TORNADO injection model, where each IP sends a packet at each other IPs, which ensures a complete exploration of NoC paths. One header flit containing critical information is added to each packet and we consider that it contains only critical data whatever the flit size. Thus, the standard BiSu technique and the R-BiSu approach use the header distribution on two flits to mitigate faults which impact the headers. The experiments are led for flit sizes of 32 and 64 bits with subflit sizes of 4 and 8 bits. Single Hard Errors (SHEs) are randomly injected in the NoC datapath and they impact the data following the stuck-at fault model [25]. However, we consider that the injected faults have always an impact on the data by applying a bit-flip on the affected bits to consider the worst case avoiding the masking effects due to the data values. The results used to quantify the efficiency of the R-BiSu approach are computed based on $10,000$ fault injection sets using the Mean Square Error (MSE) and the Bit-Error Rate (BER) as metrics for the data payloads and the Correct Header Transmission Rate (CHTR) for the headers. All simulations are performed on an Ubuntu 18.04.5 LTS Linux distribution on an 48-cores Intel© Xeon(R) Silver 4214 CPU @ 2.20GHz with a Quadro RTX 5000/PCIe/SSE2 graphic card.

4.2.2 Efficiency Results at the NoC Scale

In this part, the impact of the region size on the efficiency of the R-BiSu approach is explored at the NoC scale. For that, we first consider the fault impacts on the data payloads, and then, the fault impacts on the header flits. Results are compared to an extended Hamming code which protects each router and each interconnection of the NoC.

4.2.2.1 Data-Payloads Mitigation

Figures 4.6 and 4.7 depict respectively the MSE and the BER of the packets which transit on the faulty NoC according to the fault density considering different region sizes.



(a) 32-bit flits with 4-bit subflits

(b) 32-bit flits with 8-bit subflits

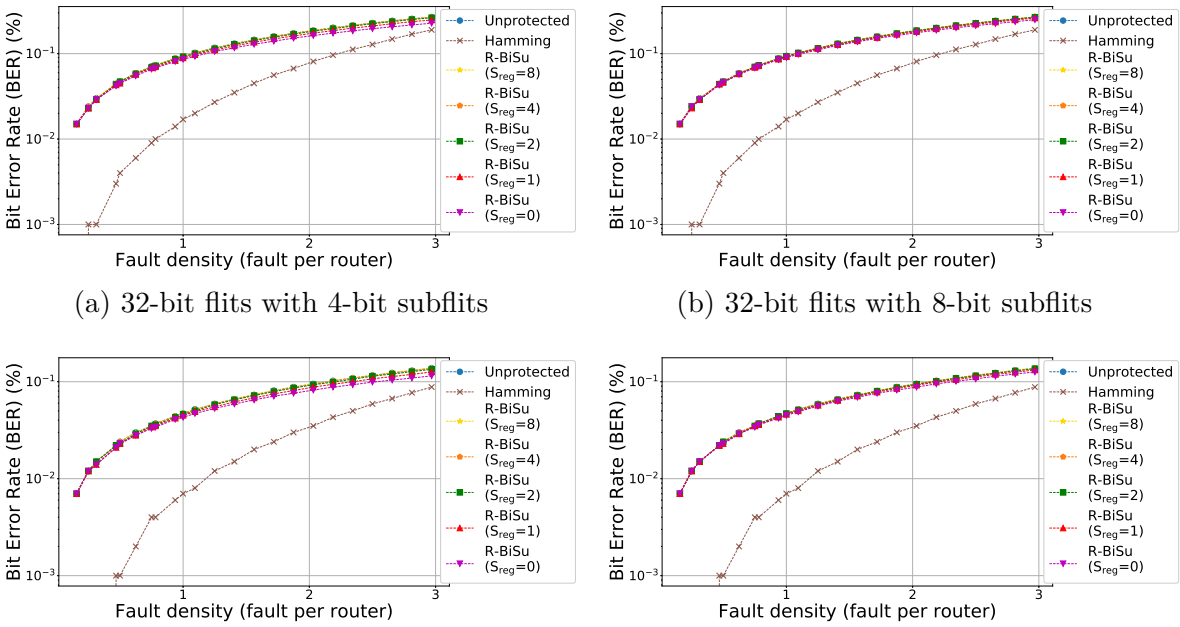
(c) 64-bit flits with 4-bit subflits

(d) 64-bit flits with 8-bit subflits

Figure 4.6: Experimental efficiency of the R-BiSu approach at the NoC scale using the MSE metric.

Figure 4.6 shows that the size of the region has an impact on the efficiency of the R-BiSu approach since the MSE increases with the region size. However, when the fault density is high, the MSE values reach a limit which seems to be equal to the results of unprotected NoC. In particular, we observe that the size of the region can be increased from 0 to 1 with a small impact on the MSE. For instance, considering 64-bit flits with 4-bit subflits (cf. Figure 4.6c) and a fault density equal to 1.00 fault per router, increasing the region size from 0 to 1 only increases the MSE from 7.80×10^7 to 1.25×10^9. In comparison, for the unprotected case, we obtain a MSE equal to 8.53×10^{35}. In addition, we observe that the Hamming code gives in general a higher MSE than the R-BiSu since, considering the previous example, we obtain a MSE equal to 8.69×10^{33}. However, we can note an exception when the flit size is equal to 32 bits with 8-bit subflits (cf. Figure 4.6b). In this case, the subflit number is too low to obtain efficient BiSu results in particular when the region sizes are greater than 1.

In Figure 4.7, we note that the BER slightly increases with the region size until reaching the same results than the unprotected case for large region sizes and high fault densities. However, this metric provides information on the number of faulty bits but it



(a) 32-bit flits with 4-bit subflits

(b) 32-bit flits with 8-bit subflits

(c) 64-bit flits with 4-bit subflits

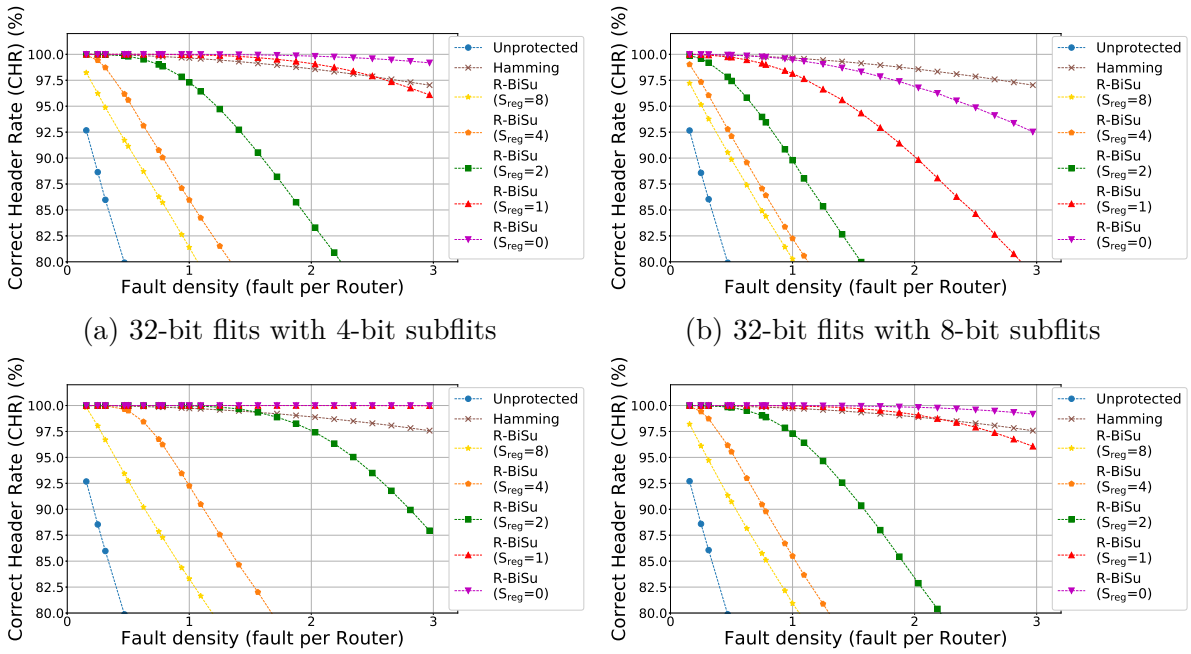
(d) 64-bit flits with 8-bit subflits

Figure 4.7: Experimental efficiency of the R-BiSu method at the NoC scale using the BER metric.

does not quantify the impact of these faulty bits on the flit information. Based on these results, two observations can be made.

First, the BER metric highlights the fact that faults are mitigated with the same Least Significant Bits (LSBs) when several shuffling operations are successively applied on the flit. As shown in Figure 4.7, this effect is more perceptible for high fault densities and small region sizes since more shuffling operations are made on the same flits. Thus, as expected, the method efficiency is higher when the region size is reduced. Moreover, it is important to note that obtaining the same BER than the unprotected case does not mean the method is inefficient since the fault impacts are deferred on the LSBs. For example, considering 64-bit flits with 4-bit subflits (cf. Figure 4.7c) and a fault density equal to 1.00 fault per router, we observe that the BER is increased by 2.20×10^{-4} when the region size is increased from the size 0 to the size 1.

Second, we see through Figure 4.7 that the BER is largely reduced using the extended Hamming code. For example, considering 32-bit flits with 4-bit subflits (cf. Figure 4.7a) and a fault density equal to 1.00 fault per router the BER is decreased by 1.50×10^{-2} with the Hamming code against only 1.21×10^{-3} with the standard BiSu technique compared



(a) 32-bit flits with 4-bit subflits

(b) 32-bit flits with 8-bit subflits

(c) 64-bit flits with 4-bit subflits

(d) 64-bit flits with 8-bit subflits

Figure 4.8: Experimental efficiency of the R-BiSu method at the NoC scale using the CHTR metric.

to the unprotected case. Indeed, Hamming code is able to correct all single faults which impact the packets transiting on the NoC. Thus, as we inject faults of size 1 bit in the NoC, most of them are corrected. However, this code becomes inefficient when two faults impact the flit at the same time, which degrade the flit information. This is why, despite a better BER, the MSE of the Hamming code is worse than those obtained with R-BiSu approach.

4.2.2.2 Header Mitigation

Figure 4.8 presents the packet CHTR, i.e. the percentage of headers which reach the destination IP without any fault on the critical bits, which transit on the faulty NoC according to the fault density. This metric is computed for different region sizes to evaluate their influence on the header transmission when the R-BiSu approach is applied.

In this figure, we observe that the correct transmission of the headers is maintained when the subflit number per flit stays sufficient. For example, considering flits of 32 bits segmented into subflits of 8 bits, we can see in Figure 4.8b that the R-BiSu approach is less efficient to mitigate the faults in the headers than the Hamming code whatever the region

size. Nevertheless, we note that on the three other cases the use of the R-BiSu approach with a region size equal to 1 offers better header protection. For instance, considering 32-bit flits with 4-bit subflits (cf. Figure 4.8a), the use of the R-BiSu approach with a region size equal to 1 ensures 99% of correct header transmissions for fault density of up to 2.03 fault per router. In comparison, the extended Hamming code can maintain the accuracy on the control bits above 99% only for fault density of up to 1.56. Moreover, we observe in Figure 4.8c that the gap between the two methods widens when we consider higher flit sizes.

However, we have to note that the headers do not contain any unused bits, as mentioned in the experimental setup. Thus, the results present the worst case which can be obtained with the different flit sizes. Indeed, as mentioned in Section 3.1.4, the headers often contain unused bits when the flit size is high, which has for effect to increase the efficiency of the proposed method concerning the header protection.

4.3 R-BiSu Hardware Evaluation

In this section, we study the hardware costs for the region size implementation of the R-BiSu approach, in terms of area and power consumption. The impacts on the critical path and the latency are not detailed again since the R-BiSu approach is a lightweight implementation version of the BiSu method. First, the experimental setup is presented. Then, the evolution of the number of shuffler and de-shuffler blocks according to the region size is studied. After that, we explore the hardware overheads induced by the R-BiSu approach. Finally, the hardware overheads are evaluated when the register computing is supported by dedicated circuits, called register updater blocks.

4.3.1 Experimental Setup

For the experiments, we consider a 8×8 NoC using the CONNECT router [228] hich is based on a 5-ports router with four virtual channels of 8-flit depth, a round-robin arbitration and a XY routing algorithm. The R-BiSu approach is implemented considering regular square regions and we consider the header distribution on two flits which correspond to the worst case. The results are synthesized on 28 nm FDSOI technology through High-Level Synthesis (HLS) tools of Mentor Graphic, targeting a clock frequency of 1 GHz. All syntheses are performed on the Fedora 28 Linux distribution with 8-cores

Approach	**BiSu** ($S_{reg} = 0$)	**R-BiSu** ($S_{reg} = 1$)	**R-BiSu** ($S_{reg} = 2$)	**R-BiSu** ($S_{reg} = 4$)	**R-BiSu** ($S_{reg} = 8$)
Block Number	1344	640	384	256	192

Table 4.1: Comparison of the required number of shuffler and de-shuffler blocks between the BiSu and the R-BiSu methods considering a 8×8 NoC.

Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz.

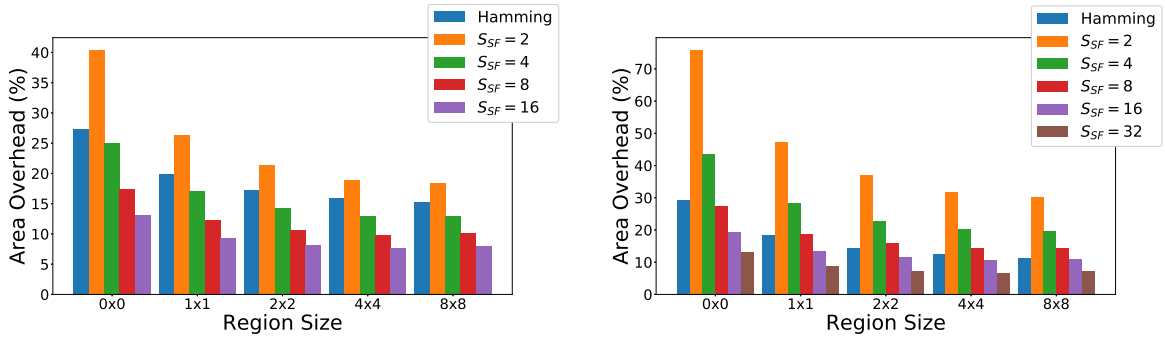
4.3.2 Shuffler/De-shuffler Block Number Comparison

In this part, we explore the evolution of the number of shuffler and de-shuffler blocks according to the region size. As the R-BiSu method targets the reduction of the hardware costs by reducing the implementation density of the BiSu technique, this number gives a first overview of the region size influence on the hardware costs. Indeed, with the R-BiSu method the number of shuffler and de-shuffler blocks and the number of merger and de-merger blocks stay constant in the Network Interfaces (NIs), nevertheless the number of shuffler and de-shuffler blocks is drastically reduced in the routers and in the interconnections of the NoC.

Table 4.1 displays the required number of shuffler and de-shuffler blocks according to the region size when the R-BiSu approach is implemented. These results are computed using Equation 4.7 of Section 4.1.2. They are compared to the required block number of the standard BiSu technique, i.e. region of size 0. In this table, we observe that the number of required blocks is drastically reduced when the region size is increased. For example, implementing R-BiSu^1 instead of R-BiSu^0 decreases the number of required blocks by 52%. And more, increasing the region size from 1 to 2 reduces the number of required blocks by 33%. Based on this, we can argue that the R-BiSu approach drastically reduces the hardware costs.

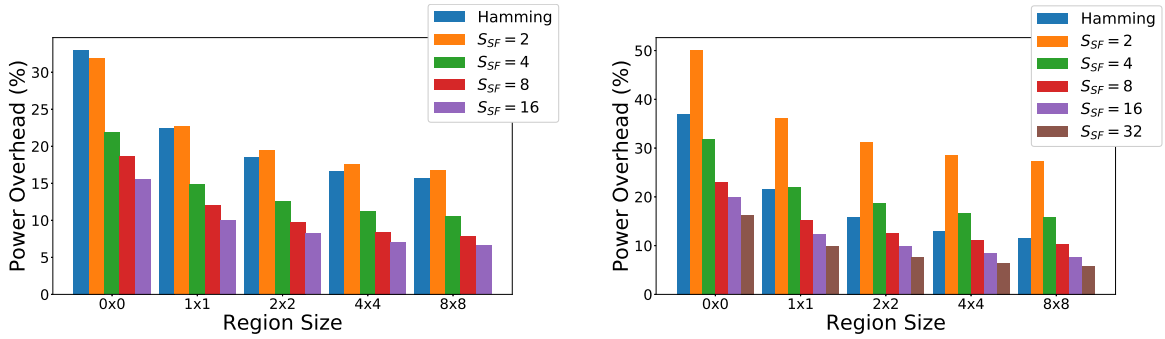
4.3.3 R-BiSu Hardware Results

In this part, we evaluate the influence of the region size on the hardware overheads induced by the R-BiSu approach considering as baseline the unprotected 8×8 CONNECT NoC. Figures 4.9 and 4.10 display respectively the area and power overheads according to the region size considering flit sizes of 32 and 64 bits with different subflit sizes. The results are compared to the Hamming case.



(a) 32-bit flits (b) 64-bit flits

Figure 4.9: Area overhead comparison for the R-BiSu method.



(a) 32-bit flits (b) 64-bit flits

Figure 4.10: Power overhead comparison for the R-BiSu method.

In Figure 4.9, we can observe that the region size influences the area overhead of the R-BiSu approach. Indeed, increasing the region size leads to area overhead reduction. For instance, increase the region size from 0 to 1 allows to decrease the area overhead from 25.1% to 17.1% considering 32-bit flits with 4-bit subflits and from 27.5% to 18.8% considering 64-bit flits with 8-bit subflits. Moreover, we observe in this figure that the area overhead of the R-BiSu approach stays higher than those induced by the Hamming code considering the same region size. However, we saw in Section 4.3.2 that the efficiency of the R-BiSu approach stays considerably greater than Hamming code efficiency when it is implemented for the protection of each router and each interconnection of the NoC. Based on this, we note that increasing the region size allows equal or inferior R-BiSu^n area overhead compared to the Hamming code.

The same observations can be made concerning the power overhead which is displayed in Figure 4.10. In this figure, we can note that the power overhead decreases when the region size increases. For instance, considering flit size of 32 bits with 4-bit subflits, the

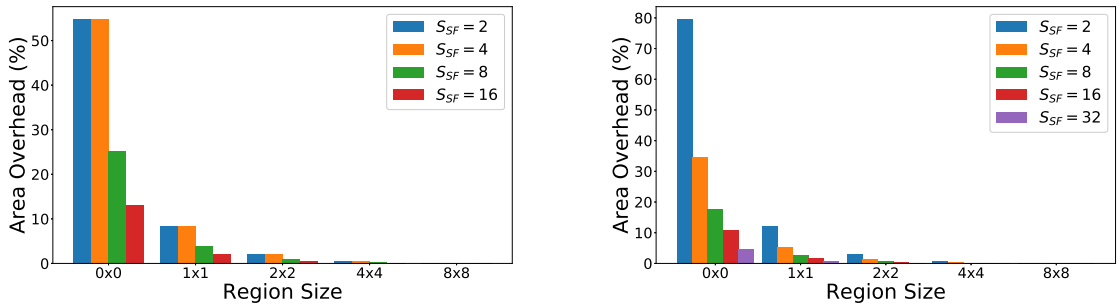
power overhead is reduced from 21.9% to 14.9% when the region size is increased from 0 to 1 and from 23.0% to 15.2% for flit size of 64 bits with 8-bit subflits. Moreover, we observe that increasing the region size of the R-BiSu approach significantly reduces the power overhead compared to Hamming code implemented with a region size equal to 0.

Through Figures 4.9 and 4.10, we can observe that the hardware overheads tend to converge toward a bound when the region size is increased. Indeed, as shown in Section 4.3.2, the R-BiSu approach permits to decrease the number of shuffler and de-shuffler blocks in the routers and in the interconnections. However, the number of merger and de-merger blocks added to the NIs for the flit organization are not affected by this approach, as the number of shuffler and de-shuffler blocks which are located to the input and output of the NIs. Thus, the constant number of hardware blocks leads to these bounds hardware results when the region size is increased. Moreover, we can observe that the values of these bounds seem to be equal to the hardware overhead of the NI part which is detailed in Section 3.3.3 for the region size 0.

4.3.4 Hardware Overheads of the Register Updater

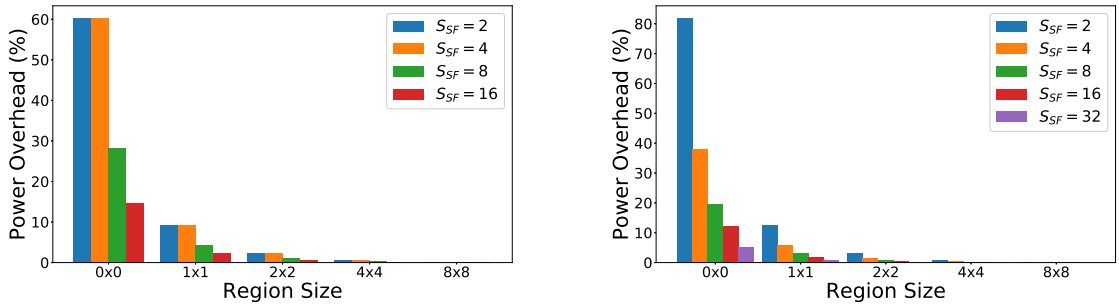
In Section 3.1.3, we proposed a way to compute the shuffler and de-shuffler registers with a dedicated circuit relaxing the pressure on the dedicated cores of the IPs. However, we saw in Section 3.3.4 that the induced area and power overheads were too high due to the high number of registers which have to be computed. As the register number is correlated to the implementation density, i.e. more shuffler and de-shuffler blocks means more registers, the use of dedicated circuits to compute these registers can be re-considered with the R-BiSu approach. Indeed, this latter allows to drastically reduce the register number with the high shuffler and de-shuffler block density. For that, we compute the supplementary overheads required by the register updater blocks for the computation of the shuffler and de-shuffler registers. Figures 4.11 and 4.12 display respectively the induced area and power overheads based on the unprotected 8×8 CONNECT NoC.

In these two figures, we observe that the R-BiSu approach ensures important area and power reductions compared to the standard BiSu technique. For instance, considering 64-bit flits with 4-bit subflits, the supplementary area and power overheads are respectively decreased from 34.8% and 38.0% to 5.3% and 5.8% compared to the standard BiSu technique when the region size is up to 2. Moreover, we can observe that the induced overheads can become negligible when the region size is increased to a size greater than 2. Based on these results, we note that the shuffler and de-shuffler registers needed for



(a) 32-bit flits (b) 64-bit flits

Figure 4.11: Area overhead of the register updater for the R-BiSu method.



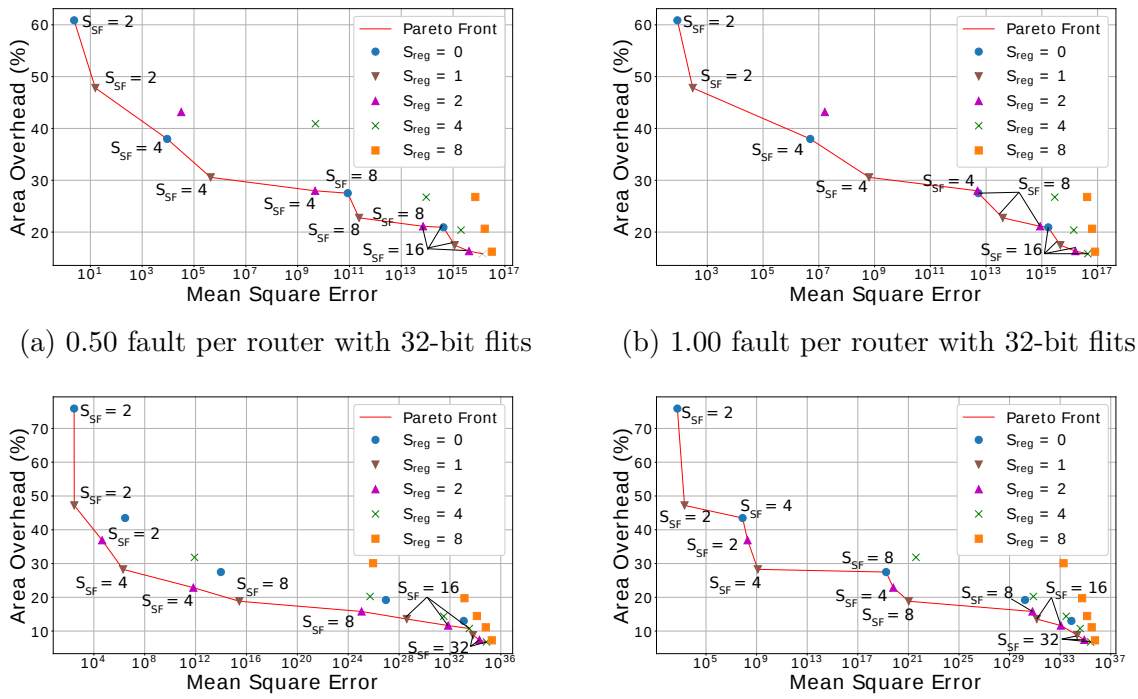
(a) 32-bit flits (b) 64-bit flits

Figure 4.12: Power overhead of the register updater for the R-BiSu method.

BiSu technique can be computed by dedicated circuits with negligible hardware overheads allowing for the pressure on the dedicated cores to be removed.

4.4 Efficiency Versus Hardware Cost Trade-off

In this section, we confront the results obtained concerning the efficiency and the hardware overheads of the R-BiSu approach. As mentioned in Section 4.2, the efficiency of the method is decreased when the region size increases. In addition, we showed in Section 4.3 that the hardware overheads are reduced when the size of the region is increased. Based on these observations, we can argue that a Pareto front exists between the hardware costs and the efficiency of the R-BiSu approach. First, we present the existing Pareto front between the efficiency and the area overhead. Then, the Pareto front between the efficiency and the power overhead is explored.



(a) 0.50 fault per router with 32-bit flits

(b) 1.00 fault per router with 32-bit flits

(c) 0.50 fault per router with 64-bit flits

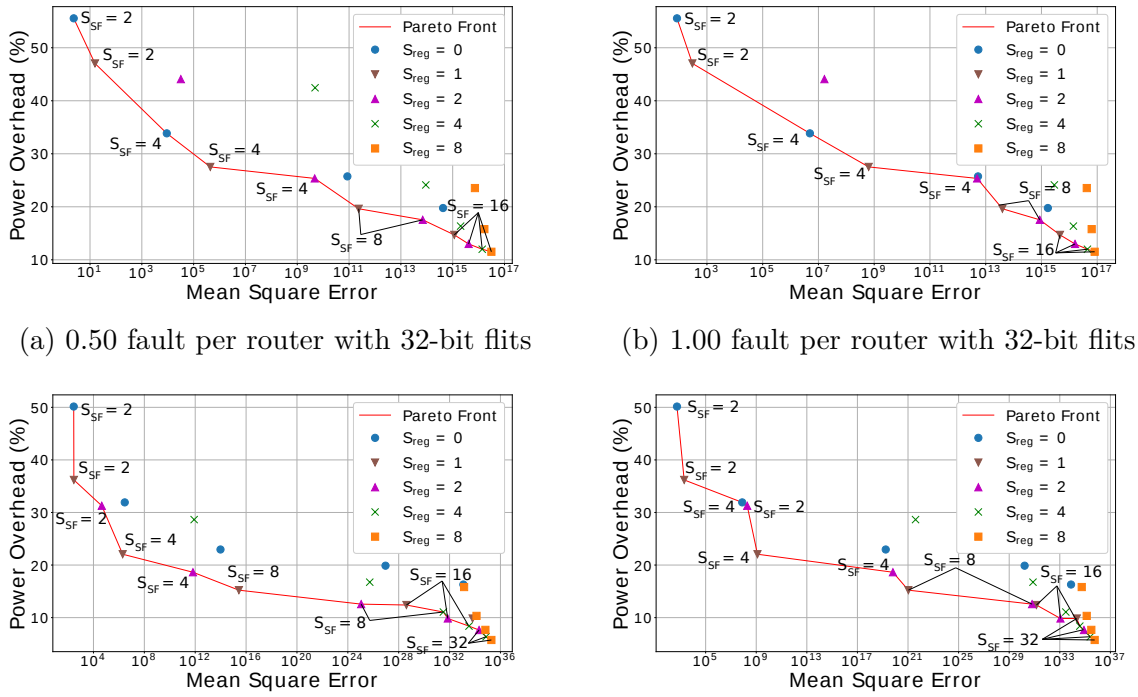
(d) 1.00 fault per router with 64-bit flits

Figure 4.13: Highlighting of the Pareto front between area costs and efficiency for the R-BiSu approach.

4.4.1 Area Overhead Versus Efficiency

Figure 4.13 plots the Pareto front between the area overhead and the efficiency of the R-BiSu approach for flit sizes of 32 and 64 bits varying the subflit size and the region size. Fault densities equal to 0.50 and 1.00 fault per router are considered. For sake of clarity, only the subflit size of the points which compose the Pareto front are detailed.

From the obtained results, we observe that the Pareto front is mainly composed of configurations with low region size, i.e. until region size equal to 2. Furthermore, we note that the region size can be increased until 2 with low impact on the efficiency reducing significantly the area overhead. Moreover, Figure 4.13 shows that increasing the subflit size reduces the area overhead with a higher impact on the efficiency. Thus, this figure shows that increasing either subflit size or the region size provides two viable solutions to reduce the area overhead by trading-off efficiency. Indeed, we observe that the Pareto front is composed of points obtained with both solutions. Finally, we can see that increasing the fault density has as effect efficiency reduction of the method for all subflit sizes and



(a) 0.50 fault per router with 32-bit flits

(b) 1.00 fault per router with 32-bit flits

(c) 0.50 fault per router with 64-bit flits

(d) 1.00 fault per router with 64-bit flits

Figure 4.14: Highlighting of the Pareto front between power consumption and efficiency for the R-BiSu approach.

region sizes. For example, considering 32-bit flits with a fault density equal to 0.50 fault per router (cf. Figure 4.13a), we note that the use of a region size equal to 1 with 2-bit subflits offers a high fault-tolerance efficiency at the price of a high area overhead. On the contrary, we observe that the use of a subflit size equal to 8 bits considering the same region size reduces significantly the area overhead at the price of a reduced fault-tolerance efficiency. Finally, we observe that the use of a region size equal to 1 with 4-bit subflits offers a good trade-off between the area overhead and the R-BiSu fault-tolerance efficiency.

4.4.2 Power Overhead Versus Efficiency

Figure 4.14 plots the Pareto front between the power overhead and the efficiency of the R-BiSu approach for flit sizes of 32 and 64 bits varying the subflit size and the region size. Fault densities equal to 0.50 and 1.00 fault per router are considered. For sake of clarity, only the subflit size of the points which compose the Pareto front are detailed.

In this figure, we observe that the Pareto front between the efficiency and the power overhead of the R-BiSu approach is affected by the region size and the subflit size in a similar way to the Pareto front between the efficiency and the area overhead. The Pareto front is mainly composed of configurations with a region size smaller or equal than 2 and reducing the region size or the subflit size are two viable solutions to reduce the power consumption when trading-off efficiency. Based on this, similar conclusions to the ones obtained when trading-off efficiency and area overhead can be made to characterize Pareto fronts of Figure 4.14. For example, considering 64-bit flits with a fault density equal to 1.00 fault per router (cf. Figure 4.14d), we observe that the use of a region size equal to 1 with 2-bit subflits offers a high fault-tolerance efficiency at the price of a high power overhead. On the contrary, we note that the use of a subflit size equal to 16 with the same region size reduces significantly the power overhead at the price of a reduced fault-tolerance efficiency. Finally, we can see that several configurations offer a reasonable trade-off between the power overhead and the fault-tolerance efficiency, such as a region size equal to 1 with 4-bit subflits.

4.5 Conclusion

In this chapter, we presented a region-based approach of the BiSu technique, called R-BiSu. This approach is an extension of the BiSu technique by considering the hardware overheads versus efficiency trade-off. Thus, the area and power overheads are reduced by relaxing the BiSu efficiency.

We observed through the efficiency and hardware evaluations that R-BiSu^1 and R-BiSu^2 offer a good trade-off between the efficiency and the hardware overheads of the method. Indeed, we saw that the hardware costs can be reduced with small impact on the fault mitigation efficiency. Moreover, we saw that the transmission of sensible flits, i.e. headers, is also managed by the BiSu method.

Finally, we showed that, contrary to the standard BiSu technique, the R-BiSu approach can support hardware computation of shuffler and de-shuffler registers by dedicated circuits. Indeed, as the number of registers which need to be computed is largely decreased when the region size increases, the hardware overheads induced by the register adapter blocks are drastically reduced, and more, they can become negligible for high region sizes. Moreover, the pressure induced on the dedicated IP cores is removed by using the R-BiSu approach.

Conclusion and Perspectives

Conclusion

Due to the increasing intrinsic failure rate in the electronic field, Network-on-Chips (NoCs) became more sensitive to faults affecting their functionalities. In particular, permanent faults are critical for NoCs since they affect them with no possibility of recovery contrary to transient faults. To address this issue, we saw that the literature provides many methods to detect, diagnose, correct and mitigate the occurring faults. However, these methods often induce high hardware costs in terms of area and power consumption and their efficiencies are drastically impacted, and may even become inoperative in presence of multiple permanent faults. As saw in Chapter 2), the approximate-communication field is a new domain offering many perspectives as, for example, fault tolerance to address this gap.

Therefore, we proposed as first contribution the Bit-Shuffling (BiSu) technique [P2, P1] for fault mitigation in NoC datapath. For that, our approach re-organizes the flits at the subflit scale to ensure that the faults only impact the Least Significant Bits (LSBs) keeping the Most Significant Bits (MSBs) safe. This technique is particularly efficient to protect routers and interconnections in NoC architectures and it allows to protect a flits which transit along a NoC path which is scattered with errors located on different bits. Furthermore, redundancy approach is presented to handle critical data, such as headers, by distributing critical information on two flits allowing to artificially increase the number of unused bits to enable the BiSu methods. In addition, hardware blocks are used in Network Interfaces (NIs) to manage the difference between the flit size and the data size keeping the proposed method efficient whatever the NoC architecture and the considered application. We demonstrated through exhaustive evaluations made at flit, NoC, and application levels that this contribution is able to manage high fault densities contrary to state-of-the-art methods, such as the extended Hamming code. Finally, we implement this contribution with 28 nm technology through High-Level Synthesis (HLS), and the results highlight low hardware overheads, and in particular, low power consumption. In addition, this method has small critical path and latency overheads limiting the impacts

on the NoC performances.

Following this work, we proposed as second contribution the Region-based Bit-Shuffling (R-BiSu) approach [P3] which consists in protecting large regions with the standard BiSu technique to reduce the hardware overheads of this latter at the cost of efficiency reduction. To achieve that, a hierarchical method is proposed to compute the error mask of a complete region. The exploration of the R-BiSu approach allowed to highlight the existing trade-offs between the efficiency and the hardware costs of the method. We saw that these trade-offs are influenced by the subflit size and the region size. Moreover, we noted through the obtained Pareto fronts that increasing the region size from 0 (standard BiSu) to 2 (R-BiSu^2), considering regular square regions, allows to drastically reduce the hardware costs with small impact on the method efficiency. In addition, this approach enables the computing of the register values by using a designed hardware block, called register updater block. In this way, the software algorithm executed on Intellectual Property (IP) core can be replaced by this dedicated block with negligible hardware overheads.

Perspectives

However, the R-BiSu method reaches its limits when packets contain only critical data, i.e. which cannot tolerate faults. In this case, applying the flit distribution on two flits (cf. Section 3.1.4) can drastically impact the NoC performances since the number of flits per packet is doubled. To tackle this issue, we propose to associate our method with region-based adaptive routing algorithms [134, 18, 229, 114] (cf. Section 2.3.1) which manage faulty regions by circumventing them. Thus, while region-based adaptive routing algorithms provide accurate paths for packets which cannot tolerate faults, the R-BiSu method forwards packets which can be approximated through faulty regions, exploiting the application fault resilience. Hence, critical packets are able to reach their destination without any errors. In addition, as only critical packets circumvent the faulty region, the congestion induced around this latter due to the adaptive routing algorithm is reduced limiting the NoC performance degradation. Moreover, isolated IPs, i.e. localized in the faulty region or in an isolated NoC part, stay reachable by using the flit distribution of the header on two flits. The distinction between the two paths provided by these two methods is depicted in Figure C.1. In this figure, we consider one packet transmission from the IP A towards the IP B using the XY routing algorithm. The R-BiSu approach and the adaptive routing algorithm are used to manage permanent faults considering regions

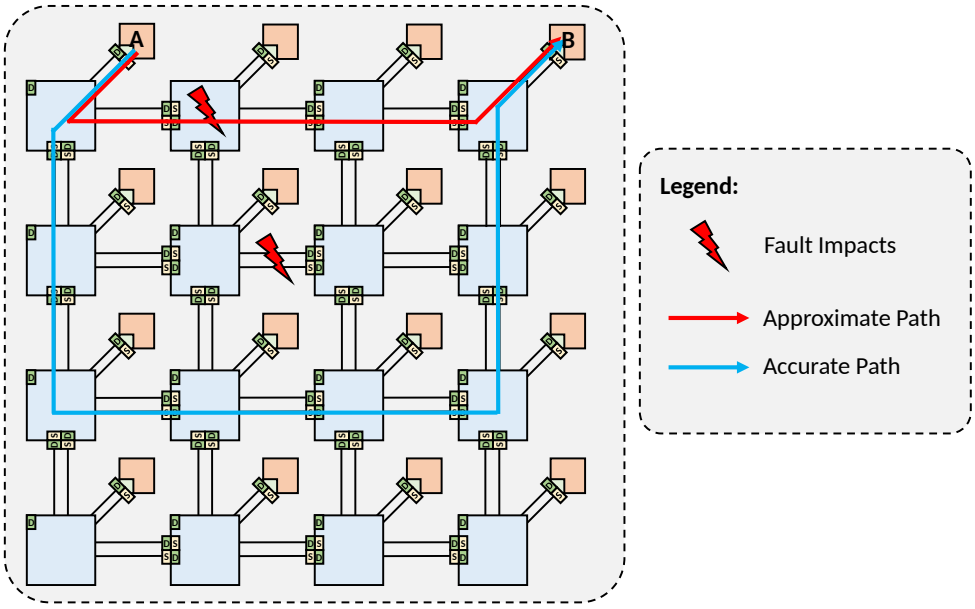


Figure C.1: R-BiSu associating with a region-based adaptive routing algorithm.

of size 1. While the R-BiSu technique provides an approximate path through the faulty region (cf. red path in Figure C.1), the adaptive routing algorithm ensures the correct transmission of the packet using an accurate path (cf. blue path in Figure C.1). Finally, the BiSu and R-BiSu techniques can be associated with other state-of-the-art methods to enhance the fault-tolerant efficiency and NoC performances.

Furthermore, in this manuscript we focused on electrical 2-D NoCs. However, the proposed methods can be extended to other NoC architectures. For instance, the BiSu method can be especially interesting in 3-D NoC architectures where the Through Silicon Vias (TSVs) are particularly sensible to aging effects such as electromigration [230, 231, 232].

In addition, the BiSu technique can be used to protect communications at different scales, i.e. from multi-processor-based on-chip to data center. In general, our approach proposes an efficient fault-tolerant solution to manage the permanent faults in electrical parallel interconnections, i.e. busses. Thus, it can be used as well for Application-Specific Integrated Circuit (ASIC) target, and for Field Programmable Gate Array (FPGA) target where long distance communications can be particularly sensible [233, 234].

Finally, as bit-shuffling is widely used in other fields, such as in cybersecurity [235, 236, 237] where data are secured during communications by shuffling them using an encryption key. Thus, our method may be used to encrypt the data using one or several shuffling

patterns during the transmissions. In this case, the encryption keys are the shuffling registers. In addition to this field, the BiSu technique can offer promising solution for other application fields.

Publications and Communications

Publications

[P1] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "Multiple Permanent Faults Mitigation Through Bit-Shuffling for Network-on-Chip Architecture," in *International Conference on Computer Design (ICCD)*, pp. 205–212, IEEE, Oct. 2020.

[P2] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "BiSuT: A NoC-Based Bit-Shuffling Technique for Multiple Permanent Faults Mitigation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–14, 2021.

[P3] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "A Region-Based Bit-Shuffling Approach Trading Hardware Cost and Fault Mitigation Efficiency," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 205–212, IEEE, Oct. 2021.

Communications

[C1] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "BiSu: A Low Cost Bit-Shuffling Technique for Permanent Fault Mitigation in NoC Architecture." Conférence Francophone d'Informatique en Parallélisme, Architecture et Système (COMPAS), July 2021.

[C2] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "BiSu: A NoC-Based Bit-Shuffling Technique For Multiple Permanent Faults Mitigation." Colloque National du GDR SOC^2, June 2021.

[C3] R. Mercier, C. Killian, A. Kritikakou, Y. Helen, and D. Chillet, "BiSu: A NoC-Based Bit-Shuffling Technique For Multiple Permanent Faults Mitigation." Séminaire sur la Tolérance Aux Fautes des Équipements Électroniques pour la Défense (STAFEED), June 2021.

Bibliography

[1] R. Schaller, "Moore's Law: Past, Present and Future," *IEEE Spectrum*, vol. 34, pp. 52–59, June 1997.

[2] T. Kuroda, "CMOS Design Challenges to Power Wall," in *International Conference on Microprocesses and Nanotechnology*, pp. 6–7, IEEE, Oct. 2001.

[3] D. Suggs, M. Subramony, and D. Bouvier, "The AMD Zen 2 Processor," *IEEE Micro*, vol. 40, pp. 45–52, March 2020.

[4] W. Dally and B. Towles, "Route Packets, Not Wires: On-chip Inteconnection Networks," in *Design Automation Conference (DAC)*, pp. 684–689, ACM, June 2001.

[5] D. Bertozzi, G. Dimitrakopoulos, J. Flich, and S. Sonntag, "The Fast Evolving Landscape of on-Chip Communication," *Design Automation for Embedded Systems*, vol. 19, pp. 59–76, Mar. 2015.

[6] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, pp. 70–78, Jan. 2002.

[7] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A Methodology for Design, Modeling, and Analysis of Networks-on-chip," in *International Symposium on Circuits and Systems (ISCAS)*, vol. 2, pp. 1778–1781, IEEE, May 2005.

[8] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel, "Network-on-Chip Programmable Platform in VersalTM ACAP Architecture," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 212–221, ACM/SIGDA, Feb. 2019.

[9] B. D. Dinechin and A. Graillat, "Feed-Forward Routing for the Wormhole Switching Network-on-Chip of the Kalray MPPA2 Processor," in *International Workshop on Network on Chip Architectures (NoCArc)*, ACM, Oct. 2017.

[10] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The Impact of Technology Scaling on Lifetime Reliability," in *International Conference on Dependable Systems and Networks (DSN)*, pp. 177–186, IEEE, June 2004.

[11] M. T. Bohr, "Logic Technology Scaling to Continue Moore's Law," in *Electron Devices Technology and Manufacturing Conference (EDTM)*, pp. 1–3, IEEE, Mar. 2018.

[12] "Space Product Assurance: Techniques for Radiation Effects Mitigation in AASIC and FPGAs Handbook," tech. rep., ESA Requirements and Standards Division, Sept. 2016.

[13] E. Dubrova, *Fault-Tolerant Design.* Fault-Tolerant Design, Springer, 2013.

[14] S. Kundu and S. Chattopadhyay, *Network-on-Chip: The Next Generation of System-on-Chip Integration.* Taylor & Francis, 2014.

[15] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for Fault Tolerance in Networks-on-Chip," *Computing Surveys (CSUR)*, vol. 46, pp. 1–38, July 2013.

[16] S. Werner, J. Navaridas, and M. Luján, "A Survey on Design Approaches to Circumvent Permanent Faults in Networks-on-Chip," *ACM Computing Surveys (CSUR)*, vol. 48, pp. 1–36, May 2016.

[17] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Self-Healing Hardware Systems: A Review," *Microelectronics Journal*, vol. 93, p. 104620, Nov. 2019.

[18] B. Fu, Y. Han, H. Li, and X. Li, "ZoneDefense: A Fault-Tolerant Routing for 2-D Meshes Without Virtual Channels," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 113–126, Jan. 2014.

[19] G. Liva, L. Gaudio, T. Ninacs, and T. Jerkovits, "Code Design for Short Blocks: A Survey," *Computing Research Repository (CoRR) - arXiv*, Oct. 2016.

[20] M. F. Reza and P. Ampadu, "Approximate Communication Strategies for Energy-Efficient and High Performance NoC: Opportunities and Challenges," in *Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 399–404, ACM, May 2019.

[21] A. B. Ahmed, D. Fujiki, H. Matsutani, M. Koibuchi, and H. Amano, "AxNoC: Low-power Approximate Network-on-chips Using Critical-path Isolation," in *International Symposium on Networks-on-Chip (NOCS)*, no. 6, pp. 1–8, IEEE, Oct. 2018.

[22] V. Y. Raparti, N. Kapadia, and S. Pasricha, "CHARM: A Checkpoint-Based Resource Management Framework for Reliable Multicore Computing in the Dark Silicon Era," in *International Conference on Computer Design (ICCD)*, pp. 201–208, IEEE, Oct. 2016.

[23] B. N. K. Reddy, M. H. Vasantha, and Y. B. Nithin Kumar, "A Gracefully Degrading and Energy-Efficient Fault Tolerant NoC Using Spare Core," in *Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 146–151, IEEE, July 2016.

[24] A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, J. Hu, and G. Chen, "A Reliable Routing Architecture and Algorithm for NoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, pp. 726–739, May 2012.

[25] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* Elsevier, Mar. 2004.

[26] D. Jung, S. Davidson, C. Zhao, D. Richmond, and M. Taylor, "Ruche Networks: Wire-Maximal, No-Fuss NoCs," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 1–8, IEEE/ACM, Sept. 2020.

[27] Y. Qiu, C. Xiao, L. Peng, J. Wang, Z. Kang, S. Li, and L. Wang, "A Novel Ring-based Small-World NoC for Neuromorphic Processor," in *International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 234–241, IEEE, July 2021.

[28] V. Rantala, T. Lehtonen, J. Plosila, *et al.*, *Network on Chip Routing Algorithms.* Citeseer, 2006.

[29] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, pp. 51–61, Nov. 2007.

[30] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC," in *Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 178–183, IEEE, Sep. 2005.

[31] B. Mohamed Bakhouya, "Evaluating the Energy Consumption and the Silicon Area of on-Chip Interconnect Architectures," *Journal of Systems Architecture*, vol. 55, no. 7, pp. 387–395, 2009.

[32] C. Fallin, C. Craik, and O. Mutlu, "CHIPPER: A Low-Complexity Bufferless Deflection Router," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 144–155, IEEE, Feb. 2011.

[33] J. Jose, B. Nayak, K. Kumar, and M. Mutyam, "DeBAR: Deflection Based Adaptive Router With Minimal Buffering," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1583–1588, IEEE, 2013.

[34] B. Daya, L. Peh, and A. Chandrakasan, "Towards High-Performance Bufferless NoCs with SCEPTER," *IEEE Computer Architecture Letters*, vol. 15, pp. 62–65, Jan. 2016.

[35] K. Jain, S. Singh, A. Majumder, and A. Mondai, "Problems Encountered in Various Arbitration Techniques Used in NOC Router: A Survey," in *International Conference*

on Electronic Design, Computer Networks Automated Verification (EDCAV), pp. 62–67, IEEE, Jan. 2015.

[36] M. Palesi and M. Daneshtalab, *Routing Algorithms in Networks-on-Chip.* Springer, Oct. 2014.

[37] S. Chawade, M. Gaikwad, and R. Patrikar, "Review of XY Routing Algorithm for Network-on-Chip Architecture," *International Journal of Computer Applications (IJCA)*, vol. 43, pp. 0975–8887, Apr. 2012.

[38] C. Glass and L. Ni, "The Turn Model for Adaptive Routing," *ACM SIGARCH Computer Architecture News*, vol. 20, pp. 278–287, Apr. 1992.

[39] G. Chiu, "The Odd-Even Turn Model for Adaptive Routing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 11, pp. 729–738, July 2000.

[40] É. Cota, A. de Morais Amory, and M. Lubaszewski, *Reliability, Availability and Serviceability of Networks-on-Chip.* Springer Science & Business Media, Sept. 2011.

[41] C. Killian, *Réseaux Embarqués sur Puce Reconfigurable Dynamiquement et Sûrs de Fonctionnement.* PhD thesis, Université de Lorraine, Dec. 2012.

[42] S. Tan, M. Tahoori, T. Kim, S. Wang, Z. Sun, and S. Kiamehr, *Long-Term Reliability of Nanometer VLSI Systems, Modeling, Analysis and Optimization.* Springer, 2019.

[43] P. Dodd, M. Shaneyfelt, J. Schwank, and J. Felix, "Current and Future Challenges in Radiation Effects on CMOS Electronics," *IEEE Transactions on Nuclear Science*, vol. 57, pp. 1747–1763, Aug. 2010.

[44] I. Koren and Z. Koren, "Defect Tolerance in VLSI Circuits: Techniques and Yield Analysis," *Proceedings of the IEEE*, vol. 86, pp. 1819–1838, Sept. 1998.

[45] R. Baumann, "Soft Errors in Advanced Semiconductor Devices-Part I: The Three Radiation Sources," *IEEE Transactions on Device and Materials Reliability*, vol. 1, pp. 17–22, Mar. 2001.

[46] R. Ecoffet, "Overview of In-Orbit Radiation Induced Spacecraft Anomalies," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 1791–1815, June 2013.

[47] X. Zhu, L. Massengill, C. Cirba, and H. Barnaby, "Charge Deposition Modeling of Thermal Neutron Products in Fast Submicron MOS Devices," *IEEE Transactions on Nuclear Science*, vol. 46, pp. 1378–1385, Dec. 1999.

[48] L. Mutuel, "Single Event Effects Mitigation Techniques Report," *Federal Aviation Administration*, vol. 15, p. 470, Feb. 2016.

[49] D. Fleetwood, "Total-Ionizing-Dose Effects, Border Traps, and 1/f Noise in Emerging MOS Technologies," *IEEE Transactions on Nuclear Science*, vol. 67, pp. 1216–1240, July 2020.

[50] D. Fleetwood, "Evolution of Total Ionizing Dose Effects in MOS Devices With Moore's Law Scaling," *IEEE Transactions on Nuclear Science*, vol. 65, pp. 1465–1481, Aug. 2018.

[51] J. Srour and J. Palko, "Displacement Damage Effects in Irradiated Semiconductor Devices," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 1740–1766, June 2013.

[52] C. Zhang, F. Jazaeri, A. Pezzotta, C. Bruschini, G. Borghello, F. Faccio, S. Mattiazzo, A. Baschirotto, and C. Enz, "Characterization of GigaRad Total Ionizing Dose and Annealing Effects on 28-nm Bulk MOSFETs," *IEEE Transactions on Nuclear Science*, vol. 64, pp. 2639–2647, Oct. 2017.

[53] J. Schwank, M. Shaneyfelt, J. Felix, P. Dodd, J. Baggio, V. Ferlet-Cavrois, P. Paillet, G. Hash, R. Flores, L. Massengill, and E. Blackmore, "Effects of Total Dose Irradiation on Single-Event Upset Hardness," *IEEE Transactions on Nuclear Science*, vol. 53, pp. 1772–1778, Aug. 2006.

[54] V. Ferlet-Cavrois, L. Massengill, and P. Gouker, "Single Event Transients in Digital CMOS–A Review," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 1767–1790, June 2013.

[55] A. Sogoyan, A. Chumakov, A. Smolin, A. Ulanova, and A. Boruzdina, "A Simple Analytical Model of Single-Event Upsets in Bulk CMOS," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 400, pp. 31–36, June 2017.

[56] N. Yusop, A. Nordin, M. Azim Khairi, and N. F. Hasbullah, "The Impact of Scaling on Single Event Upset in 6T and 12T SRAMs from 130 to 22 nm CMOS Technology," *Radiation Effects and Defects in Solids*, vol. 173, pp. 1090–1104, Jan. 2018.

[57] Y. Bentoutou and M. Djaifri, "Observations of Single-Event Upsets and Multiple-Bit Upsets in Random Access Memories on-Board the Algerian Satellite," in *Nuclear Science Symposium Conference Record*, pp. 2568–2570, IEEE, Oct. 2008.

[58] R. Koga, S. Penzin, K. Crawford, and W. Crain, "Single Event Functional Interrupt (SEFI) Sensitivity in Microcircuits," in *European Conference on Radiation and its Effects on Components and Systems (RADECS)*, pp. 311–318, IEEE, Sept. 1997.

[59] F. W. Sexton, "Destructive Single-Event Effects in Semiconductor Devices and ICs," *IEEE Transactions on Nuclear Science*, vol. 50, pp. 603–621, June 2003.

[60] L. Artola, G. Hubert, and T. Rousselin, "Single-Event Latchup Modeling Based on Coupled Physical and Electrical Transient Simulations in CMOS Technology," *IEEE Transactions on Nuclear Science*, vol. 61, pp. 3543–3549, Dec. 2014.

[61] A. H. Johnston, "The Influence of VLSI Technology Evolution on Radiation-Induced Latchup in Space Systems," *IEEE Transactions on Nuclear Science*, vol. 43, pp. 505–521, Apr. 1996.

[62] A. Ochoa, F. Sexton, T. Wrobel, G. Hash, and R. Sokel, "Snap-Back: A Stable Regenerative Breakdown Mode of MOS Devices," *IEEE Transactions on Nuclear Science*, vol. 30, pp. 4127–4130, Dec. 1983.

[63] J. Titus, "An Updated Perspective of Single Event Gate Rupture and Single Event Burnout in Power MOSFETs," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 1912–1928, June 2013.

[64] F. Sexton, D. Fleetwood, M. Shaneyfelt, P. Dodd, and G. Hash, "Single Event Gate Rupture in Thin Gate Oxides," *IEEE Transactions on Nuclear Science*, vol. 44, pp. 2345–2352, Dec. 1997.

[65] W. Roesch, "Using a New Bathtub Curve to Correlate Quality and Reliability," *Microelectronics Reliability*, vol. 52, pp. 2864–2869, Dec. 2012.

[66] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky, "BulletProof: A Defect-Tolerant CMP Switch Architecture," in *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 5–16, IEEE, Feb. 2006.

[67] C. Hu, L. Gignac, G. Lian, C. Cabral, K. Motoyama, H. Shobha, J. Demarest, Y. Ostrovski, C. Breslin, M. Ali, J. Benedict, P. McLaughlin, J. Ni, and X. Liu, "Mechanisms of Electromigration Damage in Cu Interconnects," in *International Electron Devices Meeting (IEDM)*, pp. 5.2.1–5.2.4, IEEE, Dec. 2018.

[68] D. Rossi, V. Tenentes, S. Yang, S. Khursheed, and B. Al-Hashimi, "Aging Benefits in Nanometer CMOS Designs," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, pp. 324–328, Mar. 2017.

[69] C. Guerin, V. Huard, and A. Bravaix, "The Energy-Driven Hot-Carrier Degradation Modes of nMOSFETs," *IEEE Transactions on Device and Materials Reliability*, vol. 7, pp. 225–235, June 2007.

[70] R. Moazzami, J. C. Lee, and C. Hu, "Temperature Acceleration of Time-Dependent Dielectric Breakdown," *IEEE Transactions on Electron Devices*, vol. 36, pp. 2462–2465, Nov. 1989.

[71] K. Kim, "Analysis of Electromigration in Nanoscale CMOS Circuits," *Journal of the Korea Industrial Information Systems Research*, vol. 18, pp. 19–24, Feb. 2013.

[72] J. G. Massey, "NBTI: What we Know and What we Need to Know - A Tutorial Addressing the Current Understanding and Challenges for the Future," in *International Integrated Reliability Workshop (IIRW)*, pp. 199–211, IEEE, Oct. 2004.

[73] H. Yang, S. Yang, W. Hwang, and C. Chuang, "Impacts of NBTI/PBTI on Timing Control Circuits and Degradation Tolerant Design in Nanoscale CMOS SRAM," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, pp. 1239–1251, June 2011.

[74] A. Houadef and B. Djezzar, "HCI Degradation of LOCOS-based LDMOS Transistor Fabricated by 1 μm CMOS Process," in *International Conference on Electrical Engineering (ICEE)*, pp. 1–6, IEEE, Sept. 2020.

[75] Y. Kim and K. Kim, "The Impact of TDDB Failure on Nanoscale CMOS Digital Circuits," *Journal of the Korea Industrial Information Systems Research*, vol. 17, pp. 27–34, June 2012.

[76] A. Vassighi, O. Semenov, M. Sachdev, A. Keshavarzi, and C. Hawkins, "CMOS IC Technology Scaling and its Impact on Burn-in," *IEEE Transactions on Device and Materials Reliability*, vol. 4, pp. 208–221, June 2004.

[77] J. Wang, M. Ebrahimi, L. Huang, A. Jantsch, and G. Li, "Design of Fault-Tolerant and Reliable Networks-on-Chip," in *Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 545–550, IEEE, July 2015.

[78] T. Sedighi, P. Phillips, and P. Foote, "Model-based Intermittent Fault Detection," *Procedia CIRP*, vol. 11, pp. 68–73, 2013.

[79] B. Aghaei, A. Khademzadeh, M. Reshadi, and K. Badie, "Link Testing: A Survey of Current Trends in Network on Chip," *Journal of Electronic Testing (JETTA)*, vol. 33, pp. 209–225, Apr. 2017.

[80] A. Louri, J. Collet, and A. Karanth, "Limit of Hardware Solutions for Self-Protecting Fault-Tolerant NoCs," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, pp. 1–16, Jan 2019.

[81] B. Fu and P. Ampadu, *Error Control for Network-on-Chip Links.* Springer Science & Business Media, Oct. 2011.

[82] Q. Yu and P. Ampadu, *Transient and Permanent Error Control For Networks-on-Chip.* Springer Sci. & Business Media, 2011.

[83] S. Jain and S. Chouhan, "Cyclic Redundancy Codes: Study and Implementation," *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, vol. 4, pp. 213–217, Apr. 2014.

[84] L. Xie, K. Mei, and Y. Li, "REPAIR: A Reliable Partial-Redundancy-Based Router in NoC," in *International Conference on Networking, Architecture and Storage (NAS)*, pp. 173–177, IEEE, July 2013.

[85] V. Fochi, E. Wächter, A. Erichsen, A. M. Amory, and F. G. Moraes, "An Integrated Method for Implementing Online Fault Detection in NoC-Based MPSoCs," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1562–1565, IEEE, May 2015.

[86] E. Wächter, F. Barreto, V. Fochi, A. M. Amory, and F. G. Moraes, "A Layered Approach for Fault Tolerant NoC-Based MPSoCs – Special Session: Dependable MPSoCs," in *Latin-American Test Symposium (LATS)*, pp. 189–194, IEEE, Apr. 2016.

[87] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri, "Dynamic Error Mitigation in NoCs Using Intelligent Prediction Techniques," in *International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE/ACM, Oct. 2016.

[88] A. Sánchez-Macián, P. Reviriego, and J. A. Maestro, "Hamming SEC-DAED and Extended Hamming SEC-DED-TAED Codes Through Selective Shortening and Bit Placement," *IEEE Transactions on Device and Materials Reliability*, vol. 14, pp. 574–576, Mar. 2014.

[89] F. Chiaraluce and R. Garello, "Extended Hamming Product Codes Analytical Performance Evaluation for Low Error Rate Applications," *IEEE Transactions on Wireless Communications (TWC)*, vol. 3, pp. 2353–2361, Nov. 2004.

[90] S. Murali, D. Atienza, L. Benini, and G. De Micheli, "A Multi-Path Routing Strategy with Guaranteed in-Order Packet Delivery and Fault-Tolerance for Networks on Chip," in *Design Automation Conference (DAC)*, pp. 845–848, ACM/IEEE, July 2006.

[91] A. Dutta and N. A. Touba, "Reliable Network-on-Chip Using a Low Cost Unequal Error Protection Code," in *International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, pp. 3–11, IEEE, Sept. 2007.

[92] T. Majumder, M. Suri, and V. Shekhar, "NoC Router Using STT-MRAM Based Hybrid Buffers with Error Correction and Limited Flit Retransmission," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 2305–2308, IEEE, May 2015.

[93] Q. Yu and P. Ampadu, "Adaptive Error Control for NoC Switch-to-Switch Links in a Variable Noise Environment," in *International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, pp. 352–360, IEEE, Oct. 2008.

[94] S. Kanakala, K. Ashok Kumar, and P. Dananjayan, "High Reliability NoC switch using Modified Hamming Code with Transient Faults," in *International Conference on System, Computation, Automation and Networking (ICSCAN)*, pp. 1–5, IEEE, July 2018.

[95] L. Huang, X. Lin, J. Wang, and Q. Li, "A Low Latency Fault Tolerant Transmission Mechanism for Network-on-Chip," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, IEEE, May 2017.

[96] S. Shamshiri, A. Ghofrani, and K. Cheng, "End-to-End Error Correction and Online Diagnosis for on-Chip Networks," in *International Test Conference (ITC)*, pp. 1–10, IEEE, Sept. 2011.

[97] H. Yathiraj and M. R. Hiremath, "Implementation of BCH Code (n, k) Encoder and Decoder for Multiple Error Correction Control," *International Journal of Computer Science and Mobile Applications (IJCSMA)*, vol. 2, pp. 45–54, May 2014.

[98] A. Helmy and A. R. Alameldeen, "Redundancy and ECC Mechanisms to Improve Energy Efficiency of on-Die Interconnects," in *International Conference on Energy Aware Computing (ICEAC)*, pp. 1–6, IEEE, Dec. 2012.

[99] A. A. Salem, M. A. A. E. Ghany, and K. Hofmann, "Performability Measurement of Coding Algorithms for Network on Chip," in *International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 691–694, IEEE, Dec. 2013.

[100] D. Rossi, N. Timoncini, M. Spica, and C. Metra, "Error Correcting Code Analysis for Cache Memory High Reliability and Performance," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, Mar. 2011.

[101] M. Sim and Y. Zhuang, "Design of Two Interleaved Error Detection and Corrections using Hsiao Code and CRC," in *Industrial Electronics Society (IES)*, pp. 2261–2266, IEEE, Oct. 2020.

[102] Y. Ueng, C. Yang, K. Wang, and C. Chen, "A Multimode Shuffled Iterative Decoder Architecture for High-Rate RS-LDPC Codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, pp. 2790–2803, Oct. 2010.

[103] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Transactions on Communications (TCOM)*, vol. 44, pp. 1261–1271, Oct. 1996.

[104] F. Silva, W. Magalhães, J. Silveira, J. M. Ferreira, P. Magalhães, O. A. Lima, and C. Marcon, "Evaluation of Multiple Bit Upset Tolerant Codes for NoCs Buffering," in *Latin American Symposium on Circuits Systems (LASCAS)*, pp. 1–4, IEEE, Feb. 2017.

[105] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing Transient and Permanent Faults in NoC With Efficient Fault-Tolerant Deflection Router," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, pp. 1053–1066, June 2013.

[106] M. Ibrahim, N.K.Baloch, S. Anjum, Y. Zikria, and S. Kim, "An Energy Efficient and Low Overhead Fault Mitigation Technique for Internet of Thing Edge Devices Reliable on-Chip Communication," *Software: Practice and Experience*, Feb. 2020.

[107] S. Rajagopal, M. Vinodhini, and N. S. Murty, "Multi-Bit Error Correction Coding with Crosstalk Avoidance Using Parity Sharing Technique for NoC," in *International Symposium on Smart Electronic Systems (iSES)*, pp. 249–254, IEEE, Dec. 2018.

[108] M. Gul, M. Chouikha, and M. Wade, "Joint Crosstalk Aware Burst Error Fault Tolerance Mechanism for Reliable on-Chip Communication," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, pp. 889–896, Dec. 2017.

[109] B. Wang, J. Xie, Z. Mao, and Q. Wang, "Multiple Continuous Error Correct Code for High Performance Network-on-Chip," in *Asia Pacific Conference on Postgraduate Research in Microelectronics Electronics*, pp. 98–101, IEEE, Oct. 2011.

[110] K. Wang, A. Louri, A. Karanth, and R. Bunescu, "High-Performance, Energy-Efficient, Fault-Tolerant Network-on-Chip Design Using Reinforcement Learning," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1166–1171, IEEE, Mar. 2019.

[111] B. Anitha and B. Jeevidha, "Low Overhead Decimal Matrix Code with Dynamic Network on Chip Against Multiple Cell Upsets," in *International Conference on Inno-*

vations in Information, Embedded and Communication Systems (ICIIECS), pp. 1–6, IEEE, Mar. 2015.

[112] X. Chen, Z. Lu, Y. Lei, Y. Wang, and S. Chen, "Multi-Bit Transient Fault Control For NoC Links Using 2D Fault Coding Method," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 1–8, IEEE/ACM, Aug. 2016.

[113] K. A. Satya Sai Lakshmi, A. M. Keerthi, K. M. Sri, and M. Vinodhini, "Code With Crosstalk Avoidance and Error Correction for Network on Chip Interconnects," in *International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 75–79, IEEE, June 2020.

[114] C. Killian, C. Tanougast, and A. Dandache, "Hybrid Fault Detection for Adaptive NoC," *IEEE Embedded Systems Letters (ESL)*, vol. 5, pp. 69–72, Dec. 2013.

[115] Q. Yu and P. Ampadu, "Dual-Layer Adaptive Error Control for Network-on-Chip Links," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, pp. 1304–1317, July 2012.

[116] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of Error Recovery Schemes for Networks on Chips," *IEEE Design & Test of Computers*, vol. 22, pp. 434–442, Sep. 2005.

[117] J. Wang, L. Huang, Q. Li, G. Li, and A. Jantsch, "Optimizing the Location of ECC Protection in Network-on-Chip," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 1–10, IEEE, Oct. 2016.

[118] C. Yuan, L. Huang, J. Wang, and Q. Li, "Micro-Architecture Design for Low Overhead Fault Tolerant Network-on-Chip," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, May 2018.

[119] J. Hou and M. Radetzki, "A Methodology to Compute Long-Term Fault Resilience of NoCs Under Fault-Tolerant Routing Algorithms," in *Forum for Specification and Design Languages (FDL)*, pp. 1–7, IEEE, Sept. 2019.

[120] L. Wang, X. Wang, and T. Mak, "Adaptive Routing Algorithms for Lifetime Reliability Optimization in Network-on-Chip," *IEEE Transactions on Computers*, vol. 65, pp. 2896–2902, Sept. 2016.

[121] Y. Zhang, X. Hong, Z. Chen, Z. Peng, and J. Jiang, "A Deterministic-Path Routing Algorithm for Tolerating Many Faults on Very-Large-Scale Network-on-Chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, pp. 1–26, Oct. 2020.

[122] Z. Chen, Y. Zhang, Z. Peng, and J. Jiang, "A Deterministic-Path Routing Algorithm for Tolerating Many Faults on Wafer-Level NoC," in *Des. Automat. Test in Europe Conf. Exhib. (DATE)*, pp. 1337–1342, Mar. 2019.

[123] S. Priya, S. Agarwal, and H. K. Kapoor, "Fault Tolerance in Network on Chip Using Bypass Path Establishing Packets," in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*, pp. 457–458, IEEE, Jan. 2018.

[124] A. Jain, V. Laxmi, M. Tripathi, M. S. Gaur, and R. Bishnoi, "S2DIO: An Extended Scalable 2D Mesh Network-on-Chip Routing Reconfiguration for Efficient Bypass of Link Failures," *The Journal of Supercomputing*, vol. 75, pp. 6855–6881, June 2019.

[125] J. Cano, J. Flich, J. Duato, M. Coppola, and R. Locatelli, "Efficient Routing Implementation in Complex Systems-on-Chip Designs," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 1–8, ACM, May 2011.

[126] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 25–32, IEEE, May 2010.

[127] J. Khichar, S. Choudhary, and R. Mahar, "Fault Tolerant Dynamic XY-YX Routing Algorithm for Network-on-Chip Architecture," in *International Conference on Intelligent Computing and Control (I2C2)*, pp. 1–6, IEEE, June 2017.

[128] T. Wehbe and X. Wang, "Secure and Dependable NoC-Connected Systems on an FPGA Chip," *IEEE Transactions on Reliability*, vol. 65, pp. 1852–1863, Dec. 2016.

[129] S. Y. Jiang, G. Luo, Y. Liu, S. S. Jiang, and X. T. Li, "Fault-Tolerant Routing Algorithm Simulation and Hardware Verification of NoC," *IEEE Transactions on Applied Superconductivity (TAS)*, vol. 24, pp. 1–5, Oct. 2014.

[130] A. Charif, N. Zergainoh, and M. Nicolaidis, "A New Approach to Deadlock-Free Fully Adaptive Routing for High-Performance Fault-Tolerant NoCs," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 121–126, IEEE, Sept. 2016.

[131] S. Moriam and G. P. Fettweis, "Fault Tolerant Deadlock-Free Adaptive Routing Algorithms for Hexagonal Networks-on-Chip," in *Euromicro Conference on Digital System Design (DSD)*, pp. 131–137, IEEE, Aug. 2016.

[132] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Fault Tolerant Algorithms for Network-on-Chip Interconnect," in *Computer Society Annual SympEosium on VLSI (ISVLSI)*, pp. 46–51, IEEE, Feb. 2004.

[133] T. Dumitras and R. Marculescu, "On-Chip Stochastic Communication [SoC Applications]," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 790–795, IEEE, Mar. 2003.

[134] R. Fernandes, C. Marcon, R. Cataldo, and J. Sepúlveda, "Using Smart Routing for Secure and Dependable NoC-Based MPSoCs," *IEEE/ACM Transactions on Networking (TON)*, vol. 28, pp. 1158–1171, June 2020.

[135] V. Catania, A. Mineo, S. Monteleone, and D. Patti, "A First Effort for a Distributed Segment-Based Approach on Self-Assembled Nano Networks," in *International Workshop on Network on Chip Architectures (NoCArc)*, pp. 59–64, ACM, Dec. 2013.

[136] A. Mejia, J. Flich, J. Duato, S. A. Reinemo, and T. Skeie, "Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm for Meshes and Tori," in *International Parallel Distributed Processing Symposium (IPDPS)*, pp. 1–10, IEEE, Apr. 2006.

[137] Y. Chen, E. Chang, H. Hsin, K. Chen, and A. Wu, "Path-Diversity-Aware Fault-Tolerant Routing Algorithm for Network-on-Chip Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 28, pp. 838–849, Mar. 2017.

[138] H. Zhao, N. Bagherzadeh, and J. Wu, "A General Fault-Tolerant Minimal Routing for Mesh Architectures," *IEEE Transactions on Computers*, vol. 66, pp. 1240–1246, July 2017.

[139] J. Liu, J. Harkin, Y. Li, and L. Maguire, "Low Cost Fault-Tolerant Routing Algorithm for Networks-on-Chip," *Microprocessors and Microsystems (MICPRO)*, vol. 39, pp. 358–372, Aug. 2015.

[140] R. Parikh and V. Bertacco, "UDIREC: Unified Diagnosis and Reconfiguration for Frugal Bypass of NoC Faults," in *International Symposium on Microarchitecture (MICRO)*, pp. 148–159, IEEE/ACM, Dec. 2013.

[141] A. Kohler and M. Radetzki, "Fault-Tolerant Architecture and Deflection Routing for Degradable NoC Switches," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 22–31, ACM/IEEE, May 2009.

[142] D. Xiang, K. Chakrabarty, and H. Fujiwara, "Fault-Tolerant Unicast-Based Multicast for Reliable Network-on-Chip Testing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, pp. 1–23, Dec. 2018.

[143] W. Zong, X. Wang, and T. Mak, "On Multicast for Dynamic and Irregular On-Chip Networks Using Dynamic Programming Method," in *International Workshop on Network on Chip Architectures (NoCArc)*, pp. 17–22, ACM, Dec. 2013.

[144] A. Charif, N. Zergainoh, and M. Nicolaidis, "Addressing transient routing errors in fault-tolerant networks-on-chips," in *2016 21th IEEE European Test Symposium (ETS)*, pp. 1–6, May 2016.

[145] C. Killian, C. Tanougast, F. Monteiro, and A. Dandache, "Smart Reliable Network-on-Chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 242–255, Feb. 2014.

[146] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi, "Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model," in *International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, pp. 21–29, IEEE, Sept. 2007.

[147] J. Duato, "A Theory of Fault-Tolerant Routing in Wormhole Networks," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 8, pp. 790–802, Aug. 1997.

[148] J. Han, J. Gao, P. Jonker, Y. Qi, and J. A. B. Fortes, "Toward Hardware-Redundant, Fault-Tolerant Logic for Nanoelectronics," *IEEE Design & Test of Computers*, vol. 22, pp. 328–339, July 2005.

[149] R. E. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability," *IBM Journal of Research and Development*, vol. 6, pp. 200–209, Apr. 1962.

[150] C. Li, M. Yang, and P. Ampadu, "An Energy-Efficient NoC Router with Adaptive Fault-Tolerance Using Channel Slicing and On-Demand TMR," *IEEE Transactions on Emerging Topics in Computing (TETC)*, vol. 6, pp. 538–550, Oct. 2018.

[151] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, and R. Reis, "Analyzing Area and Performance Penalty of Protecting Different Digital Modules with Hamming Code and Triple Modular Redundancy," in *Symposium on Integrated Circuits and Systems Design*, pp. 95–100, IEEE, Sept. 2002.

[152] I. Wali, A. Virazel, A. Bosio, L. Dilillo, and P. Girard, "An Effective Hybrid Fault-Tolerant Architecture for Pipelined Cores," in *European Test Symposium (ETS)*, pp. 1–6, IEEE, May 2015.

[153] P. Balasubramanian and R. T. Naayagi, "Redundant Logic Insertion and Fault Tolerance Improvement in Combinational Circuits," in *International Conference on Circuits, Systems and Simulation (ICCSS)*, pp. 6–13, IEEE, July 2017.

[154] A. Mukherjee and A. S. Dhar, "Triple Transistor Based Triple Modular Redundancy With Embedded Voter Circuit," *Microelectronics Journal*, vol. 87, pp. 101 – 109, May 2019.

[155] J. Han, E. Leung, L. Liu, and F. Lombardi, "A Fault-Tolerant Technique Using Quadded Logic and Quadded Transistors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, pp. 1562–1566, Aug. 2015.

[156] E. Suvorova, Y. Sheynin, and N. Matveeva, "Reconfigurable NoC Development with Fault Mitigation," in *Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, pp. 335–344, IEEE, Apr. 2016.

[157] S. Shamshiri and K. Cheng, "Yield and Cost Analysis of a Reliable NoC," in *VLSI Test Symposium (VTS)*, pp. 173–178, IEEE, May 2009.

[158] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu, "Self-Adaptive System for Addressing Permanent Errors in On-Chip Interconnects," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 527–540, Apr. 2010.

[159] F. Refan, H. Alemzadeh, S. Safari, P. Prinetto, and Z. Navabi, "Reliability in Application Specific Mesh-Based NoC Architectures," in *International On-Line Testing Symposium (IOLTS)*, pp. 207–212, IEEE, July 2008.

[160] H. S. Kia and C. Ababei, "Improving Fault Tolerance of Network-on-Chip Links Via Minimal Redundancy and Reconfiguration," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 363–368, IEEE, Nov. 2011.

[161] Y. Ren, L. Liu, S. Yin, J. Han, Q. Wu, and S. Wei, "A Fault Tolerant NoC Architecture Using Quad-Spare Mesh Topology and Dynamic Reconfiguration," *Journal of Systems Architecture (JSA)*, vol. 59, pp. 482–491, Aug. 2013.

[162] N. Chatterjee, S. Chattopadhyay, and K. Manna, "A Spare Router Based Reliable Network-on-Chip Design," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1957–1960, IEEE, June 2014.

[163] Y. Chang, C. A. Gong, and C. Chiu, "Fault-Tolerant Mesh-Based NoC With Router-Level Redundancy," *Journal of Signal Processing Systems*, vol. 92, pp. 345–355, Sept. 2019.

[164] H. J. Mohammed, W. N. Flayyih, and F. Z. Rokhani, "Tolerating Permanent Faults in the Input Port of the Network on Chip Router," *Journal of Low Power Electronics and Applications*, vol. 9, pp. 1–11, Feb. 2019.

[165] N. K. Machal, B. T. Repalle, S. Geedipally, R. Rios, and M. Rahman, "A New Paradigm for Fault-Tolerant Computing with Interconnect Crosstalks," in *International Conference on Rebooting Computing (ICRC)*, pp. 1–6, IEEE, Nov. 2018.

[166] J. Collet, A. Louri, V. Bhat, and P. Poluri, "ROBUST: A New Self-Healing Fault-Tolerant NoC Router," in *International Workshop on Network on Chip Architectures (NoCArc)*, pp. 11–16, ACM, Dec. 2011.

[167] M. Koibuchi, H. Matsutani, H. Amano, and T. Pinkston, "A Lightweight Fault-Tolerant Mechanism for Network-on-Chip," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 13–22, ACM/IEEE, Apr. 2008.

[168] M. Ebrahimi, M. Daneshtalab, J. Plosila, and H. Tenhunen, "Minimal-Path Fault-Tolerant Approach Using Connection-Retaining Structure in Networks-on-Chip," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 1–8, IEEE/ACM, Apr. 2013.

[169] H. S. Castro and O. A. de Lima, "A Fault Tolerant NoC Architecture Based Upon External Router Backup Paths," in *International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, IEEE, June 2013.

[170] A. Ahmed and A. Abdallah, "Adaptive Fault-Tolerant Architecture and Routing Algorithm for Reliable Many-Core 3D-NoC Systems," *Journal of Parallel and Distributed Computing*, vol. 93-94, pp. 30–43, July 2016.

[171] J. Heisswolf, S. Friederich, L. Masing, A. Weichslgartner, M. A. Zaib, C. Stein, M. Duden, J. Teich, A. Herkersdorf, and J. Becker, "A Novel NoC-Architecture for Fault Tolerance and Power Saving," in *International Conference on Architecture of Computing Systems (ARCS)*, pp. 1–8, VDE VERLAG, Apr. 2016.

[172] E. W. Wächter, V. Fochi, F. Barreto, A. M. Amory, and F. G. Moraes, "A Hierarchical and Distributed Fault Tolerant Proposal for NoC-Based MPSoCs," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, pp. 524–537, Oct. 2018.

[173] M. Ali, M. Welzl, and S. Hessler, "A Fault Tolerant Mechanism for Handling Permanent and Transient Failures in a Network on Chip," in *International Conference on Information Technology (OCIT)*, pp. 1027–1032, IEEE, Apr. 2007.

[174] K. Janson, R. Pihlak, S. P. Azad, B. Niazmand, G. Jervan, and J. Raik, "Handling of SETs on NoC Links by Exploitation of Inherent Redundancy in Circular Input Buffers," in *Baltic Electronics Conference (BEC)*, pp. 1–4, IEEE, Oct. 2018.

[175] K. Janson, R. Pihlak, S. P. Azad, B. Niazmand, G. Jervan, and J. Raik, "AWAIT: An Ultra-Lightweight Soft-Error Mitigation Mechanism for Network-on-Chip Links," in *International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pp. 1–6, IEEE, July 2018.

[176] C. Liu, L. Zhang, Y. Han, and X. Li, "A Resilient on-Chip Router Design Through Data Path Salvaging," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 437–442, IEEE, Jan. 2011.

[177] C. Chen, Y. Fu, and S. Cotofana, "Towards Maximum Utilization of Remained Bandwidth in Defected NoC Links," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, pp. 285–298, Feb. 2017.

[178] M. Braga, É. Cota, F. L. Kastensmidt, and M. Lubaszewski, "Efficiently Using Data Splitting and Retransmission to Tolerate Faults in Networks-on-Chip Interconnects," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 4101–4104, IEEE, May 2010.

[179] P. Poluri and A. Louri, "A Soft Error Tolerant Network-on-Chip Router Pipeline for Multi-Core Systems," *IEEE Computer Architecture Letters (CAL)*, vol. 14, pp. 107–110, July 2015.

[180] S. Tosun, V. B. Ajabshir, O. Mercanoglu, and O. Ozturk, "Fault-Tolerant Topology Generation Method for Application-Specific Network-on-Chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, pp. 1495–1508, Sept. 2015.

[181] Z. Li, J. Huang, Q. Xu, and S. Chen, "Integer Linear Programming Based Fault-Tolerant Topology Synthesis for Application-Specific NoC," in *International Conference on ASIC (ASICON)*, pp. 96–99, IEEE, Oct. 2017.

[182] W. Fan,, J. He, Z. Han, P. Li, and R. Wang, "Reconfigurable Fault-Tolerance Mapping of Ternary N-Cubes Onto Chips," *Concurrency and Computation: Practice and Experience*, vol. 32, Jan. 2020.

[183] C. Angeli and A. Chatzinikolaou, "On-Line Fault Detection Techniques for Technical Systems: A Survey," *International Journal on Computational Science & Applications (IJCSA)*, vol. 1, no. 1, pp. 12–30, 2004.

[184] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Flexible Self-Healing Router for Reliable and High-Performance Network-on-Chips Architecture," in *International System-on-Chip Conference (SOCC)*, pp. 152–157, IEEE, Sept. 2018.

[185] A. Dalirsani and H. Wunderlich, "Functional Diagnosis for Graceful Degradation of NoC Switches," in *Asian Test Symposium (ATS)*, pp. 246–251, IEEE, Nov. 2016.

[186] S. Babaei, M. Mansouri, B. Aghaei, and A. Khadem-Zadeh, "Online-Structural Testing of Routers in Network on Chip," *World Applied Sciences Journal (WASJ)*, vol. 14, no. 9, pp. 1374–1383, 2011.

[187] B. Bhowmik, S. Biswas, J. K. Deka, and B. B. Bhattacharya, "A Low-Cost Test Solution for Reliable Communication in Networks-on-Chip," *Journal of Electronic Testing*, vol. 35, pp. 215–243, Apr. 2019.

[188] B. Aghaei, A. Khademzadeh, K. Badie, M. Reshadi, and S. Sarhangi, "OSDTM: An Offline-Structural Distributed Test Mechanism for Data Links in NoC," *International Journal of Information & Communication Technology Research (IJICTR)*, vol. 10, pp. 1–12, Jan. 2018.

[189] C. Marcon, A. Amory, T. Webber, T. Volpato, and L. B. Poehls, "Phoenix NoC: A Distributed Fault Tolerant Architecture," in *International Conference on Computer Design (ICCD)*, pp. 7–12, IEEE, Oct. 2013.

[190] B. Bhowmik, J. K. Deka, and S. Biswas, "Reliability Monitoring in a Smart NoC Component," in *International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, IEEE, Nov. 2020.

[191] B. Aghaei, A. Khademzadeh, M. Reshadi, and K. Badie, "A New BIST-Based Test Approach with the Fault Location Capability for Communication Channels in Network-on-Chip," *Journal of Electronic Testing*, vol. 33, pp. 501–513, June 2017.

[192] B. Bhowmik, J. K. Deka, S. Biswas, and B. B. Bhattacharya, "On-Line Detection and Diagnosis of Stuck-at Faults in Channels of NoC-Based Systems," in *International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 004567–004572, IEEE, Oct. 2016.

[193] J. Zhan, L. Huang, J. Wang, M. Ebrahimi, and Q. Li, "Online Path-Based Test Method for Network-on-Chip," in *International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, May 2019.

[194] B. Bhowmik, J. K. Deka, S. Biswas, and B. B. Bhattacharya, "Performance-Aware Test Scheduling for Diagnosing Coexistent Channel Faults in Topology-Agnostic Networks-on-Chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, pp. 1–29, Jan. 2019.

[195] G. Ganzer and M. D. Berejuck, "Dedicated Monitoring Service without Routing Mitigation for Networks-on-Chip," in *Latin American Symposium on Circuits Systems (LASCAS)*, pp. 121–124, IEEE, Feb. 2019.

[196] R. Abdel-Khalek and V. Bertacco, "Post-Silicon Platform for the Functional Diagnosis and Debug of Networks-on-Chip," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, pp. 1–25, Mar. 2014.

[197] M. Dehbashi and G. Fey, "Transaction-Based Online Debug for NoC-Based Multiprocessor SoCs," *Microprocessors and Microsystems (MICPRO)*, vol. 39, pp. 157–166, May 2015.

[198] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," in *International Symposium on Microarchitecture (MICRO)*, pp. 60–71, IEEE/ACM, Dec. 2012.

[199] K. Chrysanthou, P. Englezakis, A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, and G. Dimitrakopoulos, "An Online and Real-Time Fault Detection and Localization Mechanism for Network-on-Chip Architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, pp. 1–26, June 2016.

[200] J. Liu, J. Harkin, Y. Li, L. Maguire, and A. Linares Barranco, "Low Overhead Monitor Mechanism for Fault-Tolerant Analysis of NoC," in *International Symposium on Embedded Multicore/Manycore Systems-on-Chip (MCSoC)*, pp. 189–196, IEEE, Sept. 2014.

[201] D. Zoni and W. Fornaciari, "Sensor-Wise Methodology to Face NBTI Stress of NoC Buffers," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1038–1043, IEEE, Mar. 2013.

[202] M. Ali, M. Welzl, S. Hessler, and S. Hellebrand, "An Efficient Fault Tolerant Mechanism to Deal with Permanent and Transient Failures in a Network on Chip," *Interna-*

tional Journal of High Performance Systems Architecture (IJHPSA), vol. 1, pp. 113–123, Oct. 2007.

[203] G. Schley, N. Batzolis, and M. Radetzki, "Fault Localizing End-to-End Flow Control Protocol for Networks-on-Chip," in *International Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 454–461, IEEE, Feb. 2013.

[204] Y. Zhang, K. Chakrabarty, H. Li, and J. Jiang, "Software-Based Online Self-Testing of Network-on-Chip Using Bounded Model Checking," in *International Test Conference (ITC)*, pp. 1–10, IEEE, Oct. 2017.

[205] N. Karimi, A. Alaghi, M. Sedghi, and Z. Navabi, "Online Network-on-Chip Switch Fault Detection and Diagnosis Using Functional Switch Faults.," *Journal of Universal Computer Science (JUCS)*, vol. 14, pp. 3716–3736, Dec. 2008.

[206] G. Schley, A. Dalirsani, M. Eggenberger, N. Hatami, H. Wunderlich, and M. Radetzki, "Multi-Layer Diagnosis for Fault-Tolerant Networks-on-Chip," *IEEE Transactions on Computers (TC)*, vol. 66, pp. 848–861, May 2017.

[207] L. Anghel, M. Benabdenbi, A. Bosio, M. Traiola, and E. Vatajelu, "Test and Reliability in Approximate Computing," *Journal of Electronic Testing*, vol. 34, pp. 375–387, May 2018.

[208] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys (CSUR)*, vol. 48, pp. 1–33, Mar. 2016.

[209] Z. Zhang, R. Wang, Z. Zhang, R. Huang, C. Meng, W. Qian, and Z. Zhou, "Reliability-Enhanced Circuit Design Flow Based on Approximate Logic Synthesis," in *Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 71–76, ACM, Sept. 2020.

[210] K. Parasyris, V. Vassiliadis, C. Antonopoulos, S. Lalis, and N. Bellas, "Significance-Aware Program Execution on Unreliable Hardware," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, pp. 1–25, Apr. 2017.

[211] V. Chippa, S. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and Characterization of Inherent Application Resilience for Approximate Computing," in *Design Automation Conference (DAC)*, pp. 1–9, ACM, May 2013.

[212] G. Rodrigues, F. Kastensmidt, and A. Bosio, "Survey on Approximate Computing and its Intrinsic Fault Tolerance," *Electronics*, vol. 9, p. 557, Mar. 2020.

[213] R. Boyapati, J. Huang, P. Majumder, K. Yum, and E. Kim, "APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures," in *International Symposium on Computer Architecture (ISCA)*, pp. 666–677, ACM, June 2017.

[214] A. Najafi, L. Bamberg, A. Najafi, and A. Garcia-Ortiz, "Integer-Value Encoding for Approximate On-Chip Communication," *IEEE Access*, vol. 7, pp. 179220–179234, Dec. 2019.

[215] A. Najafi, L. Bamberg, G. P. Vayá, and A. Garcia-Ortiz, "A Coding Approach to Improve the Energy Efficiency of Approximate NoCs," in *International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pp. 74–81, IEEE, July 2019.

[216] Y. Chen and A. Louri, "An Online Quality Management Framework for Approximate Communication in Network-on-Chips," in *International Conference on Supercomputing (ICS)*, pp. 217–226, ACM, June 2019.

[217] L. Wang, Y. Wang, and X. Wang, "An Approximate Multiplane Network-on-Chip," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 234–239, IEEE, Mar. 2020.

[218] Z. Wang, D. Larso, M. Barker, S. Mohajer, and K. Bazargan, "Deterministic Shuffling Networks to Implement Stochastic Circuits in Parallel," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, pp. 1821–1832, Aug. 2020.

[219] S. Jafri, L. Guang, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, "Energy-Aware Fault-Tolerant Network-on-Chips for Addressing Multiple Traffic Classes," *Microprocessors and Microsystems (MICPRO)*, vol. 37, pp. 811–822, Nov. 2013.

[220] R. Parikh and V. Bertacco, "Resource Conscious Diagnosis and Reconfiguration for NoC Permanent Faults," *IEEE Transactions on Computers (TC)*, vol. 65, pp. 2241–2256, July 2016.

[221] P. Ren, M. Kinsy, M. Zhu, S. Khadka, M. Isakov, A. Ramrakhyani, T. Krishna, and N. Zheng, "FASHION: Fault-Aware Self-Healing Intelligent On-chip Network," *Computing Research Repository (CoRR)*, pp. 1–14, Sept. 2017.

[222] L. Wang, S. Ma, C. Li, W. Chen, and Z. Wang, "A High Performance Reliable NoC Router," *Integration*, vol. 58, pp. 583–592, June 2017.

[223] O. Astrachan, "Bubble Sort: An Archaeological Algorithmic Analysis," in *Technical Symposium on Computer Science Education (SIGCSE)*, pp. 1–5, ACM, Jan. 2003.

[224] A. Gulli and S. Pal, *Deep Learning With Keras*. Packt Publishing Ltd, Apr. 2017.

[225] A. Krizhevsky, G. Hinton, *et al.*, "Learning Multiple Layers of Features From Tiny Images," *University of Toronto*, pp. 1–60, Apr. 2009.

[226] O. Vincent, O. Folorunso, *et al.*, "A Descriptive Algorithm for Sobel Image Edge Detection," in *Informing Science & IT Education Conference (InSITE)*, vol. 40, pp. 97–107, 2009.

[227] B. Barrois and O. Sentieys, "Customizing Fixed-Point and Floating-Point Arithmetic - A Case Study in K-Means Clustering," in *International Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, IEEE, Oct. 2017.

[228] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining Conventional Wisdom for Designing Nocs in the Context of FPGAs," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 37–46, ACM/SIGDA, Feb. 2012.

[229] Y. Fukushima, M. Fukushi, and I. Yairi, "A Region-based Fault-Tolerant Routing Algorithmfor 2D Irregular Mesh Network-on-Chip," *Journal of Electronic Testing*, vol. 29, pp. 415–429, May 2013.

[230] T. Frank, C. Chappaz, P. Leduc, L. Arnaud, F. Lorut, S. Moreau, A. Thuaire, R. El Farhane, and L. Anghel, "Resistance Increase Due to Electromigration Induced Depletion Under TSV," in *International Reliability Physics Symposium (IRPS)*, pp. 3F.4.1–3F.4.6, IEEE, Apr. 2011.

[231] Y. Lin, C. Zhan, J. Juang, J. H. Lau, T. Chen, R. Lo, M. Kao, T. Tian, and K. Tu, "Electromigration in Ni/Sn Intermetallic Micro Bump Joint for 3D IC Chip Stacking," in *Electronic Components and Technology Conference (ECTC)*, pp. 351–357, IEEE, May 2011.

[232] M. Pathak, J. Pak, D. Z. Pan, and S. K. Lim, "Electromigration Modeling and Full-Chip Reliability Analysis for BEOL Interconnect in TSV-Based 3D ICs," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 555–562, IEEE/ACM, Nov. 2011.

[233] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications," *ACM Computing Surveys (CSUR)*, vol. 47, pp. 37:1–37:34, Jan. 2015.

[234] M. Wirthlin, "High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond," *Proceedings of the IEEE*, vol. 103, pp. 379–389, Mar. 2015.

[235] I. T. Almalkawi, J. N. Al-Karaki, A. Alsarhan, R. Abu-Ajamiyah, and D. Al-Mughrabi, "An Efficient Digital Image Encryption Using Pixel Shuffling and Substitution for Wireless Networks," in *Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 266–271, IEEE, Apr. 2019.

[236] K. Patro and B. Acharya, "A Simple, Secure, and Time-Efficient Bit-Plane Operated Bit-Level Image Encryption Scheme Using 1-D Chaotic Maps," in *Innovations in Soft Computing and Information Technology*, pp. 261–278, Springer, Jan. 2019.

[237] S. Janakiraman, K. Thenmozhi, J. Rayappan, and R. Amirtharajan, "Lightweight Chaotic Image Encryption Algorithm for Real-Time Embedded System: Implementation and Analysis on 32-bit Microcontroller," *Microprocessors and Microsystems (MICPRO)*, vol. 56, pp. 1–12, Feb. 2018.

List of Figures

A.1 Formatage des messages : Paquets, flits et sous-flits. 4
A.2 NoC classique étendu avec la technique BiSu. 5
A.3 Valeurs des masques d'erreur dans un NoC défectueux avec la technique R-BiSu en considérant différentes tailles de régions. 7

1.1 Description of the main elements present in network-on-chips. 14
1.2 Several network-on-chip topologies. 15
1.3 Flit-based message formatting for NoC communications. 17
1.4 Classic router architecture with input buffering. 19
1.5 Pipeline of a classic router architecture with input buffering. 19
1.6 Architecture of virtual channels in a NoC router. 20
1.7 Definition of the router position in a mesh NoC. 22
1.8 XY routing algorithm in a 4×4 mesh NoC. 23
1.9 PMOS transistor ionization due to a particle strike. 26
1.10 Failure rate during the system lifetime. 28

2.1 CRC-4 encoding and decoding. 39
2.2 Encoding and decoding of a 8-bit flit with the Hamming code. 40
2.3 ECC distributions in a 3×3 NoC. 43
2.4 DMR and TMR principles. 46
2.5 Router architecture implementing BIST technique. 51
2.6 Test vectors used in the BIST methods. 51

3.1 Fault impacts in the NoC architecture. 58
3.2 Message formatting: Packets, flits and subflits. 59
3.3 Classic NoC extended with the BiSu technique. 60
3.4 Shuffler and de-shuffler block architecture. 62
3.5 Protection of headers and sensible flits with the BiSu approach. 65
3.6 Second approach for header protection with the BiSu technique. 66
3.7 Organization of 32-bit flits for different data sizes considering 8-bit subflits. 67

3.8 Mathematical computation of the fault impact on a 32-bit flit with 4-bit subflits. 70
3.9 Theoretical payload flit accuracy for one Multiple Hard Error (MHE). . . . 72
3.10 Experimental payload flit accuracy for one MHE. 73
3.11 Experimental payload flit accuracy for several SHEs using MSE metric. . . 74
3.12 Experimental payload flit accuracy for several SHEs using BER metric. . . 75
3.13 Influence of the method on the header accuracy considering 32-bit flit. . . . 77
3.14 Influence of the method on the header accuracy considering 64-bit flit. . . . 78
3.15 Influence of the subflit size on the header flit accuracy. 78
3.16 BiSu efficiency for payload protection at the NoC-level using the MSE metric. 80
3.17 BiSu efficiency for payload protection at the NoC-level using the BER metric. 81
3.18 BiSu efficiency for header protection at the NoC-level using the Correct Header Transmission Rate (CHTR) metric. 83
3.19 Fault injection in the Keras Convolutional Neural Network (CNN) benchmark. 84
3.20 Classification accuracy of the CNN using the CIFAR10 database under different sizes of MHE. 85
3.21 Fault localization on the load and the store paths on a part of mesh NoC. . 88
3.22 State of the initial image in the core after data loading from memory to core via the loading path. 89
3.23 Results of the Sobel-filter computing in the core using the loaded images via the loading path. 90
3.24 Results of the Sobel-filter computing after data storing in the memory using the storing path. 91
3.25 K-means clustering results for the first data set. 94
3.26 Overhead comparison between the BiSu technique with header distribution on two flits and the Hamming code considering a 8×8 NoC. 101

4.1 NoC decomposition into regular regions. 106
4.2 Enhanced NoC regions with the R-BiSu technique considering different region sizes. 107
4.3 NoC enhanced with the R-BiSu approach considering different region sizes. 108
4.4 Error mask values in a faulty NoC with the R-BiSu technique considering different region sizes. 111
4.5 Region error mask computation. 112

4.6 Experimental efficiency of the R-BiSu approach at the NoC scale using the MSE metric. 114
4.7 Experimental efficiency of the R-BiSu method at the NoC scale using the BER metric. 115
4.8 Experimental efficiency of the R-BiSu method at the NoC scale using the CHTR metric. 116
4.9 Area overhead comparison for the R-BiSu method. 119
4.10 Power overhead comparison for the R-BiSu method. 119
4.11 Area overhead of the register updater for the R-BiSu method. 121
4.12 Power overhead of the register updater for the R-BiSu method. 121
4.13 Highlighting of the Pareto front between area costs and efficiency for the R-BiSu approach. 122
4.14 Highlighting of the Pareto front between power consumption and efficiency for the R-BiSu approach. 123
C.1 R-BiSu associating with a region-based adaptive routing algorithm. 127

List of Tables

3.1 Notation summary. 60
3.2 PSNR results for the computation of the Sobel filter. 92
3.3 Results of the K-means clustering algorithm using the MSE and the CER metrics. 93
3.4 Hardware implementation costs for the proposed BiSu method. 96
3.5 Hardware implementation costs for Hamming technique. 96
3.6 Hardware implementation overhead for packetization/de-packetization blocks extended with merger/de-merger blocks and with 2-flit header distribution capabilities. 99
3.7 Hardware implementation costs for the register updater block. 102

4.1 Comparison of the required number of shuffler and de-shuffler blocks between the BiSu and the R-BiSu methods considering a 8×8 NoC. 118

Acronyms

Ack Acknowledgement. 24

ASIC Application-Specific Integrated Circuit. 26, 27, 127

BCH Bose–Chaudhuri–Hocquenghem. 40

BER Bit-Error Rate. 73, 74, 75, 80, 81, 82, 113, 114, 115, 116

BIST Built-In Self-Test. 50, 52, 54, 58, 62

BiSu Bit-Shuffling. 9, 10, 55, 57, 58, 59, 60, 61, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 76, 77, 79, 80, 81, 82, 83, 84, 86, 87, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 100, 101, 102, 103, 105, 106, 107, 113, 114, 115, 117, 118, 120, 121, 124, 125, 126, 127, 128, 155, 156

CER Clustering-Error Rate. 93

CHTR Correct Header Transmission Rate. 82, 113, 116

CNN Convolutional Neural Network. 68, 83, 84, 86

CRC Cycle Redundancy Check. 38, 41

DMR Double Modular Redundancy. 45, 46

ECC Error-Correcting Code. 36, 37, 38, 39, 40, 41, 42, 46, 54, 55, 103

EDC Error-Detecting Code. 36

FPGA Field Programmable Gate Array. 26, 27, 127

HCI Hot Carrier Injection. 28, 29

HLS High-Level Synthesis. 95, 117, 125

IP Intellectual Property. 9, 14, 16, 17, 18, 20, 21, 22, 23, 45, 47, 48, 49, 50, 61, 80, 83, 95, 100, 102, 107, 109, 112, 113, 116, 120, 124, 126

ISFS Intermediary Significant Faulty Subflit. 69, 70

LDPC Low-Density Parity-Check. 40

LSB Low Significant Bit. 10, 59, 61, 64, 66, 67, 69, 75, 82, 103, 115, 125

LSFS Least Significant Faulty Subflit. 69, 70, 71

LSS Least Significant Subflit. 66, 67, 69, 112

MAE Maximum Absolute Error. 68, 71

MBU Multiple Bit Upset. 27

MCU Multiple Cell Upset. 27

MHE Multiple Hard Error. 30, 68, 69, 72, 73, 84, 86, 87, 88, 156

MSB Most Significant Bit. 10, 53, 57, 59, 60, 61, 64, 65, 66, 67, 75, 76, 77, 82, 86, 103, 125

MSE Mean Square Error. 68, 71, 72, 73, 74, 80, 81, 93, 113, 114, 116

MSFS Most Significant Faulty Subflit. 69, 70, 71

MSS Most Significant Subflit. 66, 67

NAck Non-Aknowledgement. 49

NBTI Negative Bias Temperature Instability. 28, 29, 52

NI Network Interface. 14, 16, 42, 65, 66, 67, 95, 98, 100, 118, 120, 125

NMR N-Modular Redundancy. 45

NoC Network-on-Chip. 3, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 52, 53, 54, 55, 57, 58, 59, 60, 61, 62, 64, 65, 66, 67, 68, 79, 80, 81, 82, 84, 87, 88, 89, 91, 95, 97, 98, 100, 101, 102, 103, 105, 106, 107, 110, 112, 113, 114, 116, 117, 118, 119, 120, 125, 126, 127, 155, 156

ORA Output Response Analyzer. 50

PBTI Positive Bias Temperature Instability. 29

PSNR Peak Signal-to-Noise Ratio. 89, 92

QoS Quality of Service. 9, 15, 19, 20, 32

R-BiSu Region-based Bit-Shuffling. 6, 10, 11, 105, 106, 107, 109, 110, 112, 113, 114, 116, 117, 118, 119, 120, 121, 122, 123, 124, 126, 127

REM Region Error Mask. 110

RS Reed-Solomon. 40

SaF Store and Forward. 18

SEB Single-Event Burnout. 27

SEDR Single-Event Dielectric Rupture. 27

SEE Single Event Effect. 25, 26

SEFI Single-Event Functional Interrupt. 27

SEGR Single-Event Gate Rupture. 27

SEHE Single-Event Hard Error. 27

SEL Single-Event Latch-up. 27

SESB Single-Event Snap-Back. 27

SET Single Event Transient. 26

SEU Single Event Upset. 26, 27

SF Subflit. 10, 59, 60, 61, 64, 66

SHE Single Hard Error. 30, 68, 73, 74, 75, 76, 80, 82, 87, 88, 113

SoC System-on-Chip. 9, 13

TDDB Time Dependent Dielectric Breakdown. 28, 29

TID Total Ionizing Dose. 25, 26

TMR Triple Modular Redundancy. 45, 46

TNID Total Non-Ionizing Dose. 25, 26

TPG Test Pattern generator. 50

TSV Through Silicon Via. 127

VCT Virtual Cut Through. 18

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Atténuation des Défaillances Multiples dans les Architectures de Réseau sur Puce par une Méthode de Brassage de Bits

Mot clés : Réseau sur Puce, Atténuation des Défaillances, Communication Approximative, Brassage de Bits, Défaillances Permanentes Multiples

Résumé : Depuis plusieurs décennies, la tolérance aux fautes est devenue un domaine de recherche majeur en raison du rétrécissement des transistors et de l'augmentation de la puissance des systèmes sur puce. En particulier, les défauts survenant dans les réseaux sur puce (*Network-on-Chips* - NoCs) de ces systèmes ont un impact significatif en raison de la grande quantité de données qui traversent les NoCs. De plus, les approches existantes de tolérance aux fautes ne peuvent pas traiter efficacement plusieurs fautes permanentes. Pour remédier à ces limitations, nous proposons la technique de brassage de bits (*Bit-Shuffling* - BiSu) qui réduit l'impact des fautes survenant dans le chemin de données des NoCs. Pour ce faire, l'approche proposée exploite, au moment de l'exécution, la position des défauts permanents et modifie l'ordre des bits à l'intérieur d'un flit. Notre méthode réduit, autant que possible, l'impact des fautes en les reportant sur les bits les moins significatifs, au lieu de les garder sur les bits les plus significatifs. Les résultats obtenus par des évaluations approfondies montrent que la méthode BiSu peut réduire l'impact de multiples défauts permanents, avec des coûts matériels faibles, par rapport aux approches existantes, comme le code de Hamming. Ensuite, nous proposons une approche de brassage de bits basée sur des régions (*Region-based BiSu* - R-BiSu) qui réduit les coûts matériels de la technique BiSu en réduisant son efficacité de tolérance aux fautes.

Title: Multiple Fault Mitigation in Network-on-Chip Architectures Through a Bit-Shuffling Method

Keywords: Network-on-Chip, Fault Mitigation, Approximate Communication, Bit-Shuffling, Multiple Permanent Faults

Abstract: Since several decades, fault tolerance has become a major research field due to transistor shrinking and power scaling in system-on-chips. Especially, faults occurring to Network-on-Chips (NoCs) of those systems have significant impacts due to the high amount of data crossing NoCs. Furthermore, existing fault-tolerant approaches cannot efficiently deal with several permanent faults. To address these limitations, we propose the Bit-Shuffling (BiSu) technique which reduces the impact of faults occurring in NoC datapath. To achieve that, the proposed approach exploits, at run-time, the position of permanent faults and changes the bit order inside a flit. Our method reduces, as much as possible, the impact of faults by deferring them on least significant bits, instead of keeping them on most significant bits. The results obtained by extensive evaluations show that the BiSu method can reduce the impact of multiple permanent faults, with low hardware costs, compared to existing approaches, like Hamming code. Then, we propose a Region-based Bit-Shuffling (R-BiSu) approach which reduces the hardware costs of the BiSu technique relaxing its fault-tolerance efficiency.