Development of On-Chip Communication Fault-Resilient Adaptive Architectures and Algorithms for 3D-IC Technologies

Dang Nam Khanh

A DISSERTATION
SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems
The University of Aizu
2017

The thesis titled

Development of On-Chip Communication Fault-Resilient Adaptive Architectures and Algorithms for 3D-IC Technologies

by

Dang Nam Khanh

is reviewed and approved by:

Chief referee
Professor
Abderazek Ben Abdallah

Professor
Toshiaki Miyazaki

Professor
Tsuneo Tsukahara

Professor
Junji Kitamichi

Professor
Tomohiro Yoneda

The University of Aizu
2017

Contents

List of Figures

List of Tables

List of Abbreviations

2D-NoC	Two dimensional Network-on-Chip
3D-IC	Three dimensional Integrated Circuit
3D-NoC	Three dimensional Network-on-Chip
3D-SIC	Three dimensional Stacked Integrated Circuit
ASIC	Application-Specific Integrated Circuit
BLoD	Bypass-Link-on-Demand
CAC	Crosstalk Avoidance Codes
CAD	Computer-Aided Design
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
CT	Crossbar Traversal stage
DfR	Design for Reliability
DSM	Deep Sub-micron
DRAM	Dynamic Random Access Memory
ECC	Error Correction Codes
EM	Electromigration
FAIT	Fabrication, Assembly, Integration and Test
FIFO	First-In-First-Out
HDL	Hardware Description Language
IC	Integrated Circuit
ITRS	International Technology Roadmap for Semiconductors
KoZ	keep-out-zone
LAFT	Look-Ahead-Fault-Tolerant
NI	Network Interface
NMR	N-Modular Redundancy
NoC	Network-on-Chip
NPC	Next-Port-Calculation stage
P&R	Place and Route step
PE	Processing Element
PV	Process Variation
RAB	Random-Access-Buffer
RC	Routing Computation stage
RTL	Register-Transfer Level

SA	Switch Allocation stage
SAIF	Switching Activity Interchange Format
SDF	Standard Delay Format
SoC	System-on-Chip
SSD	Solid State Drive
TSV	Through Silicon Via
WLP	Wafer Level Packaging

To my parents and my family.

Acknowledgments

I would like to express my thanks and gratitude to Prof. Abderazek Ben Abdallah for his support, encouragement, and guidance to achieve this project. Also, I would like to thank Prof. Toshiaki Miyazaki, Prof. Tsuneo Tsukahara, Prof. Junji Kitamichi of The University of Aizu and Prof. Tomohiro Yoneda of National Institute of Informatics for taking the time to revise my thesis.

Moreover, my sincere gratitude to Prof. Yuichi Okuyama for his help and support during the past three years. I also would like to thank Prof. Xuan-Tu Tran for leading me into this research field and his supports, and Dr. Akram Ben Ahmed of Keio University for his help on the NoC platform and the later works.

I want to thank all the members of the Adaptive Systems Laboratory and my friends at the University of Aizu. Their supportive words and encouraging messages kept me motivated to work harder and be a better researcher and person. Not to forget to appreciate the staff of the University of Aizu for their assistance.

My gratitude to my mom and my dad, who have been strongly supporting me through my whole life. They have been inspiring and pushing me to achieve my goals. And to my dear adoptive parents, who encouraged me in all aspects of my life and I am happy to have them. I also thank my big family for their support, to my friends whom I spent hours talking to, for helping me passing through countless tough times.

This thesis document partly belongs to OASIS - a Network-on-Chip project, in Adaptive Systems Laboratory, The University of Aizu. I inherited the basic router design, the hard fault tolerant mechanisms and the TSV tutorial from the senior students' works. I would like to thank all of them for the solid platforms that allowed me to achieve a part of this research.

Thesis advisor: Professor Abderazek Ben Abdallah Dang Nam Khanh

Development of On-Chip Communication Fault-Resilient Adaptive Architectures and Algorithms for 3D-IC Technologies

Abstract

Multicore processing is predicted to be the backbone of future complex embedded architectures. By distributing the tasks into multiple processing elements, the system's frequency and operation voltage can be reduced; thus, a decrease in total power consumption can be obtained. However, due to the high complexity in terms of organization, communication and operation, multicore processing demands in high scalability, efficient bandwidth, and better power efficiency solution have become primordial. Notably, wires have overcome gates to become the most dominant source of delay in the deep sub-micron era. Consequently, the power consumption caused by additional buffers and wires is considered as a critical obstacle. Moreover, conventional communication paradigms (e.g., bus, point-to-point) also encounter several scalability and latency issues. In the past few years, the benefits of 3D Integrated Circuits (3D-ICs) and mesh-based Network-on-Chips (NoCs) have been fused into a promising architecture, called 3D-Network-on-Chip (3D-NoC). In fact, the scalability and parallelism of NoCs can be enhanced in the third dimension thanks to the short wire length and the low power consumption of 3D-ICs interconnects. As a result, the 3D-NoC paradigm is considered to be one of the most advanced and auspicious architectures for the future of IC designs, as it is capable of providing extremely high bandwidth, efficient scalability and low power interconnects.

While the 3D-NoC paradigm has been increasing in popularity with several commercial chips, it is threatened by the decreasing reliability of aggressively scaled transistors as they are approaching the fundamental physical limits. In deep sub-micron processes, gates have become more vulnerable to soft errors which can affect the operation accuracy of control logics and buffers in NoCs' routers; thus, leading to chip failure. In addition, low supply voltages enforce a very narrow noise margin, which makes the architecture more vulnerable and more sensitive to faults. In particular, hard faults, including both permanent and intermittent, can occur during the manufacturing stage or under specific operating circumstances. Because the intermittent faults do not permanently damage a given component, they can pass through several testing stages, but can still cause operating failures. Furthermore, and by shifting to 3D-ICs, 3D-NoCs are introduced to a new major challenge. That is, the high probability for TSV (Through Silicon Via) defects to occur. With high defect-rates and the clustering effect, TSVs need a proper fault-tolerance methodology to ensure the overall reliability. By accumulating all the failure sources, 3D-NoCs' reliability is expected to be one of the most critical issues in future System-on-Chips (SoCs) designs.

Due to the numerous types of faults, many studies have proposed solutions for various individual aspects of on-chip reliability; however, a comprehensive approach encompassing soft errors, hard

faults, and TSV defects pertaining to NoCs' reliability has yet to evolve. In addition, the error detection and diagnosis in 3D-NoC architectures have been studied thoroughly in the scope of offline-testing. On the other hand, with soft errors and intermittent faults becoming a dominant failure mode in modern NoCs and general VLSI systems, a widespread deployment of online-testing approaches has become crucial.

In addition to the variety and complexity of failure modes, the rapid development of fault-tolerance for NoC has become exposed to a new challenge: NoCs' reliability needs to be evaluated and quantitatively assessed in the early design stages. As a matter of fact, most of the existing evaluation methodologies use the simple fault insertion and correctness-verification method. Such a method only ensures the functionality of a given technique. Moreover, this type of evaluation requires the complete design to be performed which may lead to a significant redesign time risk. To solve this issue, early reliability assessment is needed. After satisfying the performance requirements in the early assessment stage, the reliability of the design is also needed to be fully simulated and analyzed.

Starting from the above facts, fault resilient adaptive architectures and algorithms for 3D-ICs, especially for 3D-NoCs based systems, are developed in this research. With the aid of efficient detection, diagnosis and recovery mechanisms and algorithms, the proposed system is capable of detecting and recovering from soft errors occurring in the routing pipeline stages. It also leverages configurable components to handle permanent faults' occurrences in links, input buffers, and crossbars by adopting our previous works. For integrating these hard error fault tolerant techniques, a detection, diagnosis, and recovery mechanism is proposed. This mechanism analyzes the transmitting operation and its failure state to determine the fault's position. Based on the position of the fault, it issues signals to handle it. Moreover, this work also proposes a dedicated fault-tolerant technique for TSV-cluster defects, which are the most vulnerable components of 3D-IC technology.

From another important perspective, this work presents a platform of reliability assessment for NoC systems. An analytical is used to help designers quickly estimate the efficiency of potential fault tolerant schemes. The result of this assessment can indicate the reliability enhancement of the evaluated technique. Later, the complete architecture is put into a netlist-based simulation process to estimate other results. The development of the reliability assessment, fault-tolerance architecture, and algorithms are integrated into the flow of *Design for Reliability*. In this flow, the analytical model helps the early assessment of the proposed fault-tolerant techniques, and the netlist simulations are conducted to confirm the reliability of the design.

The final goal of this dissertation is to propose a comprehensive fault-resilient architectures, algorithms, and a design methodology for highly reliable 3D-NoC systems development. In addition to providing the fault-tolerance techniques to deal with soft errors, hard fault and TSV defects, a working flow is also presented. The complete working design stages are also provided to help designers understand their proposals, know how to approach the fault-tolerance challenge and complete a robust and graceful design.

1

Introduction

This chapter discusses the future of Integrated Circuits (ICs), where Three dimensional ICs (3D-ICs) are widely used. It first highlights the emergence of the multi-core era and the numerous challenges in interconnection. Next, these challenges are analyzed, and Through-Silicon-Via (TSV) based Three dimensional Network-on-Chips (3D-NoCs) are considered as the solution for future 3D-ICs. However, by inheriting the faults from ICs and TSVs, and due to the increase in high power density and thermal removal difficulty, 3D-NoCs are predicted to have a reliability crisis which needs to be properly addressed. In order to solve these problems, the design goals show the objectives of this dissertation and the contribution section highlights the proposed methodologies. Finally, this chapter concludes with the structure of this dissertation.

1.1 Computing Architecture: The Road to Multi-Core Era

Moore's law [1] and *Dennard's scaling* [2] have been the driving forces for decades of Integrated Circuit (IC) development. With the prediction of doubling the integration density every 18 months and keeping similar power density, silicon semiconductor industry has shown extraordinary achievements throughout its history. Unfortunately, *Dennard's scaling*, which claims that transistors get smaller; but their power density stays constant, was questioned in the last decade due to the domination of dynamic power and the increase in leakage current [3–5]. Consequently, increasing the frequency is no longer the concern of most CPU manufacturers. Instead, reducing the power consumption is the big challenge to deal with. Although Moore's law is still long lasting, the deep sub-micron (DSM) technology is reaching the molecular scale which is the limitation of transistors' sizes. Nevertheless, state-of-the-art systems have been already integrated onto a single chip (or package) with billions of transistors. Consequently, the System-on-Chip (SoC) integration paradigm becomes more complex, requires better performance and more power efficiency, and demands higher reliability. In order to satisfy these requirements, shifting to a new architecture has become primordial [6, 7].

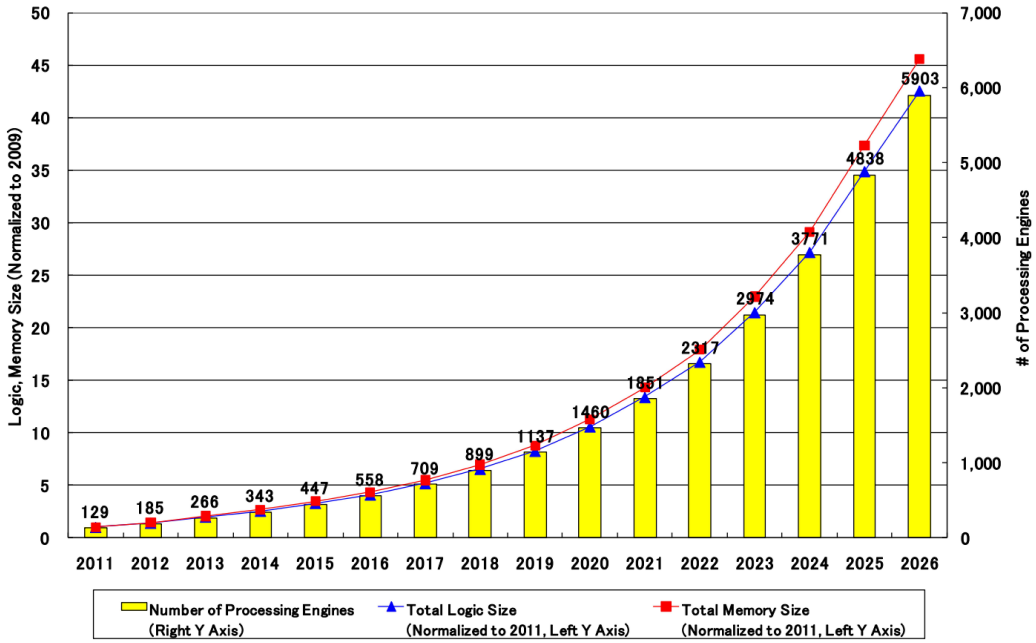

Figure 1.1: Many-core system complexity trends [8].

The improvements of computing performance and energy efficiency are given by the develop-

ment of technology scaling. Since *Dennard's scaling* was disclaimed, increasing the frequency is no longer a suitable solution for improving the system performance. Therefore, to satisfy the hunger for highly complex and stressful applications, the number of cores has been increased to improve the parallelism and reduce the clock frequency while enhancing the overall performance. For ease of understanding, Figure 1.1 presents the International Technology Roadmap for Semiconductors (ITRS) trends in many-core systems' complexity [8]. In comparison to systems in 2011, the current logic and memory integration of today increases by nearly 5.5 times. By 2026, the integration technology is predicted to be 8.3 times more than this year (2017). This complexity incrementation is led by the increasing of the number of processing engines. As a result, the era of thousand-cores per chip has been revealed. In the year of 2016, *KiloCore* [9], with the ability to execute *1.78 trillion instruction/sec*, has been recorded as the first *1,000 cores* processor. Such as system has come as a solid evidence for the predicted future of computing. As predicted in 2026, nearly 6000 cores are expected to be embedded on a single chip.

Shifting to the thousand-cores era brings numerous challenges on both software and hardware perspectives [10]. With its rapid development, software has been innovated with parallel processing to keep up with multi-core systems. According to *Amdahl's Law* [11], the speed up of an algorithm on a parallel computation platform, as shown in Eq. 1.1, depends on the speed up of each part (s) and the percentage of parallelism (p). Since the task speed up has been limited by the stabilization of the maximum frequency, increasing the parallelism is one of the obvious solutions.

$$S_{latency}(s) = \frac{1}{1 - \frac{p}{s}} \tag{1.1}$$

Von Neumann architecture, which was proposed a half century ago, is still a solid base for computer design. However, it is also exposed to several bottleneck drawbacks. To alleviate this issue, several multi-core approaches, such as SIMD (Single Instruction Multiple Data), MIMD (Multiple Instruction Multiple Data), have offered significant improvements over the traditional methods. By parallelizing the data flow and instructions, the performance of computer architectures has significantly improved. On the other hand, with thousands of cores per chip, the traditional interconnection paradigms (e.g., bus-like fabrics or point-to-point) are no longer able to keep up with such improvement. To handle thousands of Processing Elements (PEs), the arbitration modules

of the bus or point-to-point systems have to be overwhelmingly complex. Due to the low scalability of these traditional paradigms, designers also struggle to expand and manage their multi-core systems. Consequently, the need for a more scalable, lower power and higher performance interconnection platform has become unavoidable.

To answer these limitations, *Network-on-Chip* [12] (NoCs) have been widely accepted as the most suitable solution. In such systems, a network is created from micro-routers which are connected to PEs. Instead of establishing a communication channel beforehand, the data is split into sub-sets (e.g., messages, packets and flits) and sent throughout a micro-network. By creating a dedicated topology (e.g., mesh, torus, circle), expanding to thousands of cores with NoC is simpler than traditional paradigms. In fact, several recent multi-cores systems [9, 13, 14] have used NoCs as the backbone of their interconnection.

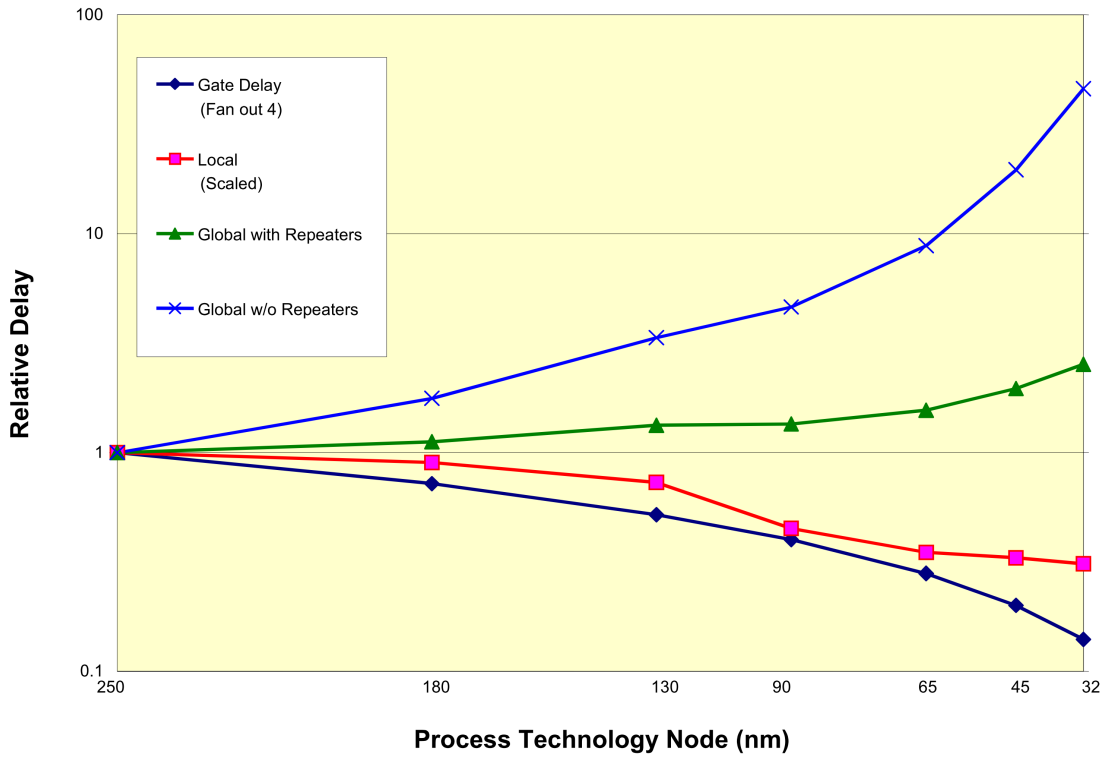

Figure 1.2: Wire vs. gate delay [15].

Besides the scalability and parallelism issues, wire delays has become more dominant than the logic gate delay in DSM technologies, as shown in Fig. 1.2. From the relative delay in 250*nm* technology node, the gate delay has decreased (about 5 times faster than the 32*nm* technology) thanks to the scaling of the feature sizes while the global wire delay has significantly increased

(about 30 times faster than $32nm$ technology) due to the increase in system complexity. As reported in [16], wiring delay accounts for about 75% of the overall delay in the $90nm$ technology node. Table 1.1 also depicts the dominance of wire latency over the transistor delay with smaller feature sizes. As expected at $12nm$ feature size, a $1mm$ interconnect delay is 3×10^5 times slower than the transistor one. From these reports, the integrated circuit delay optimization has to shift from logic gate delay to wire delay in order to maintain the performance improvement requirements. As a consequence of the increasing in wire delays, more repeaters are needed which increase the area cost and the overall power consumption. An analysis of power dissipation in microprocessor [17] showed that interconnection power occupied over 50% of the dynamic power. For the smaller technology node, there is an expectation of a higher domination of wires in the overall delay and power consumption. Consequently, and despite the tremendous obtained benefits, conventional NoC systems cannot handle alone the high requirements of the future IC needs.

Table 1.1: Interconnects delay domination over technology scaling [18].

	90 nm	45 nm	22 nm	12 nm
Transistor delay (ps)	1.6	0.8	0.4	0.2
Delay of 1mm interconnect (ps)	5×10^2	2×10^2	1×10^4	6×10^4
Ratio	3×10^2	2.5×10^3	2.5×10^4	3×10^5

In order solve the interconnection delay and power consumption issues, several emerging interconnect paradigms such as *RF/Wireless* communication [19, 20], Carbone Nanotube [21, 22], Photonic[23]and 3D Integrated Circuits (3D-ICs) [24, 25] have been proposed. On-chip *RF/Wireless* systems replace wires by antennas to handle the communication using electromagnetic waves. Since this technology does not require physical wires, it does not encounter the interconnect delay issue when the system keeps scaling up; however, it may have some interference issues by putting multiple terminals in a small area. *Carbone Nanotube* uses carbon-based interconnect instead of Cu/low-k. Thanks to their outstanding electric and thermal properties, *Carbon Nanotubes* can outperform Cu interconnections, especially for long lengths[26]. On the other hand, *Photonic* aims to transfer data by using photons instead of electrons which require dedicated modulators and receivers to convert data from electric to photonic domain and vise versa. In order to help reducing the wire length, 3D-ICs use short intra-layer wires and allow multiple layers to be stacked in order to make modules closer. Among the above paradigms, 3D-IC [24, 25] is one of the most prominent solutions. There are two kinds of 3D-IC: *monolithic* and *stacking*. Monolithic 3D-ICs [27, 28]

are manufactured by fabricating all layers in the same wafer. It is still unrealistic due to several issues, and it is considered as a long-term research activity. On the other hand, 3D-IC stacking technology is immediately available by using similar 2D wafer fabrication and establishing a layer-to-layer connection [24] using short vertical wires, called Through-Silicon-Vias (TSVs). Thanks to the shorter wire length of TSVs and the 3D structure, the wire delay is reduced in 3D-ICs; thus, resulting in power consumption reduction. As shown in Fig. 1.3, by stacking two layers, the wires of TSV-based 3D-ICs can be shorter in comparison to other approaches. Moreover, the chip footprint is significantly smaller, and the stacking structure even allows multiple technologies to be deployed on a single chip. This can open new and promising horizons for ICs designs where an individual chip can be tiny while being able to handle multiple tasks: processing, memory, RFID, mobile network, and so on.

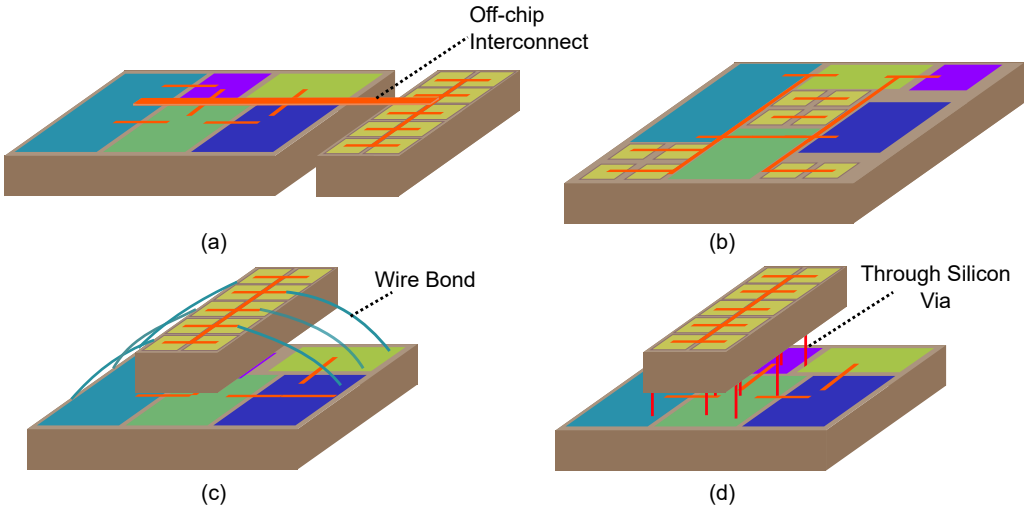

Figure 1.3: A comparison between alternative design implementations: (a) Separated chips; (b) System-on-Chip; (c) Wire-bonding-based 3D-ICs; (d) TSV-based 3D-ICs.

By combining the 3D integration benefits with Network-on-Chips scalability and parallelism, a new hybrid architecture, called 3D-Network-on-Chip [29] (3D-NoC), can offer one of the most advanced interconnection solutions for future IC designs. In order to do that, the most basic idea is to expand from a 2D mesh topology to the third dimension by adding vertical connections. As a result, 3D-NoCs not can only solve the interconnection issues by using TSVs; but, they can also provide better scalability for the thousand-cores era. Furthermore, 3D-NoCs not only reduce the wire delay by using TSVs; but; they also offer a lesser number of hops in the routing path.

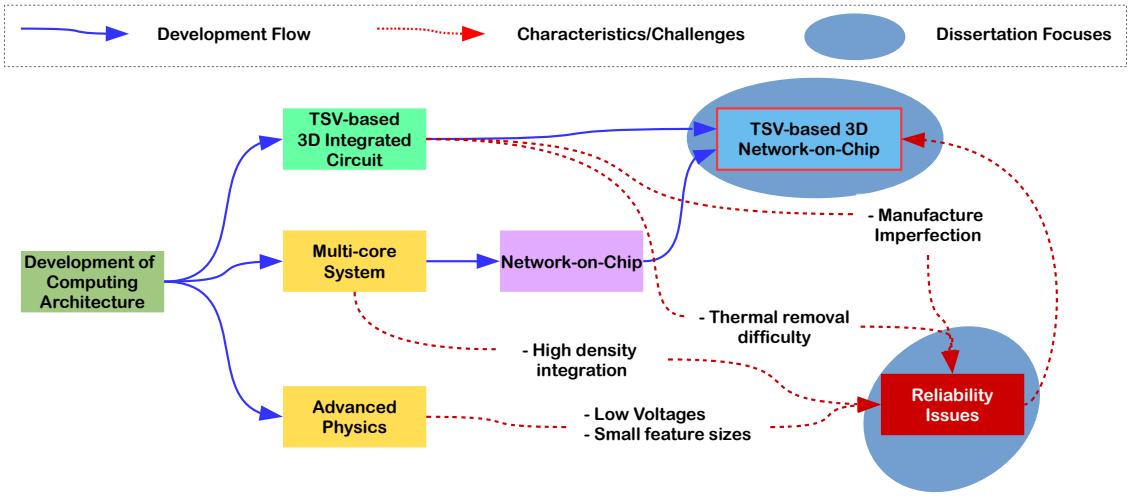

Figure 1.4: 3D-NoCs as one of the future solutions for computing paradigms and the predicted reliability issues.

1.2 Motivation: Interconnection Reliability Crisis

Despite the advantages gained from 3D-ICs, as well as 3D-NoCs, IC designs are still facing several challenges related to reliability, thermal management, heat removal and stress issues. Notably, reliability is one of the major challenges to deal with. This is due to the imperfection of the manufacturing process [30, 31] and the vulnerability of 3D-ICs due to operation voltages, high density, and thermal removal difficulties. As depicted in Fig. 1.4, TSV-based 3D-NoCs is encountering significant reliability crisis which prevents it from being widely used. Therefore, this dissertation is motivated to solve the predicted vulnerability of 3D-NoCs, being considered as the interconnection paradigm for 3D-ICs.

Inherited from 3D-ICs problems, the TSVs' reliability constitutes one of the most challenging design issues in 3D-NoCs. As a matter of fact, in the manufacturing process, the TSV defect-rates have been recently reported to be nearly 0.63% [32]. Moreover, 3D-ICs suffer from the stress issues due to the difference between thermal expansion coefficients of the implementation materials [33]. The temperature variation between two layers has been reported to reach up to 10°C [34] which negatively effects the *Time Dependent Dielectric Breakdown* and *Thermal Cycling* [35]. Not forget to mention that *Electromigration* [36] can also be a major concern.

Besides the high defect rates of TSVs, soft errors and hard faults also occur inside the system and corrupt the operation. In fact, and as previously mentioned, transistors are approaching the

fundamental limits of scaling. Gate widths are nearing the molecular scale, resulting in breakdown and wear-out in end products [37, 38]. Moreover, the anticipated fabrication geometry in 2018 scales down to $8nm$ with a projected 0.6V supply voltage [39]. In such node scaling, a higher rate of soft errors affects the control logics and buffers in NoC routers, leading to chip failure. In addition, the low supply voltage enforces a very narrow noise margin, which makes the architecture more vulnerable and sensitive to faults. As reported in [40], the soft error rate increases by about 30% for each 100 mV decrease in the supply voltage. With the rising power density and the non-ideal threshold and supply voltage scaling, soft errors have become increasingly common during a chip's lifetime [35].

Because of their high complexity, 3D-NoCs using TSVs also face difficulties regarding testing, diagnosis and recovery of faults which need to be properly addressed. In addition, the error detection and diagnosis in NoC architectures have been studied thoroughly in the scope of offline-testing. Nevertheless, and taking into consideration the fact that soft errors and intermittent faults are becoming a dominant failure modes in modern NoCs and general VLSI systems, a widespread deployment of online-testing approaches has become crucial.

To solve the reliability issues, there is a substantial number of fault-tolerance works for NoCs, which have been recently summarized in [41]. However, designing fault-tolerance techniques is only a part of the whole process, which is *Design for Reliability* [42]. According to *the IEEE Guide on Reliability* [43], there are five basic phases in reliability assessment: *System definition*, *Preliminary design*, *Detailed design*, *FAIT* (Fabrication, Assembly, Integration, and Test) and *Product/support* phase. Notably, reliability prediction in the three early phases is very important to prevent the wasted time of manufacturing and designing (FAIT phase). The system still needs to be carefully investigated before selecting the appropriate method in the specification/preliminary stage. With the system's specification, an analytical model can be used to early estimate the reliability of a given NoC system. After completing the design, the system's reliability has to be analyzed. The prototype is also studied carefully to extract the finest results. If the product can satisfy the predefined requirements, it passes the process. Otherwise, designers need to investigate again. Therefore, design with reliability awareness [42] is a methodology to ensure the robustness of the system which needs to be early predicted and carefully evaluated. In fact, early reliability analysis in the *Preliminary design* phase is one of the most important stages that can prevent the

cost of re-design and re-implementation. Even after completing the design, the system needs to be carefully assessed not only with design constraints; but, also in terms of reliability.

Because NoC reliability assessment is still immature, it becomes one of the most crucial issues for designers. Therefore, besides the fault-tolerance solutions, this dissertation also considers the reliability assessment as an important issue to be addressed.

1.3 Dissertation Goals

The objective of this research is to propose a comprehensive set of faults-tolerant architectures and algorithms for soft errors, hard faults, and TSV defects. In addition, a reliability assessment platform is presented to help designers analyze the fault-tolerance mechanism. Hereafter, the design goals are defined for this dissertation:

1. *Reliability*: The proposed 3D-NoC architecture must be able to deal with the most common faults of 3D-ICs: soft errors, hard faults, and TSV defects.

2. *Modularity*: The proposed design methodology should allow the components' update. Due to the rapid development in computer architecture, more and more advantaged techniques are constantly proposed. To keep up with this development, modularity is a necessary criterion.

3. *Scalability*: The proposed fault-tolerant methods have to maintain the scalability of 3D-NoCs. This allows the proposed methodology to be widely applied in multi-core systems.

4. *Efficiency*: Because the fault-tolerant mechanisms may have negative effects on the system performance, and naturally bring additional numbers of faults, they need to ensure the efficiency in terms of performance and fault resilience. The reliability assessment should offer a low complexity method that reduces the development time.

5. *Adaptivity*: The system should adapt different scenarios of applications and faults. Moreover, the reaction/configuration time is also an important criterion. Because of the difficulty in terms of repairability, the highly reliable 3D-NoC not only have to recover from faults; but, it can also detect them and quickly respond to the situations.

1.4 Contributions

With the above predefined goals that this work aims to satisfy, this dissertation comprises the following contributions:

1. **Complete and comprehensive soft-errors and hard-faults resilient architectures, algorithms, and design methodologies.** This thesis presents a soft error resilient method which is specified for the pipeline stage in NoCs. Because soft errors are unpredictable and unavoidable while occurring with a high probability (80% [44]), the recovery for them needs to be completed to avoid catastrophic consequences. Besides several well-known techniques on tackling data corruption, computation process is one of the most critical parts. To deal with computation corruptions, this work presents *Pipeline Computation Redundancy*. With the aid of *Error Correcting Code*, they complement each other to endorse the soft error resiliency of a given NoC system based on an online-testing method. Along with the previous work on hard fault tolerance, the soft error and the TSV-defect (the second contribution) resiliency complement the final 3D-NoC and its ability to deal with all types of faults. This comprehensive and complete architecture is considered as the ultimate design goal for 3D-NoCs, which not only offers several benefits in terms of performances and power consumption; but, it is also able to deal with the most common faults.

2. **Scalable design methodology and online algorithm for TSV-cluster defects' recovery in highly reliable 3D-NoC systems.** Defects on TSVs have significant impacts on the yield and also reduce the reliability of post-manufacturing products. While random TSV defects have been efficiently dealt with using redundancy, the cluster TSV defect is a raising reliability issue. In order to deal with the cluster defect, this thesis proposes a technique where the TSV clusters are shared between neighboring routers. Thanks to a light-weight and online arbitration algorithm, the system can maintain the operation under a significant amount of cluster defects. Several optimizations and arbitrations are added to improve the overall reliability.

3. **A platform for early reliability assessment for NoC designs.** To help designers understand the different trade-offs between the performance degradation, area cost, power consump-

tion, maximum frequency and fault-tolerance capacity, this dissertation build a platform for early reliability prediction using a fast and light-weight analytical model. By modeling the network with the fault-tolerant techniques to states and exploring the reactions of the system to failure, this proposed platform provides a quantitative solution that depicts the reliability of the system. Hence, given a fault-tolerant technique and a NoC architecture, designers can early assess the reliability without implementing and performing real tests.

By bringing the aforementioned contributions together, this dissertation offers a whole process consisting of specification with prediction, system-level and modular design, in addition to the layout stage.

1.5 Dissertation Structure

In the remaining parts of this thesis, the contents are described and organized as shown in Fig 1.5. The details of the remaining chapters are as follows:

- In Chapter 2, this thesis first overviews 3D integration and Network-on-Chip interconnect paradigms. It also highlights the reliability issues in 3D-NoC systems and provides an overview of the proposed solution.

- Chapter 3 presents some of the important related works that dealt with fault-tolerance in NoC systems. Moreover, it also provides a survey of TSV fault-tolerance. Later, a summary of reliability prediction is presented.

- Chapter 4 is dedicated to the soft error and hard fault tolerant NoC architecture where it addresses both types of faults. A detection, diagnosis, and recovery mechanism is proposed to ensure the online fault-tolerance capacity of the network. Evaluations are performed and analyzed to show the impacts on the system performance and reliability under several fault-rates.

- Chapter 5 introduces the proposed TSV fault-tolerant mechanism. It firstly starts by presenting a brief overview of the proposed structure. Second, it introduces the sharing algorithm and several performance and reliability enhancement techniques. Finally, it evaluates

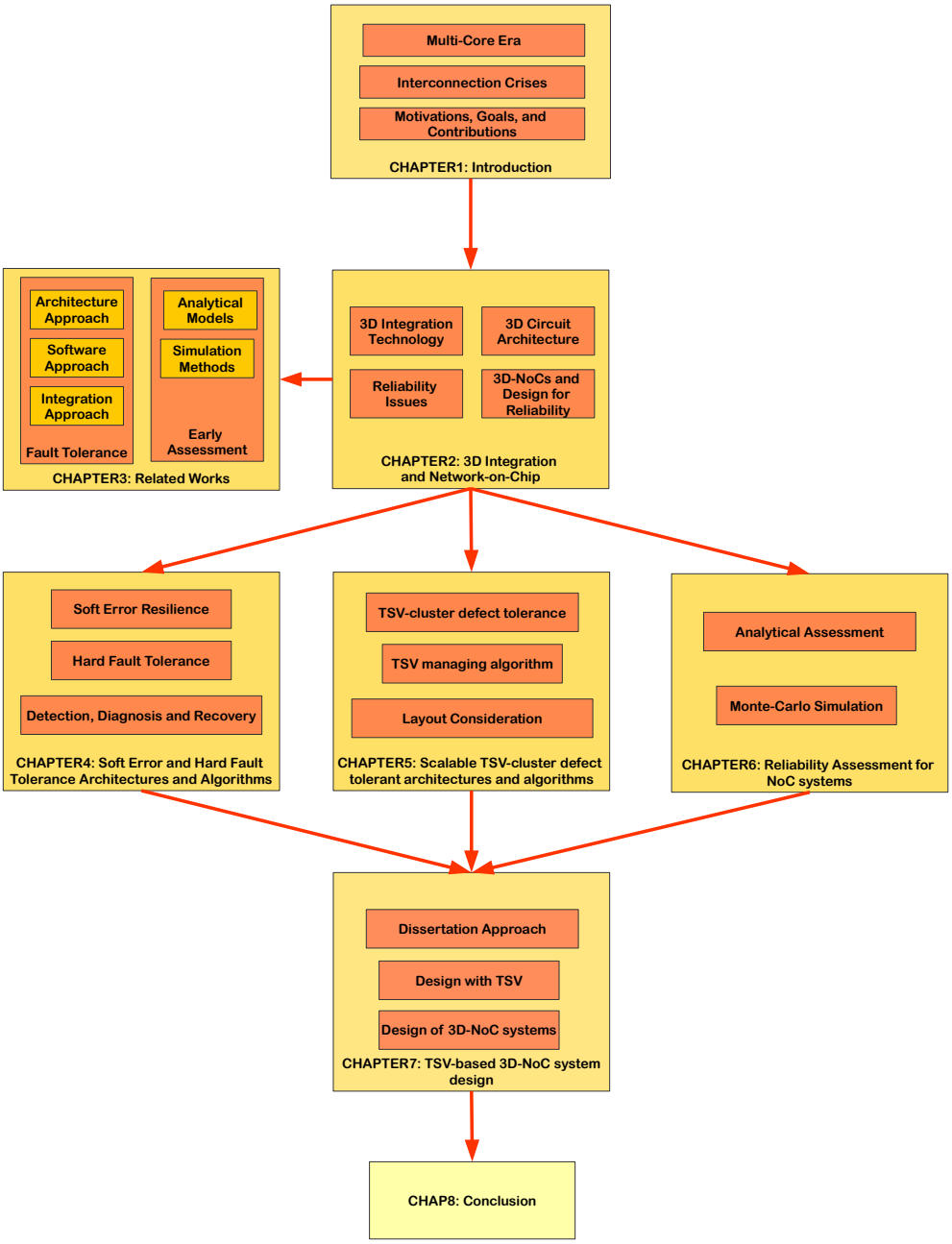

Figure 1.5: Dissertation structure.

the system to analyze the impact on performance and reliability when adopting the proposed TSV fault-tolerant mechanism.

- The dissertation dedicates Chapter 6 to present the proposed reliability assessment platform in a fair amount of details. The definitions and the backbone of this assessment are firstly provided. Later, reliability is addressed by using specific strategies and merging techniques. For the accuracy evaluation, it firstly presents the Monte-Carlo Mean Time To Failure simulation and compares the results with the proposed platform. Finally, assessment results for several network sizes are provided and analyzed.

- Chapter 7 depicts the dissertation approach regarding the design for TSV-based 3D-NoC systems. Firstly, the dissertation approach with *Design for Reliability* is presented. Finally, the discussion of design with TSV and 3D-NoC systems is described.

- Finally, in Chapter 8, this dissertation ends with the conclusion. It also discusses the potential optimization that can be further undertaken.

2

3D Integration and Network-on-Chip

THREE DIMENSIONAL INTEGRATION is an emerging technology offering lower power consumption, reduced wire delay and allowing heterogeneous integration. Among several technologies, Through-Silicon-Via (TSV) is rising as a prominent interconnection method. By creating a via, then thinning the dies and stacking them together, TSVs are created for handling the inter-die connections. Although the 3D-integration has brought significant benefits, it is still immature due to various reliability issues. The defect-rates of TSVs are still extremely high and they require more advanced development on quality improvement and fault-tolerance. This chapter presents the fundamentals of 3D integration and addresses the critical reliability challenges. Because 3D-Network-on-Chips (3D-NoCs) are considered as the backbone of future 3D Integrated Circuits (3D-ICs), this chapter also provides a brief description of 3D-NoCs and specifies their main design challenges. An overview of existing fault-tolerant

mechanisms for 3D-ICs and 3D-NoCs is also described. Furthermore, the reliability assessment demands are presented and analyzed.

2.1 3D Integration Technology

2.1.1 Overview

In the early years of computing systems, the birth and scale-down of the transistor have introduced a new different focus: *the gate*. Scientists and engineers, in the last five decades, have focused on how to reduce the size of transistors while improving their performance. As envisioned by Gordon Moore, transistors are becoming smaller [1] and will reach the modular scale soon. However, the focus has been shifted to a retro obstacle: *the wire*. While the gate delay and performance are improved, wire's delay has been increased and has become the major challenging part inside ICs. Because of the domination of wire delay, scientists and engineers need a better method of integration. The answer, in fact, has been imagined for decades, is 3D (or vertical) integration [45].

Because the System-on-Chip (SoC) development has shifted from off-chip to on-chip interconnections, and that allowed all the system's components to be integrated on a single package, 3D-ICs are widely considered as the promising solution for future SoCs. By increasing the system integration at a lower cost, reducing the footprint, improving the performance and reusing the existing technologies, 3D integration is a solid step forward for manufacturing integrated circuits. Recently, 3D-ICs have been introduced in several applications such as DRAM stacking [32], camera sensors [46–48], SSD (Solid State Drive) [49], processors [50] and many others.

In the most simple way, 3D integration is an extension of conventional 2D integration. By vertically integrating, designers expect to have smaller packages, shorter wires that reduce the power consumption, and better overall performances. Figure 2.1 shows how the 3D structure can reduce the footprint and wire length. With the same area A, the die width of 2D-IC is $\sqrt{A}$ while the two layers and four layers 3D-ICs die widths are $\sqrt{A/2}$ (29.29% of reduction)and $\sqrt{A/4}$ (50% of reduction), respectively. Thanks to the diminution of the die width, the wire length is also reduced. As shown in Fig. 2.1, by converting to two layers, the wire length in the same planar is reduced, and there is an additional length of the vertical wire, which is significantly small (layer's thickness).

On the other hand, by following *Moore's law*, transistors are reaching the modular scale. To

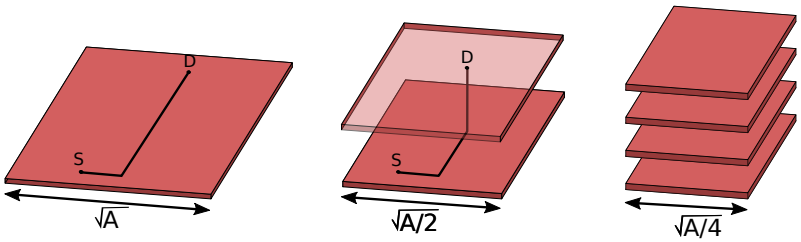

Figure 2.1: Footprint and wire length reduction in 3D-stacked structures.

keep adding more gates in the same area, reducing the transistor's size is no longer the correct answer. To solve these limitations, or even advance to *more than Moore*, going vertical is a better solution. As earlier mentioned, the 3D integration is not a brand new idea; but, it is recently considered thanks to its several benefits. The first benefit of going vertical is having shorter wires, as analyzed before. Second, the footprint can be smaller. Third, some 3D integration technologies allow heterogeneous stacking, where each die can be fabricated in different technologies and can be stacked together later. This also helps to reduce the development time.

2.1.2 3D Integration Methodologies

Because 3D integration has been early considered and recently researched, there are several methods for vertical integration. The approach can be *Wire bonding* [51], *Solder balls* [52], *Through-Silicon-Via* [24] or *Wireless stacking* [19, 53, 54]. Figure 2.2 shows the most common methods for 3D-ICs fabrication. *Wire bonding* uses dedicated wires to establish the connection between layers. *Solder balls* is an alternative method for wire bonding, where layers are connected by solder balls to establish the intra-layer connections. For both *wire bonding* and *solder-balls*, the major objective is the global interconnection or the I/O of the dies. Although they improve the performance and power consumption in comparison to off-chip interconnections, the long wire length problem is still not completely solved. The *Through-Silicon-Via* and *Wireless stacking* aim to solve this issue. By establishing vias throughout the layer, creating micro-bumps as contact points, and connect layers together, TSVs can act as a the global interconnection with large TSV sizes (2-10μm) or even local ones with smaller sizes (less than 2μm). *Wireless stacking*, which can use capacitive or inductive couplings, does not require any vias for the communication. It only demands *Vdd* and *GND* supplies. A single communication can be extremely fast; however, the interference between terminals is one of major obstacles to allow high density integration. The wireless method

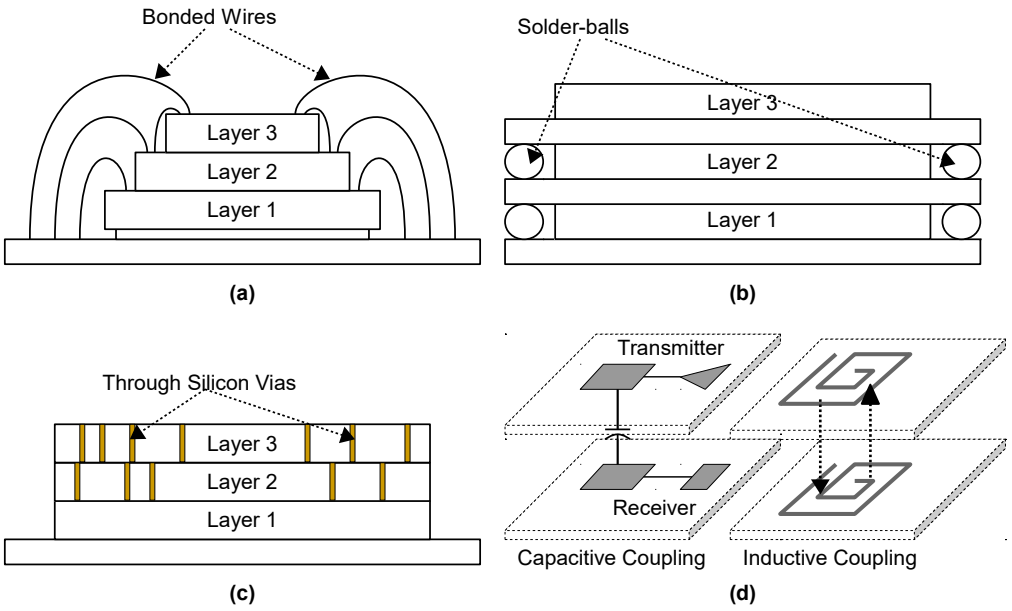

Figure 2.2: 3D integration methodologies: (a) Wire bonding; (b) Solder balls; (c) Through Silicon Vias; (d) Wireless stacking.

Table 2.1: 3D and 2D technologies comparison.

Technology	2D SoC	Wire bonding	Solder balls	TSVs	Wireless stacking
Integration Capacity	low	high	high	very high	very high
Bandwidth	high	high	medium	very high	very low
Interconnect density	medium	low	low	very high	medium
Yield	medium	medium	medium	considerable	N/A
Cost	very low	high	low	very high	N/A
Power Consumption	medium	low	low	very low	N/A

is prominent when eliminating the use of TSVs; however, it is still immature for wide usage. Table 2.1 shows a brief comparison between 2D and 3D technologies.

Scientists are also doing a lot of research about 3D VLSI [55], where no TSVs are required for global or local interconnection. In fact, the transistor is fabricated not only in a planar fashion; but, also vertically. Instead of TSVs, *Monolithic Inter-Tier Vias* (MIVs) are used to connect layers. However, the CAD tools' support and fabrication difficulties still limit the feasibility of this method. Despite of the above challenges, *Through-Silicon-Vias* is predicted to play a major part of 3D integration in the next decades.

TSV-based 3D Integration

Figure 2.3 shows the recent implementation of TSVs. The available technology is 3D WLP (Wafer Level Packaging) bond-pad where the TSV size is larger than 10 μm and it serves as an I/O

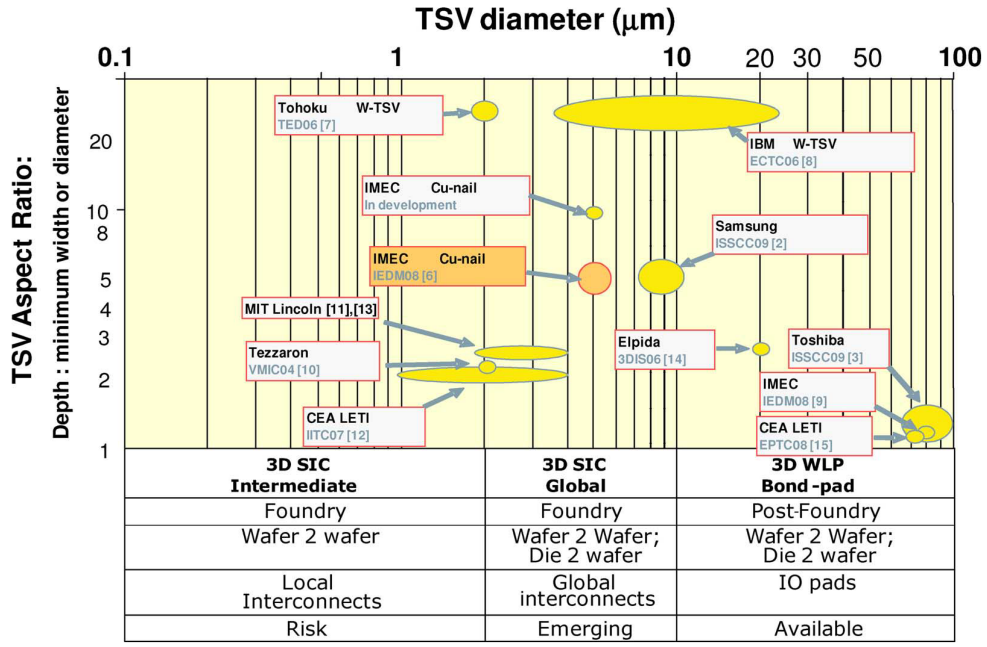

Figure 2.3: TSV fabrication technology [56].

interconnection in some systems [56]. TSVs serve as global interconnects and they are designed for 3D SIC (3D Stack IC) where their average size is between 2 μm and 10 μm. These are emerging technologies and have been researched in the past few years. The smaller size (less than 2 μm) is aimed for local interconnects and is still risky. For each TSV technology, the TSV aspect ratio shows the ratio between the depth to the diameter. For the lower ratios, they require thinner wafers/dies in order to be connected.

By stacking multiple layers and connecting them via TSVs, the wire lengths can be significantly shortened. As shown in Figure 2.3, the depth of TSVs are nearly or smaller than 20 times the diameter, which is extremely shorter than normal wires. Moreover, the 3D structure also makes blocks in the system closer, which also creates shorter wires. As a result, the power consumption is reduced because of the lesser buffers in wires.

TSV's material can be *Copper Tungsten (W)* [57, 58], *Copper (Cu)* [59, 60], or *Poly-Silicon (Poly-Si)* [61]. Notably, the TSVs' materials have different Coefficient of Thermal Expansion (CTE) when compared with the substrate, which is considered as the most important issue of TSV-based 3D-ICs under operating conditions. Specifically, Copper TSVs have lower resistivity and film stress; but, with a higher CTE difference than Tungsten [58]. On the other hand, Poly-Si is the most stable material among all of them; but, it suffers from higher resistance [62]. Figure 2.4

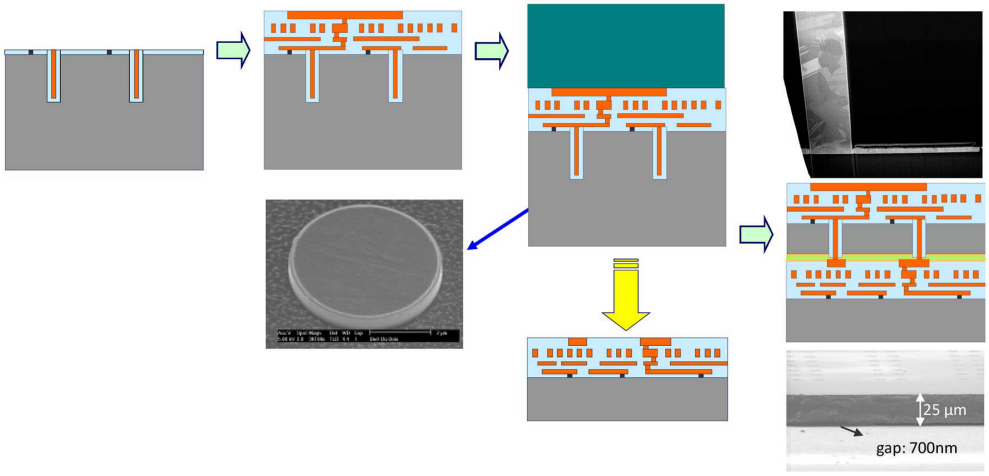

Figure 2.4: Process flow of a 3D-Cu TSV technology: Die-to-wafer stacking is performed with simultaneous Cu-Cu thermo-compression to create mechanical and electrical connections simultaneously [56].

shows the fabrication process for Cu-based TSV.

2.1.3 Challenges of 3D-integration

Because TSV-based 3D-IC technology is still new and immature, it is challenged by several obstacles. They need to be overcome to be widely adopted. Below is a summary of the key challenges:

- **Manufacturing Cost.** TSV fabrication requires extra steps: via etching, thinning, micro-bump forming, bonding and testing. In addition, the low yield of TSV-based systems (see Table 2.2) also increases the manufacturing cost.

- **Design, Tool and Flow.** Because 2D-integration is different in comparison to 3D-integration, design tools and flows need to be modified to support TSVs.

- **Reliability and Testing.** Due to the high defect-rate of TSVs, the reliability is an important challenge. Moreover, by stacking multiple dies together, a failed die also leads to the total system failure.

- **Thermal hotspots, Stress and Power Dissipation.** By stacking multiple layers, thermal removal becomes extremely difficult. Therefore, thermal hotspots are expected to appear in 3D-ICs. Moreover, due to the difference in thermal expansion coefficients of materials and hotspots, 3D-stacked ICs encounter the mechanical stress.

Because TSVs and 3D-ICs are presenting a major shift in the system architecture, the risk of early fabrication is obvious. Therefore, recent researches have more and more focused on TSV-based structure, and are expecting to be widely used in the near future. In the scope of this dissertation, the reliability of TSV-based 3D-ICs is discussed.

2.1.4 Dissertation focus

As previously discussed, several technologies have been proposed to adopt the 3D integration technology. In fact, there are existing products using such a technology [32, 46–50]. By following the current trends of vertical integration, this dissertation aims to focus on the TSV technology, which is predicted to be maturely available soon and can significantly benefit the existing techniques. Although TSV-based integration is the focus of this dissertation, the proposed architectures, algorithms and techniques can be applied for future technologies with few modifications and considerations.

Section 2.1.3 discussed the challenges of 3D integration, which mainly focuses on TSV-based 3D-ICs. Among these challenges, the reliability and testing are one of the most critical issues as they will decide the popularity of 3D integration in the near future. In fact, reliability is considered as the most important criteria for critical applications: space, medical, autonomous, and so on. Since the future of 3D integration is undeniable, considering highly reliable architectures and algorithms is definitely demanded. Meanwhile, due to the vulnerability of 3D-ICs, the reliability crisis for interconnection is foreseen.

2.2 Interconnection Reliability Crisis

Figure 2.6 shows a taxonomy of the reliability issues in 3D-ICs. Here, they are classified into three branches: soft errors, hard faults and TSV defects. Examples of hard faults and soft errors are illustrated in Fig. 2.5. Figure 2.5 (a) shows an open wire defect where there is a crack inside the wire. Consequently, the resistance of wires is increased. Instead of transmitting the signal '1' (the voltage value is *Vdd*), the output of the wire does not pass the threshold voltage due to the high wire resistance. As a result, at the sampling time, the output value is determined as '0' instead of '1'. Figure 2.5(b) shows a single-event transient where a possible particle effects the output of an AND gate. At the output port, a glitch occurs and is sampled as logic value '1' instead of the

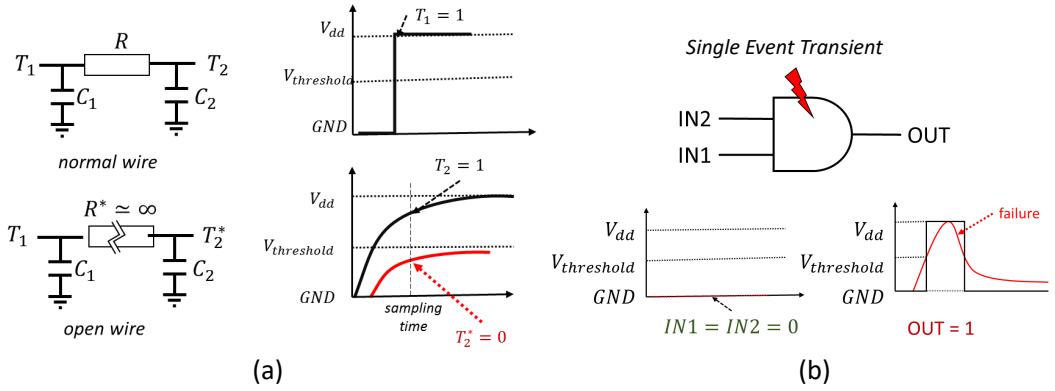

Figure 2.5: Illustration of faults: (a) Open wire defect (hard fault); (b) Single event transient (soft error).

correct value '0'; thus, resulting in an error. In later clock cycles, the transient effect can disappear and the gate operates back normally.

Hard faults, including both permanent and intermittent faults, can occur during the manufacturing stage or under specific operating circumstances. Intermittent faults periodically occur during operation and can disappear after a certain time. For instance, the crack in Fig. 2.5 (a) creates timing violations at a high temperature; however, it still allows normal operation at cooler conditions. Because these faults do not permanently damage a given component, it can pass through several testing stages. Nevertheless,they can still cause operation failures. Although intermittent faults can disappear after a specific period of time, their inconsistency can be treated as permanent faults to avoid complex situations. For both permanent and intermittent faults, the most natural solution is using redundant components [63, 64].

On the other hand, soft errors arise from energetic particles, such as alpha particles and neutrons from cosmic rays, generating electron-hole pairs as they pass through a device. A sufficient amount of accumulated charge may invert the state of a logic device such as a latch, gate or SRAM cell; thereby, introducing a logic fault into the NoC's operation. Soft errors do not permanently defect the gate and only occur over a short period of time. Because of their special characteristics, they are unpredictable and unavoidable. Unlike permanent and intermittent faults, transient faults cannot be fixed by replacing the affected components. Instead, they can be recovered by repeating the erroneous operation. A transient failure inside the data path can also be fixed by using code-based techniques (e.g., Error Correction Code (ECC) [65]). Statistically, transient faults are the most common kind of faults accounting for 80% of failures, as reported in [44]. Therefore, without

Table 2.2: TSV Defect Rate Summary.

Work	TSV Pitch	Defect Rate	TSV Number	Yield w/o Spare
IBM'05 [75]	$0.4\mu m$	13.9E-6	1k-10k	95% 98%
IMEC'06 [76]	$10\mu m$	40.0E-6	10k	67%
HRI'07 [77]	-	9.75E-6	100k	68%
HRI'09 [78]	-	7.95E-6	100k	$\geq$90%
SAMSUNG'09 [32]	-	0.63%	300	15%

an efficient protection mechanism, these errors can compromise the system's functionality and reliability.

Table 2.2 summarizes the defect-rates of TSV-fabrication [66]. The TSV defect-rate is considered extremely high which negatively affects the final yield. In [32], 0.63% of the TSVs is reportedly defected and the yield without spare is only 15%. Besides the high defect rates during the manufacturing stage, TSVs under operation also face several challenges related to stress and thermal issues [33, 67–71]. As a result, TSVs are one of the most vulnerable components in 3D-ICs.

Defects in TSVs can be classified into three types: open (or void), stuck-at and bridge (shorted between two or more TSVs). Figure 2.7 show cross-sectional chip photographs of the void and pinch-off defects (note that pinch-off is also classified as an open defect due to the high resistance of the TSV). If the defect of a TSV is partial, it may only lead to timing issues when the transmitted signal arrives late.

In [30, 72, 73], the authors modeled the TSV and its defects as RC models where additional open or short resistors are added. The RC models of its failure are depicted in Fig. 2.8. A TSV, which handles the connection between the top layer and the bottom layer, is modeled with its resistance and capacitance. When the open defect occurs (depicted in Fig. 2.8 (b) by the white area inside the TSV), an additional resistance R_{open} is put in the RC model [73, 74]. Similarly, R_{short} and R_{bridge} are added to represent the short-to-substrate and bridge defect, as depicted in Figs. 2.8 (c) and (d), respectively. Depending on the value or the additional resistance, the TSV function is corrupted or only timing violations occur. For example, in [73], the authors classified three levels of R_{short}: $R_{short} > 20K\Omega$ - good; $20K\Omega > R_{short} > 10K\Omega$ - repairable (creating timing issues); $R_{short} < 10K\Omega$ - total defect.

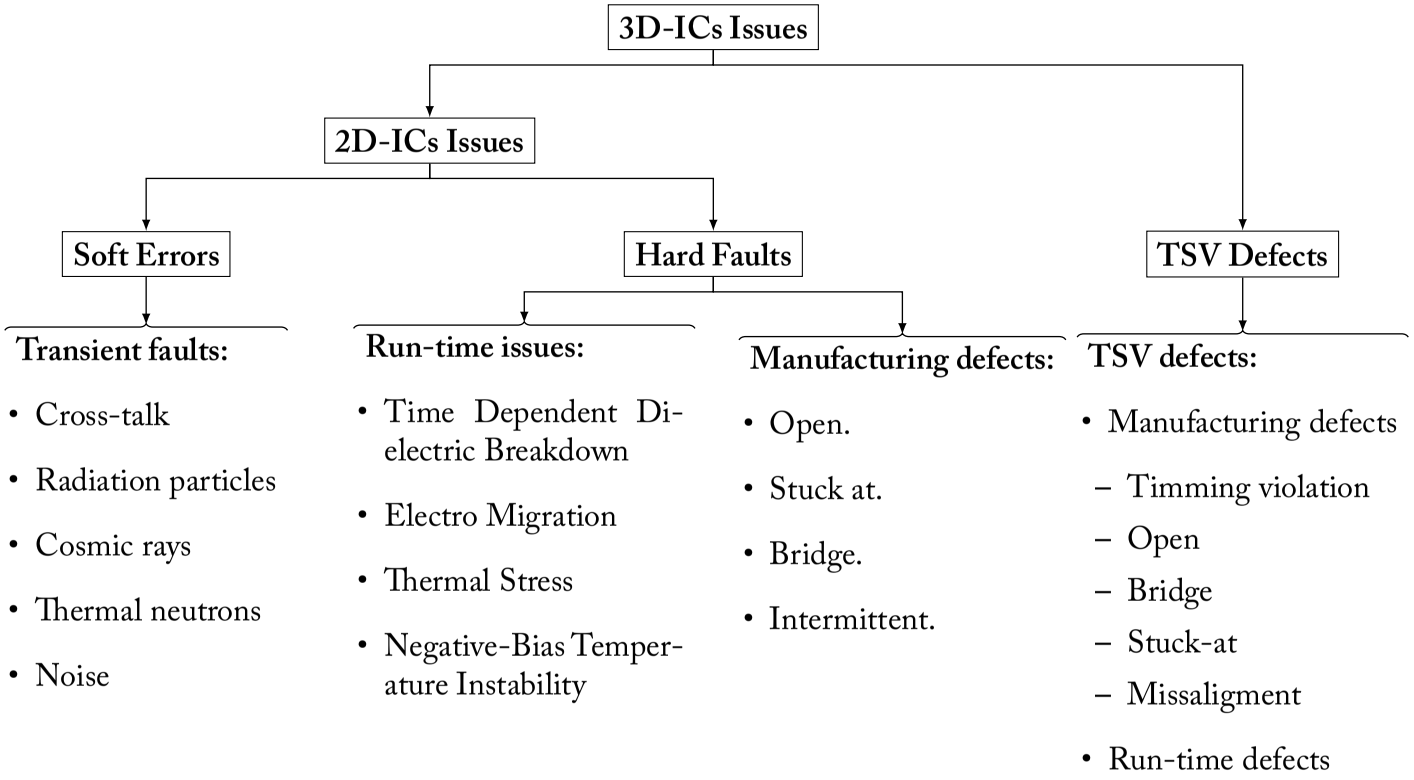

Figure 2.6: Taxonomy of reliability issues in 3D-ICs.

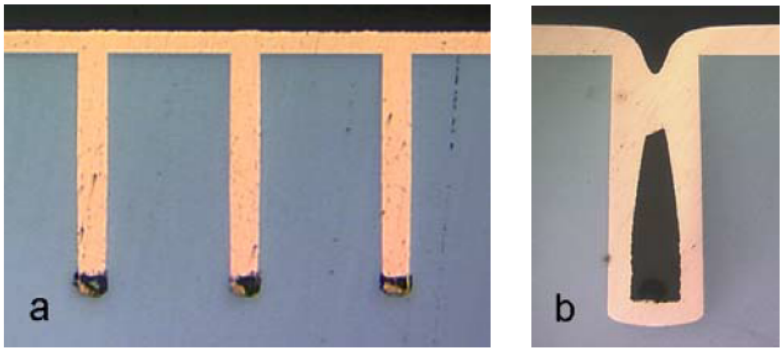

Figure 2.7: TSV defects: (a) Void; (b) Pinch-off [79].

The TSV failure distribution is still under investigation. In general, there are two main assumptions regarding failures: *Random* and *Clustering* distributions [30, 80, 81]. Random TSV defects are efficiently dealt by adding redundancy and recovery methods; however, *clustering* defects are considered as a big challenge. Moreover, TSV misalignment [82] also may occur and it is classified as a cluster defect. Because of the stress and thermal issues, TSVs may also be defected after manufacturing. In [35], the authors presented several *Mean Time To Failure* equations for 3D-ICs caused by *Time Dependent Dielectric Breakdown*, *Thermal Cycling* and *Electro-migration* where the temperature values play an important role. Because of the cluster effect on hot-spot areas in 3D-ICs [56], the obvious result is the cluster TSV defect.

2.3 3D Circuit Architecture

3D integration is proposed to solve the interconnection issues for deep sub-micron era. As a result, there are more and more gates inside a single chip. The system can be extremely complex and, consequently, 3D-ICs may require different organizations and infrastructures, or even software architectures. This section highlights the possible solutions, where 3D-NoCs are considered as the most promising one.

The most natural method of shifting to 3D integration is using vertical connections as normal wires. There are two cases in this method: (1) Splitting a module into multiple planes; (2) Changing from off-chip interconnection to vertical connection. Because the two cases do not require a complex design, they are simple to implement. However, since the system is becoming more and more complex than before, and it is expected to have multiple cores/PEs inside a single chip, the 3D architecture is investigated more carefully. Here, 3D architectures are classified into three

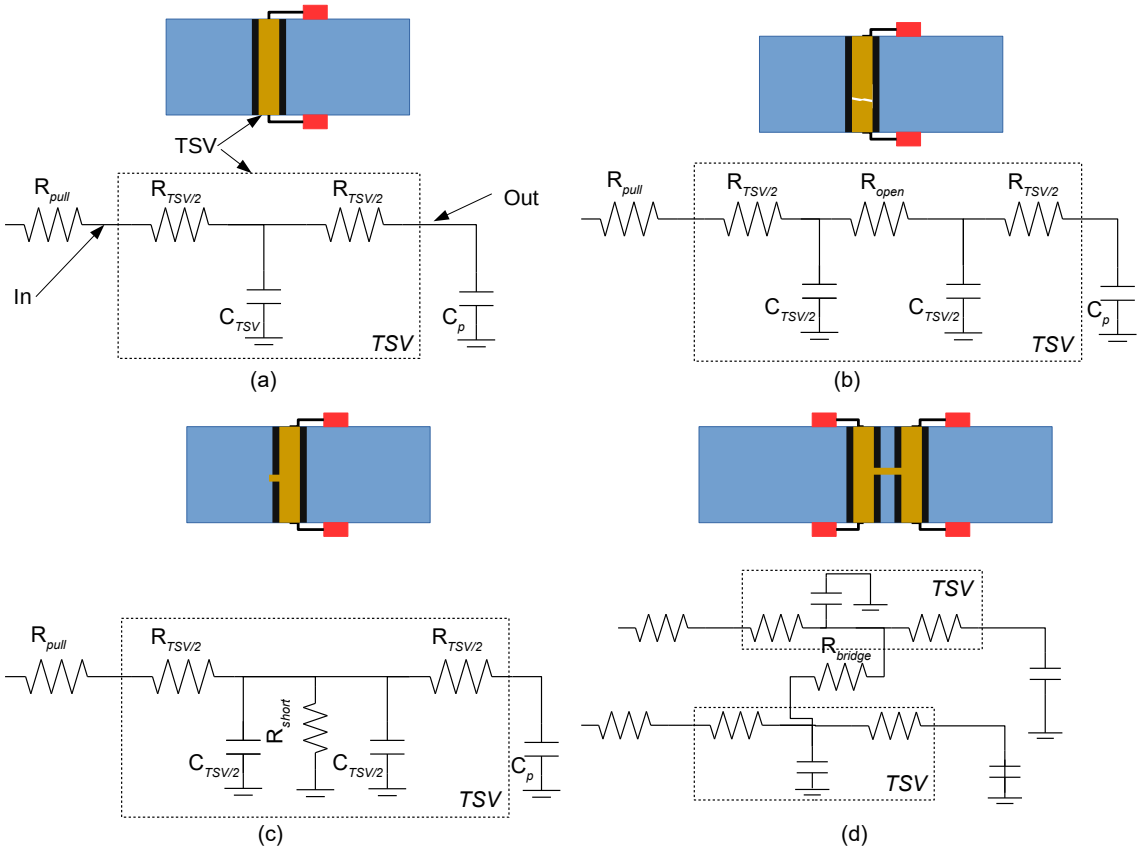

Figure 2.8: TSV RC modeling: (a) Healthy TSV; (b) Open defect TSV; (c) Short-to-substrate TSV; (d) Bridge TSVs.

main categories:

1. 3D Microprocessor and Memories.

2. 3D Field Programmable Gate Array (3D-FPGA).

3. 3D Network-on-Chips (3D-NoCs).

Since the 3D-FPGA is a development platform and 3D-NoC is an interconnection paradigm, there are overlapped areas between them. In fact, 3D microprocessor, memories and 3D-NoCs can be implemented into 3D-FPGAs without the need for significant modifications (e.g., they may only require a proper tool). Microprocessors can be replicated into multi-core systems, which require NoCs as the interconnection infrastructure instead of bus or point-to-point. With the help of 3D-NoCs, microprocessors are put in a single plane and the restrictions on their designs can be simplified. The timing, communication and reliability analyses are shifted then to NoCs.

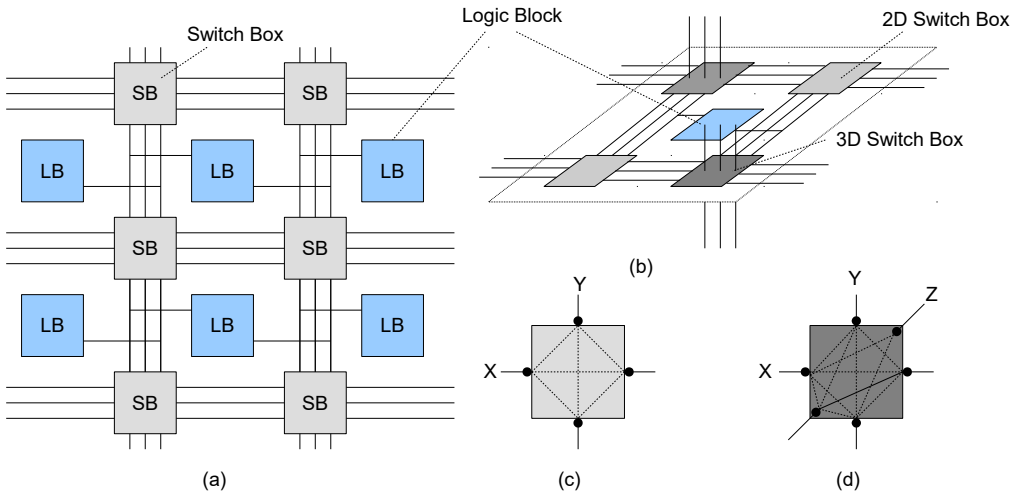

Figure 2.9: FPGA design: (a) 2D-FPGA structure; (b) 3D-FPGA structure; (c) 2D Switch Box (SB); (d) 3D Switch Box.

2.3.1 3D FPGA

Because this dissertation focuses on the 3D Integration, which mostly considers VLSI implementation, 3D-FPGAs [83–86] are discussed here as the architecture, not as the development platform. Figure 2.9 shows the brief structure of 3D-FPGA with its dedicated Switch Box (SB) [87]. The architecture of 3D-FPGAs is similar to 2D-FPGAs inside the same plane. To allow vertical signals traversal, there are some 3D SBs inside the 3D-FPGAs. These 3D-SBs handle the vertical connection by selecting the *Z* direction. Depending on the fabrication technology, vertical connections can be implemented differently. Recently, the common method is grouping them together and using TSVs as the inter-planar connections. Because TSVs utilize large area, the number of 3D-SBs is limited. Two main FPGA vendors (Xilinx and Altera) have recently launched 3D-ICs FPGA products[86, 88] which are important stepping stones for the 3D-FPGA architecture.

As a development platform, a 3D-FPGA can be programmed as a specific circuit. Because the 3D SBs allow the vertical connection, the development flow needs to be modified to adapt to the new architecture.

2.3.2 3D Microprocessors and Memories

3D Microprocessors have been extensively investigated in several works [50, 89, 90]. In these studies, TSVs, *wire bonding* or *wireless stacking* are used to handle the communication between layers. Authors in [89] rearranged the floor-planning process for 3D architecture instead of 2D.

Table 2.3: Performance and power enhancement of 3D over 2D architectures [87, 91].

# of input bits	Kogge-Stone Adder			Brent-Kung Adder
	16-bits		32-bits	
	Delay	Power	Delay	Delay
2 planes	22.23%	8%	9.6%	13.3%
3 planes	23.60%	15%	20%	18.1%
4 planes	32.70%	22%	20%	21.7%

Sean et al. [50] use the inductive-coupling based link ($f = 600MHz,\ 16bits$) instead of TSVs for the inter-layer communications. The block diagram of this architecture is shown in Figure 2.10(a). The photograph of the implementation is depicted in Figure 2.10(b). Table 2.3 shows the comparison between the processor components' implementations in 2D and 3D architecture [87, 91]. Here, the 3D architecture offers better delay and power consumption because of shorter wire length.

Memories can also be implemented into 3D architectures [32, 92, 93]. Figure 2.10(c) shows the block diagram of an 8Gb DDR3 memory which is implemented using TSVs. The master chip is put on the bottom layer and the DRAM cores are put on the top one. The communication between the master and the slave are handled by TSVs. In comparison to traditional 2D memories, 3D ones have smaller footprint and shorter wire length between the master core and the slaves.

2.3.3 3D-Network-on-Chips

With the continuous trend of IC development, hundreds then thousands of cores are expected to be integrated into a single chip [8]. Because the traditional on-chip communication paradigms (e.g., bus, point-to-point) do not provide better scalability, future ICs need alternative interconnect fabrics. Recently, several multi-cores systems [9, 13, 14] have used packet-switched NoCs as the backbone of their communication. In NoC architectures, PEs are connected to the network through Network Interfaces (NIs) and the communications are performed in parallel using specific protocols.

For 3D-ICs, NoCs can evolve into 3D-NoCs [94, 95, 95, 96] where a 3D mesh topology can be used. By adding the vertical connections, 3D-NoCs can support the inter-die communication without overwhelmingly complicating the infrastructure. However, 3D-NoCs are also exposed to several challenges such as synchronization and reliability. As the main theme of this dissertation, the reliability issue is focused on. Since solutions to solve the reliability in 2D-NoCs have been

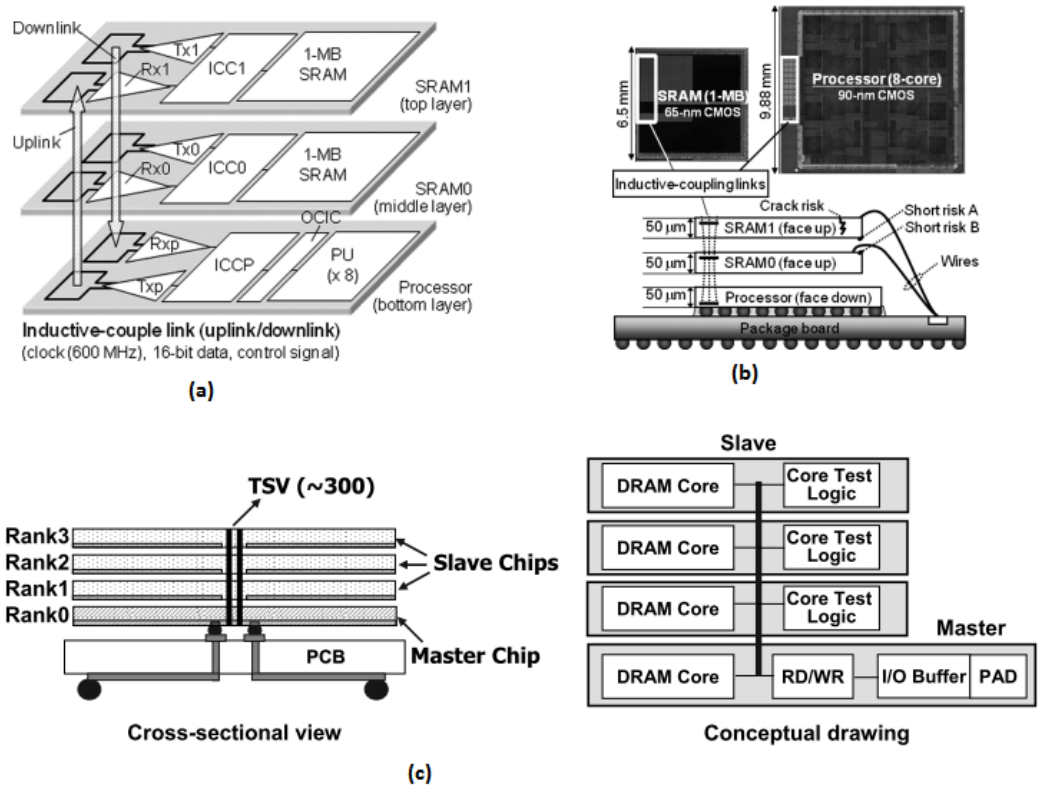

Figure 2.10: 3D architectures: (a) 3D Microprocessors and memories systems [50]; (b) DDR3 memory block diagram [32].

extensively researched in the last decade, some of them can be adopted in 3D-NoCs. This section shows the architecture of 3D-NoCs, the reliability challenges and the possible solutions.

As depicted in Fig. 2.11, a TSV works as an inter-layer wire in 3D-NoCs, as well as in 3D-ICs. By creating vias, thinning the wafer and performing a thermo-compression [56], the TSVs are established and the two wafers can connect through them. TSVs are usually fabricated regularly into a group or a cluster, or irregularly in random positions. For 3D-NoCs, the most typical method is putting TSVs near their corresponding routers.

A NoC system, as shown in Fig. 2.12, consists of three main elements: (1) Routers: they are connected between each other via point-to-point channels. (2) Network Interfaces (NIs): constitute the interface between the router and the attached PE. (3) Processing Elements (PEs): execute the program and they are connected to the NoC via NIs.

A router includes three main parts: *input ports*, *routing units*, and a *crossbar*. Data is divided into a set of packets where each packet consists of several flits which are obtained by the flitizing process. A flit travels from source to destination through the network with the help of routers.

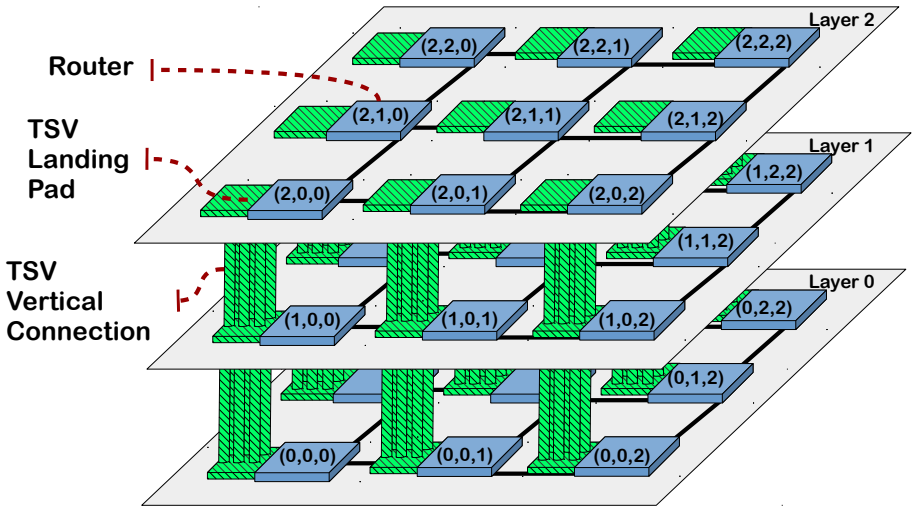

Figure 2.11: Structure of a typical 3D Network-on-Chip system.

Inside a router, an incoming flit is stored in an *input port*, routed by *routing units* (virtual channel, switch allocation, and intra-router arbitration) and physically forwarded to the next node by a *crossbar* channel. The flit is transmitted via an inter-router channel to the next router or to an NI. The routing path of a packet is decided by the *routing unit*. After completely transmitting a packet, the routing configurations of the packet is released for future packets.

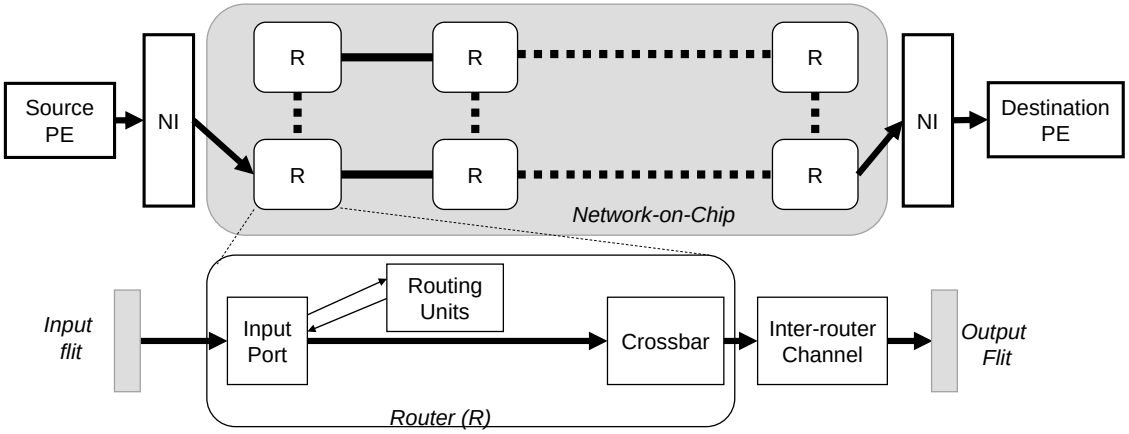

Figure 2.12: Network-on-Chip simplified block diagram.

2.4 3D-NoCs: Design for Reliability

As discussed in Section 1.2, *Design for Reliability* is the method to ensure a highly reliable design [42, 97]. For 3D-ICs and 3D-NoCs, designers also need a similar approach: from *Specification/Preliminary Design* to *Design* and *FAIT* [43]. Figure 2.13 depicts a brief overview of *Design*

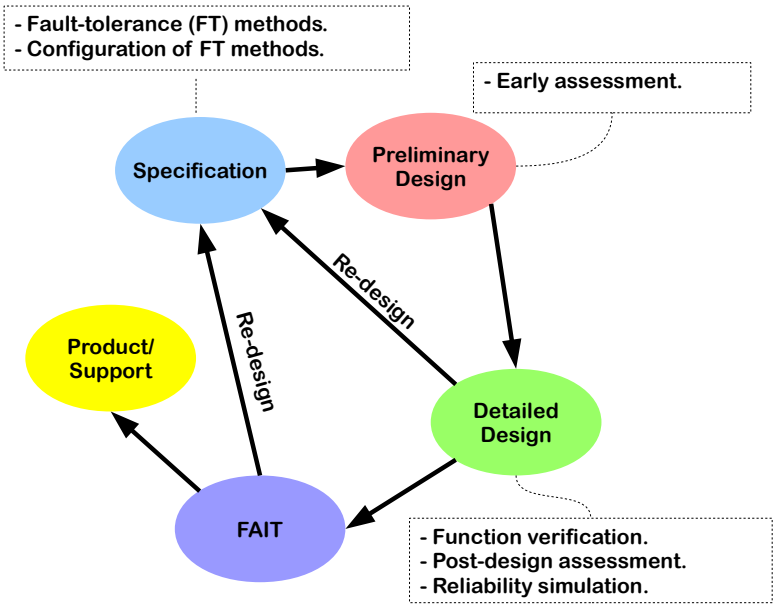

Figure 2.13: Design for Reliability.

for Reliability. In the *Specification* stage, designers select the fault-tolerant (FT) methods and their configurations (e.g., number of redundancies). In the second stage, designers should do early assessment to estimate the reliability of the system with the aid of the FT methods. After finalizing the specifications, the system is designed and forwarded to the next stage: *Detailed Design*. By having a completed design (e.g., Hardware Description Level code, layout), designers can first verify the correctness of the FT methods. Here, they can manually inject faults and check the operation accuracy. Later, they can perform post-design assessment or reliability simulations. If the system matches the reliability requirements, it is forwarded to the *FAIT* stage. Otherwise, re-design is required to enhance the reliability. Similarly to the *Detail Design*, *FAIT* also checks the reliability and decides whether to finish the development or require a re-design process. Among the above processes, *FAIT* is the most time-consuming one. Therefore, the reliability of the system needs to be carefully assessed before this step. This is because designing a complex system requires numerous hours of development where the redesign time should be eliminated or minimized as much as possible. In summary, in *Design for Reliability*, the early assessment in *Preliminary* roles an important part to reduce the risk of re-design.

In this dissertation, the main focuses are fault-resilient architectures and algorithms for 3D-ICs where 3D-NoCs are the main target. Therefore, the *FAIT* stage is not addressed. Here, the thesis

mainly works with FT methods and the reliability assessment. The remaining parts of this section provides an overview of FT methods and reliability assessment for 3D-NoCs. Later, Chapter 3 describes the already proposed approaches in more details.

2.4.1 Fault-Tolerance for 3D NoCs

Similarly to the fault-type classification, fault-tolerance for 3D-NoCs is classified into three types: soft errors, hard faults and TSV defects. This classification is necessary as these types have different characteristics, probabilities and impacts on the system.

Hard fault handling schemes are based on two main approaches: (a) FT routing algorithms, which enable packets to avoid faulty nodes in the network [63, 94]; (b) architecture-based methods, which use hardware (components) redundancy and/or reconfiguration to recover from faults [63, 64, 98]. Soft error recovery is also performed by two main schemes: (a) data corruption handling using Error Correction Code (ECC) based methods [65, 99, 100]; (b) control logic handling using temporal redundancy based methods [101–103].

Existing works on TSV-defect presented so far have dealt with reliability in different approaches: (a) improving the manufacturing process to enhance the reliability of TSVs [104]; (b) accounting the potential defects in the design stage [33]; (c) correcting the defected TSVs by using supporting circuits [73, 105], redundancy [74, 80, 82], or *ECC* [65]; (d) using an alternative channel to avoid the defected TSV channel (e.g., using FT routing [94] in 3D-NoCs).

Although these works have impressively enhanced the reliability of TSV-based systems, there is still an existing issue in the fault distribution. Most of the first conducted works addressed the random distributions [82, 106]; however, the cluster defect distributions [30, 80, 81] are recently considered as the most realistic one. In order to deal with the cluster TSV defect, most works aim to select a suitable grouping configuration [81] to distribute TSVs on different positions [30] or to enhance the redundancy correction rate [80]. Although these methods can improve the reliability of the system, adding extra redundancy and complex arbitration result in penalties in terms of area cost, wire latency and power consumption. Moreover, if the number of defective TSVs is larger than the number of assigned redundant ones, the vertical connection is corrupted. Therefore, we observe that a better management solution can help to deal with this issue, especially for 3D-NoCs, where low utilization rate of the TSVs has been reported [107, 108].

2.4.2 Reliability Assessment

For the early reliability assessment, there are three basic methods: *Physical-analysis*, *System-level simulation* and *Analytical-model*. The *Physical-analysis* method [109–111] focuses on the device reliability under specific conditions. The whole system's reliability can be synthesized from its all elements. A major drawback of physical-analysis is the complexity of evaluation for large systems which may involve a huge amount of analyses. *System-level simulation* [41, 43, 65, 112, 113] requires having the fully implemented system characteristics (in *Detail Design* stage) which is difficult to be obtained in the system definition or the preliminary design phase. On the other hand, the *Analytical model* is popular in the computer network's reliability assessment [114, 115]; but, it is not widely applied in integrated systems. Specifically, NoCs reliability assessment is still immature. Recently, works in [35] and [116] have provided good analyses regarding *TSV failure* and a *baseline NoC system*. Because of the continuous increase in terms of complexity, NoCs assessment using physical-analysis and system-level simulations is no longer suitable enough, and the need for a more efficient analytical method has become primordial.

2.5 Conclusion

This section has introduced the background of the conducted research. It also classified the issues of future 3D-ICs and the motivation underlying this dissertation. In summary, because of the vulnerability of 3D-ICs, their reliability need to be enhanced. First, fault-tolerant methods are needed to help the system dealing with the possible failures. The fault-tolerant methods should well cover the most common issues: hard faults, soft errors and TSV defects. Second, these methods need to be properly assessed to help designers understand the enhancement in terms of reliability. In the next section, some of the important related works to this research are summarized and discussed.

3

Related Works

In the last decade, numerous of works have addressed the related issues to fault-tolerance in NoC architectures. Recently, 3D-NoCs also bring several fault characteristics that need to be tolerated. This chapter reviews some of the well-known solutions presented to solve the potential reliability issues. The existing techniques are classified in three main categories: architecture, software, and integration. As a part of *Design for Reliability*, this chapter also reviews the reliability assessment for IC design and focuses on NoC systems.

3.1 Fault-Tolerance Approach

Table 3.1 shows the taxonomy of fault tolerance in 3D-NoCs. Here, the FT methods are classified into three approaches:

- *Architecture*: use built-in circuits to recover the faults (e.g. adding redundancy or self-

configuring the system to deal with the presence of faults). The failed part is abandoned and possibly replaced by a redundant component or its task is handled by other parts.

- *Software*: use alternative revisions of the system's program to handle the possible failure. This method is normally applied to the computing-related block, where the FT methods can refine the output to help the system deal with the faults. Re-executing the operation is also considered as a viable method. For instance, fault tolerant routings can aware the occurrences of faults and avoid them in the selection process.

- *Integration*: this method focuses on the physical layer, where the defect is repaired or considered in the design process.

Table 3.1: Taxonomy of different error recovery protocols and architectures in NoCs. Classification: "A" for architecture, "S" for software and "I" for integration.

Fault Type	Position/Type	Fault Tolerant Method	Approach
Soft Errors	Data Path	Automatic Re-transmission Request [99]	S
		Error Detecting/Correcting Code [65, 100]	S
	Control Logic	Logic/Latch Hardening [41, 101]	A
		Pipeline Redundancy [103]	S
		Monitoring and Correcting model [102, 117, 118]	S
Hard Faults	Routing Technique	Spare wire [119, 120]	A
		Split transmission [121]	A
		Fault-Tolerant routing algorithm [63, 94]	S
	Architecture-based Technique	Hardware Redundancy [64]	A
		Reconfiguration architectures [63]	A
TSV Defects	Redundancy	Shifting [32, 81]	A,I
		Crossbar [82]	A,I
		Network [80]	A,S,I
	Management	Design awareness [33, 73]	I
		Randomly distributed redundancy [30]	A,I

3.1.1 Architecture approach

Figure 3.1 depicts the two basic methods in the architecture approach. Besides the original module, the redundancy technique adds a replica for recovery. On the other hand, self-configuration does not utilize extra area cost and use the remained modules to handle the task of the failed module. As shown in Figure 3.1(c), the task of Module 1 is handled by Module 3 and 5.

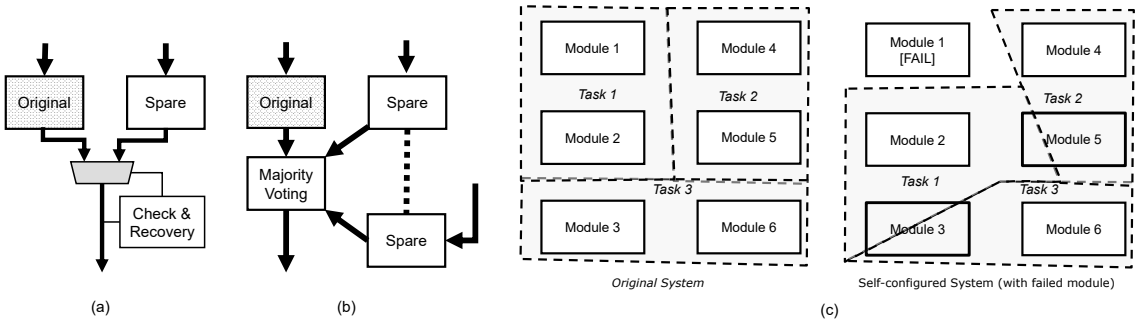

Figure 3.1: Hard fault tolerance using architectural approaches: (a) Redundancy with checking; (b) Redundancy with majority voting; (c) Self-configuration.

Hard Fault Tolerance

The architecture approach is efficient for permanent faults: hard faults or TSV defects. The defected part is replaced by a healthy part to keep the system run correctly. For NoCs, adding redundancy [64] or self-configuration [63] to the system in order to work around the defect can be found as two common solutions. In previous works conducted at our laboratory[1], several hard fault-tolerant methods were developed to deal with faults on input buffer [98], crossbar [98], and inter-router link [94, 122, 123]. This dissertation also adopts these techniques to enhance the reliability of the proposed system. More details about these techniques are presented in Section 4.2.

In the routing approach of hard fault tolerance, the failed parts can be isolated and the connections are maintained by using spare-wires [119, 120], split transmission [121] or fault-tolerant routing algorithms [63, 94]. Figure 3.2 shows three examples of spare-wires, split routing, and fault-tolerant routing. When a wire is failed, the spare-wire technique uses a redundant wire as the replacement to maintain the connection. On the other hand, the fault-tolerant routing technique uses an alternative transmitting path to deal with the fault. Split routing, or serialization, uses a part of the healthy links for maintaining the connection. When wires are failed, the system can use serialization and deserialization which require multiple clock cycles to transmit a single flit. Figure 3.2(c) shows the 50% serialization mode, where only half of the wires are used and the system needs two clock cycles for transmitting a flit. Note that the fault-tolerant routing can be considered as a software approach.

[1]Adaptive Systems Laboratory, The University of Aizu, Aizu-Wakamatsu, Fukushima, Japan.

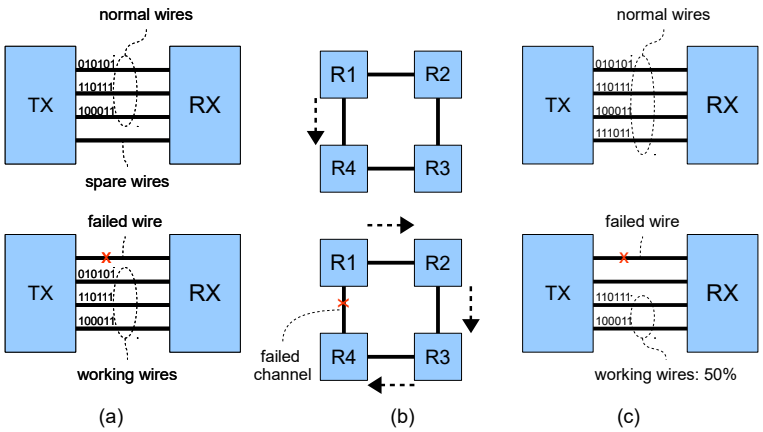

Figure 3.2: Hard fault tolerance using routing approaches: (a) Spare wire; (b) Fault-tolerant routing; (c) Split routing.

Soft Error Resilience

On the other hand, soft error resilience can use Triple Modular Redundancy (TMR) (see Fig. 3.1(b)) which can deal with both soft errors and hard faults at the same time. By triplicating the original module, the system gets three results at the same time [41]. The three results are sent to a *Majority Voting* module to decide the accurate result. Although this technique suffers from high area overhead and power consumption (about 300%), it is easy to implement and effective for both soft errors and hard faults. This method is feasible for TSV defects; however, due to the high area overhead, it is rarely considered.

TSV Defect Recovery

For TSV defect recovery, the most common method is adding redundant TSVs to correct the defected ones [74, 80, 82]. The major concern in this method is to efficiently route from a defected TSV to a spare one. There are four basic solutions: (a) signal switching [32], (b) single shifting [106], (c) crossbar [82] and (d) Network TSV routing [80]. Figure 3.3 shows examples of shifting techniques, Double TSV and Network TSV. The redundancy technique, which is the shifting, switching and crossbar method, use redundant TSV for recovery. When a TSV defects, its function is handled by a redundant TSV. Network TSV routing [80] is one of the most advanced techniques in redundancy mapping. A light-weight network is created for routing the terminal having failed TSVs to a spare/redundant ones. By creating a network, it can utilize the redundancies better than other approaches. Because of the cluster defect, the TSVs located nearby

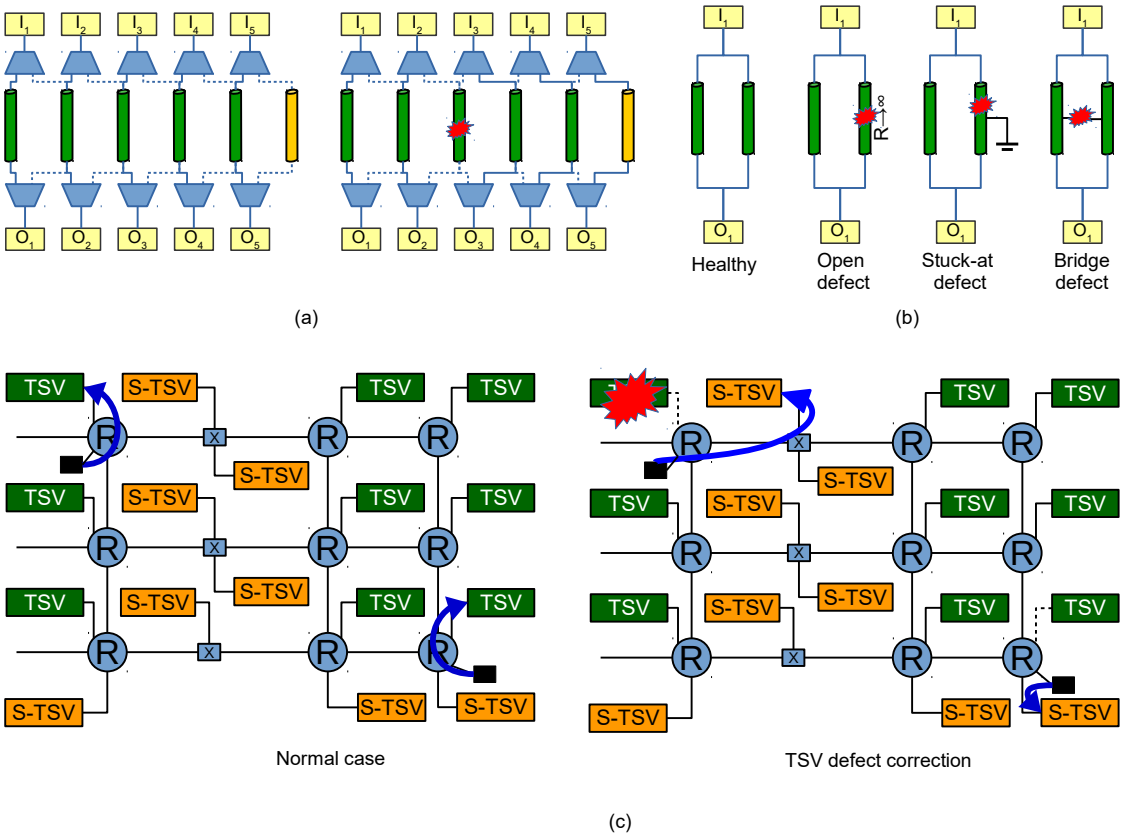

Figure 3.3: TSV Fault Tolerance: (a) Redundancy Technique; (b) Double TSV; (c) Network TSV.

the defected one have a high probability of failure. Therefore, grouping them together makes a group with many defected TSVs. This leads to increase the redundancy to deal with this kind of defects. In [30], the authors propose a mapping method to reduce the impact of cluster defects. TSVs in the same group are mapped to a random position with the help of an optimization process. On the other hand, *Zhao et. al* [81] analyze the grouping method to achieve the best recovery. The work presented in [80] introduces an innovated method for TSV mapping by creating a network, and implements an algorithm for re-routing the defected TSVs.

3.1.2 Software approach

In the software approach, the basic methods are (1) program redundancy [124, 125] and (2) creating a checkpoint and rolling back when a fault is detected [103]. Figure 3.4 depicts these two methods. In the *programming redundancy*, copies of the program are executed in parallel to detect and correct the occurred faults. A voting module can be used as a recovery. The second method creates checkpoints of the process and when a fault is detected, the system rolls back to

the previous checkpoint or completely restart.

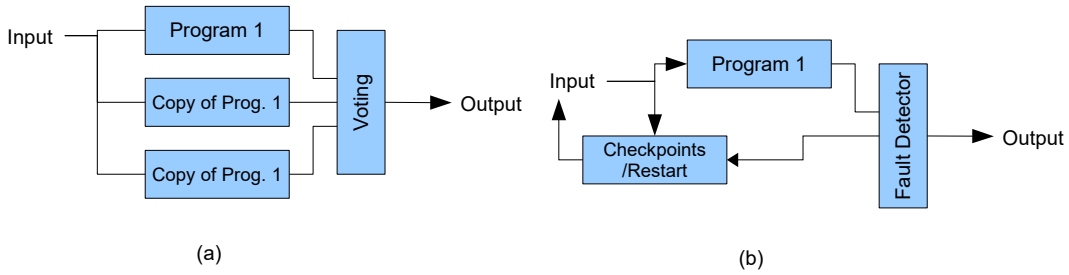

Figure 3.4: Software approach: (a) Program redundancies; (b) Checkpoint/Restart and Roll-back.

Hard Fault Tolerance

Software approaches are limited to hard fault tolerance. Because the presence of hard faults is permanent, the system has to roll-back and usually correct. Even the fault can be corrected, it significantly effect the system performance.

However, using software for testing and finding the best recovery methods and configurations is a promising solution. In [66], the authors propose an optimization program to re-route the TSV. This program is implemented in software [80]. The work by *Shamshiri et al.* [126] and *FoReVer framework* [118] are also examples of using software programs to detect and recover the faults.

Soft Error Tolerance

The software approach is more common in soft error resilience. For soft errors, the fault-tolerance is classified into two main objects: data path and control logic. In the data path, most works use code-based techniques that not only detect the integrity of the received data; but, also provide a correction function up to an acceptable number of faults. For instance, *Bertozzi et al.* [65] analyzed several low-cost coding techniques for on-chip communication. As an adaptive solution, *Yu et al.* [100] presented a dynamic *ECC* based on the quality of the wire connection by using a configurable *ECC* with two Hamming codes to adapt to several probabilities of faults. Although this adaptive *ECC* provides energy efficiency, its area overhead is problematic. Even if the soft error cannot be corrected, the data can also be forwarded (as known as FEC: Forward Error Coding) to be recovered later.

For soft errors in the control logic, there are several techniques with cross-layer resolution. In the *End-to-End* level, *Shamshiri et al.* [126] proposed error-correction and online diagnosis using

a specific code named *2G4L*. Based on the position of the erroneous bit in the received data, the system can indicate the position of the faulty node in the network; however, when a packet is misrouted due to wrong routing information/arbitration or an adaptive routing algorithm, the path of a packet is not fixed in a way that can determine the faulty node. To ensure arbitration computation across layers, *NoCAlert* [117] implements constraints to obtain computational accuracy. By constraining the relationship between the input and output of a block, the system can detect both soft and hard faults. Although this work presents an efficient detection, it lacks efficiency in recovering from soft errors. First, the system needs to distinguish between soft and hard faults to decide the recovery method. Second, soft errors cannot be recovered by spatial redundancy and their recovery in the End-to-End level is inefficient. The *FoReVer framework* [118] also presented a network level method to detect and recover from routing errors: lost, duplicated, and misrouted packets. Since *FoReVer* is based on *End-to-End* detection and recovery, dealing with soft errors requires retransmission of the whole packet instead of an online recovery.

In [102], the authors deploy a monitoring system on important control modules. They can diagnose the output to find the failure. This technique is light-weight in both area and power and has an insignificant impact on the system performance. However, it suffers from the lack of flexibility since the monitor module has to be specifically designed depending on the target component. If any changes in the routing algorithm or pipeline stages are needed, investigation and re-design of the monitor module are mandatory.

TSV Defect Tolerance

In the *System layer*, which focuses on 3D-NoCs, fault-tolerant routing [122] is one of the most suitable solutions. To reduce the risk of thermal and stress issue in 3D-ICs, thermal-aware management [67, 127] is also a promising solution. On the other hand, most of the works proposed offline testing and recovery which is not suitable for post-manufacturing. The system operation has to be halted in order to be tested and recovered. In [74, 80], the authors presented an online testing function. Because the reliability of TSVs is a critical issue, the need for online testing and recovery is important.

3.1.3 Integration approach

Unlike the architecture approach, the integration one focuses on improving the quality of devices. Here, it mostly focuses on the *Physical layer* and physical design phases. The improvements in manufacturing technologies are considered.

Hard Fault and Soft Error Tolerance

In this type of approach, designers focus on how to protect the circuit against the potential fault source. For example, to prevent cross-talk, designers insert GND wires solely with signal wires [128]. Radiation hardened systems [129, 130] are also popular in military and space applications.

For soft errors, *Ernst et al.* [101] presented a *Razor D Flip-flop* with an additional shadow latch sampled by a delayed clock for checking the occurrence of transient faults (see Fig. 3.5). Furthermore, a soft error detection solution based on redundant latches was also presented by *Ravindan et al.* [131]. Although these techniques obtain more efficient detection results, they nearly double the area overhead and power consumption to maintain the redundant latches.

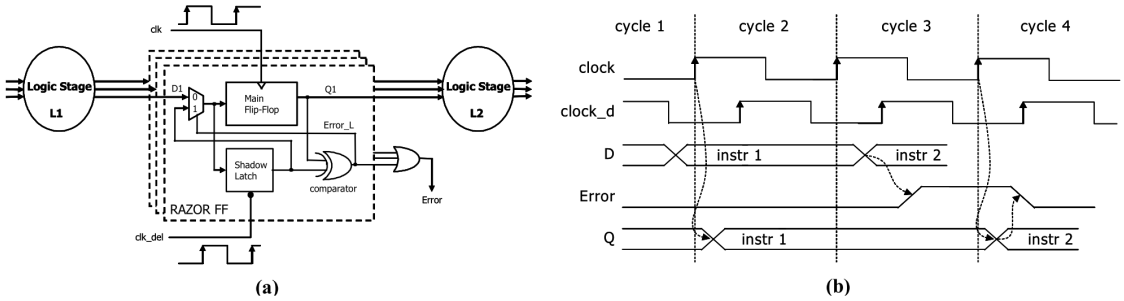

Figure 3.5: Razor D Flip-flop [101]: (a) Architecture; (b) Waveform.

TSV Defect Recovery

In *Physical layer*, the improvement in TSV manufacturing can help to reduce the defect rate [104]. Designers can optimize the physical layout, use thermal-aware routing and placement methods to improve the reliability of 3D-ICs [33, 67, 71]. Even when a fabricated TSV has a short defect, a correction circuit uses a voltage comparator to gain the output voltage of the TSV [73]. To enhance the reliability of TSVs, [105] proposed a method named *Double TSV* which uses two TSVs instead of one for maintaining the communication (see Fig 3.3(b)). If an open, short-to-

substrate or bridge defect occurs in one TSV, the communication is still performed by the other TSV.

3.2 Reliability Assessment

The last section have shown the existing fault-tolerant techniques for soft errors, hard faults and TSV defects. Most of the works use fault injection and verification as the methodology of reliability evaluation. However, designers have to fully implement the system for the test. To help them understand the reliability in the early stages, there are numerous works on reliability assessment which are presented in this section.

Table 3.2: Reliability assessment methodologies.

Method	Description	Type
Physical Failure Analysis [109]	The acceleration factors of the failure rate are defined as: Oxide, Metalization, Hot Carrier, Contamination, Package, EOS/ESD (Electrical Overstress/ Electrostatic Discharge) and Miscellaneous Failure Rate	physical-level
Soft Error Rate Estimation [110]	This simulation flow uses a random function to generate hit locations and perform analysis using HSPICE simulation. This aims to obtain the probability of soft errors with several physical parameters.	physical-level
Chip-level Soft Error Estimation [132]	The authors present a method to estimate the failure of a full-chip. It is based on timing analyses of basic devices. A combination of these analyses help build the final full-chip failure rate.	physical-level and system-level simulation
Monte-Carlo Simulation [43, 133]	A random error injection and function verification to measure the reliability of the system under failure.	system-level simulation
System Reliability Block Diagrams [43]	This model presents the logic relationship of the system components. There are five basic systems: series, parallel, stand-by, k-out-of-n and complex systems. All of these systems are analyzed using the probability theory.	analytical model
Fault-tree Analysis [43]	A graphical and logical representation of possible events occurring in a system.	analytical model
Markov Model [43, 134]	It models the system's possible failure situations as a set of states. There are possible failure rates and repair rates between two states. The final reliability of the system is obtained by calculating the probability of dead-end state.	analytical model
TSV Failure Analysis [35]	This analytical model analyzes the probability of failure of 3D-NoCs under the failure of TSVs. In 3D-NoCs, TSVs are used as the vertical connection wires.	analytical model (for NoCs)
NoCs Reliability Analysis [116]	This model estimates the reliability of a NoC system using probability theory.	analytical model (for NoCs)

Table 3.2 shows the different methodologies used to analyze a given system's reliability. Here, we categorized them into three basic methods: *Physical-level analysis*, *System-level simulation*, and *Analytical models*.

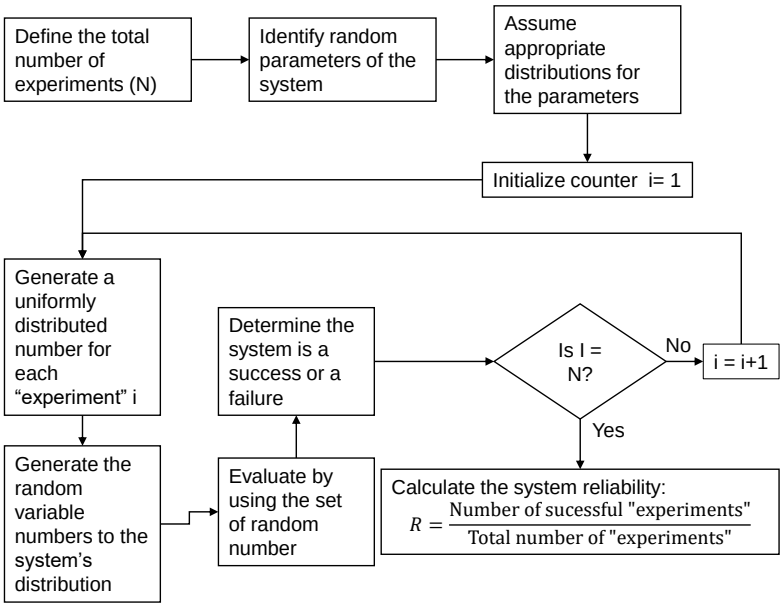

Figure 3.6: Monte-Carlo Simulation.

The *Physical-level analysis* [109–111] can obtain accurate results for gates or small devices. The task of the *Physical-level analysis* is to analyze the circuit using a specified simulator or physical analysis; however, it is time-consuming and requires a huge computation resource to analyze complex systems. The *System-level simulation* can be performed by Monte-Carlo simulations [43] or simple RTL-level simulations [41, 112, 113]. Figure 3.6 shows the flow-chart of Monte-Carlo simulation. Faults are inserted randomly and the final reliability is the average value of the successful tests. In order to evaluate the system's reliability, the *Mean Time To Failure* [42] estimation can be used. Beside analysis, design-awareness is one of the most important keys to obtain reliable systems [135–138].

The *Analytical models* [35, 38, 139–143] can be used to save a significant amount of time wasted on redesign. The IEEE 1413.1 [43] recommends several methods: *System Reliability Block Diagrams*, *Fault-tree Analysis*, and *Markov-model* for repairable systems. The most basic method is the *System Reliability Block Diagrams* where the system is modeled in logic relationships and the results are obtained by using probability theories. To reduce the system complexity, the *cut-sets* method can calculate the sub-system and merge them into the final system. The *Fault-tree Analysis* exploits the operation of a given design to obtain the possible events of the system. The above methods have a common drawback which manifests in their lack of flexibility and reconfigurability when it

comes to a reconfigurable and repairable system. This can be dealt with using the *Markov model* which analyzes the system's events, configurations, and behaviors as states and builds a graph-based model. After that, the reliability of the system can be obtained. More details about Markov state model are discussed in Section 6.2.1. Analytical models for NoCs are recently presented [144–147]. A reliability assessment for TSV failure in 3D-NoCs is addressed in [35, 148]. The reliability of a NoC is also analyzed in [116]. These methods have provided promising solutions for NoCs' reliability assessment; however, they lack the support for fault-tolerant and highly complex systems.

4

Hard Fault and Soft Error Tolerant Architecture

Reliability is becoming the critical requirement for future Systems-on-Chip, especially 3D-NoCs. In spite of the benefits of using 3D-NoCs, systems are threaten by multiple types of faults. To solve this issue, many researchers have proposed solutions for various individual aspects of on-chip reliability; however, a comprehensive approach encompassing both soft errors and hard faults pertaining to NoC reliability has yet to evolve. In addition, the error detection and diagnosis in NoC architectures have been studied thoroughly in the scope of offline testing; however, with soft errors and intermittent faults becoming a dominant failure mode in modern NoCs and general VLSI systems, a widespread deployment of on-

line test approaches has become crucial. In this chapter, comprehensive soft error and hard fault tolerant 3D-NoC architecture, named 3D-Hard-Fault-Soft-Error-Tolerant-OASIS-NoC (3D-FETO), is presented. With the aid of efficient mechanisms and algorithms, 3D-FETO is capable of detecting and recovering from soft errors occurring in the routing pipeline stages and leverages reconfigurable components to handle permanent fault occurrences in links, input buffers, and crossbars.

The organization of this chapter is as follows: Section 4.1 presents the adaptive router architecture (SHER-3DR) which is the backbone component of the 3D-FETO system. Section 4.2 and 4.3 depicts the hard fault and soft error tolerant techniques. In Section 4.4, comprehensive techniques, which include fault detection, diagnosis and recovery, are presented. Section 4.5 provides the implementation and evaluation results. Finally, this chapter presents conclusion and discussion in the last section.

4.1 Adaptive 3D Router Architecture

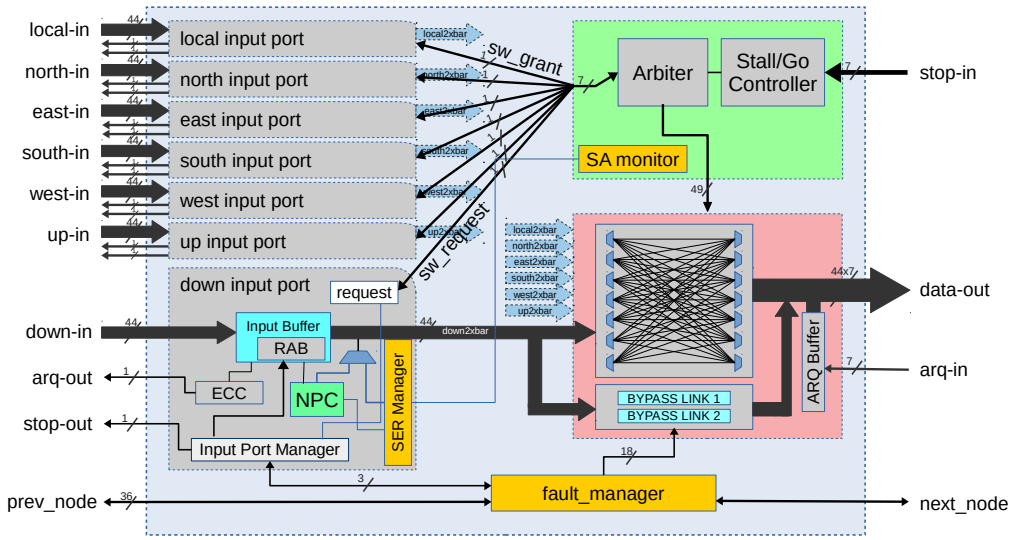

Figure 4.1: Adaptive 3D router (SHER-3DR) architecture.

Figure 4.1 shows the block diagram of the proposed adaptive 3D router architecture (SHER-3DR). The router relies on simple recovery techniques based on system reconfiguration with redundant structural resources to contain hard faults in the input-buffers, crossbar, and links, in addition to soft errors in the routing pipeline stages.

The SHER-3DR router is the backbone component of the 3D-FETO system. Each router

has a maximum of 7-input and 7-output ports, where 6 input/output ports are dedicated to the connection to the neighboring routers and one input/output port is used to connect the switch to the local computation tile. As shown in Fig. 4.1, the SHER-3DR contains seven *Input-port* modules for each direction in addition to the *Switch-Allocator*, and the *Crossbar* module which handles the transfer of flits to the next node. An *Input-port* module is composed of two main elements: an *Input-buffer* and the *LAFT routing* (Next-Port-Computing) module. Incoming flits from different neighboring routers, or from the connected computation tile, are first stored in the *Input-buffer*. This step is considered as the first pipeline stage of the flit's life-cycle, Buffer-Writing (BW). After receiving and storing the flits, their routing information is read and processed by a *LAFT-Routing* module (*Next-Port-Computing*) and an arbitration module (*Switch-Allocator*). This step is the second stage - Next-Port-Computing/Switch-Allocator (NPC/SA). After the NPC/SA pipeline stage, the *next-port* value is embedded in the flit and the *grant* signal allows the flit to traverse from its input port to an output port (Crossbar-Traversal (CT) stage).

An augmented Look-Ahead-Fault-Tolerant routing algorithm (LAFT) [149, 150] is used to perform the routing decision. If a given flit is routed to the local port, there is no routing calculation. If the flit is to be routed to another node, the faulty links information of all neighboring nodes is read by each input-port and LAFT routing is executed. The first phase of the algorithm is calculating the next node's address and its fault output information. In the next phase, the LAFT routing algorithm determines the minimal paths which are valid for routing after eliminating the faulty paths. The final routing path is selected by evaluating two factors of all the possible routing paths: (1) the diversity of the routing path to the destination node and (2) the congestion value of the connection. If there is non-minimal routing path, a similar approach is applied for the non-minimal routing paths. Finally, an output port for the selected routing is calculated. This information is embedded in the flit as *next-output-port* bits for routing in the downstream node [98].

4.2 Hard Fault Tolerance

In this work, the hard fault tolerant architectures and algorithms are adopted from the previous works in our laboratory.[1]

[1]Readers who are interested in the complete details of the hard fault-tolerance techniques, are recommended to check the works in [94, 98, 122, 123].

4.2.1 Fault-tolerant routing algorithm

To keep the benefits of look-ahead routing [149, 150], the augmented Look-Ahead-Fault-Tolerant routing algorithm (LAFT) [94] is able to perform the routing decision for the next node taking its link status into consideration and selecting the best minimal path. The fault link information received from the DDRM (*Detection Diagnosis Recovery Mechanism*) is read by each input-port where LAFT is executed. Algorithm 4.1 illustrates the LAFT algorithm. The first phase of this algorithm calculates the next node address depending on the *Next-port* identifier read from the flit. For a given node wishing to send a flit to a given destination, there is a maximum of three possible directions through the X, Y, and Z dimensions. In the second phase, LAFT performs the calculation of these three directions by comparing the X, Y and Z coordinates of both the current and the destination node concurrently. As these directions are being computed, the *LAFT routing* algorithm reads the *Next-port* identifier from the flit and sends the appropriate fault information to the corresponding input-port. By the end of this second phase, LAFT has information about the next node's fault status and also the three possible directions for minimal routing. In the next phase, the routing selection is performed. For this decision, we adopted a set of prioritized conditions to ensure fault-tolerance and high performance either in the presence or absence of faults:

1. The selected direction should ensure a minimal path and it is given the highest priority in the routing selection.

2. The algorithm should select the direction with the largest next router path diversity.

3. The congestion status is given the lowest priority.

Depending on these priorities, LAFT reads the fault status of the next node received from the *Fault-manager* module and checks the number of possible non-faulty minimal directions. As illustrated in Algorithm 4.1, if only one non-faulty minimal direction is obtained, this direction is selected as the output port for the next node. If more than one possible minimal direction is available, the algorithm selects the direction which leads to the node with higher path diversity. The diversity value for a given node is the number of possible directions leading to the destination through a minimal path. A node with high diversity results in more routing choices. This means that the probability of finding a non-faulty link is greater when considering faults. When no faults

are detected in the system, selecting the direction with the highest diversity gives more choices to find the least congested direction. As stated in [151], to obtain directions with high diversity, we should select those leading to nodes located in the center of the mesh and avoid routing to the edges of the network.

Algorithm 4.1: Look-Ahead-Fault-Tolerant routing algorithm

```
   // Destination address
   Input: X_dest, Y_dest, Z_dest
   // Current node address
   Input: X_cur, Y_cur, Z_cur
   // Next-port identifier
   Input: Next-port
   // Link status information
   Input: Fault-in
   // New-next-port for next node
   Output: New-next-port
   // Calculate the next-node address
1  Next← Next-node (X_cur, Y_cur, Z_cur, Next-port);
   // Read fault information for the next-node
2  Next-fault← Next-status (Fault-in, Next-port);
   // Calculate the three possible directions for the next-node
3  Next-dir← poss-dir (X_dest, Y_dest, Z_dest, Next_x, Next_y, Next_z);
   // Evaluate the diversity number of three minimal paths
4  Div_1 ← path-div (X_dest, Y_dest, Z_dest, poss − dir_1);
5  Div_2 ← path-div (X_dest, Y_dest, Z_dest, poss − dir_2);
6  Div_3 ← path-div (X_dest, Y_dest, Z_dest, poss − dir_3);
   // Evaluate the New-next-port direction
7  if (|Next-dir| > 1) then
8      if (Div_1==Div_2==Div_3) then
9          New-next-port ← min-congestion (poss − dir_1, poss − dir_2, poss − dir_3);
10     else
11         New-next-port ← max-diversity (poss − dir_1, poss − dir_2, poss − dir_3);
12 else
13     if (Next-dir == 1) then
14         New-next-port← Next − dir_1;
15     else New-next-port← nonminimal (X_dest, Y_dest, Z_dest, X_cur, Y_cur, Z_cur, Fault-in);
```

When the three possible directions are minimal and have the same diversity, the routing selection is made depending on the congestion of each output port. This congestion information is obtained by the *Stop* signal issued from the flow-control used in our 3D-FETO system. When no valid minimal route is available, LAFT chooses a non-minimal route while also considering the second and third priorities (path diversity and congestion), as illustrated in Algorithm 4.1.

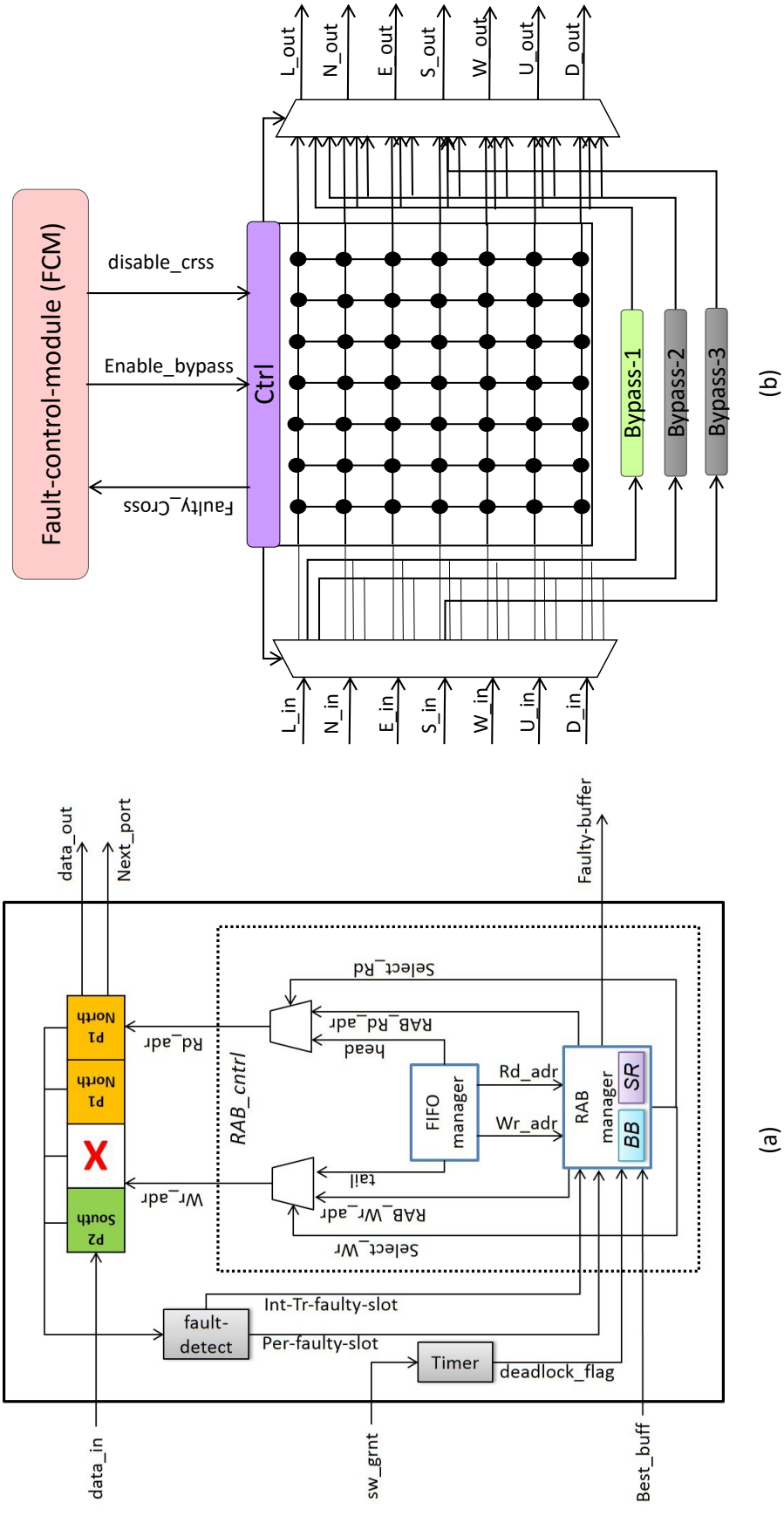

Figure 4.2: Hard-fault tolerant mechanism [98]: (a) Random Access Buffer (RAB); (b) Bypass-Link-on-Demand (BLoD)

4.2.2 Buffer Fault Tolerance

The block diagram of the hard fault recovery mechanism is shown in Fig. 4.2. The Random Access Buffer mechanism (RAB) [98] solves the deadlock problem that can occur with the look-ahead fault-tolerant routing algorithm (LAFT), and is able to recover from transient, intermittent, and permanent faults in the input-buffer. When a fault is detected in one of the slots, the main controller (located in *input port manager* in Fig. 4.1) considers the flagged slots when assigning the write and read addresses. It remains to check the flagged slots for recovery from the faults.

4.2.3 Crossbar Fault Tolerance

The Bypass Link on Demand mechanism (BLoD) [98] provides additional escape channels whenever the number of faults in the baseline 7x7 crossbar increases. When a fault is detected in one or several crossbar links, the *fault_manager* (depicted in Fig. 4.1) disables the faulty crossbar links and enables the appropriate number of bypass channels. The number of Bypass-links is very important and it should be minimized as much as possible to reduce the area and power overhead. In a case where the number of faulty links is larger than the number of backup links, the system needs to mark the links as faulty and use the LAFT algorithm to avoid routing through this defective connection.

4.3 Soft Error Tolerance

As represented in Fig. 4.3, the principal soft-error handling method in the proposed 3D-FETO system relies on a solution called *Pipeline Computation Redundancy* (PCR) [103, 152, 153].

For ease of understanding, we explain the PCR in Algorithm 4.2. The Next Port Computing (NPC) and Switch Allocator (SA) run in parallel (line 2,3) after the Buffer Writing stage. This is achieved by the LAFT routing algorithm, where the dependency between the two stages is eliminated. After the first computation, both of the two stages have an additional computation clock cycle (line 4, 5). By comparing two consecutive results, soft errors are detected. If a soft error is detected, the whole pipeline is halted for correction. A third computation is required for majority voting, which decides the final result. To recover from soft errors in the data, Single Error Correction Double Error Detection (SECDED) [154] with ARQ (Automatic Retransmission Request) [99] is adopted.

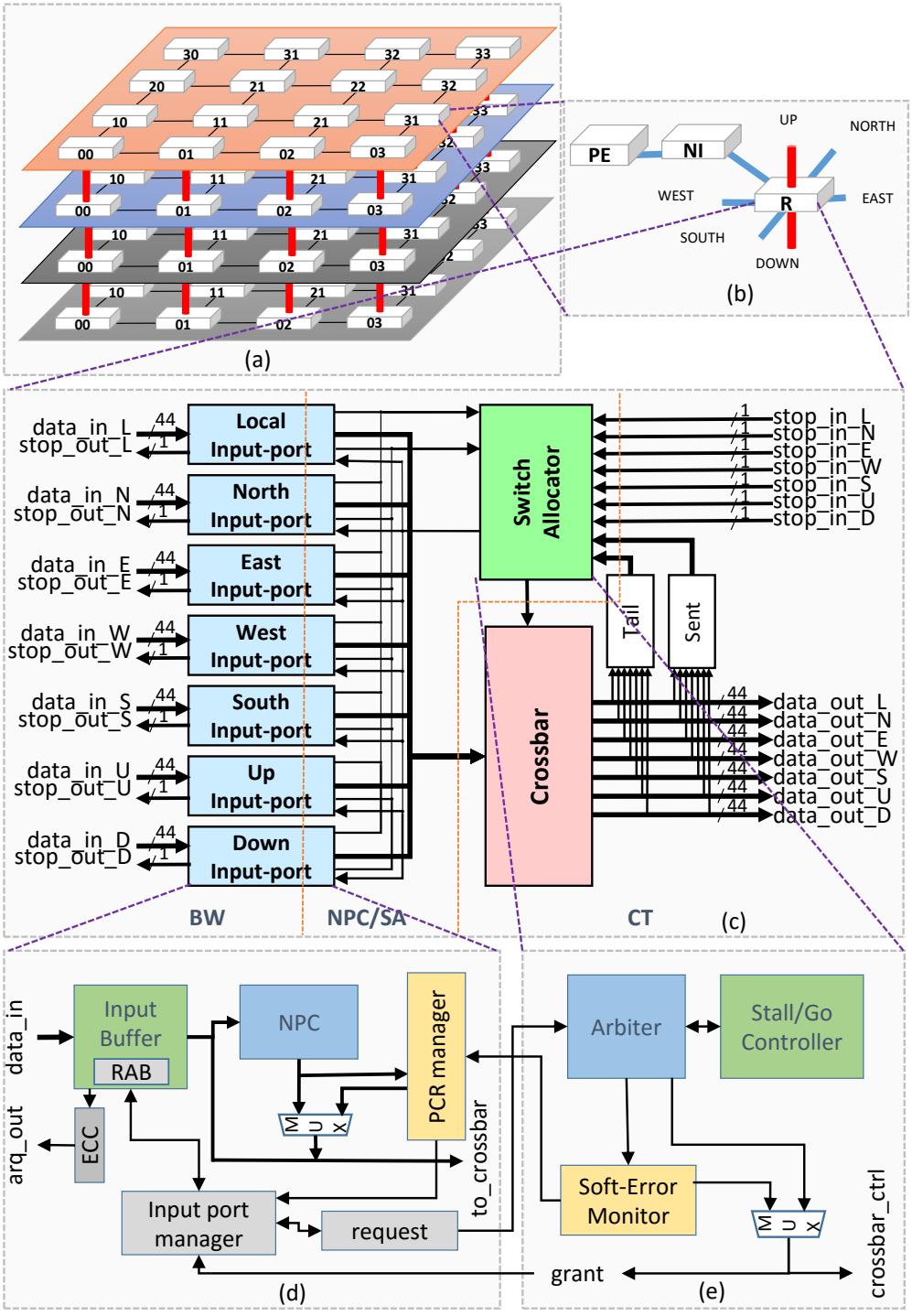

Figure 4.3: High-level view of the soft-hard error recovery approach: (a) 3D-Mesh based NoC configuration; (b) Tile organization; (c) SHER-3DR router organization; (d) Input-Port; (e) Switch allocation unit.

Algorithm 4.2: Algorithm of Pipeline Computation Redundancy (PCR).

```
   // input flit's data
   Input: in_flit
   // output flit's data
   Output: out_flit

   // Write flit's data into buffers
1  BufferWriting(in_flit)
   // Compute first time of NPC and SA
2  next_port[1] = NextPortComputing(in_flit)
3  grants[1] = SwitchAllocation(in_flit)

   // Compute redundant of NPC and SA
4  next_port[2] = NextPortComputing(in_flit)
5  grants[2] = SwitchAllocation(in_flit)

   // Compare orginal and redundant to detect soft-error
   // Soft-error on NPC
6  if (next_port[1] ≠ next_port[2]) then
       // roll-back and recalculate NPC
7      next_port[3] = NextPortComputing(in_flit)
8      final_next_port = MajorityVoting(next_port[1,2,3]);
9  else
       // No soft-error on NPC
10     final_next_port = next_port[1]
   // Soft-error on SA
11 if (grants[1] ≠ grants[2]) then
       // roll-back and recalculate SA
12     grants[3] = SwitchAllocation(in_flit)
13     final_grants = MajorityVoting(grants[1,2,3])
14 else
       // No soft-error on SA
15     final_grants = grants[1]
   // After detection and recovery, the algorithm finishes with CT
16 out_flit = CrossbarTraversal(in_flit, final_next_port, final_grants);
```

In the first stage, flits are stored in the input buffer at the Buffer Writing (BW) stage, and the ECC is used to check and correct the input data in the ECC module. In the second stage, the NPC and the SA are executed in parallel in the *LAFT routing* unit and the *Switch-Allocator* module. In the third stage, the Redundant NPC (RNPC) and the Redundant SA (RSA) are computed in parallel. Then, if the output of RNPC is equal to that of NPC, and SA is equal to RSA, the Crossbar Traversal (CT) stage is performed in the third cycle, and the flit goes to the next router via the output channel. If the RNPC is not equal to the NPC, the system rolls-back and recomputes the NPC. Moreover, if SA is not equal to RSA, the system also rolls-back and re-computes the SA stage. After rolling-back and re-computing, a majority voting module is used to decide the correct output of these modules. The rolling-back, re-computing and voting are

executed. Then, the outputs of NPC/SA are sent to the Crossbar Traversal stage to finish the flit transmission.

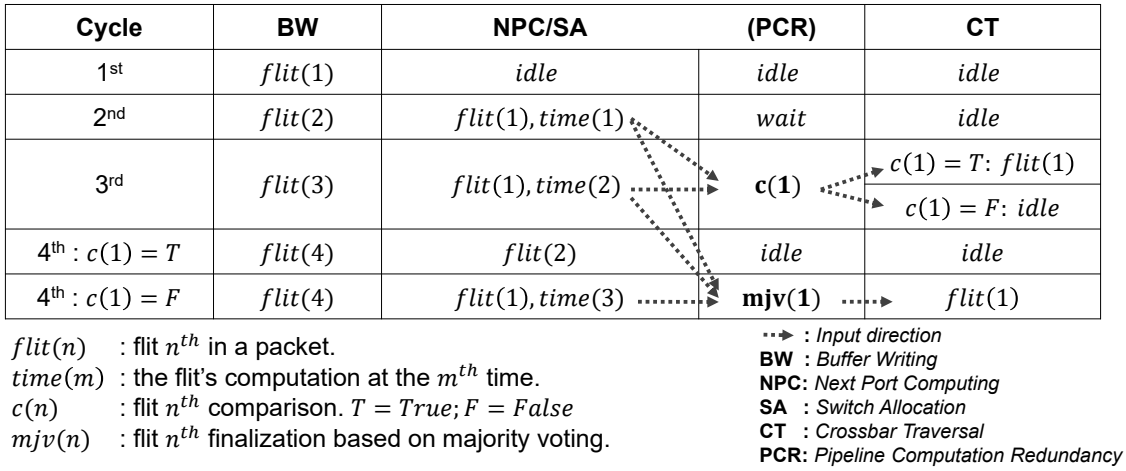

Figure 4.4: SHER-3DR working demonstration.

Figure 4.4 presents a working demonstration of the SHER-3DR router. $[flit(n)]$ represents the flit in the n^{th} position of the packet. $[time(m)]$ illustrates the m^{th} time of computation. In the first clock cycle, BW handles $[flit(1)]$ while NPC/SA and CT are idle or are handling another packet. In the second cycle, NPC/SA computes $[flit(1), time(1)]$, which means the computation of the first flit for the first time. In the third cycle, NPC/SA computes $[flit(1), time(2)]$, which means that it computes the first flit for the second time, also known as the redundant computation. $[c(1)]$ compares the results of $[flit(1), time(1)]$ and $[flit(1), time(2)]$ to detect the occurrence of a soft error. If there is no error, CT processes $[flit(1), time(1)]$ to finish the pipeline stages of the first flit. If there is an error in NPC/SA, the system requires the recovery in the fourth cycle. In this cycle, NPC/SA recalculates the first flit for the third time for recovery ($[flit(1), time(3)]$) and finalizes an accurate result by using majority voting ($[f(1)]$). After getting the final result of the first flit, CT completes the pipeline stage of the first flit based on the correct result of the two previous computations: $[flit(1), time(1)]$ or $[flit(1), time(2)]$. As shown in Fig. 4.4, the router requires one clock cycle for detecting a soft-error and one optional cycle for recovering each time an error occurs.

4.3.1 Error Correction Code

In this work, we use an ECC, which is Single Error Correct Double Error Detection (SECDED) [154], to deal with soft errors, and to support the detection and diagnosis mechanisms, as explained later.

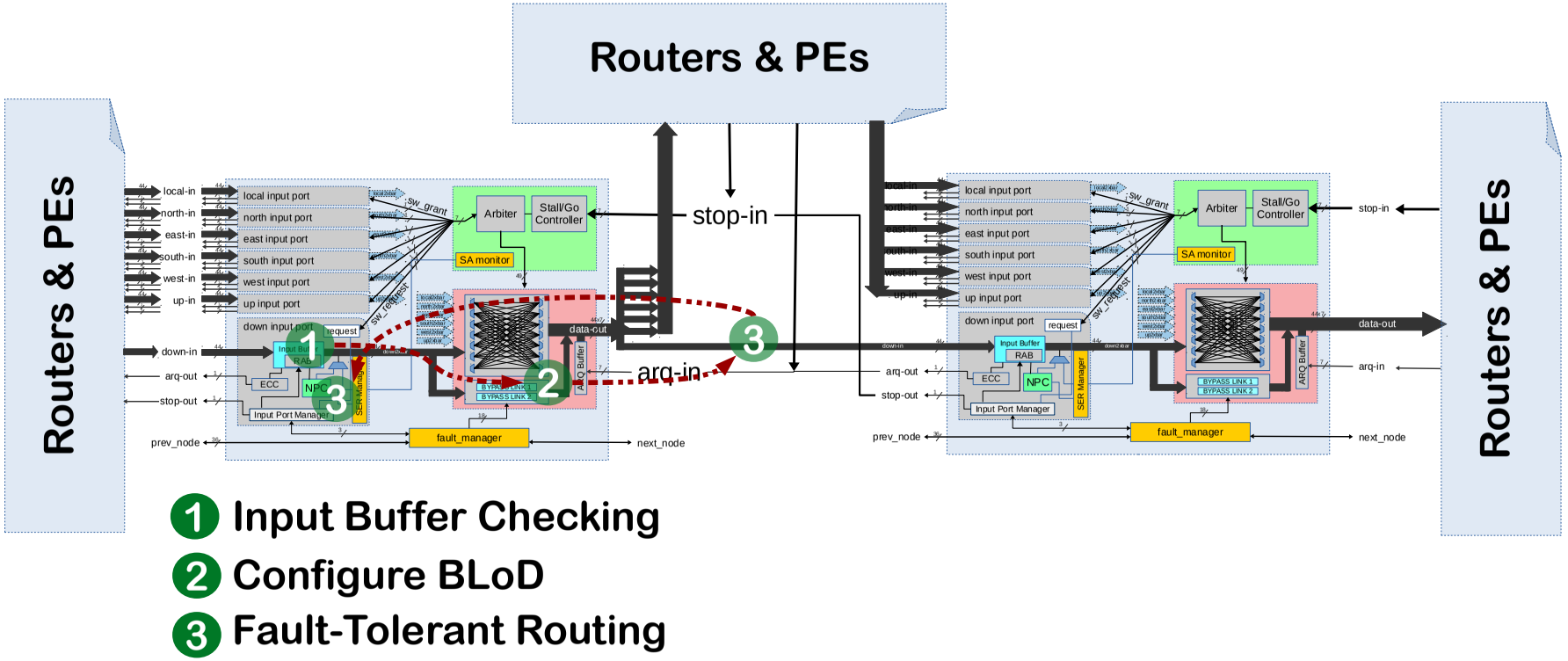

Figure 4.5: Router-to-Router interfacing and DDRM scheme.

This SECDED code is selected due to its balanced trade-off between the fault-tolerance capacity and the overhead in area cost and power consumption [65]. Moreover, we adopted the HARQ [99] mechanism, which allows to retransmit the error flit, as a flow-control. If the faulty bit can not be corrected but can be detected by the SECDED code, an ARQ signal is sent to the sender to request a retransmission. Because soft errors only occur in a short period of time, retransmitting the faulty flit can correct the faulty bit. Note that any dual detection-correction systematic coding technique can be used in our router architecture.

4.4 Detection, Diagnosis and Recovery Mechanism

Algorithm 4.3 shows the proposed *Detection, Diagnosis and Recovery Mechanism* (DDRM). It uses the feedback from the ECC and the Automatic Retransmission Request (ARQ) protocol to monitor the errors. As shown in Fig. 4.1, the input data is first verified by an ECC decoder. If the value is correct or the ECC decoder can handle the correction, the flit is written to the input buffer. Otherwise, a retransmission is requested. Since the transient fault only occurs over a short period of time, assumed to be a single clock cycle, it does not occur for two consecutive cycles. Therefore, ARQ can recover this kind of fault. However, if a permanent fault occurs, ARQ is unable to correct it and the faulty connection keeps requesting retransmission infinitely. Therefore, if the ARQ cannot correct the fault, the system considers it to be a permanent fault (line 1-10 in Algorithm 4.3).

Since a flit's correctness is verified by the ECC module before being written to the buffer, a permanent fault can only occur in the path between the input-buffer in the upstream node and the one in the downstream node. Figure 4.5 shows the high-level view of the DDRM and Router-to-Router interfacing. The transmission path of a flit consists of three main components: input buffer slots, a crossbar link and a router-to-router channel. When a fault is detected, *DDRM* diagnoses these two components to find the fault position and recover it with an appropriate mechanism.

For the diagnosis and recovery phase, the router's *Fault-manager* module initiates the diagnosis with input buffer checking. In this step, the error statuses of the following flits of the monitored input buffer are checked. If errors are detected in the following flits' transmission, it means the fault should belong to the crossbar link or the inter-router channel. The diagnosis is forwarded to check the crossbar and inter-router channel. If errors are constantly detected at the same position

Algorithm 4.3: Fault Detection, Diagnosis and Recovery.

```
   // Automatic Retransmission Request
   Input: transmitting_flit
   // Transmitted Buffer Position
   Input: buffer_position
   // Control signal to all Fault-Tolerance modules
   Output: RAB_control, BLoD_control, LAFT_control
   // Transmit the flit, get the ECC's feedback
1  Transmit(transmitting_flit);
2  ECC_result = ECC-Decoder(transmitting_flit);
   // DETECTION PHASE:
3  if ECC_result == ARQ then
       // Automatic Retransmission Request
4      increase(ARQ_counter);
5      ARQ (transmitting_flit);
6  else
       // The transmitted flit is non faulty
7      Finish;
   // Check the number of consecutive ARQs
8  if (ARQ_counter == 2) then
       // There is a permanent fault
       // Jump to DIAGNOSIS-RECOVERY PHASE
   // DIAGNOSIS-RECOVERY PHASE:
   // Start with Input Buffer Checking
9  Buffer_Failure ← Buffer_Checking(buffer_position);
10 if (Buffer_Failure == Yes) then
       // Random Access Buffer is received the position to handle.
11     RAB_Control = buffer_position;
12     Finish;
13 else
       // The buffer slot is non faulty.
       // Move to Crossbar Checking: using a Bypass-Link.
14     BLoD_control = enable;
       // Get the ECC's feedback and detect with ARQ counter.
15     if (ARQ_counter == 2) then
           // BLoD cannot fix the fault, the link is failed.
16         BLoD_control = release;
           // The LAFT routing algorithm handles the faulty link.
17         LAFT_control = faulty;
18         Finish;
19     else
           // BLoD already fixed the failure, the recovery step is finished.
20         Finish;
```

of the monitored buffer, the fault belongs to this detected position. In this fashion, the *Fault-manager* sends a signal to the *Random Access Buffer* (RAB) mechanism to indicate the faultiness of the slot in the input buffer (line 11-14). If the fault-manager indicates that the fault may belong to the crossbar or inter-router channel, the *Fault-manager* first configures the *Bypass-Link-on-Demand* (previously presented in Section 4.2) to establish an alternative connection path. Then, another flit is sent from the input buffer through a bypass-link and the router-to-router channel to the downstream node. If, at the downstream node, the flit is found to be not faulty by the ECC module, the *Fault-manager* concludes that the fault is in the Crossbar, which is already handled by the BLoD mechanism. Therefore, the configuration of the BLoD is kept as a recovery. If the flit is still faulty, the fault belongs to the inter-router channel. In this situation, the BLoD is released for further fault-tolerance and the information of the faulty channel is sent to the routing module (in LAFT algorithm). At the routing module, the *Look-Ahead Fault-Tolerant* routing algorithm uses the fault information to handle the channel's failure. The flit in the input buffer is re-routed via an alternative output port.

4.5 Evaluation Results

4.5.1 Evaluation Methodology

The proposed 3D-FETO system was designed in Verilog-HDL, synthesized and prototyped with commercial CAD tools and VLSI technology, respectively [155, 156]. We evaluate the hardware complexity of the SHER-3DR router in terms of area utilization, power consumption (static and dynamic), and speed. To evaluate the performance of the proposed system, we select both synthetic and realistic traffic patterns as benchmarks. For synthetic benchmarks, we selected Transpose [157], Uniform [158], Matrix-multiplication [159, 160], and Hotspot 10% [161]. For realistic benchmarks, we chose H.264 video encoding system [162], Video Object Plane Decoder (VOPD), Picture In Picture (PIP) and Multiple Window Display (MWD) [163]. The simulation configurations are depicted in Table 4.1.

The above synthetic benchmarks help us understand the performance of the network under stress; however, we also need several realistic benchmarks to understand the network under real application traffic. Therefore, we build a simulator in Verilog-HDL which allows us to set up the

Table 4.1: Simulation configurations.

Parameter/System		Value
Network Size ($x \times y \times z$)	Matrix	$6 \times 6 \times 3$
	Transpose	$4 \times 4 \times 4$
	Uniform	$4 \times 4 \times 4$
	Hotspot 10%	$4 \times 4 \times 4$
	H.264	$3 \times 3 \times 3$
	VOPD	$3 \times 2 \times 2$
	MWD	$2 \times 2 \times 3$
	PIP	$2 \times 2 \times 2$
Total Injected Packets	Matrix	1,080
	Transpose	640
	Uniform	8,192
	Hotspot 10%	8,192
	H.264	8,400
	VOPD	3,494
	MWD	1,120
	PIP	512
Packet's Size	Hotspot 10%	10 flits + 10% for hotspot nodes
	Others	10 flits
Flits Size		44 bits
Header Size		14 bits
Payload Bit	Baseline, 3D-FTO	30 bits
	Soft Error Tolerance, 3D-FETO	18 bits
Parity Bit	Baseline, 3D-FTO	0 bits
	Soft Error Tolerance, 3D-FETO	12 bits (2× SECDED(22,16))
Buffer Depth		4
Switching		Wormhole-like
Flow-control		Stop-Go
Routing		LAFT

traffic patterns from real applications. Based on the traffic patterns, the *Network Interfaces* send and receive packets over the networks. We select a video encoding system using a H.264 encoder, a MP3 encoder, and a OFDM [162]. Moreover, we select three applications [163]: VOPD, PIP and MWD.

We evaluate the performance of our fault-tolerant system which includes hard fault tolerance from 3D-FTO [98], Soft-Error Tolerance OASIS system, and the proposed system (3D-FETO). We measure the average packet latency, with the selected synthetic and realistic benchmarks. To understand the impact of fault-tolerance techniques on performance, we compare the obtained results with the baseline 3D-NoC system presented in [149]. We randomly inject faults at three fault-rates: 10%, 20% and 33%. The faults are injected into hard fault tolerant and soft error tolerant modules. For the soft error tolerant system, only soft errors are injected. For the hard

fault tolerant (3D-FTO) system, only hard faults are injected. For the final system (3D-FETO), both soft errors and hard faults are injected. Hard faults are injected at the beginning of simulation and their rate is measured as the percentage of routers with faults. Soft errors are injected during the system's operation and their rate is considered to be the number of soft errors per clock cycle. The injected fault rates are considered individually for each error type.

4.5.2 Complexity Evaluation

In this evaluation, we considered the hardware complexity of the proposed SHER-3DR router. For this evaluation, we use the NANGATE 45*nm* technology library [156]. Area cost and power consumption analyses are performed with Synopsys ©Design Compiler. The power consumption information is analyzed based on the switching activity of the router under the uniform benchmark. We start first by observing the additional hardware added to the baseline system when we employ the hard fault tolerance model (3D-FTO router). Then, we evaluate the impact when we consider the soft error tolerant model (Soft Error Tolerant router). Finally, we evaluate the completed 3D-FETO system including both soft and hard fault tolerant mechanisms. The configurations of the network are shown in Table 5.4 and the layout of a single SHER-3DR router is depicted in Fig. 4.7.

Table 4.2: Hardware complexity evaluation and comparison results.

Model	**Area** (μm^2)	**Power** (mW)			**Speed** (Mhz)
		Static	Dynamic	Total	
Baseline LAFT router	18,873	5.1229	0.9429	6.0658	925.28
3D-FTO router	19,143	6.4280	1.1939	7.6219	909.09
Soft Error Tolerance router	27,457	9.7314	2.6710	12.4024	625.00
SHER-3DR	29,516	10.0819	2.7839	12.8658	613.50

Table 4.2 illustrates the hardware complexity results of SHER-3DR router in terms of area, power (static, dynamic, and total), and speed. In the hard fault tolerance router (3D-FTO), the area and power consumption overheads have increased by 1.43% and 25.65%, respectively. The maximum speed has also slightly decreased. On the other hand, our soft error handling mechanism adds seven ARQ buffers and some combinational logic which increase the area and power consumption more significantly. However, SHER-3DR introduces 7.50% and 3.74% extra area and power consumption, respectively, when compared to the soft error tolerant model. In com-

parison to the baseline model, SHER-3DR increases the area and power consumption by 56.39% and 112.10%, respectively, while the maximum speed decreases by 33.70%.

The area cost and power consumption of the proposed router is given by Equation 4.1 where π_i represents the area cost or power consumption of module i. The SHER-3DR router consists of four main modules: input-ports, switch-allocator, crossbar, and fault manager.

$$\pi_{router} = \pi_{input-ports} + \pi_{switch-allocator} + \pi_{crossbar} + \pi_{fault-manager} \tag{4.1}$$

The details of an input port, a switch-allocator and a crossbar are given in Equation 4.2.

$$\begin{aligned} \pi_{input-ports} &= \pi_{original-input-ports} + \pi_{RAB-controller} + \pi_{PCR-controller} + \pi_{ECC} \\ \pi_{switch-allocator} &= \pi_{original-switch-allocator} + \pi_{PCR-monitor} \\ \pi_{crossbar} &= \pi_{original-crossbar} + \pi_{bypass-links} + \pi_{ARQ-buffers} \end{aligned} \tag{4.2}$$

We can observe the overheads in power consumption and area cost that are caused by the fault-tolerant mechanisms (RAB-controller, PCR-controller, ECC, BLoD, ARQ buffers). Figure 4.6 provides the evaluation results of power consumption and area cost of SHER-3DR. In terms of area cost, the input ports occupy the majority with over 67% which is followed by the crossbar (20%) and the switch allocator (9%). The fault manager, which supports DDRM, uses only about 4% of the overall area cost. In terms of power consumption, the input ports consume over 80% of the total value. The fault manager module also causes an insignificant increase in power consumption (3%).

When compared to the baseline OASIS router, the proposed SHER-3DR consumes more power consumption and costs more area. As shown in Fig. 4.6, SHER-3DER increases the area and power of all three main modules (crossbar, input ports, and switch-allocator). The overhead can be analyzed by Equation 4.2 where additional modules are attached to support the fault-tolerance mechanisms.

Although our proposed system is penalized in terms of area, power consumption, and maximum frequency due to additional logic and registers that are necessary for fault handling mechanisms,

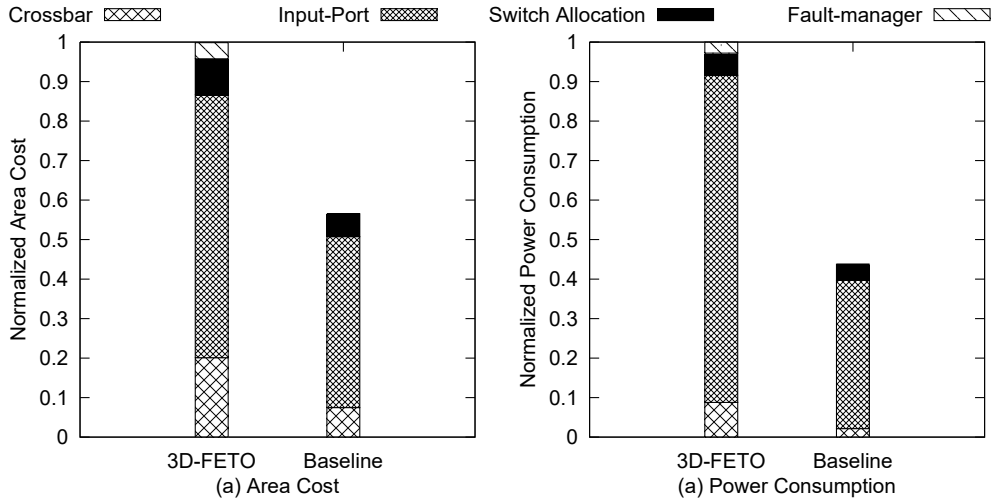

Figure 4.6: Area cost and power consumption analysis.

it provides an improved resiliency against a significant amount of soft and hard faults.

4.5.3 Latency Evaluation

In the second experiment, we evaluate the performance of the proposed architecture in terms of latency over various benchmark programs and error injection rates for three system configurations: (1) Hard-fault tolerant system (3D-FTO), (2) Soft-error tolerant OASIS system, and (3) Hard-fault and Soft-error tolerant system (3D-FETO). The simulation results are shown in Figs. 4.8 and 4.9. From these graphs, we notice that with 0% hard faults (in input buffer and crossbar only), 3D-FTO has similar performance to the baseline system (LAFT-OASIS). In addition, we found that even at a 33% fault-rate, 3D-FTO increases the latency by only 1.71%, 11.38%, 8.79% and 13.73% for Transpose, Uniform, 6×6 Matrix, and Hotspot-10%, respectively. With realistic benchmarks, the performance of 3D-FTO slightly degrades at low error-rates; but, it suffers more of an impact at high error-rates (20% and 33%) since the flit encounters bottlenecks due to errors inside the input buffers. However, the proposed 3D-FETO model still works even at high fault-rates while the baseline model collapses at a 5% error-rate. We used the same benchmark programs to evaluate the soft error tolerant system. Since both the proposed *Pipeline Computation Redundancy* mechanism and ECC require additional clock cycles, we can observe a significant effect on average packet latency. For the 0%, 10%, 20% and 33% fault-rates, the Soft Error Tolerant model increases the average delay in the Transpose benchmark by 18.57%, 28.74%, 34.54% and 49.62%, respectively. Finally, we evaluate the proposed 3D-FETO system with both soft error and hard fault handling

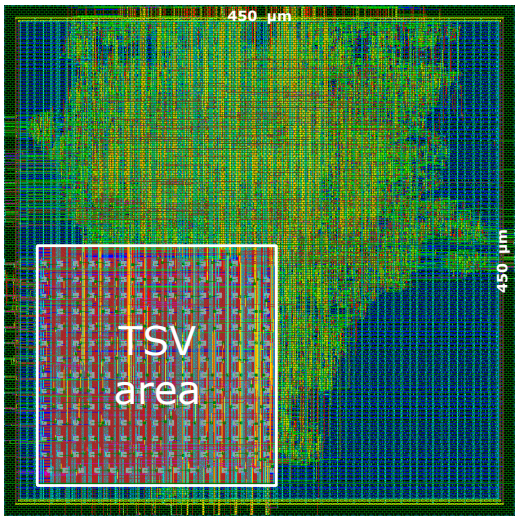

Figure 4.7: Layout of a single SHER-3DR router for the 3D-FETO system. The SHER-3DR router was designed in Verilog-HDL and synthesized using 45nm technology library. For the Through Silicon Via (TSV) integration, we used FreePDK3D45 kit compiler. The SHER-3DR router is designed on a $450\mu m \times 450\mu m$ and the TSV array contains 208 TSVs.

schemes. As shown in Figs. 4.8 and 4.9, 3D-FETO has demonstrated a significant impact on the average latency, which has mostly doubled for both realistic and synthetic benchmarks. At a 33% fault-rate using Matrix, Uniform, Transpose benchmarks, 3D-FETO's average latency increases by 78.44%, 50.73% and 67.18% in terms of average packet latency. The degradation is caused by both soft errors and hard fault tolerance mechanisms: (1) the ECC+ARQ and PCR both require additional re-transmission clock cycles; (2) the RAB and LAFT routing algorithm may disable a part of the network which causes congestion. However, it still maintains the ability to work under an extremely high fault-rate (33% for hard faults and 33% for soft errors).

4.5.4 Throughput Evaluation

Figure 4.10 depicts the throughput evaluation with the adopted synthetic benchmarks. At a 0% error rate, 3D-FTO (hard-fault tolerance) presents the best throughput which matches the capacity of the baseline LAFT-OASIS. The Soft Error Tolerant OASIS and the proposed 3D-FETO have less throughput due to their soft error tolerance mechanisms. When the errors are injected into the system, we can observe a degradation in throughput. Thanks to the efficient hard fault tolerant scheme and the fault-tolerant routing algorithm, 3D-FTO at a 33% error-rate provides a slightly decreased throughput: 40.18%, 43.96%, 43.55% and 32.59% for Transpose, Matrix, Uniform, and Hotspot 10%, respectively. For the Soft Error Tolerant OASIS, the system requires re-

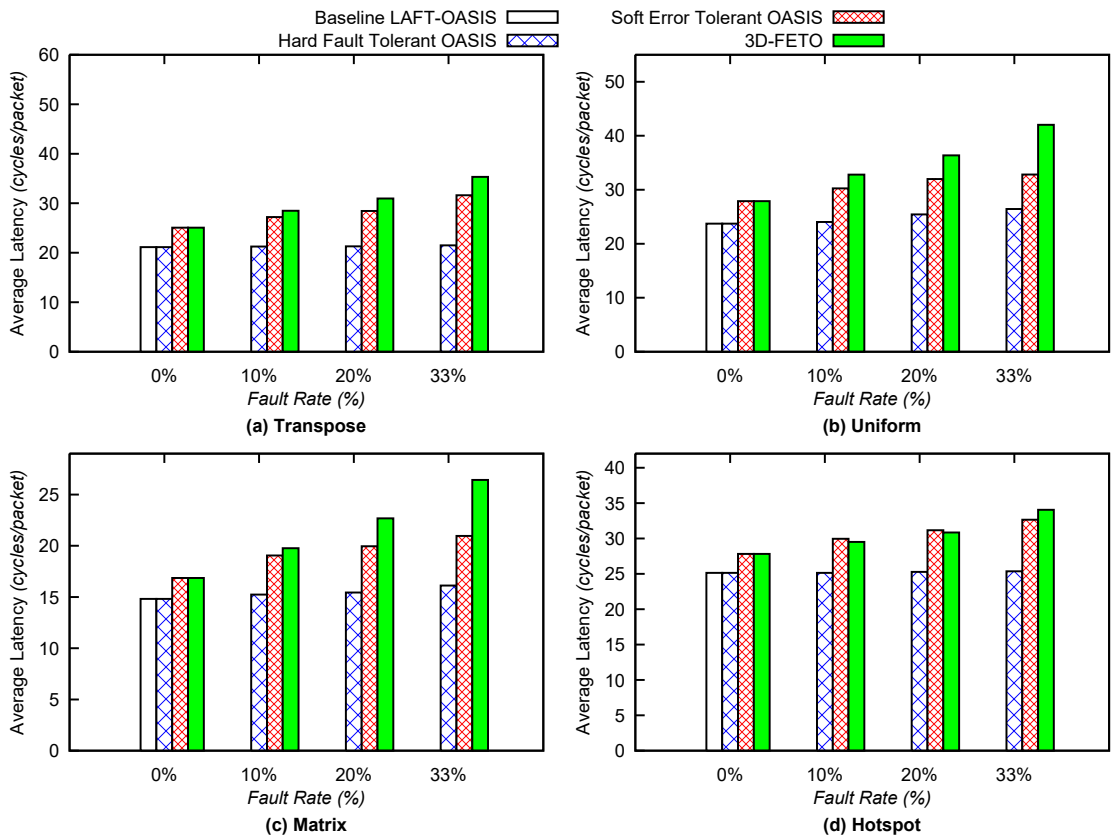

Figure 4.8: Average packet latency evaluation of the synthetic benchmarks.

transmission via the ARQ mechanism and the re-execution for the soft error mechanism. Therefore, the throughput is degraded due to the extra clock cycles. The proposed 3D-FETO, which is a fusion of both hard fault and soft error tolerant mechanisms, inherits both degradations; however, these systems provide the ability to handle up to a 33% error rate (the limitation of the soft error mechanism).

4.6 Conclusion and Discussion

In this chapter, we proposed a fault tolerant 3D-Network-on-Chip (3D-NoC) system architecture for highly-reliable many-core Systems-on-Chips (SoCs), named 3D-FETO. The proposed system is based on two approaches. First, a comprehensive mechanism to handle both soft error and hard faults in a 3D-NoC router is proposed. The hard fault support is achieved by leveraging reconfigurable components to handle permanent faults in links, input buffers, and crossbars, while soft error tolerance is obtained via efficient and light-weight software redundancy that enables fault

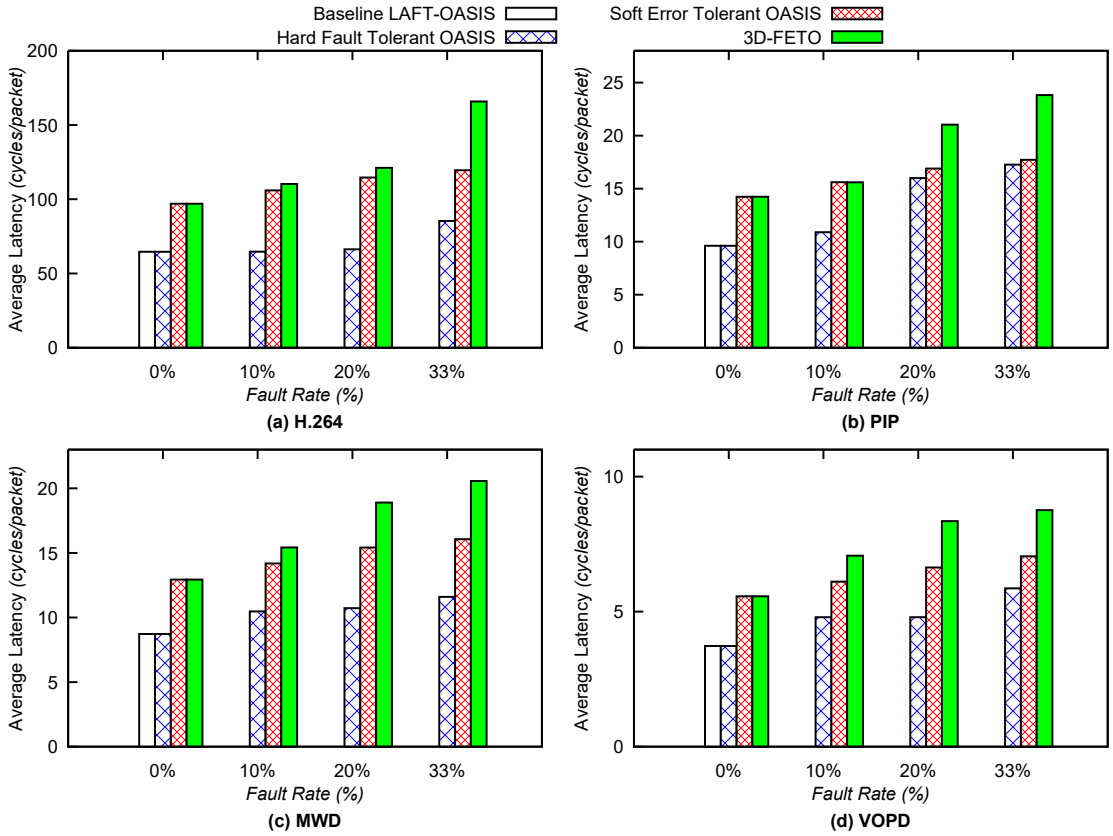

Figure 4.9: Average packet latency evaluation of the realistic benchmarks.

recovery in the router pipeline stages. In the second approach, the system can support a detection, diagnosis and recovery technique which makes it independent of any complex and costly testing mechanisms commonly found in conventional systems.

Through extensive evaluation, we showed that the proposed 3D-FETO was able to recover efficiently from a significant number of soft and hard errors at different fault-rates, reaching up to 33%. This means that 3D-FETO can provide up to a 98% packet arrival rate even when almost one-third of its components have failed. Despite the performance degradation and hardware complexity penalty, we still consider that this overhead is acceptable. This is because we made sure that the system is still functional at high fault rates where previously proposed systems fail to deliver packets. As reliability constitutes one of the main challenges in future SoC design, we demonstrated that the proposed 3D-FETO can be used as a reliable and independent system capable of ensuring fault resiliency in worst case scenarios and that it can be adopted for mission critical applications where correct data delivery is primordial.

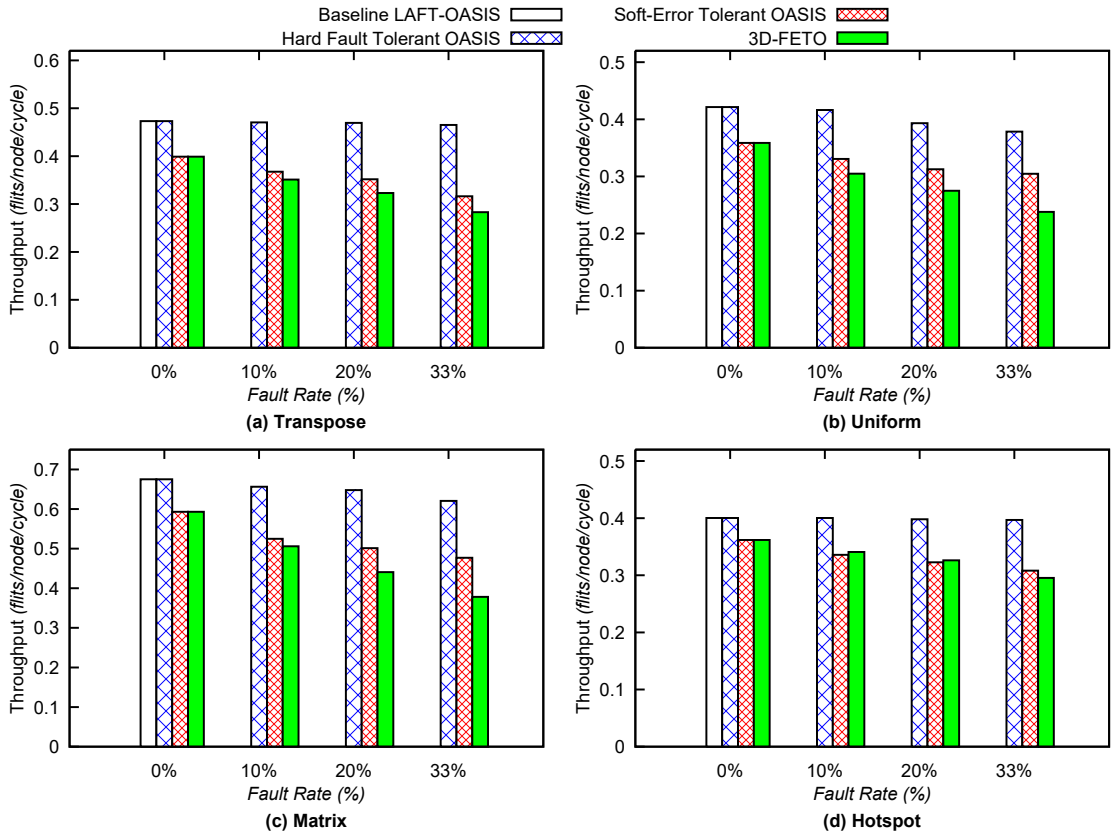

Figure 4.10: Throughput evaluation of the synthetic benchmarks.

Because this chapter has presented a complete soft error and hard fault tolerance, the fault tolerance for Through-Silicon-Vias of 3D-ICs/3D-NoCs is needed to provide a sufficient fault-tolerance method for 3D-NoC systems.

5

Scalable TSV-Cluster Fault-Tolerance

In this chapter, the TSV-cluster defect tolerant architecture and algorithm are presented. Because of the high defect rate and clustering distribution, TSV defect tolerance has become a major issue for commercializing TSVs. The proposed technique in this chapter was developed along with the work in the previous chapter, which already covers soft error and hard fault tolerance. As the final system, this chapter aims to provide a complete work on tackling soft errors, hard faults and TSV defects.

The structure of this chapter is as follows. First, Section 5.1 declares the motivations and contribution of this chapter. After that, 5.2 shows the proposed architecture with the fault assumptions. Based on this architecture, the third section shows the algorithm and optimization. Next, Section 5.4 shows the evaluation results. The final section is for conclusions and discussion.

5.1 Motivation and Contribution

As we mentioned in Chapter 3, there are two distributions for TSV defect: random and clustering. The cluster defect is predicted to be frequently occurred. The most efficient solution for correcting random defects is grouping and adding redundancy. However, they are still inefficient for the cluster defect and possibly require costly extra area for redundancy. Because the random TSV defect can be corrected by using *Error Correcting Code* (see Section. 4.3.1), the cluster defect is the main interest of this dissertation.

On the other hand, several works [95, 96, 107, 108, 164] have been reporting the low utilization of the vertical connection using TSVs in 3D-NoCs. The authors tried to reduce the number of TSVs to minimize the area overhead while maintaining a low degradation in terms of performance. Motivated by the cluster defect issue and the low utilization of the TSVs in 3D-NoC, we target a better management for TSV fault-tolerance in 3D-NoCs.

In this work, we propose a scalable TSV utilization architecture and methodology to tackle the lack of reliability in inter-layer links. To reduce the TSV-cluster defects, a router corrects its defected TSV communication by choosing one of its four neighbor TSV-clusters located on the same layer. To avoid timing violation issues, we place the TSVs of two nearby routers in between them and a TSV-cluster is only shared between its two neighboring routers. Experimental results show that the solution can help 3D-NoCs to work around TSV-cluster defects without the need for redundancy. Therefore, reliability at reasonable overhead is guaranteed.

5.2 Proposed TSV Fault Tolerance Architecture

In order to handle the TSV-cluster defects in 3D-NoCs, our solution is to share TSVs between neighboring routers. Therefore, when a TSV-cluster fails, its router can borrow a healthy cluster from one of its neighbors to maintain the connection. Moreover, we also present several design optimizations to improve the reliability of the system (Section 5.3.4).

5.2.1 Fault assumptions

Before we present the system structure, this subsection clarifies the fault assumptions taken in this proposal. Because the cluster defect [30, 80, 81] is the major obstacle to be dealt with in this work, we assume there are no random defects. Here, we consider an occurred fault makes the

whole TSVs in the cluster to become defected. For those who might be concerned about random defects, using redundancy [32, 82, 105, 106] can be easily integrated in our TSV-cluster design. For controlling signals using TSVs, they are considered as a part of the TSV cluster instead of separated TSVs, which are better dealt with than random defects (e.g. [74] uses Double TSV [105]). The detection process, which may need a Built-In-Self-Test module [165, 166], is assumed to be existing and connected to the fault-tolerance module. To synchronize the configuration, the existing NoC infrastructure is used instead of adding TSVs. Therefore, no redundancy is required in the proposed architecture.

5.2.2 System structure

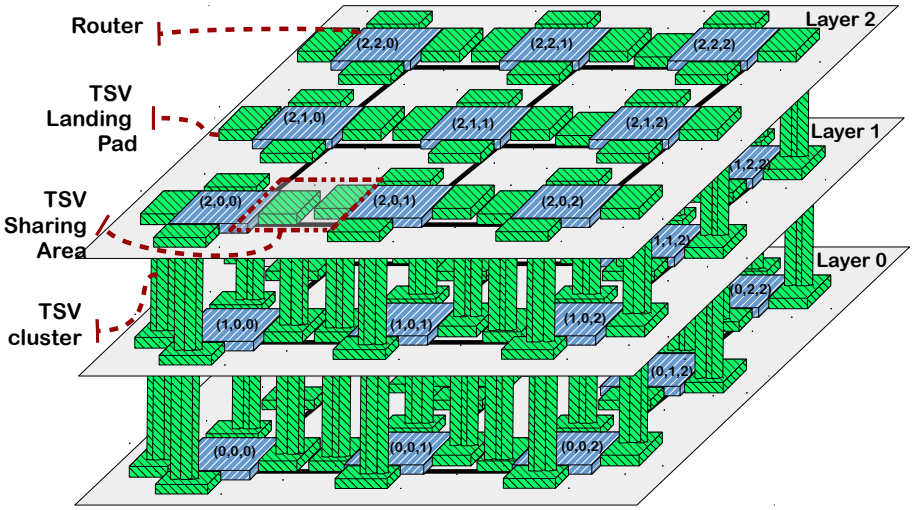

Figure 5.1: Simplified block diagram illustrating the proposed system structure.

A simplified layout example of $3 \times 3 \times 3$ 3D-NoC system using the proposed TSV usage is depicted in Fig. 5.1. For each vertical connection, a router needs a set of TSVs. Instead of grouping all TSVs together, as shown in Fig. 2.11, they are divided into four groups. As a result, a router owns four TSV-clusters and has a maximum of four nearby TSV-clusters. If a TSV-cluster of a router is defected, the router can choose one of its four neighboring clusters as a replacement without the need for redundancy. To satisfy the timing constraints, the router chooses the closest TSV-cluster among its neighbor clusters. Taking into account further TSV-clusters is not considered in order to avoid long wires that are needed to establish the connection. By structuring the TSVs into four clusters for each router, we can maintain the scalability of 3D-NoCs and avoid long wire delay. We have to note here that there are some works that consider serialization to reduce

the cost of TSVs in 3D-NoCs. In this work, we consider a normal vertical connection; however, the proposed approach can be applied for the serialized TSV structure, as shown later.

Figure 5.2 shows the placement and connections of the TSV sharing area between *R(1,1,1)* and *R(1,0,1)*. Because each router has two ports (up and down) and two directions (in and out), the number of TSV clusters is eight. Each TSV cluster handles a quarter of the vertical connection. By using the tri-stage gates, the system can control which router has access to the TSV clusters.

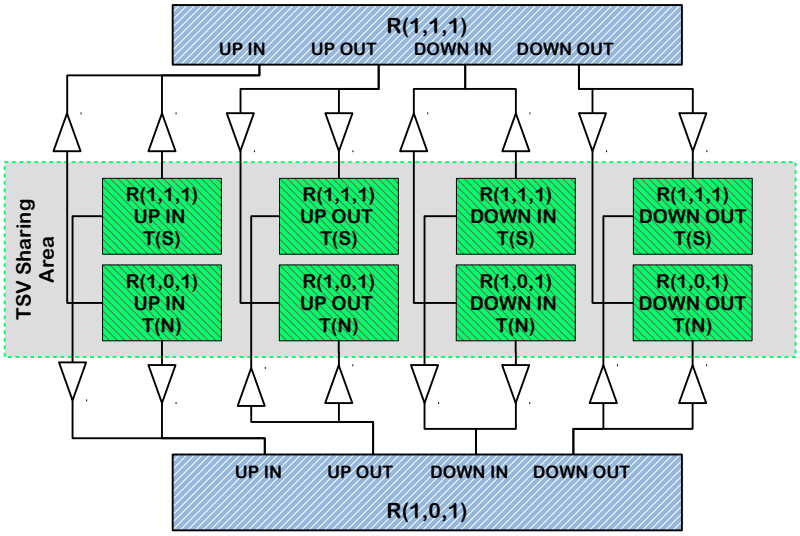

Figure 5.2: TSV sharing area placement and connectivity between two neighboring routers.

5.2.3 Sharing Circuit Design

To borrow a TSV-cluster from a neighbor, the router needs a supporting module. Figure 5.3 (a) shows the wrapper of a 3D-Router with the additional supporting modules that perform the sharing algorithm, later explained in Section 5.3. There are two identical sharing modules (S-UP and S-DOWN) for the two vertical up and down connections and each connection has two configuration registers (CR) for the input and output ports. As previously depicted in Fig. 5.1, *R(1,1,1)* shares the TSV-clusters with its four neighbors: *R(1,1,0)*, *R(1,1,2)*, *R(1,0,1)*, and *R(1,2,1)*. Figure 5.3 (b) shows the sharing circuit for a TSV-cluster. The input of this TSV-cluster is shared between *R(2,1,0)* and *R(2,1,1)* on layer2. The output of this TSV-cluster is shared between *R(1,1,1)* and *R(1,1,0)* on layer1. In the case where this TSV-cluster is defected, or borrowed, the data can be sent by using one of the four neighboring clusters.

Based on the value of the 6-bit CR, shown in Table 5.1, the input and output ports can select

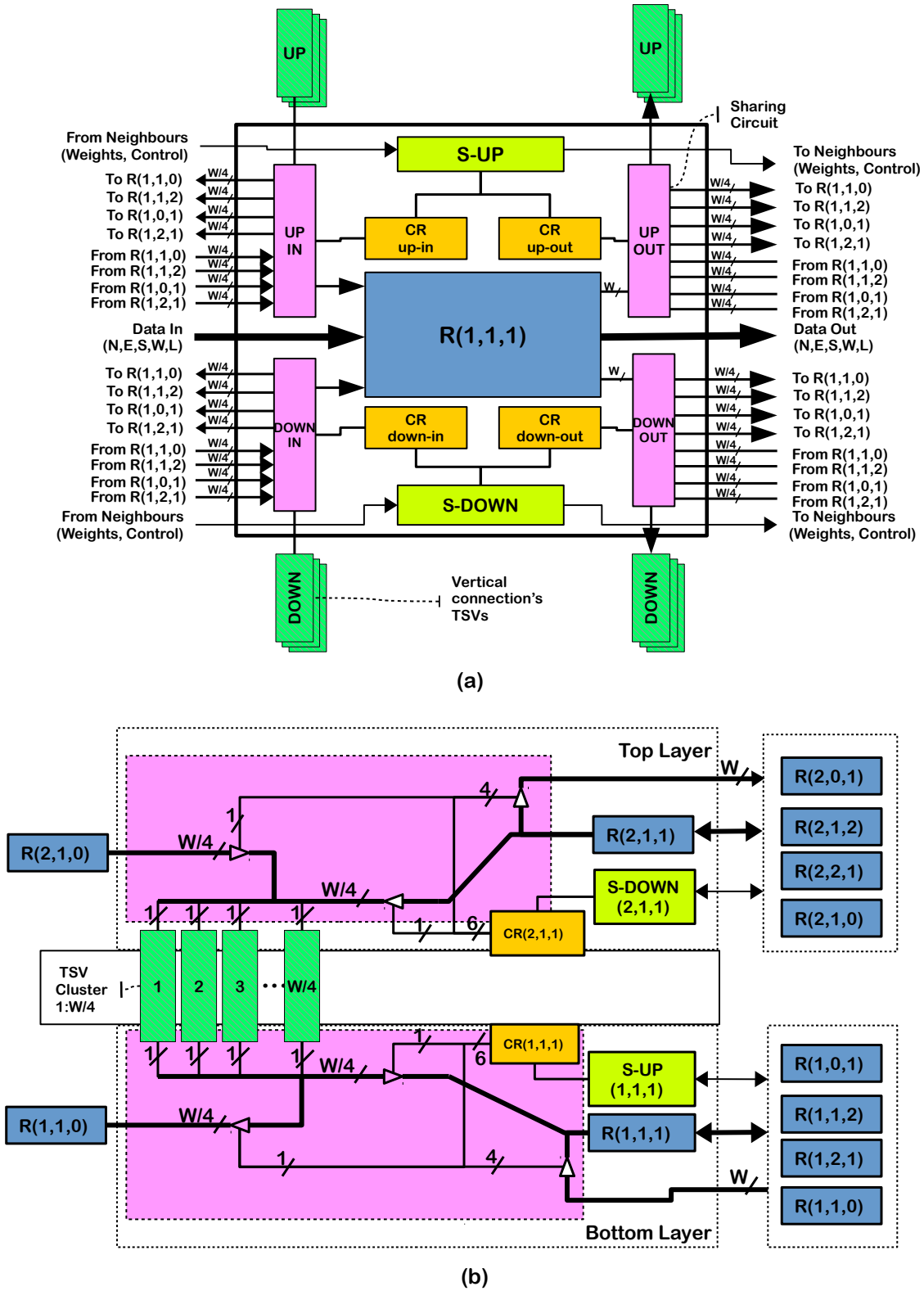

Figure 5.3: The TSV fault-tolerance architecture: (a) Router wrapper; (b) Connection between two layers. Red rectangles represent TSVs. *S-UP* and *S-DOWN* are the sharing arbitrators which manage the proposed mechanism. *CR* stands for configuration register and *W* is the flit width.

Table 5.1: Configuration register (CR) description.

Value	Description
000001	Original router connects to the cluster.
000010	Neighboring router connects to the cluster.
000100	Original router connects to the neighboring north TSV-cluster.
001000	Original router connects to the neighboring east TSV-cluster.
010000	Original router connects to the neighboring south TSV-cluster.
100000	Original router connects to the neighboring west TSV-cluster.

the data from: (1) its original TSV-cluster, (2) one of its four neighboring clusters or (3) being disconnected. As shown in Figure 5.3 (b), the output data from *R(2,1,1)* can be sent to its TSV-cluster if the least significant bit is '1'. By setting the least significant bit to '0', the original TSV-cluster is disconnected from it router. If the second bit is set as '1', the neighboring router (*R(2,1,0)*) takes the access to this cluster. When the original TSV-cluster is defected or taken, the router needs to take one of its neighbor's clusters to maintain the connection based on the last 4-bit of CR. At the receiving router (*R(1,1,1)*), a similar CR is used to establish the connection. The value of this CR is identical to the sending router's CR.

Because the CR only manages the connectivity, its value have to be set carefully to avoid the possible conflict of TSV-cluster usage and to optimize the performance. To this aim, an adaptive sharing algorithm is needed.

5.3 Adaptive Online Sharing Algorithm

In the previous section, we presented how a router can use its nearby TSV-clusters to maintain the connection and the operation on a layer. The CR values need to be configured in order to deal with the TSV defects. The simplest way for this process is to perform it offline and the configuration fuses the TSV group [80]. However, fixing the connections has two main drawbacks: (1) recovering a newly defected TSV needs to halt the system and re-perform the mapping, and (2) each application has a different distribution in the vertical connections and variations depending on the running task which is not optimized by offline mappings. Consequently, we aim to perform the mapping online so that the system can react immediately to the newly defected TSV-clusters and can consider the connectivity of the 3D-NoC system. Thus, this subsection provides an online algorithm for sharing TSVs which can be implemented onto the system.

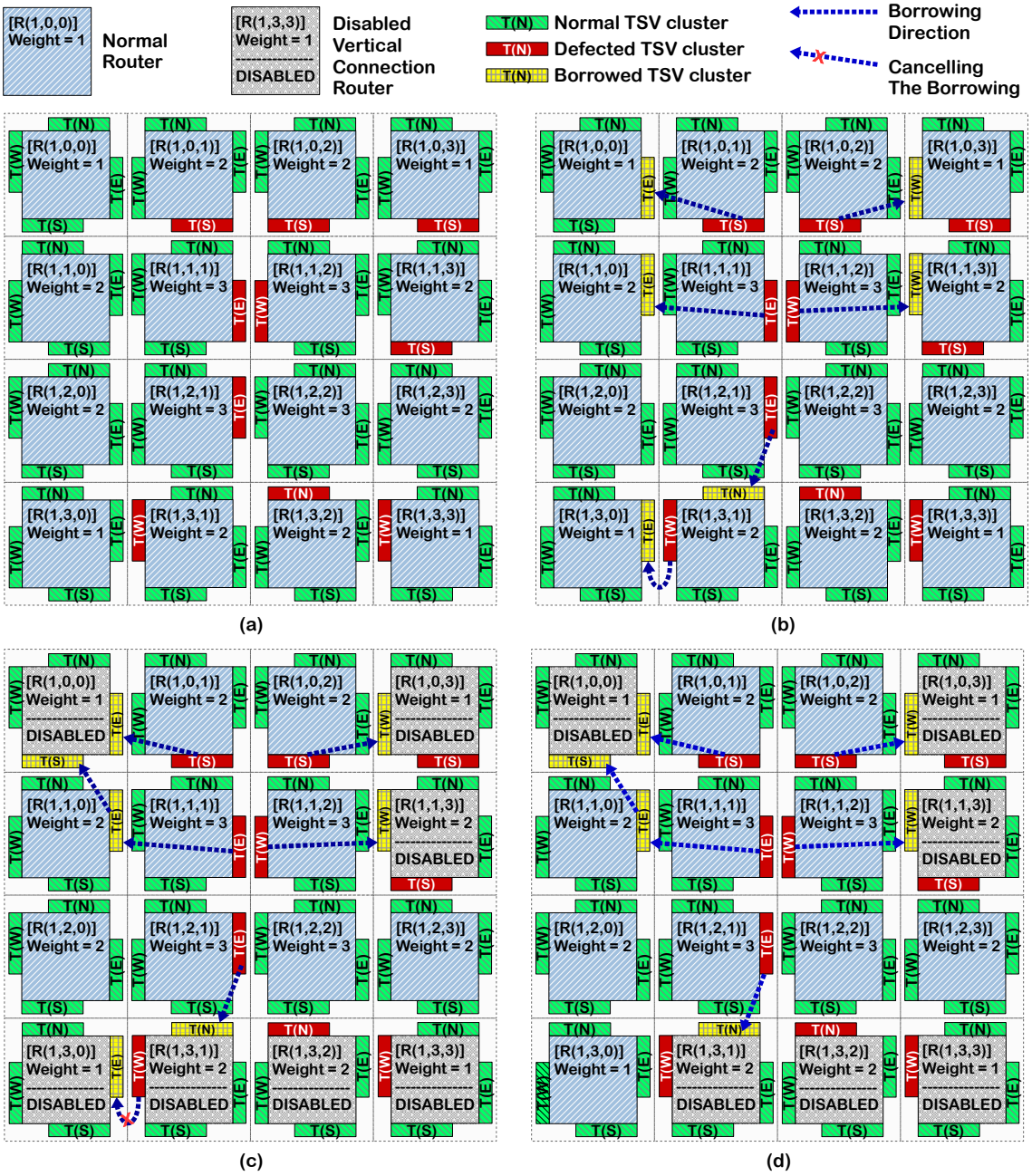

Figure 5.4: An example of the sharing algorithm on a 4 × 4 layer: (a) Initial state with ten defected TSV-clusters; (b) Best candidates selection; (c) Borrowing chain creation and selection refining. (d) Final result with six disabled routers.

Algorithm 5.1: TSV Sharing Algorithm.

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
// Status of current and neighboring TSV-clusters
Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
// Request to link TSV-clusters to neighbors
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "NORMAL" then
           // It is a healthy TSV-cluster
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is minimal
8          and TSV_Status_neighbor[c] == "NORMAL";
9          if (c==NULL) then
10             return RQ_link[i] = "NULL"
11             return Router_Status = "DISABLE"
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

Algorithm 5.1 shows the proposed algorithm for our sharing mechanism. Each router is assigned to a weight for each of the vertical connections. This weight decides its priority in sharing/borrowing. The weight can be assigned at the design process or can be updated by a dedicated module. Changing the weights of routers can create different mappings. At the initial stage, all routers in the network exchange their weights and their TSV-clusters status with their neighbors. In the next step, the algorithm performs the mapping process. If a TSV-cluster is defected, its corresponding router should find from its neighbors a possible candidate by relying on the following conditions:

- The weight of the candidate has to be smaller than the current router.
- The candidate TSV-cluster has to be healthy and not borrowed.
- The weight of the final candidate is the smallest among all the possible candidates.

At the end of the algorithm, the router finds out the possible candidate for borrowing. If no candidates are found, the router's vertical connection is disabled. If there is a candidate, the router

sends a request to the borrowing router to use its TSV-cluster as a replacement for the defected one. The routers having borrowed TSV-clusters also look for a replacement among one of their neighbors. By using a weighted system, the disabled TSV-clusters focus on smaller weight routers.

Figure 5.4 shows an example of how the sharing algorithm works on a 4×4 layer with ten defected TSV-clusters. Initially, the routers in the center, which are predefined to have higher TSV utilization rates, have higher weights than those at the edges of the network, as depicted in Fig. 5.4 (a). The sharing algorithm selects the best candidates, shown in Fig. 5.4 (b), by following the rules previously explained in Algorithm 5.1. Fig. 5.4 (c) shows that this selection must be further refined by disabling the router having less than four functional (or not borrowed) TSV-clusters and canceling their borrowing. The returning process is discussed in Section 5.3.2. Moreover, we also observe the case in Fig. 5.4 (d) where two routers *R(1,3,2)* and *R(1,3,3)* are disabled; but, *R(1,3,3)* can borrow a TSV cluster from *R(1,3,2)* to obtain full four TSV clusters. However, the borrowing is prohibited due to the higher weight of router *R(1,3,2)*. In order to optimize this case, we use a technique named *Weight adjustment*, as explained in Section 5.3.3.

As shown in the above example, the chain of sharing leads to disabling the routers on the edges. Instead of having ten defected TSV-clusters, the algorithm only disables six routers having the lowest weights (40% of reduction). Consequently, maintaining the connections of the center routers, which have higher weights and utilize more vertical communications, can reduce the impact of TSV defects in terms of overall performance.

5.3.1 Weight Generation

One of the most important parameters in the sharing algorithm is the weight values of the routers. The weights help the algorithm decide what router is suitable to be borrowed. As shown in Fig. 5.4, the routers having smaller weights are disabled after the chains of sharing are established.

Because the weights decide the priority of the routers in the sharing process, they need to be optimized to obtain a maximum system performance. In order to do that, the best solution is using a statistic-based solution where the priority of the vertical connection depends on the communication traffic [167, 168]. In other words, the vertical connections having more data transmissions are assigned higher weights; otherwise, smaller weights are assigned. Because application mapping is out of the scope of this research, we adopt a simple method where the routers in the middle of

the layer have the highest weights. The router's weights are decreased and become the lowest at the edges of the layer. Equation 5.1 shows the used weight value assignment. The output of this weight assignment on a layer of 4×4 can be seen in Fig. 5.4 where, for instance, the weights of routers *R(1,0,0), R(1,1,0)*, and *R(1,1,1)* are 1, 2, and 3, respectively.

$$\text{Weight}_{\text{router}}(x, y) = \min(x, \text{cols} - x) + \min(y, \text{rows} - y) + 1 \tag{5.1}$$

5.3.2 TSV-clusters return

After a TSV-cluster is borrowed, it is managed by the borrowing router. However, if the borrowing router is disabled later, this frees the borrowed cluster which has to be returned to its original router. As a result, if the borrowed TSV cluster created a chain of borrowing, a chain of returning is also created. This can be clearly seen in Fig. 5.4 (c) where *R(1,3,1)* has a faulty cluster and has selected the east cluster of *R(1,3,0)* to be borrowed. However, in the next step, *R(1,3,1)* is selected to borrow its north cluster to a higher weight router, *R(1,2,1)*. Because *R(1,3,1)* is unable to find any sharing TSV cluster to borrow, it is disabled and borrowing from *R(1,3,0)* is canceled. Fig. 5.4 (d) represents the final results of the sharing process. In this final stage, *R(1,3,0)* is operational again as it is no longer lending a cluster to *R(1,3,1)* which was disabled in the previous phase.

After a TSV cluster is returned, its router check whether it created a borrowing chain and release the borrowing. If there is no borrowing chain, which means the router failed to find a replacement and is disabled, the sharing algorithm is performed again to check if the router can return to normal. As shown in Fig. 5.4 (d), *R(1,3,0)* returns to normal after its TSV cluster (T(E)) is returned.

5.3.3 Weight adjustment

After applying the sharing mechanism, the disabled TSV-clusters are shifted to the region which consists of low weighted routers. Figure 5.5 (a) shows a case of three routers (*R(1,0,0), R(1,0,1)* and *R(1,0,2)*) which are disabled after the sharing process. However, there are still a chance of optimizing these routers to obtain a better mapping. In fact, *R(1,0,2)* can borrow a TSV-cluster from *R(1,0,1)*. Therefore, the number of TSV-clusters of *R(1,0,2)* can be maintained to four.

Algorithm 5.2: Weight Adjustment Algorithm.

```
   // Status of current and neighboring TSV-clusters
   Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
   // Current and neighboring routers status
   Input: Curr_Status, Neighbor_Status[1 : N]
   // Request to link TSV-clusters to neighbors
   Output: Weight_current
1  Curr_TSVs = 0;
2  foreach TSV_Status_current[i] do
3      if TSV_Status_current[i] == "NORMAL" then
4          Curr_TSVs + +;
5  Neighbor_TSVs = 0;
6  foreach TSV_Status_neighbor[i] do
7      if TSV_Status_neighbor[i] == "NORMAL" and Neighbor_Status[i] == "DISABLED" then
8          Neighbor_TSVs + +;
   // If there is at least 4 cluster, run the sharing algorithm
9  if Neighbor_TSVs + Curr_TSVs >= 4 then
10     call TSV_Sharing()
11 else
       // Reduce the current weight to allow the neighbors borrow
12     Weight_current = 0;
```

To perform this optimization, the disabled router, after the sharing process by Algorithm 5.1, is brought to a new process. First, the router counts the number of possible TSV-clusters that it can borrow. Since three routers (*R(1,0,0), R(1,0,1)* and *R(1,0,2)*) are disabled, their TSV-clusters are free to be taken. At the end of this stage, *R(1,0,0), R(1,0,1)* and *R(1,0,2)* have 1, 3, and 1 borrowed/defected TSV-clusters and are able to take 0, 1 and 1 TSV-cluster from their disabled neighbors, respectively. At the second stage, the router checks whether it can take the disabled router's cluster to obtain a full connection. Because *R(1,0,2)* has one borrowed cluster and is

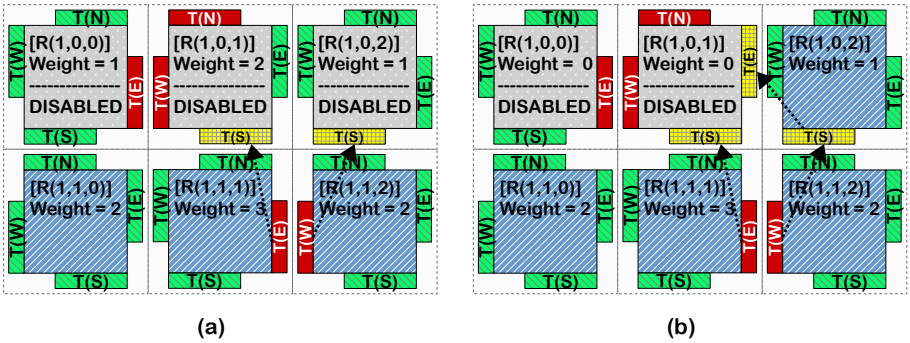

Figure 5.5: Example of the weight adjustment performed to disable routers' sharing: (a) Before weight update; (b) After weight update.

able to borrow another one from *R(1,0,1)*, its weight is kept. The other routers (*R(1,0,1)* and textitR(1,0,0) weights are reduced to zero. As a result, *R(1,0,2)* can borrow a TSV cluster from *R(1,0,1)* despite the fact that it originally has a lower weight. The result is shown in Fig. 5.5 (b) where *R(1,0,2)* vertical connection is re-enabled. If the system wants to restart the sharing mechanism, the weights of all routers need to be reinitialized.

Algorithm 5.2 shows the weight adjustment algorithm. It first calculates the total number of healthy TSVs that are possible for using. If the total number of healthy TSV-clusters is larger or equal than four, which is enough to maintain the vertical connection, the neighboring routers' weights are reduced. After that, the TSV sharing algorithm (Algorithm 5.1) is performed, where the router now can take TSV-clusters from the routers having higher weights, but being disabled.

5.3.4 Design optimization

Without adding redundancy, borrowing TSV-clusters to work around the defected ones makes some routers to have less than four accessible clusters (e.g., R(1,0,0) in Fig. 5.4 (d)). As a result, the communication of these routers have been disabled. To tackle this problem, the naive solution is using a fault-tolerant routing algorithm to re-route the packets to a neighboring router. As we mentioned in Chapter 3, this solution may lead to non-minimal routing and congestion in the network. Therefore, we propose *Virtual TSV* to help these routers maintaining the connection without using any fault-tolerant routing algorithm. In the case where the *Virtual TSV* is unable to be performed, we also implement the *Serialization* technique which helps the vertical connection establishing only one or two TSV-clusters.

Virtual TSV

When a router is not granted the access to four TSV-clusters, it is disabled. However, if the number of nearby TSVs is larger or equal than four, which is enough for maintaining vertical communication, they can be utilized to establish a connection. A possible connection, which requires four TSV-clusters, may need clusters belonging to the neighboring routers. If these routers do not use these clusters, the disabled router can borrow them for a short period to establish a communication.

Figure 5.6 (a) shows an example of how *Virtual TSV* works where *R(1,0,1)* has a defective cluster (T(N)) and borrows a cluster from *R(1,0,0)*. Because *R(1,0,0)* is unable to find any replacement

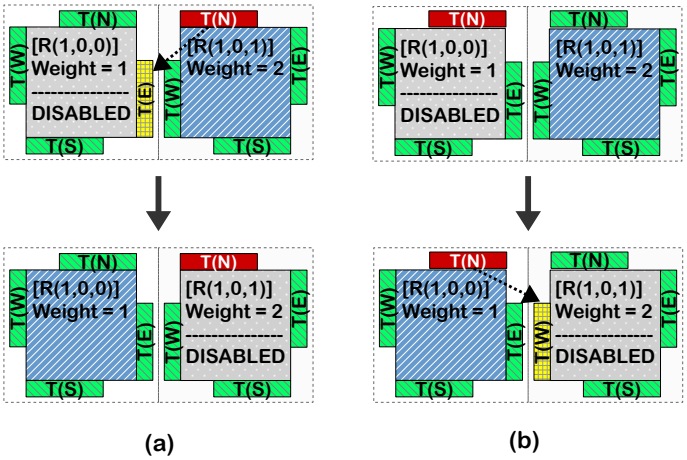

Figure 5.6: Examples of Virtual TSV: (a) return the TSV-cluster to the original router; (b) borrow a cluster from a higher weight router.

for the borrowed cluster (T(E)), it is disabled. When *R(1,0,0)* needs to establish an inter-layer communication, it needs to find at least four TSV-clusters. Assuming that *R(1,0,1)* does not use the borrowed cluster T(E), it is temporarily returned to *R(1,0,0)*. When the packet is completely transmitted, the borrowing cluster is taken back by the router *R(1,0,1)* again.

On the other hand, Fig. 5.6 (b) shows the case where a disabled router *R(1,0,0)* temporarily borrows a TSV-cluster from a higher weight router *R(1,0,1)* to establish an inter-layer connection. For selecting a suitable candidate to temporarily borrow, Algorithm 5.1 is utilized.

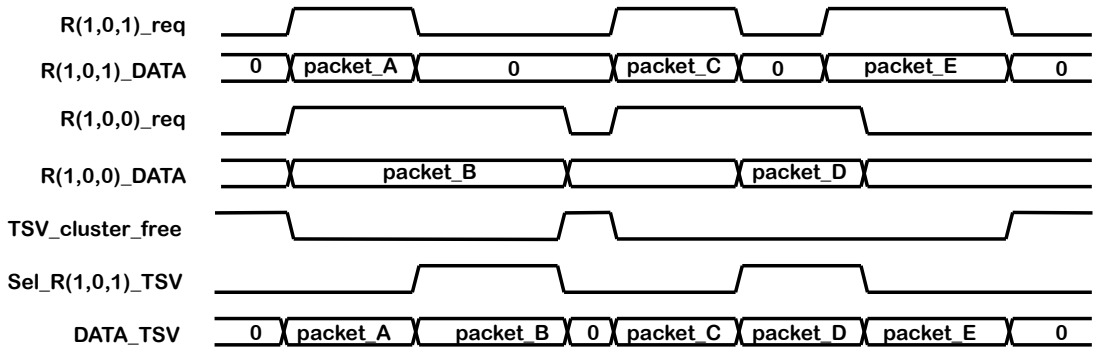

Figure 5.7: Example of Virtual TSV timing diagram.

To better understand Virtual TSVs, the timing diagram of the example previously presented in Fig. 5.6(a) is depicted in Fig. 5.7. If two routers request and use the TSV in different time slots, the request is granted to each of them to use the TSV-cluster separately. When they request at the same time, *R(1,0,1)* is prior to operation due to its higher weight value.

Because there is a case where *R(1,0,1)*, which has the higher priority, occupies the TSV for a long transmission time, *R(1,0,0)* is unable to access the TSV to establish a connection. Moreover, at a high defect-rates, *R(1,0,0)* may not find any suitable candidate for virtual TSV. In order to solve these issues, we adopt the *Serialization* [169] technique to maintain the connection.

Serialization Technique

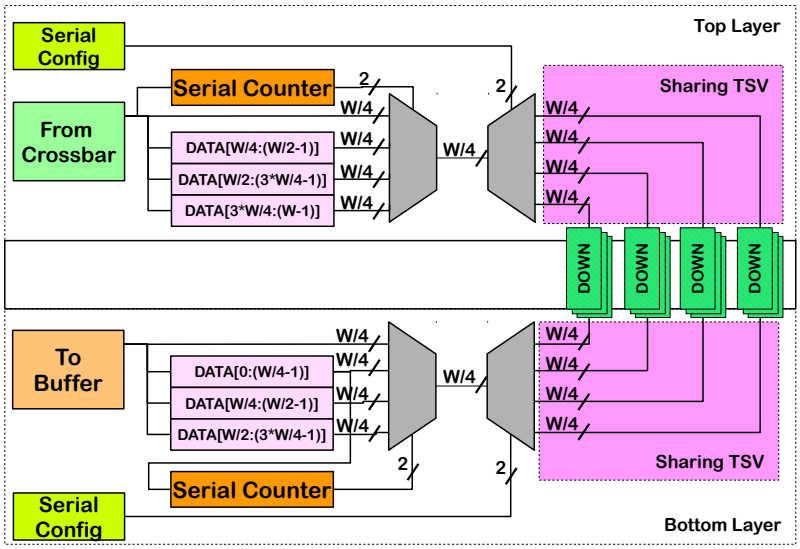

Figure 5.8: Circuit of 1:4 Serialization.

Although the *Virtual TSV* can help the disabled router maintaining its vertical connection, there are still two situations where *Virtual TSV* cannot be performed: (a) there are less than four healthy TSV-clusters, (b) the candidate TSV-cluster is occupied constantly by a higher priority router. In order to solve these cases, we use the *Serialization* technique [169] to maintain the connectivity.

For the serialization, the router needs at least one TSV-cluster to maintain its connection. If there is one available cluster, the 1:4 serialization is used, if there are two available clusters, the 1:2 serialization is established. The up and down directions' output of the crossbar is stored in a register and the serialization module transmits flits over the remained clusters. Figure 5.8 shows the vertical interface between two routers using 1:4 serialization. Two serial counters handle the connection by detecting the transmitting flit. This flit is also stored in a buffer in the transmitting router. By increasing the counter's value which selects the multiplexer, the output width is a quarter of the flit size. Because only one TSV-cluster is utilized, the controller selects the output by using a demultiplexer.

At the receiving router, the input data is cached into a register. There are also a demultiplexer and a multiplexer which are controlled by a serial configuration and a serial counter, respectively. When the corresponding counter reaches "11", the whole flit is transmitted to the buffer. For 1:2 serialization, the first half of each flit is cached and when the remainder arrivals (counter reach "01"), the whole flit is sent to the buffer.

5.4 Evaluation Results

5.4.1 Evaluation Methodology

The proposed system was designed in Verilog-HDL, synthesized and prototyped with commercial CAD tools. The hardware technology parameters are illustrated in Table 5.2. We use NANGATE 45nm library [156] and NCSU FreePDK TSV[155]. The system configurations are depicted in Table 5.3.

First, we evaluate the defect-rate by inserting faults (defects) into TSV-clusters and evaluate the reliability of the proposed 3D-NoC system. Second, we use both synthetic and realistic traffic patterns as benchmarks to study the performance of the proposed system in comparison to the baseline model [149]. Third, we evaluate the hardware complexity of a single 3D router and compare our system with other proposed approaches [80, 81].

Table 5.2: Technology parameters.

Parameter	Value
Technology	Nangate 45 nm [156] FreePDK3D45 [155]
Voltage	1.1 V
TSV's size	$4.06\mu m \times 4.06\mu m$
TSV pitch	10 μm
Keep-out Zone	15 μm

Table 5.3: System configurations.

Parameter	Value
# ports	7
Topology	3D Mesh
Routing Algorithm	Look-ahead routing
Flow Control	Stall-Go
Forwarding mechanism	Wormhole
Input buffer	4
Flit width	44

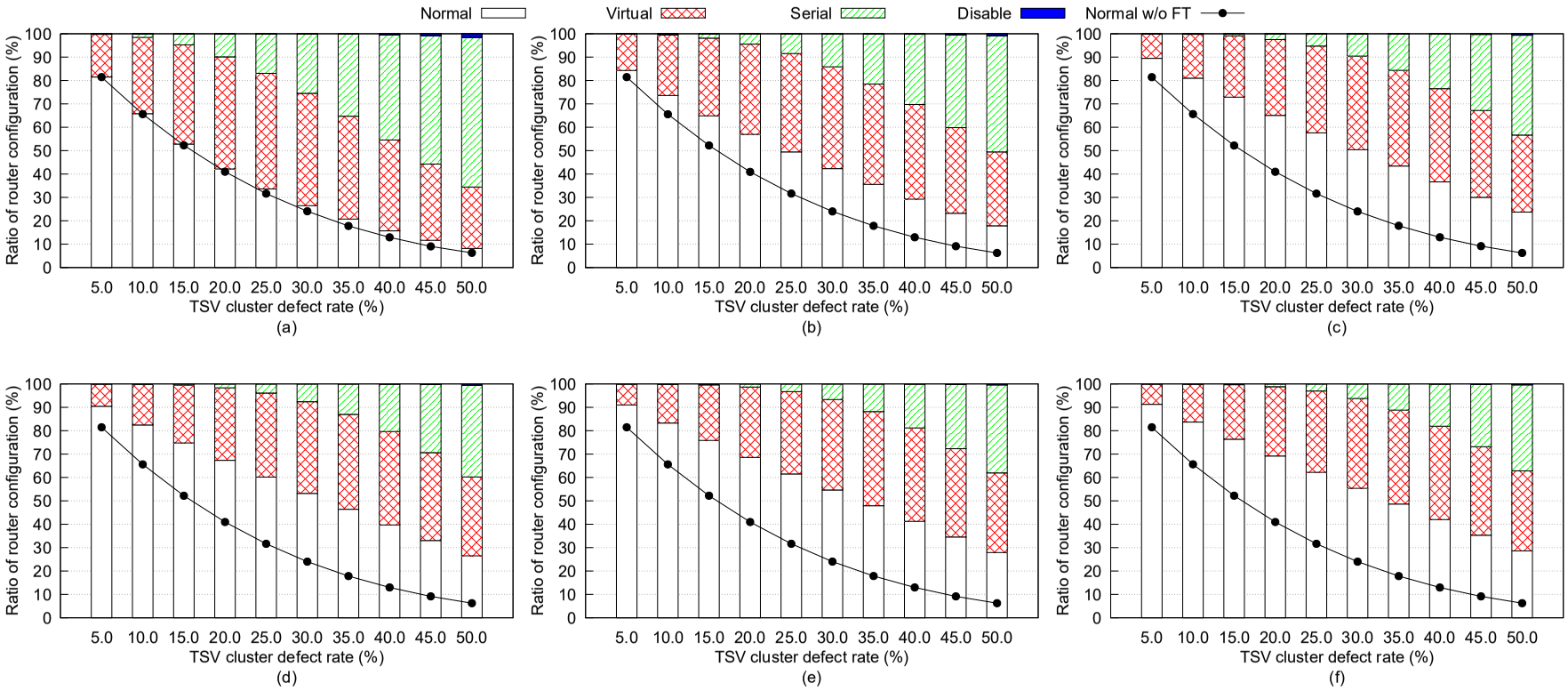

Figure 5.9: Defect-rate evaluation with different layer sizes: (a)2 × 2 (4 routers, 16 TSV clusters); (b) 4 × 4 (16 routers, 64 TSV clusters); (c) 8 × 8 (64 routers, 256 TSV clusters); (d) 16 × 16 (256 routers, 1024 TSV clusters); (e) 32 × 32 (1024 routers, 4096 TSV clusters); (f) 64 × 64 (4096 routers, 16384 TSV clusters).

5.4.2 Defect-rate evaluation

In this section, we provide the impact of the different defect-rates. To demonstrate the scalability of the proposed architecture, we set up several layer sizes: 2×2, 4×4, 8×8, 16×16, 32×32, and 64×64. TSVs are grouped in clusters as presented in Section 5.2. We also vary the TSV-cluster defect-rates: from 5% to 50%. Because our technique focuses on the cluster defect, random defects are assumed to be dealt with typical redundancy methods. The position of cluster defects are generated randomly and we perform the proposed algorithms with 100,000 different samples and calculate the average results. We measure the ratio of four types routers in the layer: *Normal* (healthy or corrected), *Virtual* (router with virtual TSV), *Serial* (router using serialization) and *Disabled* (disabled routers). We also compare the obtained results with *"Normal w/o FT"* (Normal without Fault-Tolerance), where no fault-tolerant method is used and the router vertical connection having defects is disabled.

As shown in Figure 5.9, the system mostly operates without disabling any vertical connections with fault-rates under 50%. Thanks to the *Virtual TSV* and *Serialization* techniques, the routers having less than four clusters are still able to work. Even at less than 20% of defect-rate, there are less than 10% of serialization connections in all simulated layer sizes. With 50% of defect-rate and a 2×2 layer size, the disabled router rate is negligible with about 1.565%. This can be easily dealt using a light-weight fault-tolerant routing algorithm. When the layer size increases to be larger than 8×8, the number of disabled connections is mostly insubstantial. At 50% defect-rate, the disabled router ratio is nearly 0.63%, 0.50%, 0.44% and 0.42% with 8×8, 16×16, 32×32, and 64×64 layer sizes, respectively. However, these defect-rates are extremely high; thus, our proposed mechanism can be considered as highly reliable.

In comparison to the system without fault-tolerant methods, there is a significant improvement in terms of healthy connections, especially at large layer sizes. In Figure 5.9, the percentage of routers having four healthy TSV-clusters is represented by the "Normal w/o FT" curve. At 50% defect-rate, the average ratio of normal routers has been improved by 29.83%, 186.26%, 280.76%, 324.42%, 346.74%, and 257.79% for 2×2, 4×4, 8×8, 16×16, 32×32, and 64×64 layer sizes, respectively. The improvements are lesser with small layer sizes such as: 2×2 or 4×4. However, thanks to the Virtual TSV and Serialization, the workable connection rates have nearly reached

100%.

In summary, this evaluation has shown a significant improvement in terms of reliability provided by our proposed mechanism. Thanks to the efficiency of the proposed architecture and algorithm, the system can mostly maintain all vertical connections, even at extremely high defect-rate (50%). This evaluation also shows the proposed mechanism's ability to remain efficiently scalable. The proposal can be applied from a small layer size (e.g., 2×2) to a larger one (e.g., 64×64). The evaluation is also performed with a solid number of tests (100,000) which strongly demonstrates the efficiency of the proposed approach. There were some cases where some routers were disabled; however, they can be recovered by simple and light-weight fault-tolerant routing algorithms.

5.4.3 Performance Evaluation

Table 5.4: Simulation configurations.

Parameter/System		Value
Network Size ($x \times y \times z$)	Matrix	$6 \times 6 \times 3$
	Transpose	$4 \times 4 \times 4$
	Uniform	$4 \times 4 \times 4$
	Hotspot 10%	$4 \times 4 \times 4$
	H.264	$3 \times 3 \times 3$
	VPOD	$3 \times 2 \times 2$
	MWD	$2 \times 2 \times 3$
	PIP	$2 \times 2 \times 2$
Total Injected Packets	Matrix	1,080
	Transpose	640
	Uniform	8,192
	Hotspot 10%	8,192
	H264	8,400
	VPOD	3,494
	MWD	1,120
	PIP	512
Packet's Size	Hotspot 10%	10 flits+10% on hostpot nodes
	Others	10 flits

The previous section has proved the reliability of the proposed solution. In this section, we evaluate the system performance under TSV-cluster defects. As we previously mentioned, works in [107, 108] have demonstrated the low utilization rate of the vertical connections; nevertheless, the performance degradation in highly stressed networks has to be investigated. To evaluate the performance of the proposed system and keep fair comparisons to the baseline, we adopted both synthetic and realistic traffic patterns as benchmarks. We selected Transpose [161], Uniform

[161], Matrix-multiplication [159], and Hotspot 10% [161] as the synthetic benchmarks. Within these benchmarks, Uniform and Hotspot 10% have the highest stress on the network and both Transpose and Matrix-multiplication use vertical connections for all of their connections. For realistic benchmarks, we chose H.264 video encoding system [162], Video Object Plane Decoder (VOPD), Picture In Picture (PIP) and Multiple Window Display (MWD) [163]. Moreover, the network's performance under TSV defects is the focus of these evaluations, the realistic and synthetic benchmarks provide a vast diversity to study the impact of the fault-tolerance. The configurations of these benchmarks are shown in Table 5.4. The packets are injected continuously into the network. In other words, we executed the benchmarks until the saturation point of the network is reached. In order to keep a fair comparison, only TSV defects are injected. This means that the other fault-tolerance mechanisms [153] are disabled to not affect the performance.

Latency Evaluation

In this experiment, we evaluate the performance of the proposed architecture in terms of Average packet Latency (APL) over various benchmark programs and defect-rates. The simulation results are shown in Fig. 5.10 (a). From this graph, we notice that with a 0% of defect-rate, the system's tolerance has similar performance in comparison to the baseline system.

When we increase the defect-rates in the proposed system, it has demonstrated additional impacts on APL. At a 1% fault-rate using Matrix, Uniform, Transpose, and Hotspot 10% benchmarks, the system increases the APL by 83.24%, 64.46%, 11.30% and 66,55%, respectively. These high impacts are due to the occurrence of bottlenecks inside the network. Because all vertical connections are utilized, Virtual TSV has caused congestions by sharing the TSV between two routers. The serialization is already a bottleneck technique. These bottlenecks effects are even higher at a 30% of defect-rate where the APL can be over 3 times that of the 0% case in the synthetic benchmarks.

With H.264, PIP, MWD and VOPD benchmarks, the APL incrementation are significantly reduced due to the low utilization of TSV. We can observe the identical performance of VOPD benchmark from a 1% to a 30% defect-rates. With the PIP benchmark, the system under 1% defect-rate has similar performance to 0% thank to the optimization process which disables the unused clusters. With the MWD and H.264 benchmarks, the impact on APL is gradually in-

creased when increasing the defect-rate. Even at a 30% of defect-rate, the APL values of MWD and H.264 are increased by 129.91% and 60.04%, respectively. Because there is no optimized routing technique for these benchmarks, the bottleneck effect is expected to happen.

Although there are significant impacts on latency, the system has proven to work without major issues in all benchmarks.

Throughput Evaluation

Figure 5.10 (b) depicts the throughput evaluation with different benchmarks. At 0% defect-rate, the proposed system's throughput is similar to that of the baseline. When defects are injected into the system, we can observe some degradation in throughput caused by the bottleneck effects on the system. Similarly to APL, the throughput degradation in realistic traffic benchmarks (VOPD, H.264, MWD and PIP) are significantly better than the synthetic ones. The system at a 20% defect-rate provides a decreased throughput by 71.17%, 64.36%, 67.44% and 64.37% for Transpose, Uniform, Matrix, and Hotspot 10%, respectively. At the same defect-rate, VOPD, MWD, PIP and H.264 have 46.03%, 50.04% 28.17%, and 19.79% of throughput degradation. This lower impact is caused by the low utilization of vertical connection rate and the optimization process. The throughput of realistic benchmarks are naturally smaller than the synthetic ones because of the specific tasks order of execution that was observed in the task graphs [162, 163].

Although there is a considerable degradation in the throughput evaluation, the system still maintains over 0.1 *flit/node/cycle* in the highly stressed benchmarks, even at extremely high defect-rates.

5.4.4 Router Hardware Complexity

Table 5.5: Hardware complexity of a single router.

Model		**Area** (μm^2)	**Power** (mW)			**Speed** (Mhz)
			Static	Dynamic	Total	
Baseline router [149]		18,873	5.1229	0.9429	6.0658	925.28
Proposal	Router	29,780	10.017	2.2574	12.3144	613.50
	Serialization	3,318	0.9877	0.2807	1.2684	-
	TSV Sharing	5,740	0.7863	0.2892	1.0300	-
	Total	38,838	11.7910	2.8273	14.6128	537.63

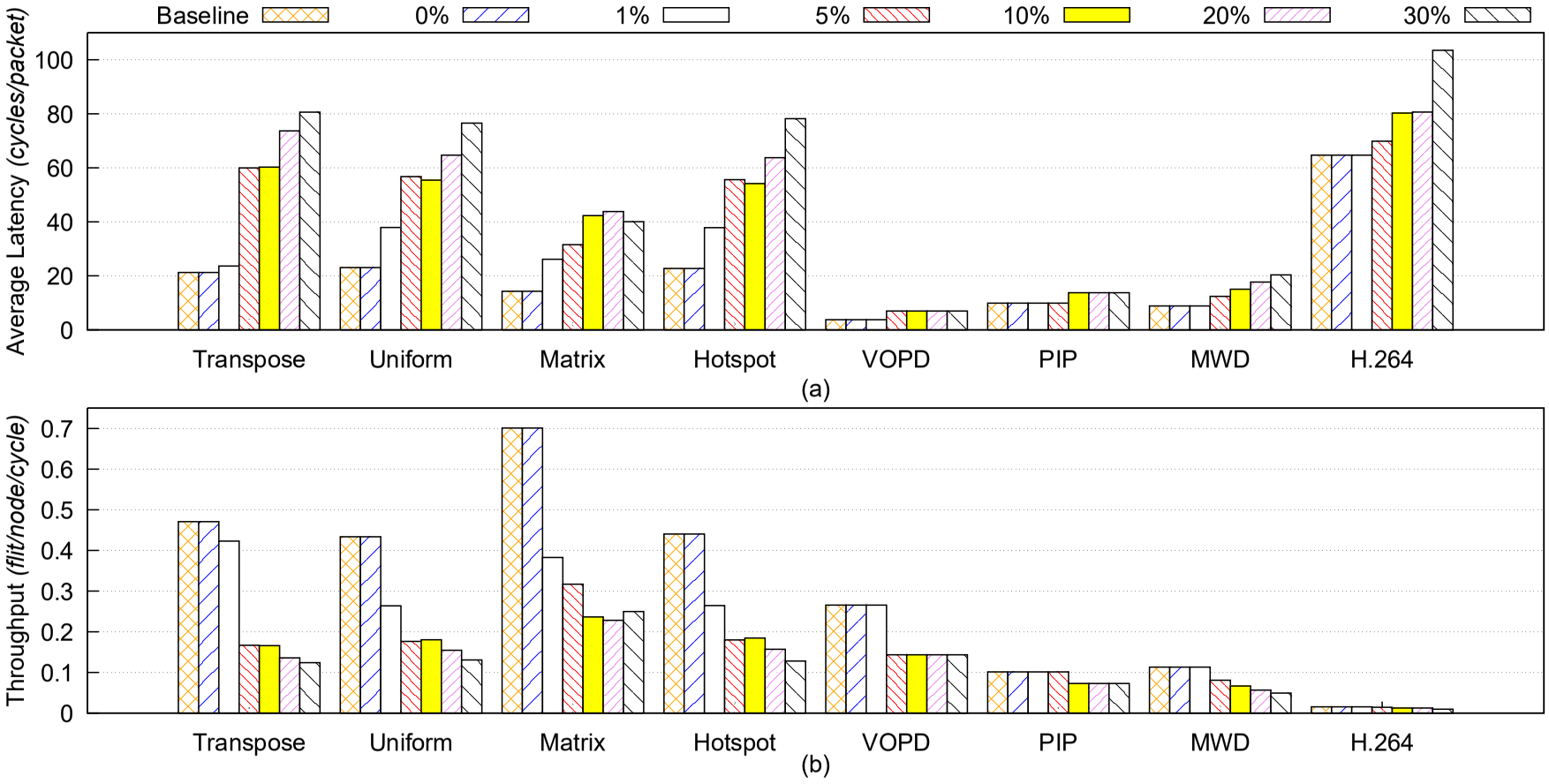

Figure 5.10: Evaluation result: (a) Average Packet Latency; (b) Throughput.

Table 5.5 illustrates the hardware complexity results of the proposed router in terms of area, power (static, dynamic, and total), and speed. In comparison to the router in which we implement the proposed techniques, the area and power consumption have increased by 30.42% and 18.66%, respectively. The maximum speed has also slightly decreased by 12.37%. In comparison to the baseline model, the proposed system almost doubles the area cost and power consumption while decreasing the maximum frequency by about 50%. However, the TSV sharing and Serialization modules incur reasonable area and power consumption overheads which are 47.99% and 38.89% in comparison to the baseline router, respectively. Here, the TSV Sharing module handles the sharing algorithm and the *Virtual TSV* process and the Serialization module helps the router communicate in *Serialization* mode.

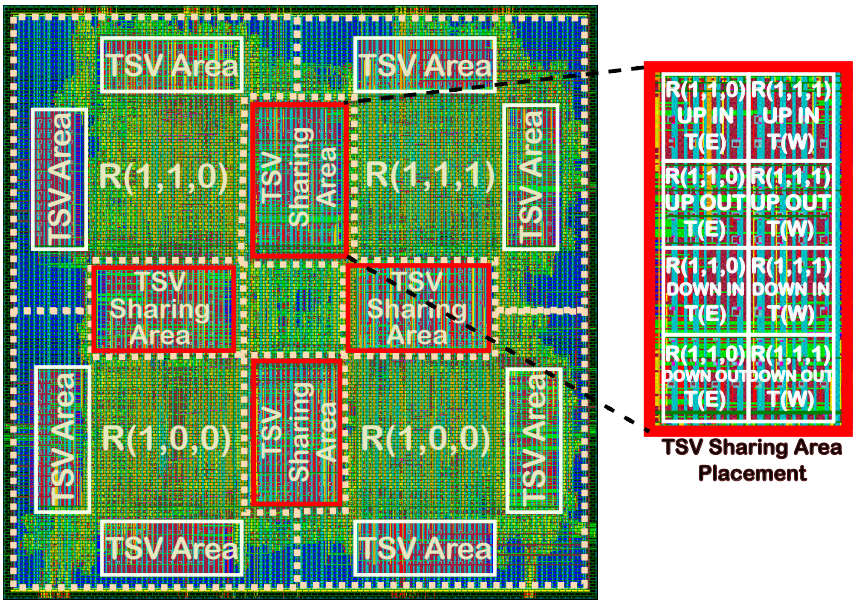

Figure 5.11: Single layer layout illustrating the TSV sharing areas (red boxes). The layout size is $865\mu m \times 865\mu m$.

The layout of a layer is shown in Fig. 5.11 where the sharing TSV areas are depicted by the red boxes. As shown in Section 5.2.2, the TSV sharing area consists of eight clusters. For each port, *R(1,1,1)* can access *T(E)* of *R(1,1,0)* and *R(1,1,0)* can access *T(W)* of *R(1,1,1)*. By placing the shared cluster areas between two routers, we can ensure a small extra wire delay for rerouting.

5.4.5 Comparison

In order to understand the efficiency of the proposed approach, we compare it with existing solutions as shown in Table 5.6. Here, we analyze our proposal with a network size of $4 \times 4 \times 4$.

Because the router and its TSV clusters structure are identical, similar results can be obtained with the others network sizes. *TSV Grouping* [81] optimized the configuration of redundancy to deal with TSV-cluster defects. *TSV Network* [80] established TSVs into networks which allow routing from defected TSVs to redundant ones. We select the best results on these two works [80, 81] for the comparison. From this table, we can see that the average area of our proposal is 151.47μm^2 per TSV and, for a TSV size of 10μm $\times$ 10μm, the area overhead is about 51.47%. The *TSV Network* [80] has similar value for 4:2 configuration (4 original TSVs and 2 redundant TSVs). With 8:4 configuration, *TSV Grouping* also obtained an average area of 151.86μm^2.

On the other hand, the other configurations obtained lower area overheads. Nevertheless, we have to note that our arbiter not only consists of the rerouting circuit (similar to the multiplexers in *TSV Network* and *TSV Grouping*); but, also includes an online adaptive algorithm designed in hardware, in addition to the *Virtual TSV* and *Serialization* techniques. Both *TSV Grouping* and *TSV Network* have to require additional dedicated circuitry to recover from the cluster defects.

In terms of reliability, the proposed approach has proven its high resiliency, as previously shown in Section 5.4.2. *TSV Grouping* demonstrated a 100% of yield rate under a defect-rate of 1% and *TSV Network* obtained nearly 100% in the most cases. However, their approaches are different than our scheme, where they add redundancy to correct the defect TSVs. As a result, if the number of defected TSVs is larger than the number of redundant ones, they are unable to recover from the defected clusters. On the other hand, our technique can significantly improve the reliability by providing 98.11% of workable routers at 50% of defected TSV-clusters. Moreover, at low defect rates (e.g. under 5%), our proposal also ensures 100% of working connection and demonstrates small performance degradations in the realistic traffic pattern benchmarks. Even with disabled vertical connections, the reliability of our system can also be improved (i.e., covering the remaining 1.89%) by using a lightweight fault-tolerant routing which would have a negligible impact on the area overhead.

Table 5.6: Comparison results between the proposed approach and the existing works.

Model	TSV Network [80]					TSV Grouping [81]			This work	
Technology	65 *nm*					N/A			45 *nm*	
#TSV	1000					6000			8448	
Configuration	4:2	8:2	$4 \times 4 : 8$	$8 \times 8 : 16$	$16 \times 16 : 32$	4:4	8:4	20:5	$11 \times 4 \times$ 4:0	
#Spare TSV	512	256	512	256	128	6000	3000	1500	0	
45*nm* Arbiter Area (μm^2)	372 [2]	744 [2]	1,116 [2]	1,116 [2]	1,116 [2]	11,160 [1]	11,160[1]	12,555[1]	434,784[3]	
Average Area/TSV (μm^2)	151.572	126.244	152.316	126.716	128.03	113.916	151.86	127.09	151.47	
Reliability	100%	99%	100%	100%	100%	100%			100%	98.11%
Fault Assumption	($\delta_{TSV} = 0.01\%$, $\alpha = 2$)[4]					($\delta_{TSV} = 1\%$,$\alpha = 2$)[4]			($\delta_{cluster} = 1\%$)[4]	($\delta_{cluster} = 50\%$)[4]

[1] The authors use 2:1 multiplexers [81]. For comparison, we use the area cost of multiplexer from Nangate 45*nm* [156] (MUX2_X1: 0.186μm^2) .
[2] The authors use 1-to-3 multiplexers [80] which consists of two MUX2_X1 multiplexers ($2 \times 0.186 \mu m^2$ [156]).
[3] For fair comparisons, our arbiter only consists of the TSV sharing and serialization modules as shown in Table 5.5.
[4] δ: defect-rate. α: parameter of Poisson distribution [80, 81].

5.5 Conclusion and Discussion

This chapter presented an adaptive and scalable sharing methodology for TSVs in 3D-NoC systems to deal with the TSV-cluster defects. The results have proven the system ability to provide high reliability that can reach up to 346.74% increase in functional routers. Moreover, the proposed approach can correctly work with a reasonable degradation, even under a 30% of defect-rate. The hardware complexity has shown a small overhead in terms of area cost (30.42%), power consumption (18.66%) and maximum frequency (12.37%) of router's logic. Since no TSV redundancy is required in the proposed architecture and algorithm, we show that it is possible to provide a highly reliable system while maintaining the overhead reasonable.

The final architecture has integrated the soft error, hard fault and TSV defect tolerance mechanisms. However, there are also numerous existing works on fault-tolerance. Depending on the requirements, designers can decide to use a suitable technique or propose a better one. During the development process, we observe the difficulty in evaluating the efficiency of the proposed system, especially in terms of reliability. Therefore, we needed to develop a platform to help assessing NoC systems. Consequently, the next chapter presents the proposed reliability assessment.

6

Reliability Assessment for 3D-NoCs

AULT TOLERANCE architectures and algorithms have been broadly developed in numerous areas, especially with the increasing of device's vulnerability. Depending on the trade-off between performance, overheads and fault-tolerance capacity, designers have to consider different techniques. To understand the reliability of the system, even before completing the design, assessment is one of the most common methods. Even after completing the design, manufacturing and testing are still costly which drives the need for reliability analyses. However, from the state-of-the-art, reliability assessment for NoC systems is still immature which needs to be properly investigated.

In this chapter, we present an analytical assessment to evaluate fault-tolerant NoC architectures as a part of *Design for Reliability* (in the preliminary design stage). Instead of building a complex analysis, we aim to provide a simple and effective method to address the reliability of NoCs. The

proposed method can work in conjunction with other reliability analyses, such as physical level analysis, simulation evaluation, and system level analysis. We build a strategy to help analyze a complex network-based system in separated parts and merge them all together. This proposal can be applied from the design definition phase, through preliminary design and up to the detailed design phase. Besides of the analytical assessment, we also introduce the Monte-Carlo MTTF simulation which helps designers evaluate the reliability of their systems.

The rest of the chapter is organized as follows: Section 6.1 classifies the fault-tolerance concept. In Section 6.2, we overview the Markov-state model in addition to the main assumptions and definitions needed for the proposed method. In Section 6.3, we present our reliability assessment methodology. The MTTF Monte-Carlo simulation is presented in Section 6.4. In Section 6.5, we provide the evaluation results. The last section is dedicated for conclusions and discussion.

6.1 Fault-tolerant Classification

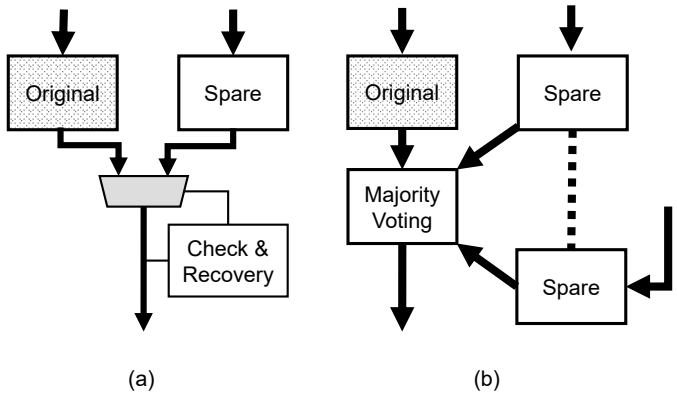

Figure 6.1: Redundancy fault-tolerant models: (a) Check and recovery; (b) Majority voting.

In [41], the fault-tolerance models and techniques for NoCs are surveyed and organized. Hereafter, we briefly summarize them and categorize them into two basic approaches:

(1) *Redundancy*: consists of temporal or spatial redundancy to handle the faults, as shown in Fig. 6.1.

(2) *Self-reconfiguration*: as represented in Fig. 6.2, the fault-tolerance adapts to the occurrence of faults to alleviate their impact.

The *redundancy* technique can use a replica of a given module as a back-up when the original system fails, as shown in Fig. 6.1(a). It also can use multiple replicates run in parallel which can

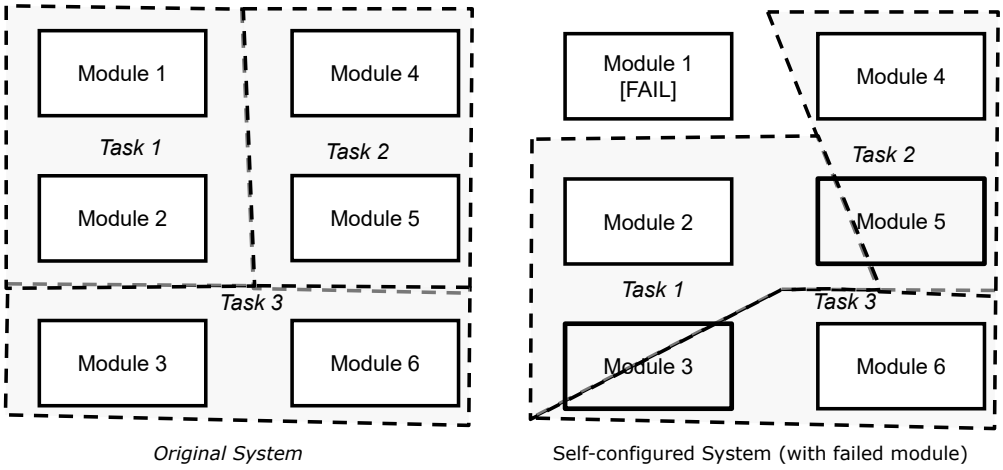

Figure 6.2: Self-configuration fault-tolerant models.

be activated on the fly, as depicted in Fig. 6.1 (b). The *self-configuration* method re-optimizes the system to ensure the system's function. As represented in Fig. 6.2, if a *module* fails, its task can be migrated and shared with other healthy modules. Thus, the system can maintain its correct functionality. The redundancy and self-configuration method can be applied in whether software (redundant execution, checkpoint) or hardware (majority voting, redundancies) approach.

Along with the recovery methods, one of the important criteria of fault-tolerance is error detection. Depending on the reliability requirements, NoC systems can use an online [44, 100, 117] or an offline [118, 126] error detection method.

6.2 Markov-state Model and Assessment Definitions

This section presents the basic concepts, definitions and assumptions used in the proposed analytical assessment methodology used to approximate and evaluate the reliability of fault-tolerant Network-on-Chip systems. Specifically, we adopt the Markov-state model to analyze the reliability of the fault-tolerance mechanisms. Therefore, we firstly start by giving an overview of the Markov-state model followed by the different assumptions and models adopted in the proposed assessment method.

6.2.1 Markov State model overview

A system operating with faults can be converted into a Markov state model [134, 170, 171] as shown in Fig. 6.3. Each state S_i of the Markov model represents a possible *status* (event, configu-

ration, behavior) of the system. A *status* can be a case where one or multiple elements of the system fail. If the system's operation in state S_i is maintained correctly, we define S_i as "healthy". If the system is unable to operate correctly in this state, we define it as "faulty".

The reliability of a system can be defined as a time-dependent probability function $R(t)$ in the time domain ($R^*(s)$ in *Laplace* domain) and can be evaluated using the *Mean Time To Failure* (MTTF) [134] calculation as follows:

$$\mathbf{MTTF} = \int_{t=0}^{\infty} R(t) = \lim_{s \to 0}(R^*(s)) \tag{6.1}$$

We assume that the system's status is converted to a set named $\mathbb{S}$ with m states: $S_0 ... S_{m-1}$. We use S_0 to represent the initial state of the system. The set of *healthy* states and the set of *faulty* states are defined, respectively, as follows:

$$\mathbb{H} \triangleq \{S_i \in \mathbb{S} | \text{the system works correctly}\} \tag{6.2}$$

and

$$\mathbb{F} \triangleq \{S_i \in \mathbb{S} | \text{the system not working}\} \tag{6.3}$$

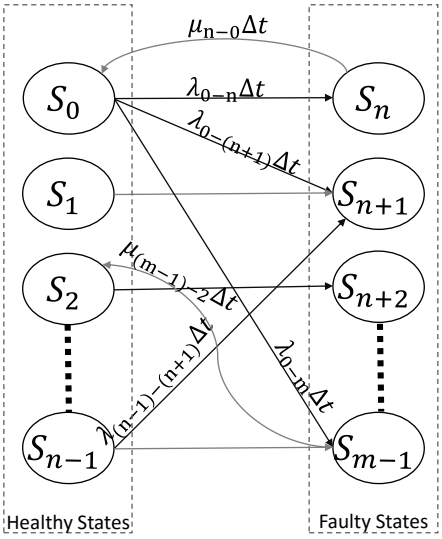

Figure 6.3: A Markov-state reliability model for an m states system with n non-faulty states.

Fig. 6.3 shows each state S_i of $\mathbb{S}$. A transition between two states has a specific rate which is defined as follows:

- λ is the fault rate of a component in the system. It can represent a transition from an element of set $\mathbb{H}$ to an element of set $\mathbb{F}$.
- μ is the repair rate of a component in the system. It can represent a transition from an element of set $\mathbb{F}$ to an element of set $\mathbb{H}$.

The reliability of a system can be obtained by summarizing the probabilities of its states and the reliability function. Given a state S_i, which has j input transitions (σ) from j states $S_{n,i}$ ($n = 1, 2, ..., j$) and k output transitions (γ) to k states $S_{m,i}$ ($m = 1, 2, ..., k$), the derivative of the probability of the state in the time domain is given as follows:

$$p'_{S_i}(t) = -\sum_{n=1}^{j} \sigma_n p_{S_{n,i}} + \sum_{m=1}^{k} \gamma_m p_{S_{m,i}} \tag{6.4}$$

Note that with each input or output transition, based the set of the state, the transition rates (σ and γ) can be either a failure (λ) or repair (μ) transition. By converting Eq. 6.4 to the *Laplace domain* [134], we obtain the below equation:

$$sP_{S_i}(s) - p_{S_i}(0) = -\sum_{n=1}^{j} \sigma_n P_{S_{n,i}} + \sum_{m=1}^{k} \gamma_m P_{S_{m,i}} \tag{6.5}$$

When applying the above for all states, we obtain m equations of $(P_{S_0}, P_{S_1}, ..., P_{S_{m-1}})$ and the reliability function $R^*(s)$ can be defined as the sum of probabilities of being in *healthy* states:

$$R^*(s) = P(\mathbb{H}) = \sum_{S_i \in \mathbb{H}} P(S_i) \tag{6.6}$$

Finally, to obtain the MTTF value, Eq. 6.1 can be used.

6.2.2 Assumptions

In a fault-tolerant system, the fault-tolerant method has to improve the reliability of the system. As a result, the MTTF value has to be also increased. Therefore, we propose the *Reliability Acceleration Factor* (RAF) to denote the efficiency of fault tolerance, represented as:

$$\mathsf{RAF} = \frac{\lambda_{original}}{\lambda_{FT}} = \frac{\mathbf{MTTF}_{FT}}{\mathbf{MTTF}_{original}} \geq 1 \tag{6.7}$$

Where:

- $\mathbf{MTTF}_{original}$ is the Mean Time To Failure of the original system.
- $\mathbf{MTTF}_{FT}$ is the Mean Time To Failure of the fault-tolerant system.
- λ is the fault rate and it is the inverse value of $MTTF$.

The transitions have dedicated "steady-state" rates which need to be predefined [134]. Therefore, we make the following assumptions regarding a given system failure.

- The system starts with a default state where all components are in the *healthy* state. In Fig. 6.3, the initial status is: $p_{S_0}(0) = 1$ and $p_{S_i}(0) = 0$ with $i \neq 0$.
- The failure rates are constant.

Since the fault-rate depends on the technology parameters, running environment and operating circumstances which are not easy to obtain in the early analysis. Therefore, we assume the "raw" fault-rate (i.e., original fault-rate with no fault-tolerance support) according to the following assumptions:

- The fault-rate has a linear relationship with the area cost of the module.
- The fault-rate has a linear relationship with the operating time of the module.
- The fault-rate is affected after a module is attached to a system.

Thus, for a system with k components, its fault rate is given by:

$$\lambda_{system} = \frac{1}{\mathbf{MTTF}_{system}} = \sum_{i=1}^{k} f_i \pi_i \lambda_{unit} \tag{6.8}$$

where *unit* is a selected module as a reference for calculation. π_i is the fault-rate ratio between the component and the *unit*. It can be defined as the area cost ratio and operating time ratio [109, 139]. f_i is the fault-rate ratio after attaching the component to the system ($f = 1/RAF$). The fault rate can be reduced ($f_i < 1$) by applying fault-tolerance; otherwise, it remains as $f_i = 1$.

In a typical fault-tolerant system, a faulty part can be repaired with a specific repair rate (μ) after being detected. This rate is given by the module managing the fault-tolerance mechanism.

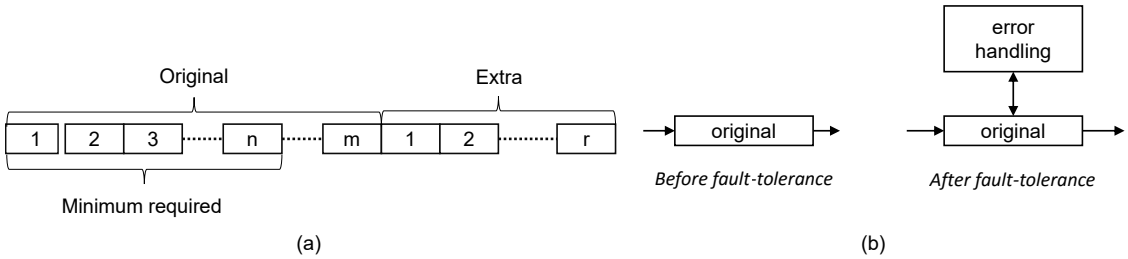

Figure 6.4: Classified Model: (a) Model 1 - Spare, (b) Model 3 - Error handling.

6.2.3 Classified Model

We categorize fault-tolerance architectures [41, 139, 172] into four basic models where each model is treated separately and differently: *Non-fault tolerant model*, *Spare model*, *Fault-reduced model*, and *Error handling model* as:

- *Model 0 - Non-fault tolerant model*: This model is applied for the module without fault-tolerance capabilities. Its fault-rate can be obtained by Equation 6.8 or based on physical-level analysis.

- *Model 1 - Spare model*: As represented in Fig. 6.4 (a), we assume the considered module has m separated identical parts which can function with at least n parts. In the redundancy method, an r extra spare parts are added in the design stage. The *Self-configuration* (previously presented in Section 6.1) can be modeled without extra parts. In fact, it has $n < m$ and allows the system to fail at most $(m - n)$ submodules.

- *Model 2 - Fault-reduced model*: This model is aimed for fault-reduced systems. The reducing of fault-rate can be given by a special technique (e.g., error correcting code [65]). This model can help applying the other former analyses (physical-level or system-level) to the new system.

- *Model 3 - Error handling model*: As shown in Fig. 6.4 (b), this model is designed for error detection and management modules. The detection module also adds a new rate to the overall system. The fault-rate of the original module can be reduced by using *Model 1* or *Model 2*.

6.3 Quantitative Reliability Assessment

Dividing:

1. A Network-on-Chip consists of N_R routers.
2. A router is divided into several modules: $M_0, M_1, ..., M_a$.
3. Each module can be modeled as one of the predefined models (Model 0-3).

Conquering:

1. For each module $M_i = M_0, M_1, ..., M_a$ of a router, analyze it using one of the given strategies:
   - If the module M_i is not fault tolerant (Model 0), it is given reliability values by Eq. 6.8.
   - If the module M_i is a spare model or a reconfigurable module (Model 1), it is given a failure rate by *Strategy 1*.
   - If the module M_i is reduced failure by applying a special technique (Model 2), use *Strategy 2*.
   - If the module M_i is a typical fault-tolerant module (Model 3), after applying *Strategy 1 or 2* its final failure rate is given by calculation in *Strategy 3*.

Merging:

1. A router reliability is obtained by *Router Merging*.
2. A network reliability is obtained by *Network Merging*.

Figure 6.5: Reliability Assessments for Fault-Tolerant Network-on-Chip.

In this section we present a detailed explanation on how to evaluate the reliability of the different components that can constitute a fault-tolerant NoC system. Figure 6.5 shows the three main steps that are necessary to obtain a comprehensive reliability assessment: A network is divided into routers which are also divided into modules (*Dividing*). After dividing, the modules are analyzed and classified according to their appropriate model, and the suitable strategy is applied (*Conquering*). The final reliability is obtained by merging all modules together (*Merging*).

6.3.1 Conquering

We consider the components of a router by using the bellow strategies which are applied to each one of the four basic models presented in the previous section.

Strategy 0

Applied for *Model 0 - Non-fault tolerant model* where if the module is not fault-tolerant, its failure rate is simply estimated using Eq. 6.8.

Strategy 1

Applied for *Model 1 - Spare model*. This strategy handles hard faults using spare modules or by reconfiguring an alternative part.

We assume that the considered module has m separate identical parts and can function with at least n parts. In order to enhance the reliability, an extra r spare parts are added in the design stage. f is the number of parts that are faulty in a state. Equation 6.2 can be then reformulated as:

$$\mathbb{H} \triangleq \{S_i \in \mathbb{S} | m + r - f \geq n\} \tag{6.9}$$

The Markov state model can be built as shown in Fig. 6.6. Each state is labeled with the number of healthy parts and the failure-rate is indicated by Eq. 6.8.

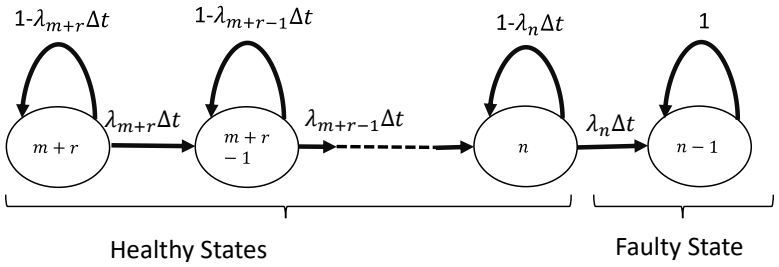

Figure 6.6: A Markov-state reliability model for spare modules.

The original system consists of m parts, and its MTTF can be expressed as:

$$\mathbf{MTTF}_{original} = \frac{1}{m}\frac{1}{\lambda_{single-part}} \tag{6.10}$$

By applying Eq. 6.1 based on the probability of healthy states, the MTTF can be expressed as:

$$\mathbf{MTTF}_{FT} = \Sigma_{i=n}^{m+r}\frac{1}{\lambda_i} = \frac{1}{\lambda_{single-part}}(\Sigma_{i=n}^{m-1}\frac{1}{i} + \frac{1}{m} + \Sigma_{i=m+1}^{m+r}\frac{1}{i}) \tag{6.11}$$

Lemma 1: The RAF values can be calculated as follows:

$$\mathsf{RAF}_{conv.} = \frac{\mathbf{MTTF}_{FT}}{\mathbf{MTTF}_{original}} = \Sigma_{i=n}^{m+r}\frac{m}{i} = 1 + \Sigma_{i=n}^{m-1}\frac{m}{i} + \Sigma_{i=m+1}^{m+r}\frac{m}{i} \tag{6.12}$$

Proof 1: We consider a system originally having m identical parts, r extra parts and requires at least n parts for maintaining its function. The derivatives of probabilities of each states in time

domain are calculated as follows:

$$p'_{m+r} = -\lambda_{m+r} p_{m+r} \tag{6.13}$$

$$p'_{m+r-1} = \lambda_{m+r} p_{m+r} - \lambda_{m+r-1} p_{m+r-1} \tag{6.14}$$

$$p'_{m+r-2} = \lambda_{m+r-1} p_{m+r-1} - \lambda_{m+r-2} p_{m+r-2} \tag{6.15}$$

$$\ldots \tag{6.16}$$

$$p'_{n} = \lambda_{n+1} p_{n+1} - \lambda_{n} p_{n} \tag{6.17}$$

$$p'_{n-1} = \lambda_{n} p_{n} - \lambda_{n-1} p_{n-1} \tag{6.18}$$

By converting to *Laplace domain*, the probabilities of states are expressed as:

$$sP_{m+r} - p_{m+r}(0) = -\lambda_{m+r} P_{m+r} \tag{6.19}$$

$$sP_{m+r-1} - p_{m+r-1}(0) = \lambda_{m+r} P_{m+r} - \lambda_{m+r-1} P_{m+r-1} \tag{6.20}$$

$$sP_{m+r-2} - p_{m+r-2}(0) = \lambda_{m+r-1} P_{m+r-1} - \lambda_{m+r-2} P_{m+r-2} \tag{6.21}$$

$$\ldots \tag{6.22}$$

$$sP_{n} - p_{n}(0) = \lambda_{n+1} P_{n+1} - \lambda_{n} P_{n} \tag{6.23}$$

$$sP_{n-1} - p_{n-1}(0) = \lambda_{n} P_{n} - \lambda_{n-1} P_{n-1} \tag{6.24}$$

By resolving the above equations, the final probabilities in *Laplace domain* are:

$$P_{m+r} = \frac{1}{s+\lambda_{m+r}} \tag{6.25}$$

$$P_{m+r-1} = \frac{\lambda_{m+r}P_{m+r}}{s+\lambda_{m+r-1}} = \frac{\lambda_{m+r}}{(s+\lambda_{m+r})(s+\lambda_{m+r-1})} \tag{6.26}$$

$$P_{m+r-2} = \frac{\lambda_{m+r-1}P_{m+r-1}}{s+\lambda_{m+r-2}} = \frac{\lambda_{m+r}\lambda_{m+r-1}}{(s+\lambda_{m+r})(s+\lambda_{m+r-1})(s+\lambda_{m+r-2})} \tag{6.27}$$

$$... \tag{6.28}$$

$$P_n = \frac{\lambda_{n+1}P_{n+1}}{s+\lambda_n} = \frac{\lambda_{m+r}...\lambda_{n+1}}{(s+\lambda_{m+r})(s+\lambda_{m+r-1})...(s+\lambda_n)} \tag{6.29}$$

$$P_{n-1} = \frac{\lambda_n P_n}{s+\lambda_{n-1}} = \frac{\lambda_{m+r}...\lambda_{n+1}\lambda_n}{(s+\lambda_{m+r})(s+\lambda_{m+r-1})...(s+\lambda_n)(s+\lambda_{n-1})} \tag{6.30}$$

The reliability of the system is given by the healthy states as follows:

$$R^*(s) = P(\mathbb{H}) = \Sigma_{i=n}^{m+r} P_i \tag{6.31}$$

$$\mathbf{MTTF}_{FT} = \lim_{s\to 0}(R^*(s)) = \lim_{s\to 0}\Sigma_{i=n}^{m+r} P_i \tag{6.32}$$

$$\mathbf{MTTF}_{FT} = \Sigma_{i=n}^{m+r}\frac{1}{\lambda_i} = \frac{1}{\lambda_{single-part}}\Sigma_{i=n}^{m+r}\frac{1}{i} \tag{6.33}$$

$$= \frac{1}{\lambda_{single-part}}(\Sigma_{i=n}^{m-1}\frac{1}{i} + \frac{1}{m} + \Sigma_{i=m+1}^{m+r}\frac{1}{i}) \tag{6.34}$$

Finally, RAF can be calculated, using Eq. 6.10 and Eq. 6.31, as follows:

$$\mathsf{RAF}_{FT} = \frac{\mathbf{MTTF}_{FT}}{\mathbf{MTTF}_{original}} = 1 + \frac{\Sigma_{i=n}^{m-1}\frac{1}{i} + \Sigma_{i=m+1}^{m+r}\frac{1}{i}}{\frac{1}{m}} \tag{6.35}$$

When simplifying Eq. 6.35, **Lemma 1** can be obtained. The enhancement of reliability is obtained thanks to the additional *extra parts (r)* and the reduction of the *minimal required part (n)*.

Strategy 2

This strategy is designed for *Model 2 - Fault-reduced model*. In this case, the efficiency can be predicted from other analyses or probability calculations. With a fault reduction value f_{FT} given by the technique, the new fault rate is obtained by Eq. 6.36

$$\lambda_{FT} = f_{FT}\lambda_{original} \tag{6.36}$$

Where f_{FT} is the inverse value of RAF:

$$f_{FT} = \frac{1}{RAF} = \frac{\lambda_{FT}}{\lambda_{original}} \tag{6.37}$$

This strategy is proposed to help designers integrate existing analyses into our method. For instance, the f_{FT} can be obtained from a physical level analysis, a simulation, another analytical model or even from the FAIT stage. By dividing the fault-rate (or MTTF) of design before and after applying the fault-tolerance methods, designers can obtain the reduction rate and integrate the fault-reduced module to the system.

Strategy 3

This strategy is applied to *Model 3 - Error handling model*. As previously mentioned, in prior strategies we demonstrate the efficiency of the fault-tolerance techniques using analytical models [143]. Because fault-tolerance requires additional modules for checking and correcting faults. These additional modules also add extra fault-rates.

We model both original and fault-tolerant systems to have two Markov states as represented in Fig. 6.7 (a) and Fig. 6.7 (b), respectively, where:

- S_0 is the initial state.
- S_F is the faulty state of the original system.
- S_1 is the faulty state of the original system which can be corrected by the fault-tolerant technique.
- S_2 is the faulty state of the original system which cannot be corrected by the fault-tolerant technique.
- S_{C-F} is the faulty state of the repair module.

Because of the protection from the fault-tolerant technique, the FT system can handle some faults. Therefore, we define the transition rates as:

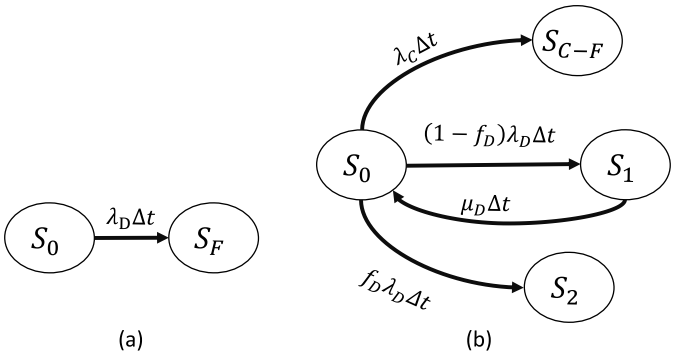

Figure 6.7: A simplified Markov-state reliability model for (a) the original system; (b) the fault-tolerant (FT) system.

- λ_D is the fault-rate of the original system (D).

- λ_C is the fault-rate of the repair module of the FT system.

- μ_D is the repair-rate which is provided by the repair module (C) on the original system (D).

- f_D is the fault reducing value by applying the fault-tolerance mechanism.

Based on the Markov state of the two systems, as shown in Fig. 6.7, the reliability function is givens as:

$$R^*(s) = P_{S0} \tag{6.38}$$

The final fault-rate of the fault-tolerance system is as follows:

$$\lambda_{FT} = f_D \lambda_D + \lambda_C \tag{6.39}$$

Lemma 2: The RAF value can be then expressed as:

$$\mathsf{RAF}_{FT} = f_D + \frac{\lambda_C}{\lambda_D} \tag{6.40}$$

Proof 2: We have the derivations of the states as the following:

$$p'_{S0} = -(\lambda_C + (1 - f_D + f_D)\lambda_D)p_{S0} + \mu_D p_{S1} \tag{6.41}$$

$$p'_{S1} = -\mu_D p_{S1} + (1 - f_D)\lambda_D p_{S0} \tag{6.42}$$

$$p'_{C-F} = (\lambda_C)p_{S0} \tag{6.43}$$

$$p'_{S2} = (f_D\lambda_D)p_{S0} \tag{6.44}$$

$$\tag{6.45}$$

And their Laplace transforms can be expressed as:

$$sP_{S0} - p_{S0}(0) = -(\lambda_C + \lambda_D)P_{S0} + \mu_D P_{S1} \tag{6.46}$$

$$sP_{S1} - p_{S1}(0) = -\mu_D P_{S1} + (1 - f_D)\lambda_D P_{S0} \tag{6.47}$$

$$sP_{C-F} - p_{C-F}(0) = (\lambda_C)P_{S0} \tag{6.48}$$

$$sP_{S2} - p_{S2}(0) = (f_D\lambda_D)P_{S0} \tag{6.49}$$

$$\tag{6.50}$$

$$sP_{S0} - 1 = -(\lambda_C + \lambda_D)P_{S0} + \mu_D P_{S1} \tag{6.51}$$

$$sP_{S1} - 0 = -\mu_D P_{S1} + (1 - f_D)\lambda_D P_{S0} \tag{6.52}$$

$$sP_{C-F} - 0 = (\lambda_C)P_{S0} \tag{6.53}$$

$$sP_{S2} - 0 = (f_D\lambda_D)P_{S0} \tag{6.54}$$

$$\tag{6.55}$$

$$sP_{S0} + (\lambda_C + \lambda_D)P_{S0} = 1 + \mu_D P_{S1} \tag{6.56}$$

$$sP_{S1} + \mu_D P_{S1} = (1 - f_D)\lambda_D P_{S0} \tag{6.57}$$

$$sP_{C-F} = (\lambda_C)P_{S0} \tag{6.58}$$

$$sP_{S2} = (f_D\lambda_D)P_{S0} \tag{6.59}$$

$$\tag{6.60}$$

$$P_{S0} = \frac{1 + \mu_D P_{S1}}{s + (\lambda_C + \lambda_D)} \tag{6.61}$$

$$P_{S1} = \frac{(1 - f_D)\lambda_D P_{S0}}{s + \mu_D} \tag{6.62}$$

$$P_{C-F} = \frac{\lambda_C}{s} P_{S0} \tag{6.63}$$

$$P_{S2} = \frac{f_D\lambda_D}{s} P_{S0} \tag{6.64}$$

$$\tag{6.65}$$

$$P_{S0} = \frac{1 + \mu_D \dfrac{(1 - f_D)\lambda_D P_{S0}}{s + \mu_D}}{s + (\lambda_C + \lambda_D)} \tag{6.66}$$

$$P_{S0}(1 - \frac{\mu_D(1 - f_D)\lambda_D}{(s + \lambda_C + \lambda_D)(s + \mu_D)}) = \frac{1}{s + \lambda_C + \lambda_D} \tag{6.67}$$

$$P_{S0}((s + \lambda_C + \lambda_D)(s + \mu_D) - \mu_D(1 - f_D)\lambda_D) = (s + \mu_D) \tag{6.68}$$

$$P_{S0} = \frac{s + \mu_D}{s^2 + (\lambda_C + \lambda_D + \mu_D)s + \mu_D(\lambda_C + f_D\lambda_D)} \tag{6.69}$$

$$P_{S0} = \frac{s+\mu_D}{s^2+(\lambda_C+\lambda_D+\mu_D)s+\mu_D(\lambda_C+f_D\lambda_D)} \tag{6.70}$$

$$P_{S1} = \frac{(1-f_D)\lambda_D}{s^2+(\lambda_C+\lambda_D+\mu_D)s+\mu_D(\lambda_C+f_D\lambda_D)} \tag{6.71}$$

$$P_{C-F} = \frac{\lambda_C(s+\mu_D)}{s^2+(s(\lambda_C+\lambda_D+\mu_D)s+\mu_D(\lambda_C+f_D\lambda_D))} \tag{6.72}$$

$$P_{S2} = \frac{f_D\lambda_D(s+\mu_D)}{s(s^2+(\lambda_C+\lambda_D+\mu_D)s+\mu_D(\lambda_C+f_D\lambda_D))} \tag{6.73}$$

The MTTF of the final system is given by the healthy state *S0*:

$$\mathbf{MTTF}_{FT} = \lim_{s\to 0}(R^*(s)) = \lim_{s\to 0} P_{S0} \tag{6.74}$$

$$\mathbf{MTTF}_{FT} = \frac{1}{(\lambda_C+f_D\lambda_D)} \tag{6.75}$$

$$\tag{6.76}$$

Because the MTTF value of the original system is $1/\lambda_D$, the RAF of this model is given as:

$$\mathsf{RAF}_{FT} = \frac{\mathbf{MTTF}_{FT}}{\mathbf{MTTF}_{original}} = \frac{\lambda_C+f_D\lambda_D}{\lambda_D} = f_D + \frac{\lambda_C}{\lambda_D} \tag{6.77}$$

When simplifying the above equation, **Lemma 2** can be obtained.

Discussion: The repair does not impact too much into the system's reliability. In fact, it gives a high impact on the availability of the system. The availability function is given as follows:

$$\begin{aligned}\mathbf{A}_{FT}(s) = \frac{E_{up-time}}{E_{up-time}+E_{down-time}} &= \frac{P_{S0}}{P_{S0}+P_{S1}} \\ &= \frac{s+\mu_D}{s+\mu_D+(1-f_D)\lambda_D} \\ &= 1-\frac{(1-f_D)\lambda_D}{s+\mu_D+(1-f_D)\lambda_D} \\ \mathbf{A}_{FT} = \lim_{s\to 0}\mathbf{A}_{FT}(s) &= \frac{\mu_D}{\mu_D+(1-f_D)\lambda_D}\end{aligned} \tag{6.78}$$

6.3.2 Merging

After analyzing all modules, the system reliability is calculated based on its sub-modules. Therefore, we start first by merging the router components and then merge all the network components to obtain the entire system reliability.

Router merging

We first determine the router is reliable only if it is able to transmit correctly a given data from any input to any output port. By applying equation 6.8, the fault rate of a router is obtained as follows:

$$\lambda^*_{router} = \sum_{i=1}^{N} f_{M_i} \lambda_{M_i} \tag{6.79}$$

Where λ_{M_i} is the fault-rate of module M_i and f_{M_i} is the fault reduction rate given by attaching this module to the system. By applying Eq. 6.8 with *unit*, which is defined as a baseline router, the new failure-rate of a router is given as follows:

$$\lambda^*_{router} = \sum_{i=1}^{N} f_{M_i} \pi_{M_i} \lambda_{router} \tag{6.80}$$

The RAF value also can be obtained by the following equation:

$$RAF_{router} = \frac{\lambda^*_{router}}{\lambda_{router}} = \sum_{i=1}^{N} f_{M_i} \pi_{M_i} \tag{6.81}$$

Network merging

The next step is to evaluate the network reliability. Unlike the router, the network has a high flexibility in the routing process. For example, if a link along the path between two distant routers is faulty, the network can avoid it in the routing path.

Among several existing network's reliability terminologies [134], this work uses "all-terminal reliability" for the analysis. "All-terminal reliability" is defined as all PEs are connected to the network. In other words, all PEs have the ability to communicate with any PE in the network.

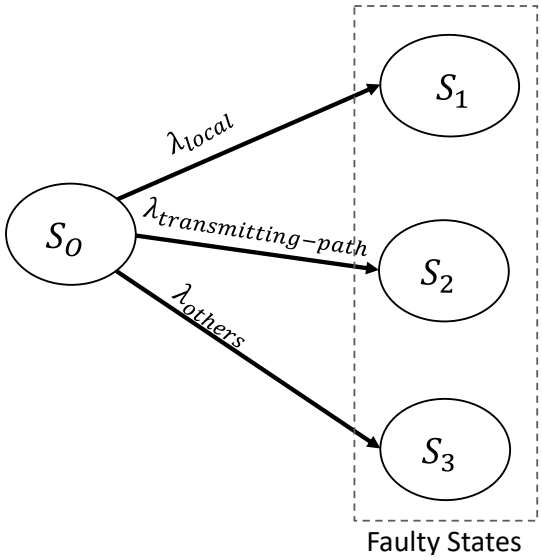

Figure 6.8: A Markov state of a mesh-based network.

According to [134, 137], the most common method is using *Node-based reliability*. Unlike computer or transportation network, NoCs have a small granularity in terms of fault occurrence and handling (FIFO, wires, logic circuit). Therefore, a low-level approach is more suitable. In this assessment, we analyze the network reliability in terms of connection between two routers, or a pair of router and PE.

To analyze the network's reliability, we analyze the possible failure cases that may corrupt the system. We define the major failure cases as follows:

- Failure on local connection: a failure on the connections, which are handled by NIs, between routers and PEs can corrupt the network's reliability. This involves two channels (input and output) and an input buffer which is constantly attached to its input channel.

- Failure on transmitting path: a failure on the transmitting path can corrupt the network's reliability. The transmitting path is considered as a set of connections between routers.

- Failed on other router modules: A failure in non-transmitting parts (e.g., switch allocator, management modules) of a router may malfunction the router.

From the failure cases, a Markov state model is built as shown in Fig. 6.8. As a result, the fault

rate of a network of N_R routers is given as follows:

$$\lambda_{network} = \lambda_{local} + \lambda_{transmitting-path} + \lambda_{others} \tag{6.82}$$

Where:

- $\lambda_{local} = N_R \times (2\lambda_{1-channel} + \lambda_{input-buffer})$ is the fault-rate of all local connections. N_R is the number of routers in the network.
- $\lambda_{transmitting-path}$ is the fault-rate of a transmitting paths between routers inside the network. Designers can estimate this value based on analytical analysis or simulation model proposed in [134][137][43][173]. In here, we apply *k-failure* [173] model to assess the reliability.
- λ_{others} is given by the fault-rates of other parts (non-routing parts) of routers.

In this work, we consider $\lambda_{transmitting-path} = \lambda_{RTR} \times N_R$. λ_{RTR} is the fault-rate of a router-to-router (RTR) connection which represents one node connection from a router to any adjacent router. The RTR connection failure rate depends on the position of the router in the network. Here, we use the *k-failure* [173] model: a router is disconnected at the presence of k failures. We adopted this model with a modification: the failure value k is defined as a connection and it depends on the router's position. For example, the corners, the edges, the side and the middles of the 3D Mesh NoCs have three, four, five and six routing selections, respectively. This condition is the maximum value that fault-tolerant routing algorithms can achieve. With the fault assumption in Eq. 6.8, the routers located at a similar position (corner, edge, side or middle) have a similar fault rate. The failure rate of RTR connection is expressed as follows:

$$\lambda_{RTR} = P_{corner} \times \lambda_{corner} + P_{edge} \times \lambda_{edge} + P_{side} \times \lambda_{side} + P_{middle} \times \lambda_{middle} \tag{6.83}$$

Where the failure rates: λ_{corner}, λ_{edge}, λ_{side}, and λ_{middle} are obtained from the fault-rate of a connection. P_{corner}, P_{edge}, P_{side} and P_{middle} are the probability of having a router in corner, edge, side and middle of the network, respectively. A connection is defined as a data path from input buffer to the next input buffer or NI's buffer. As shown in Figure 2.12, a connection consists of an input-buffer, a crossbar link, an inter-router channel. Because these components are independent,

the fault-rate of a connection ($\lambda_{conn.}$) is defined as follows:

$$\lambda_{conn.} = \lambda_{1-input-buffer} + \lambda_{1-crossbar-link} + \lambda_{1-router-channel} \tag{6.84}$$

In order to compute λ_{RTR}, the fault rate of each position in Eq. 6.83 can be calculated by using Eq. 6.11 of Strategy 1 as follows:

$$\lambda_{position} = \frac{1}{\mathbf{MTTF}_{position}} = \frac{1}{\sum_{i=n}^{m+r} \frac{1}{\lambda_{conn.}}} \tag{6.85}$$

Where the identical part is a connection (its fault-rate is $\lambda_{conn.}$), r=0, n=1, and m= 3, 4, 5 and 6 for the *corner*, *edge*, *side*, and *middle*, respectively. The non-fault tolerant routing fault-rate (from MTTF) can be calculated using Eq. 6.10.

In the case where the routers in a network have different fault-rates, and despite having the same architecture, we need to manually calculate each router's $\lambda_{RTR}(i = 1, 2, ..., N_R)$. The fault-rate of a transmitting path is obtained by the following equation:

$$\lambda_{transmitting-path} = \sum_{i=1}^{N_R} \lambda_{RTR}(i) \tag{6.86}$$

Where $\lambda_{RTR}(i)$ is the fault-rate of router-to-router connection from router i of the network.

6.4 MTTF Monte-Carlo Simulation

Figure 6.9 shows the Mean Time To Failure (MTTF) Monte-Carlo setting up flow [153]. The goal of this simulation is to measure the MTTF value of the system. The flow of this simulation consists of the following:

- The NoC architecture is designed in Verilog HDL and synthesized using Synopsys Design Compiler to obtain a post-synthesis netlist model.
- A fault distribution system is automatically integrated inside the NoC netlist model by using Python's Regular Expression scripts [174]. The faults are modeled in stuck at "0" and "1". Our method is similar to the fault injection methods in [102, 133].
- The post error injection netlist model is used to simulate and find the number of faults

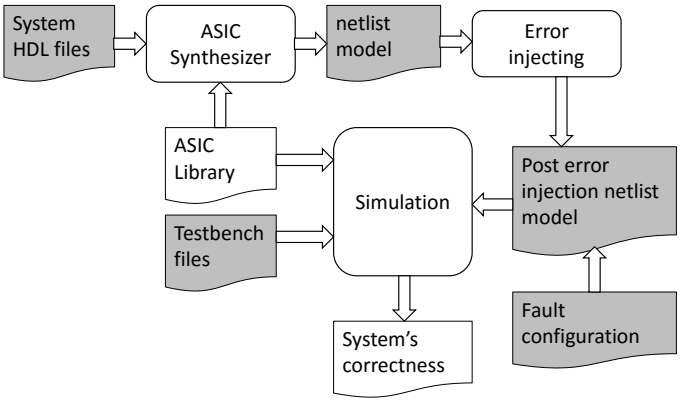

Figure 6.9: Monte-Carlo setting up flow.

leading to failure.

- The number of faults is recorded for further processing.

The error injector architectures are shown in Figure 6.10 with two models: normal gate and flip-flop gate. When the error injector is enabled (by setting up $C = 1$), it forces the output of its attached gate to "0" or "1". If the error injector is disabled, the correct output is forwarded.

The simulation process is depicted in Figure 6.11:

- At the first iteration, the injected position is generated and the corresponding position is distributed with a fault.
- Hard faults are injected until finishing the experiment. Soft errors disappear after one clock cycle.
- A testbench is designed to verify the system correctness after injecting a fault (100% of the PEs are connected and 100% of the packets are correctly delivered). If the system is still functioning correctly after a fault is injected, the simulation injects more faults. Otherwise, the number of faults and working time are recorded.
- The time-to-failure is calculated based on the number of faults or the working time. The final MTTF is the average value of all iterations.

For iteration i, the time-to-failure is measured as the number of faults injected as:

$$TTF_i = f_i \times \frac{1}{\lambda_{system}} \tag{6.87}$$

Table 6.1: Router's Weight and Gate Ratio.

Module	Submodule	Weight	Gate Ratio
Network	Network	100%	100%
	Routers	70%	100%
	Channels	30%	0%
Router	Router	100%	100%
	Input Buffer	69.72%	7.90%
	Crossbar	8.00%	11.43%
	Switch-Allocator	7.00%	16.97%
	Others	15.28%	63.7%

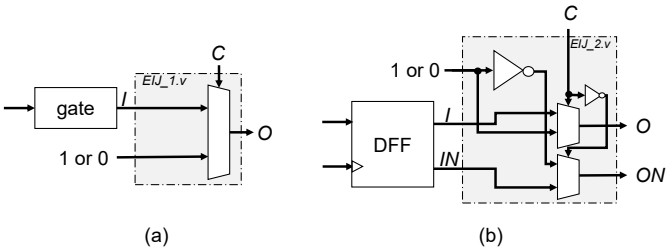

Figure 6.10: Error Injector architecture (a) Single output gate, (b) Flip-flop with two outputs.

Where λ_{system} is the raw fault-rate of the original system. After finishing a simulation of N iterations, the MTTF value of a system is given by Eq. 6.88 where f_i is the number of faults causing failure in experiment i.

$$MTTF_{system} = \frac{\sum_{i=1}^{N} TTF_i}{N} = \frac{\sum_{i=1}^{N} f_i}{N} \times \frac{1}{\lambda_{system}} \tag{6.88}$$

In order to verify the analytical model, we use two configurations: (1) *Gate Ratio* where the fault-rate is linear to the number of utilized gates in the netlist file, and (2)*Weight* where the fault-rate is higher in the data transmission modules which are likely to have a significant impact on the system correctness. The fault-rate ratio can be seen in Table 6.1 where the errors are focused on input buffers, crossbar and links.

6.5 Evaluation Results

6.5.1 Evaluation Methodology

In this section, we evaluate the reliability of NoC systems with two methodologies: the proposed analytical model and a system-level simulation. To demonstrate the efficiency of the proposed analytical analysis, we used our proposed 3D NoC (3D-FETO) as a case of study previously presented

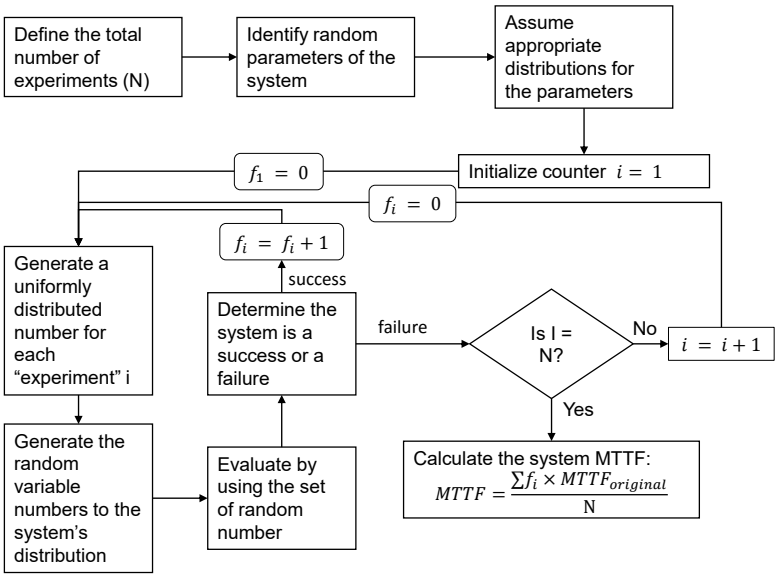

Figure 6.11: MTTF Monte-Carlo simulation process.

Table 6.2: Simulation configurations.

Parameters	Value
Flits Size	44 bits
Header Size	14 bits
Buffer Depth	4
Switching	Wormhole-like
Flow-control	Stop-Go
Routing	Look-ahead Fault-Tolerant

in 4. We select a baseline NoC model (OASIS) [149] in these evaluations. To compare between the two methods, we use a similar error-rate then calculate the MTTF and RAF values. The configurations are shown in Table 6.2. The final system-level simulation results are compared with the analytical model results to study the accuracy of the proposed model.

6.5.2 Accuracy Evaluation

Figure 6.12 and 6.13 shows the comparison of RAF values between the proposed analytical model and the simulation results. For a single router assessment, the analytical method predicts the RAF values with acceptable deviations (under 33%). Moreover, there is a significant amount of hidden faults in the baseline model. In this kind of situations, faults are injected; but, they do not give a detectable impact on the system. Especially, soft errors, which are injected in a single clock cycle, are likely to become hidden faults. In contrast, hard faults on the router give a higher

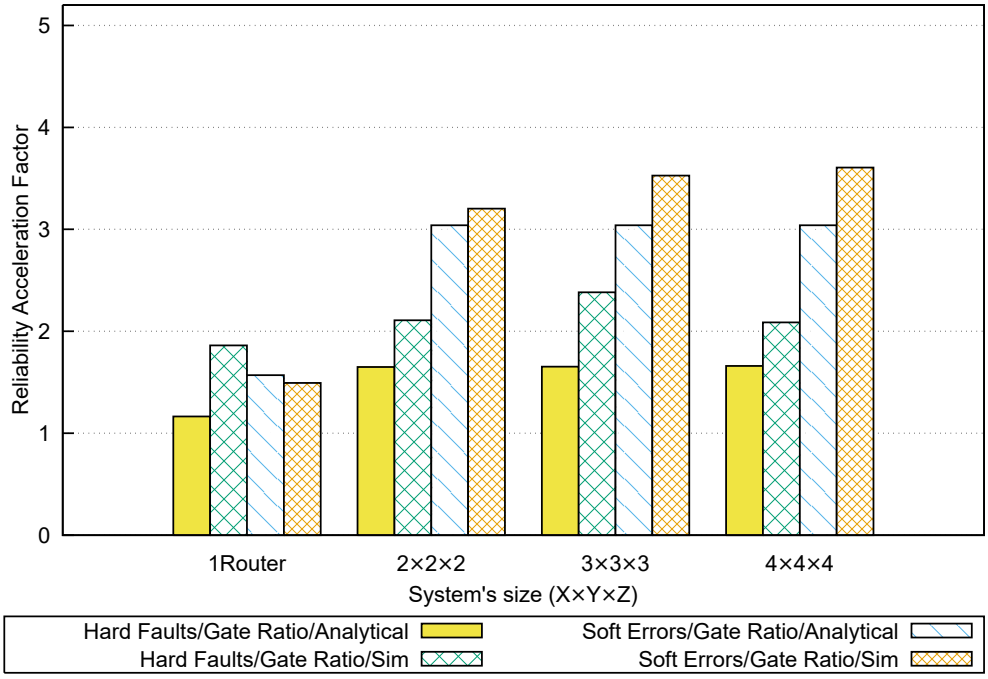

Figure 6.12: Comparison results with gate ratio distributions.

impact on the system.

For networks' assessments, we also simulated and compared the RAF values for four different network sizes: $2 \times 2 \times 2$, $3 \times 3 \times 3$ and $4 \times 4 \times 4$. The worst cases can be observed in the soft-error simulation with weight distribution and hard-fault simulation with gate ratio distribution. However, the accuracy is still better than a single router where most of the deviations are less than 23%. The worst case is hard fault tolerance with the gate ratio distribution of a $3 \times 3 \times 3$ network where the difference can reach up to 31.64%.

The deviation in the accuracy values are mostly caused by the occurrence of hidden faults. They are defined as the faults that can be injected without causing the system crash. Such hidden faults cause the observed difference between the analytical model and the Monte-Carlo simulation. For instance, a fault on the routing unit or output port may lead to a misrouted packet. Nevertheless, at the next router the packet can be routed correctly without causing dead-lock or live-lock in the network. Another example is the presence of hidden faults in the intra-router routing where the employed routing algorithm chooses either the X, Y, or Z direction depending on the used routing algorithm. At the presence of hidden faults, the order of dimensions might be altered. That is, instead of routing a packet through X, Y then Z, it is sent through Y, X then Z. However, and

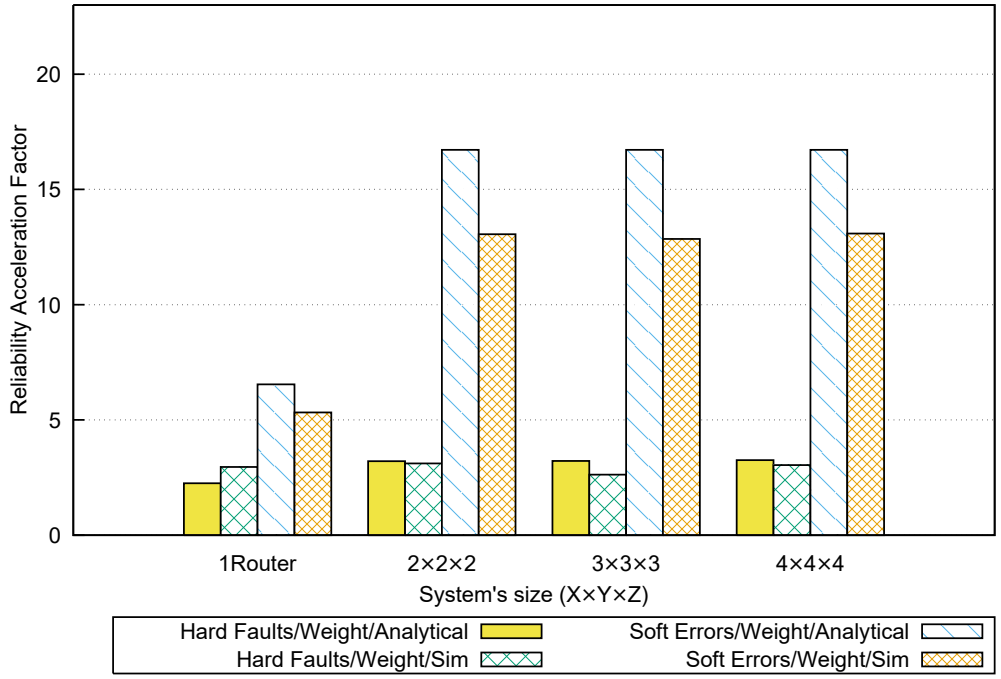

Figure 6.13: Comparison results with weight distributions.

despite this change, the packet can still reach its destination correctly without the system crashing. Furthermore, and as shown in Fig. 6.13, the difference in soft errors is slightly higher than hard faults. This is because soft errors are injected within one clock cycle which may not affect the operation of the system if this latter is idling. Because this type of faults did not crash the system, they make the non-fault-tolerant system become more resilient and increase the MTTF values, while our model cannot estimate it. Since we adopted the weight distribution, where faults are injected more on the fault-tolerant modules, the assessment results are closer than those of the gate ratio.

In summary, the overall accuracy is acceptable with most cases having less than 23% of deviations. There are some cases where the difference is considerable; however, the deviation is logical due to the low granularity of the reliability assessment. In fact, when designers apply the assessment method before obtaining the design characteristics (e.g., gate ratios), the deviations are still reasonable.

6.5.3 Reliability Assessment Speedup

The proposed analytical method offers faster estimation time in comparison to other conventional methods. Table 6.3 shows the reliability assessment simulation time and the speedup ob-

Table 6.3: Reliability Assessment Speedup.

Evaluated module	A MTTF simulation	Proposed method	Speedup
A router	11 hours	0.090 second	440,000
A $2 \times 2 \times 2$ network	20 hours	0.091 second	791,209
A $3 \times 3 \times 3$ network	2 days	0.092 second	1,878,261
A $4 \times 4 \times 4$ network	3.5 days	0.109 second	2,774,312

tained with the proposed analytical model when compared to the conventional MTTF simulation, previously explained in Section 6.4. The proposed method's calculation is performed using GNU Octave, and the MTTF simulation performs netlist simulations for 1000 cases using Cadence NC-Sim CAD tool. Both simulations were conducted on a Linux CentOS 6.4 machine using Intel Xeon E5-2620 (8 cores, 2.10Ghz) and 64 GB of RAM.

Thanks to the low complexity of the proposed method, the simulation time is always in the hundreds of milliseconds range for all cases. On the other hand, the MTTF simulation requires more than 11 hours of computation for just a single router. This results in a speedup of 440,000 times with our proposed method. The long execution time of the MTTF simulation is caused by the main following factors: (1) the high complexity of the netlist files where a router's netlist file consists of over 15,000 separated gates; (2) the high complexity of the fault injection (i.e., each gate requires an error injection module and a fault distribution system); (3) the verification complexity which usually tries to cover all possible operational situations (e.g., a seven-ports router has 49 (7^2) cases of communication); and (4) it requires four different simulations for four different types of fault: soft error, hard fault, stuck-at-0 and stuck-at-1. When we increase the network size, the MTTF simulation time has significantly increased. On the other hand, the assessment time of the proposed method does not scale up with the network size. This is given by the assumption that the routers in the same position (corner, edge, side or middle) inside the network have a similar fault-rate. Therefore, the calculation can be reduced into Eq. 6.83 and 6.82. In fact, the obtained speedup with the proposed method is 791209, 1878261, and 2774312 for $2 \times 2 \times 2$, $3 \times 3 \times 3$, and $4 \times 4 \times 4$ network sizes, respectively. The speedup values are expected to be much higher with larger network sizes.

In summary, the proposed analytical method provides an extremely fast solution to estimate the reliability of NoC systems. Although the proposed method is not as accurate as the MTTF sim-

ulation, its tremendous speedup values are very compelling for early system reliability assessment. In fact, to perform the MTTF simulation, we need to obtain a complete design and verification test which may take several months of development. If the design cannot pass the reliability requirements, the waste in re-design time and resources can be extremely critical.

6.6 Conclusion

In this chapter, we proposed a reliability assessment for fault-tolerant NoCs. The proposed method is based on three basic steps. First, the system is divided into sub-systems. Second, a state model is built upon each sub-system and the reliability value can be obtained. Last, the final reliability of the system can be generated from its sub-systems. In addition, we present an extended method for network's reliability that also helps designers.

Through extensive evaluations, we showed that the proposed method was acceptably matched with the simulation method while it reduces a large amount of modeling and simulation time and effort. This means that our method can provide a faster solution for fault-tolerant systems. Before designing, researchers can apply the proposed method to estimate the enhancement in terms of reliability which can help them understand the efficiency of the system.

In our assessment, we also point out the benefits of the fault-tolerant system in terms of reliability improvement. In addition, the impact on the system performance is also analyzed. Based on the pros and cons of the fault-tolerant system, designers can select the appropriate mechanisms and configurations to match their requirements.

7

TSV-based 3D-NoC System Design

The previous chapters have proposed and evaluated complete fault tolerant architectures and algorithms for 3D-NoCs. The reliability analytical models of these systems are also well addressed. To explore the capacity of 3D-IC technology, this chapter aims to advance to the system-level architecture where application-specified systems are presented. In the first part, the dissertation approach for design for reliability is depicted. In the second section, the working flow of TSV-based design is presented. After that, the design with TSV and the related issues are discussed. This chapter concludes with the implementation results.

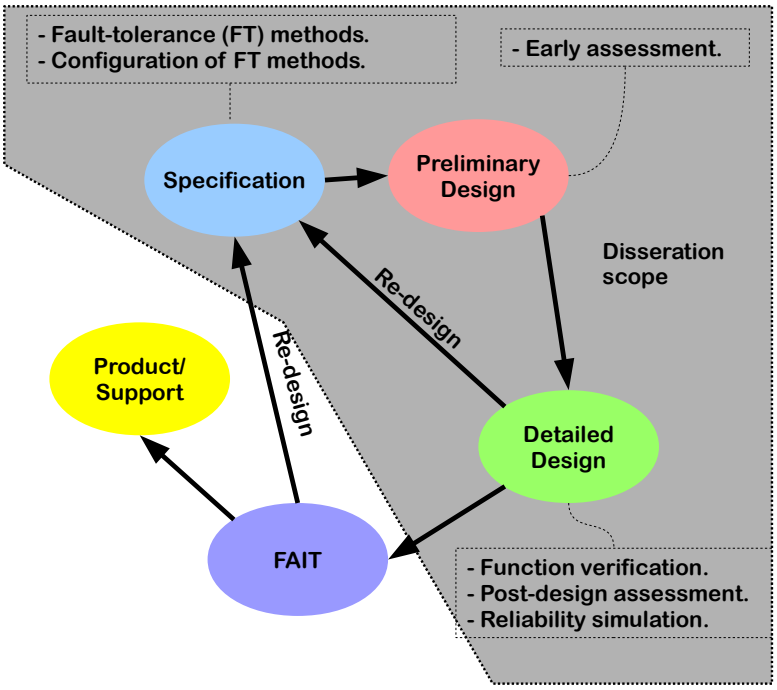

Figure 7.1: Design for Reliability: Dissertation scope.

7.1 Design for Reliability and Dissertation Approach

7.1.1 Design for Reliability

The different proposals presented in the previous chapters are firstly summarized hereafter. Figure 7.1 shows the design for reliability method with the gray polygon representing the scope of this dissertation. This work covers the specification stage where the fault-tolerant architectures and algorithms (e.g. PCR, Sharing TSV, Virtual TSV, ...) are proposed. The configurations of the fault-tolerance is also briefly discussed. In order to help designers early assess their NoC designs, the preliminary design stage is handled by the reliability analytical model in Chapter 6. At the end of this stage, designers have an early assessed reliability improvement obtained with the adopted fault-tolerant techniques. In the detailed design, a conventional working flow is adopted with some adjustments. After passing the function verification and getting the netlist design, the design is put under MTTF Monte-Carlo simulation which again help designers to obtain finer reliability results. Depending on the level of satisfaction with the results, it can be forwarded to *FAIT*, or a re-design is required.

7.1.2 Dissertation Approach

Figure 7.2 shows how this dissertation is involved in the design flow. In addition to the conventional design steps represented in Fig. 7.2 (a), this dissertation scope consists of three early design stages for reliability. These stages are *Analytical Analysis*, *Monte-Carlo MTTF simulation* and *TSV Replacement Calculation*, which are presented in pink boxes in Fig. 7.2 (b). Besides the *Specification*, the *Analytical Analysis* is employed to help designers early estimate the improvement in terms of reliability. Other parameters, such as possible area overhead or power consumption, can be also estimated in this step. After the *Specification* stage finds the suitable solutions and configurations, the *RTL-level Design* stage is where the architecture is created. Later, a functional simulation/verification is used to ensure the correctness of the design and its performance. The RTL architecture undergoes the *Synthesis* step. At the synthesis output, netlist designs have to be verified with two steps: function and reliability. To help estimate the reliability of the netlist design, the *Monte-Carlo MTTF Simulation* is added. If the design cannot pass these two steps, it needs to be adjusted in either specification or RTL-level design. The passed design is forwarded to the layout steps. In these steps, with the involvement of TSVs, their positions are calculated. Later, the flow is completed with *Place and Route* (P&R), *Post P&R Simulation* and *IO pad Insertion*.

In this dissertation, 3D-NoCs are the main target for fault-resilience. Therefore, the fault-tolerance for the cores/PEs and the whole system is out-of-scope. However, if designers aim to perform *Design for Reliability*, this proposed working flow is definitely suitable. Of course, designers have to provide the fault-tolerant techniques and the analytical model for their own circuits. This dissertation is carefully designed for being compatible with any cores/PEs type. There are no issues when adding cores or PEs for completing the architecture.

Table 7.1: Estimated development time of the fault-tolerant 3D-NoC executed by a single developer.

Fault-Tolerant Method	Design Time	Simulation Time
Pipeline Computation Redundancy	2 months	1 hour (RTL, 8 benchmarks)
Hard Fault Tolerance (LAFT, RAB, BLoD)	3 months	1 hour (RTL, 8 benchmarks)
Detection, Diagnosis, and Recovery Mechanism	1 month	1 hour (RTL, 8 benchmarks)
MTTF Monte-Carlo Simulation	2 months	3.5 days ($4 \times 4 \times 4$, Uniform)
Reliability assessment using analytical model	3 months	less than 1 second
Cluster-TSV fault-tolerance	3 months	1 hour (RTL, 8 benchmarks)

As shown in Fig. 7.1, the risk of re-design can role an important part in deciding the fault-tolerant techniques and configurations. In this part, we extend to the development breakdown

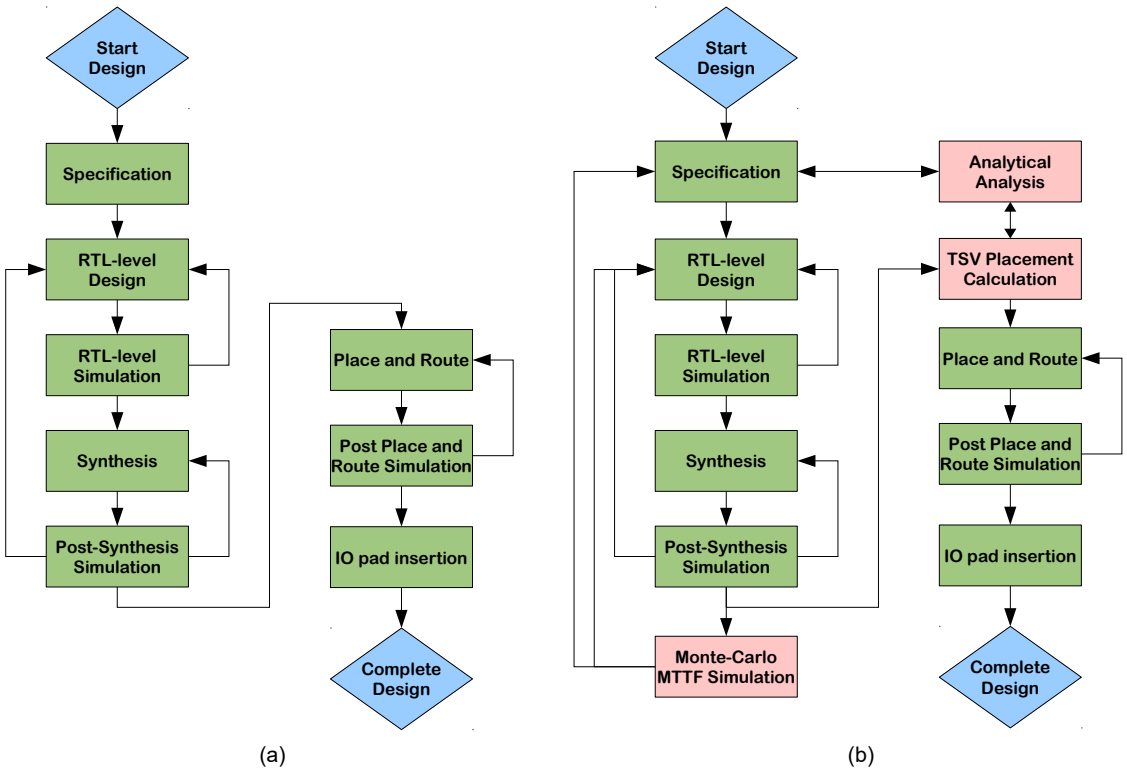

Figure 7.2: Design flow for fault-resilient 3D-ICs: (a) Traditional flow; (b) Dissertation approach.

of the proposed architectures and algorithm and our experiments. Table 7.1 shows the estimated development and simulation time of the fault-tolerant 3D-NoC. Design time consists of RTL design, verification and back-end design. In summary, the total time that one designer may need is around 11 months to complete the design in RTL and perform the MTTF simulation. This can be considered as a significant amount of time to be risked. In contrast, reliability assessment required approximately 3 months. Nevertheless, it can reduce the risk of re-design. Moreover, it can be reutilized for other circuits and future projects. On the other hand, developing MTTF Monte-Carlo simulation is also a promising solution where the finest level of reliability evaluation can be assessed. Because the fabrication and testing steps are time consuming, and demand a significant budget, MTTF simulation can alleviate the risk of unmet requirement situations.

7.1.3 Dissertation Goals and Discussions

In the introduction section, the goals of this dissertation have been declared. Based on them, this dissertation has proposed multiple architectures and algorithms and also created a working flow for *Design for Reliability*. Table 7.2 summarizes the approaches on this dissertation and the

predefined goals.

Table 7.2: Dissertation goals and the proposed methodologies.

Methodology	Step	Reliability	Modularity	Scalability	Adaptivity
Analytical assessment	Preliminary	-	Yes	Yes	NoC-based
Soft Error Tolerance	Spec./RTL	Yes	Yes	Yes	Yes
Soft-Hard Fault Tolerance	Spec./RTL	Yes	Yes	Yes	Yes
Cluster-TSV Fault Tolerance	Spec./RTL/P&R	Yes	Yes	Yes	Cluster-based
Monte-Carlo MTTF simulation	Synthesis	-	No	No	any design

For the preliminary design, reliability assessment does not help enhance the fault resilience; however, it is proposed to help designers understand the reliability and how to approach it in a preventive and proactive fashion. At the later steps, the Monte-Carlo MTTF simulation also roles the same part which can be adapted to any design.

For fault-tolerance, the soft error resiliency has significantly enhanced the reliability and can be easily adapted to any system. Because the proposed method (PCR) works on the pipeline stage, it has a high level of modularity. By combining the soft error and TSV-defect resiliency with the existing hard fault tolerant methods, this dissertation presents a complete reliable architecture that achieves our objectives. Because the cluster-TSV has been proposed with the cluster distribution, its adaptivity is limited. However, as mentioned in Chapter 5, using ECC codes or adopting a TSV redundancy method can help designers handle the random type of TSV defects.

7.2 Design with Through-Silicon-Via

In this section, the design of TSV is briefly summarized and presented[1]. As one of the dissertation approach (see Fig. 7.2), the TSV placement flow is also explained. Here, the layout of a TSV is designed as an OBS macro [176] and prevents the layout CAD tool from putting any standard cells on it as the Keep-out-Zone (KoZ). The layout of a TSV from an abstract level (LEF file) is shown in Fig 7.3 [175, 176].

Figure 7.4 shows the working flow of designing the TSV-based system. The TSV macro (LEF file) was previously designed and checked in our laboratory [176]. Here, the macro file is used to integrate the YSV macro in the final layout. The design flow starts with a complete RTL design. Before undergoing the synthesis stage, the design is wrapped by a *TSV.v* - a dummy Verilog model of TSV. For each vertical wire, a *TSV.v* is inserted between the design and I/O ports. A wrapper is

[1]Readers who are interested in more details and completed tutorials, please visit our laboratory website at: http://adaptive.u-aizu.ac.jp/?page_id=592.

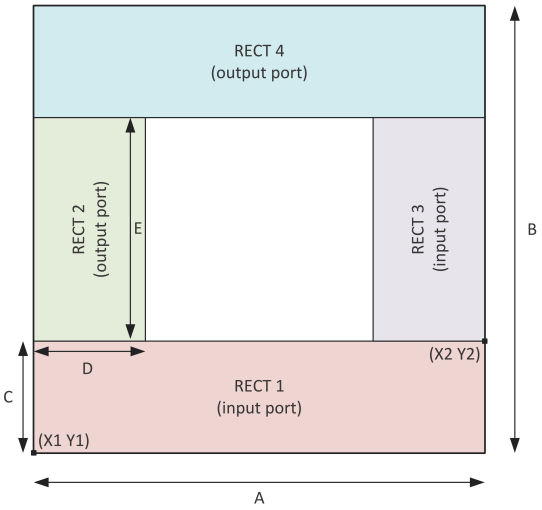

Figure 7.3: Layout of a TSV from a LEF macro definition [175]. Values: $A = 4.06$, $B = 4.06$, $C = 1$, $D = 1$ and $E = 2.06$ [176].

used to wrap all TSVs and the original design together. Note that designers can put TSV.v inside their design; but, it can maybe removed if the complete flattening option [2] is used.

```
1 module TSV(input i, output o);
2   assign o = i;
3 endmodule
```

Listing 7.1: *TSV.v.*

After obtaining the wrapped module, it is put through the synthesis flow as a normal design. The netlist file from this step is brought to the *modify the netlist file* step. In this step, the netlist file consists of multiple TSV modules. The objective of this step is to keep only one TSV module which is placed and routed as a TSV macro. At the end of this step, all TSV modules are unified and the new netlist file is forwarded to the Place and Route (P&R) step. In the P&R, the TSV macro is used for each TSV module, and standard cells are put for the normal gates. The positions of TSVs are also calculated in advance as shown in Fig. 7.5 where a port has 44 data TSVs (4 clusters, each cluster: 1,2,..11) and one control TSV (C). Finally, a layout with TSVs is obtained. Example layouts are shown in Fig. 7.6. The design is described in details in Chapter 4. The parameters of the TSV design is shown in Table 7.3 [177]. Depending on the design requirements, the positions of TSVs have to be calculated carefully.

[2] If full flattening is used, the compiler un-groups all sub-modules which omits the TSV.v module (see Listing 7.1). As a result, the *modify the netlist file* step fails to find TSV instantiations.

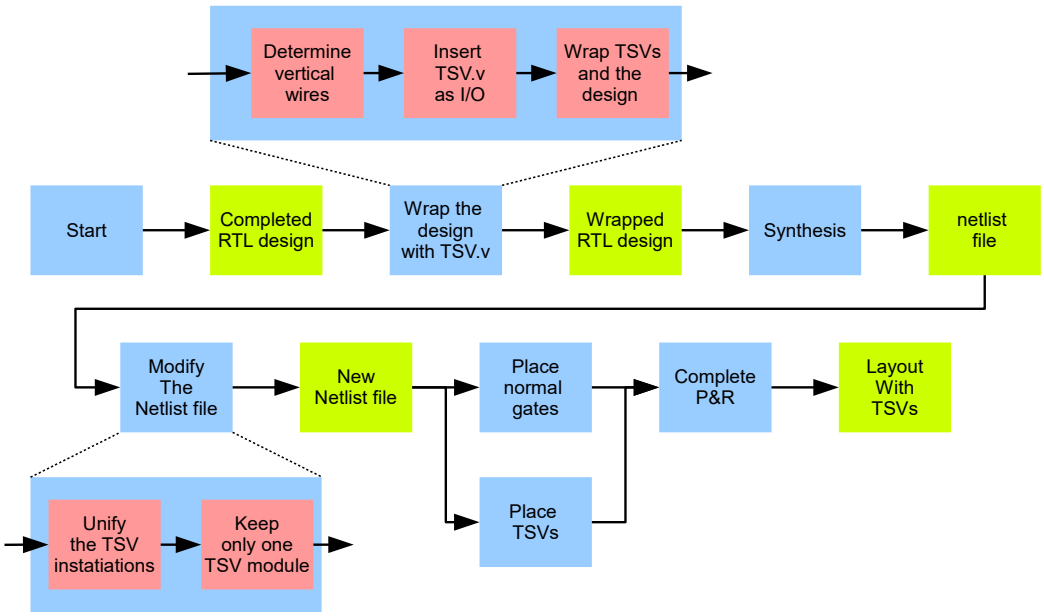

Figure 7.4: Design flow for TSVs. Green boxes are the results from their processes. Blue and pink boxes are the processes and sub-processes, respectively.

Table 7.3: Technology parameters.

Parameter	Value
Technology	Nangate 45 nm [156] FreePDK3D45 [155]
Voltage	1.1 V
TSV's size	$4.06\mu m \times 4.06\mu m$
TSV pitch	10 μm
Keep-out Zone	15 μm

7.3 Design of 3D-NoC systems

7.3.1 Specification

In the Section 7.2, the design flow with TSVs has been presented. This section discusses in details the design of 3D-NoC systems. In fact, this section is based on the design with TSV; however, there are several extra steps that need to be carefully considered.

Figure 7.7 depicts the considerations of developing NoC systems. From the target application, there are several requirements such as: reliability, performance, parallelism, area, power consumption and thermal issue. For each of these requirements, there is a set of development requirement. Here, the requirements of reliability are depicted with: (1) type of issue (soft errors, hard faults, TSV defects), (2) detection solution, (3) recovery solution and (4) the target MTTF or the quan-

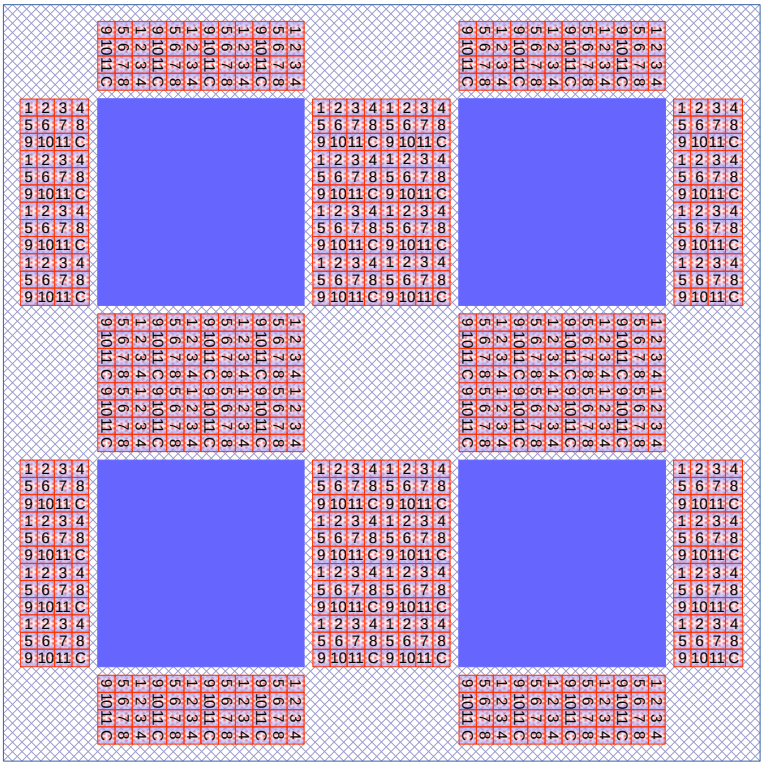

Figure 7.5: Sketch of 2×2 layout.

titative value.

To satisfy the requirements of the target application, designers can select a proper topology (mesh/torus, 2D/3D), a routing solution (deterministic/adaptive) and the protocol inside the network. If the reliability issue is the main target, fault-tolerant mechanisms can be selected to be used.

Besides of selecting the fault-tolerant mechanism, the configurations of the selected methods are also important. For instance, selecting the number of back-up channels in the *Bypass-Link-on-Demand* can decide how reliable the crossbar connections are. Moreover, it also impacts the area cost, power consumption and performance.

Because TSVs role an important part in the system, their configurations (sizes, pitches, KoZ) also need to be considered. The size and pitch of TSV impact the electronic characteristics (resistance, capacitance) which affect the its latency. The KoZ area is needed to avoid the TSV movement effects nearby devices; but, it also significantly increases the area cost.

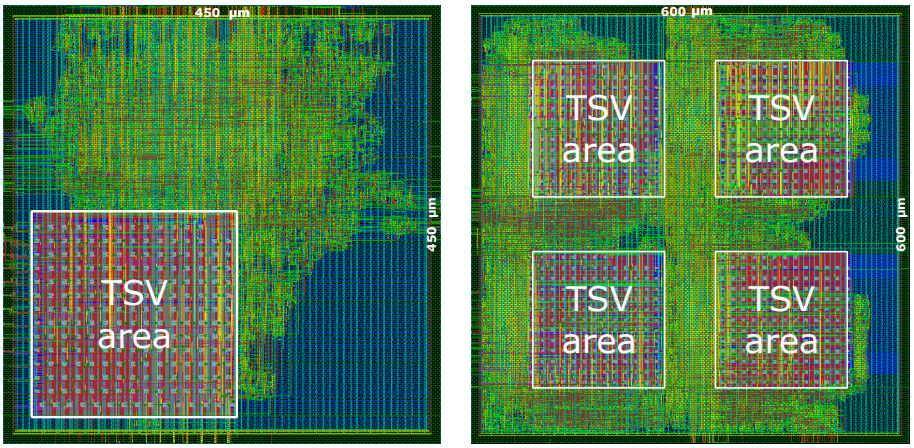

Figure 7.6: Example layout of models with TSV: (a) a single SHER-3DR router for the 3D-FETO system (see Chapter 4), #TSV: 208; (b) a layer of 2×2 3D routers, #TSV: 656.

7.3.2 Preliminary Design

In the preliminary design stage, designers need to assess the characteristic of the design. In summary, there are several aspects which are described as follows:

- **Area cost**: the possible total area cost can be estimated from the complexity of the architecture and algorithm. If this step implements a high-level model of the system, a high-level synthesis tool [178] can help estimate the design area cost. Otherwise, using profiling tools is another option to help designers estimate the complexity of the design. If the architecture is the combination of previously designed models, the final results can be obtained by accumulating them.

- **Power consumption**: this is similar to area cost estimation. For NoC-based systems, ORION [179] is a well supported tool for estimating the power consumption. If high-level synthesis tool is used, a report on power consumption can be easily obtained.

- **Performance**: The performance estimation can be precisely obtained by an accurate cycle simulator (e.g. GARNET [180] or McPAT [181]). In such estimation, designers need to investigate the impact of task mapping on the performance of the system. The possible degradation of fault-tolerant methods also need to be investigated. If the application is unspecified, using a task graph is a proper solution [163] where it provides in details the latency, bandwidth and execution order of the tasks.

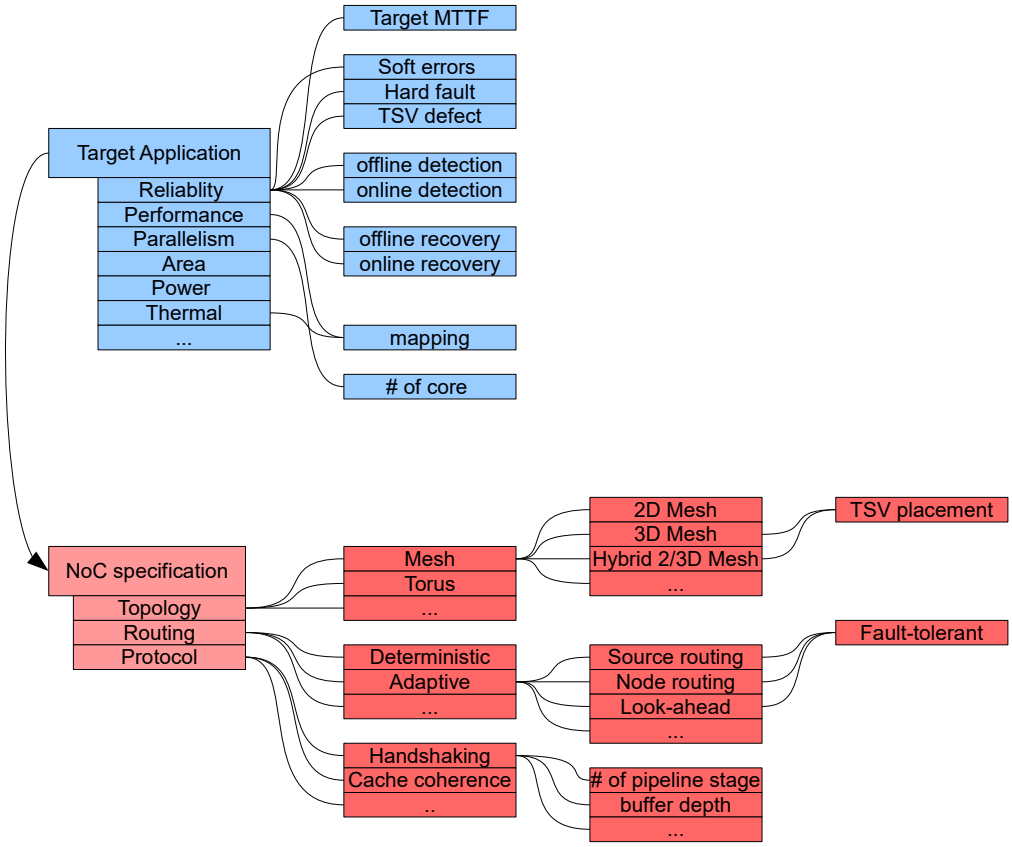

Figure 7.7: Network-on-Chip system specification.

- **Thermal hot-spot/Stress issue**: To reduce the impact of thermal or stress issues, designers also need to consider them as important criteria. In fact, if there are regions with high temperatures, they need to be addressed properly. There are also there approaches: (1) Integration: proposing a low power device and inserting thermal removal infrastructures; (2) Hardware: monitoring the temperature to reduce the operation of the hotspot area; (3) Software: using a thermal-aware program to alleviate the issue.

- **Yield & Test**: the yield rate is heavily depending on the manufacturing processes which designers has to firstly consider. In TSV-based 3D-ICs, the failure rate is accumulated by stacking layers together. Therefore, testing each layer is needed to improve the yield rate [182, 183]. Figure 7.8 depicts the possible test flow for 3D-ICs. For each completed die, it is carefully tested to avoid the defects. Then, the two healthy dies are stacked together and a test is established to ensure the correctness of logic functions and TSVs. If the stack test is done and passed, other dies are stacked and tested until finishing the stack process.

After the assembly and packaging process, one more test is performed into the final chip. As shown in this flow, there are numerous of needed tests in 3D-ICs manufacturing which impact the cost of the device.

- **Reliability issue**: as discussed in Chapter 6, designers need to obtain the reliability of the system to avoid redesign risk. Especially for highly reliable applications, designers need to select proper fault-tolerant solutions to ensure the reliability. This dissertation has presented a method to early assess the reliability of NoC systems. To obtain the reliability of the full system, the proposed method can be extended or an alternative method can be used [43].

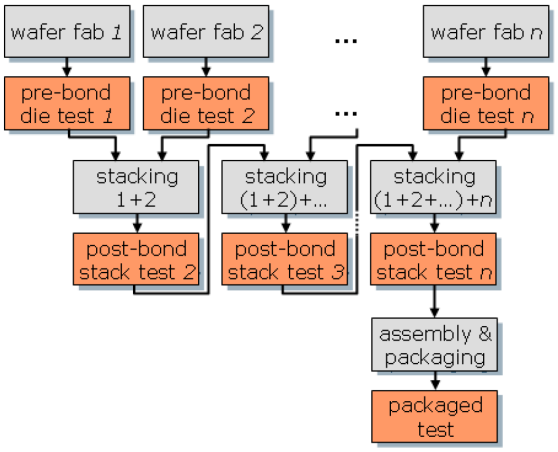

Figure 7.8: Test flow for 3D-ICs [182].

7.3.3 Detail Design

After satisfying the requirements in the preliminary design, the *detailed design* stage is fully performed from RTL design to the final layout. By using the specification and possible high-level models, the design can be coded using an HDL language. The following steps are: RTL-simulation, Synthesis, Post-synthesis simulation, *Monte-Carlo MTTF simulation*, *TSV Placement calculation*, Place and Route, P&R simulation and IO pad insertion. The fabrication and testing step (FAIT) follows this stage. Because most of the steps are well known, this section only discusses about three two additional parts: *Monte-Carlo MTTF simulation* and *TSV Placement calculation*.

Monte-Carlo MTTF simulation

The *Monte-Carlo MTTF simulation* was previously presented in Section 6.4. The flow of this simulation consists of the following:

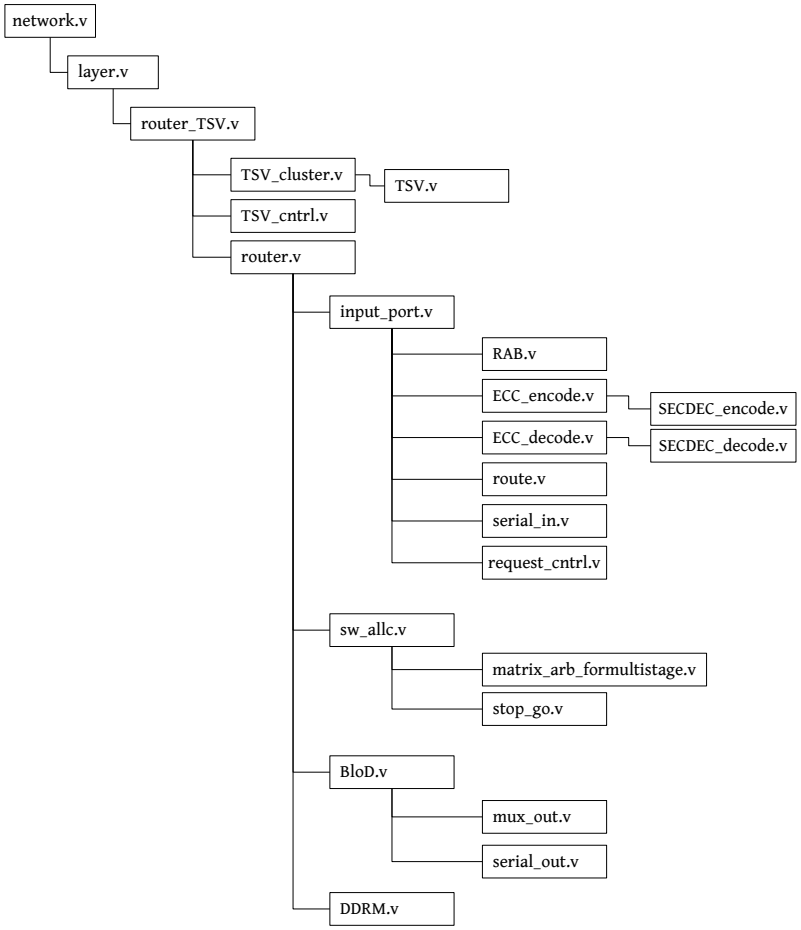

Figure 7.9: List of files in the design.

- The NoC architecture is designed in Verilog HDL and synthesized using Synopsys Design Compiler to obtain a post-synthesis netlist model.
- A fault distribution system is automatically integrated inside the NoC netlist model by using Python's Regular Expression scripts [174]. The faults are modeled in stuck at "0" and "1". Our method is similar to the fault injection methods in [102, 133].
- The post error injection netlist model is used to simulate and find the number of faults leading to failure.
- The number of faults is recorded for further processing.

The error injector architectures are shown in Figure 6.10 with two models: normal gate and flip-flop gate. When the error injector is enabled (by setting up $C = 1$), it forces the output of its attached gate to "0" or "1". If the error injector is disabled, the correct output is forwarded.

Because the netlist generated from EDA tools does not support error injection, we need to process the design to allow. In order to do that, we propose a process named as "Netlist Processing".

Netlist processing consists of two steps: (1) insert injectors into netlist file; and (2) link the error injectors. Figure 7.10 shows the flow chart of error injector inserting. In the first step, it prepares the possible gate labels and input/output labels which are extracted from library. Second, the script file matches the gate labels with the module instantiations. If they are matched, it creates new output wires, insert *EIJ.v* (the error injector module) to the output of the gate. After completing a file, a random generator (*RAN_F.v*) is inserted for generate random fault positions.

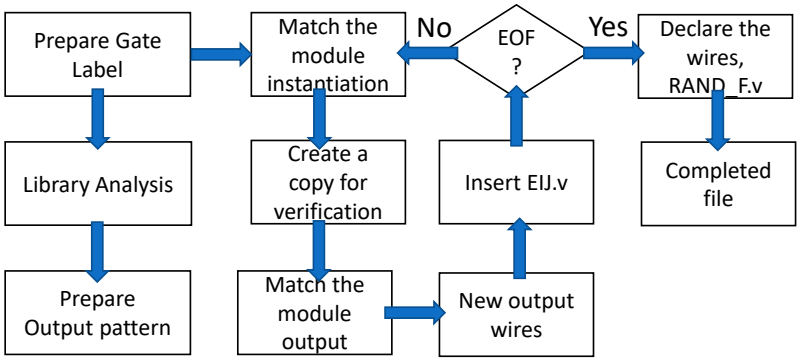

Figure 7.10: Flow chart of error injector inserting.

An example of input and output of this process is shown in Fig. 7.12. The processed module is AND 3 to 1 which consists of two AND gates. In the error injection process, faults can be inserted in these gates. In order to do that, the outputs of these gates are driven by a fixed value to create a stuck-at-0/1 failure. Therefore, beside the instantiations *AND_2x1*, we insert two *EJI* modules. Notice that the outputs of *AND_2x1s* are changed to temporary values (*temp_wire*) which are the inputs of the EIJ modules. These EIJ modules are actual driver of the correct outputs (n2 and o1). Depend on the control signal *c*, the EIJs can change the outputs or keep the correct values. To control the EIJs, we need a random generator which takes a trigger and generates a randomized position.

With the aids of trigger signals, a multiple sub-modules design can have ability to distribute the fault properly. Figure 7.11 shows how the system decide which module having a fault. If the fault is triggered in the top file (network), the router is trigger with equally probability. Therefore, one of them will have a fault which is distributed to its sub-modules. In the submodules (input port, switch allocators, ...), each of them have a dedicated probability. Depended on those probabilities,

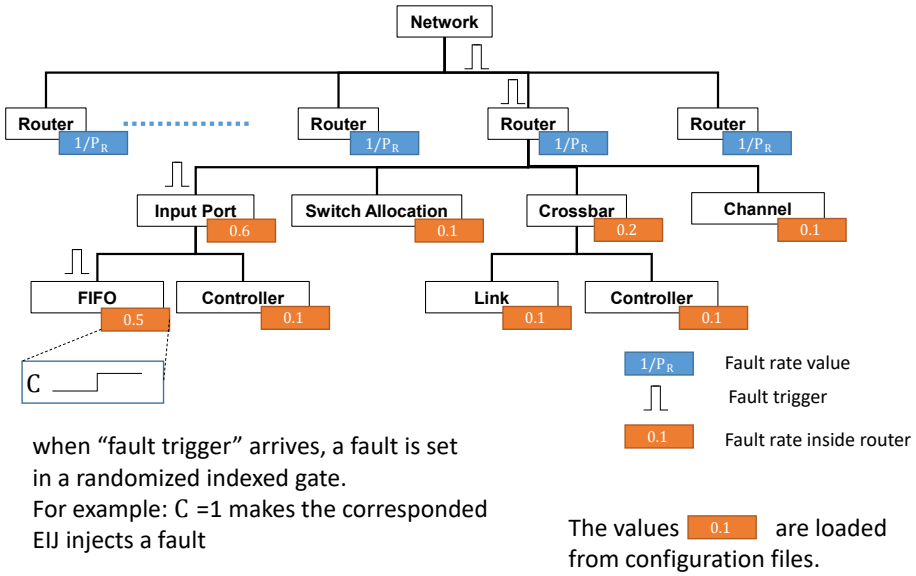

Figure 7.11: Fault trigger.

the fault is distributed. A similar process is applied to the deeper sub-modules. When the smallest module has been triggered, the signal is send to *RAND_F* to select the faulty gate.

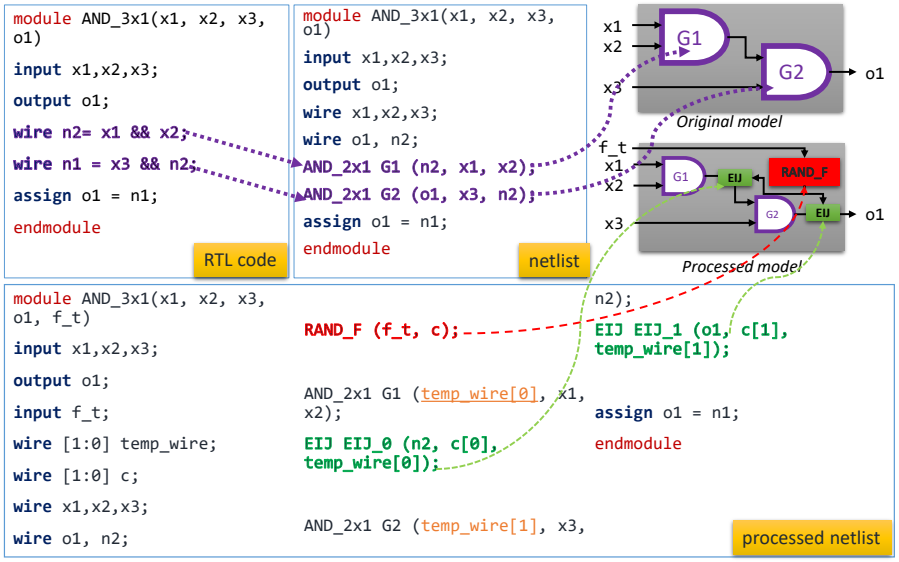

Figure 7.12: An example of input and output of the netlist processing.

Figure 7.13 shows the flow chart for multiple modules processing. This process is similar to a single module process; however, it adds a signal for trigger the fault which is used later.

The simulation process is depicted in Figure 6.11:

- At the first iteration, the injected position is generated and the corresponding position is distributed with a fault.

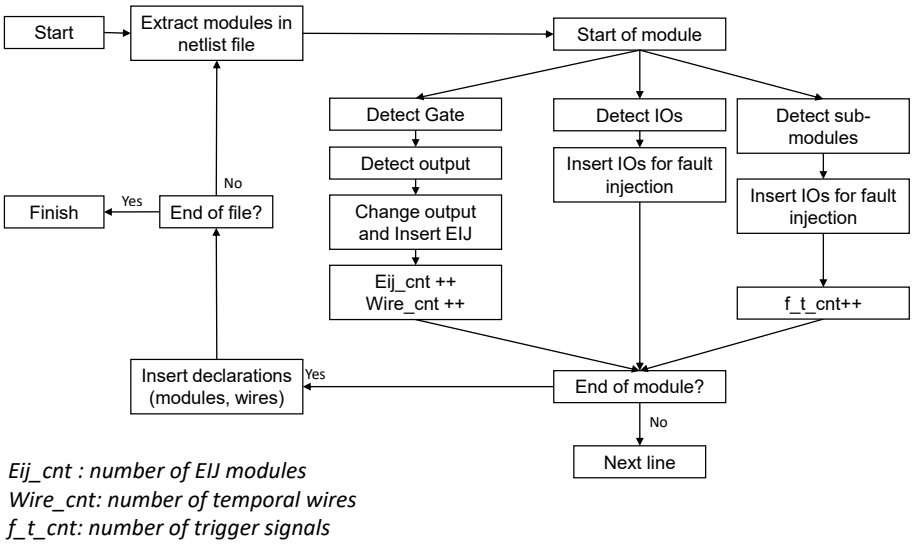

Figure 7.13: Netlist processing for multiple modules file.

- Hard faults are injected until finishing the experiment. Soft errors disappear after one clock cycle.
- A testbench is designed to verify the system correctness after injecting a fault (100% of the PEs are connected and 100% of the packets are correctly delivered). If the system is still correctly functioning after a fault is injected, the simulation injects more faults. Otherwise, the number of faults and working time are recorded.
- The time-to-failure is calculated based on the number of faults or the working time. The final MTTF is the average value of all iterations.

For iteration i, the time-to-failure is measured as the number of faults injected as:

$$TTF_i = f_i \times \frac{1}{\lambda_{system}} \tag{7.1}$$

Where λ_{system} is the raw fault-rate of the original system. After finishing a simulation of N iterations, the MTTF value of a system is given by equation 7.2 where f_i is the number of faults causing failure in experiment i.

$$MTTF_{system} = \frac{\sum_{i=1}^{N} TTF_i}{N} = \frac{\sum_{i=1}^{N} f_i}{N} \times \frac{1}{\lambda_{system}} \tag{7.2}$$

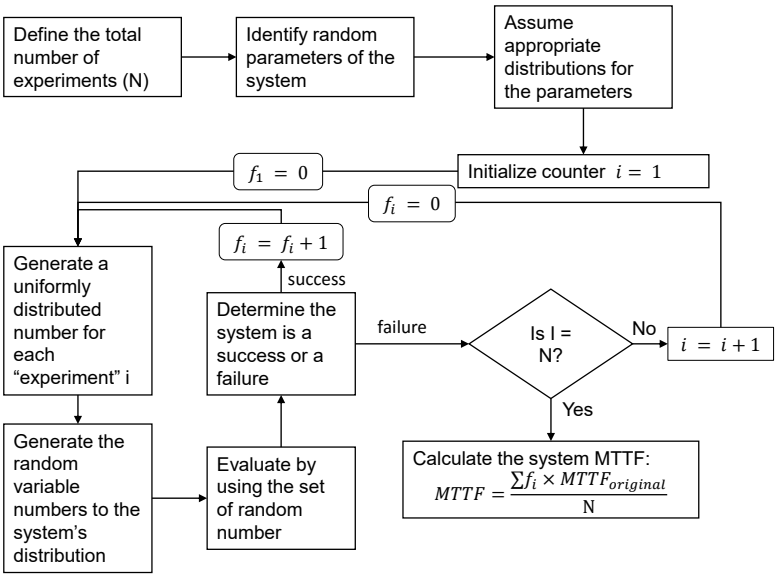

Figure 7.14: MTTF Monte-Carlo simulation process.

After the MTTF simulation finds out the reliability of the system, designers can decide to forward to the next step and keep optimizing or enhancing the design. If the design is completed, the place and route is the following step. However, before the P&R step, the positions of TSVs should be calculated.

TSV Placement Calculation

In chapter 5, the cluster-TSV method has been presented. This section discusses the mapping of the complete final system TSV connections. Figure 7.15 shows a layer layout of 2×2. Besides the PEs and the routers, TSVs are placed in a fixed area. After defining the area of TSVs, each TSV address has to be carefully calculated. To ensure the alignment, the TSVs' position on the other layer have to be properly aligned.

After calculating the TSVs' positions, TSVs are placed onto the layout. The other modules (routers, PEs, NIs) are later placed. Finally, the routing process connects them together.

7.4 Implementation Results

In the previous sections, the TSV-based 3D-NoC system design flow has been discussed. In order to summarize the methodology, this section presents the implementation results. Table 7.4 depicts the system configurations of the design.

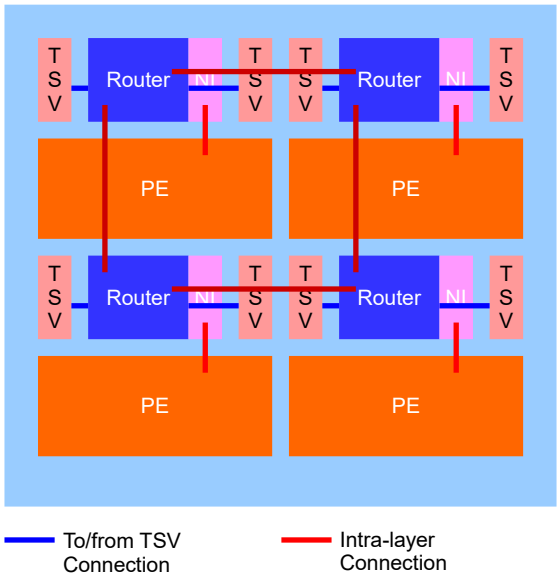

Figure 7.15: An example of layer layout.

Table 7.4: System configurations.

Parameter	Value
# ports	7
Topology	3D Mesh
Routing Algorithm	Look-ahead routing
Flow Control	Stall-Go
Forwarding mechanism	Wormhole
Input buffer	4
Flit width	44

7.4.1 Hardware Complexity

Table 7.5 shows the hardware complexity of a single router of the final system (TSV-FETO) in terms of area, power (static, dynamic, and total), and speed. In comparison to the router in which we implement the proposed techniques (3D-FETO), the area and power consumption have increased by 30.42% and 18.66%, respectively. The maximum speed has also slightly decreased by 12.37%. In comparison to the baseline model, the proposed system almost doubles the area cost and power consumption while decreasing the maximum frequency by about 50%. The degradations in this evaluation is caused by the fault-tolerant modules where additional area cost is required. The fault-tolerant modules also operate concurrently with the router which require extra power consumption. Moreover, the calculation of the fault-tolerance is more complex than the routing process in the baseline model which reduces the maximum frequency. However, the final design

is still relatively small (average router+TSVs' width: 425μm, see Section 7.4.2) and the maximum frequency is still significantly high enough for operation (≥500MHz).

Table 7.5: Hardware complexity of a single router.

Model	**Area** (μm^2)	**Power** (mW)			**Speed** (Mhz)
		Static	Dynamic	Total	
Baseline 3D router	18,873	5.1229	0.9429	6.0658	925.28
3D-FETO Router	29,780	10.017	2.2574	12.3144	613.50
TSV-FETO	38,838	11.7910	2.8273	14.6128	537.63

7.4.2 Layout

The layouts of 2 × 2 layer are depicted in Fig. 7.16. The parameters can be found in Table 7.3. The required areas for a 2 × 2 layer of 3D-FETO and TSV-FETO are 600μm × 600μm and 850μm × 850μm, respectively. In average, the final router width is 425μm.

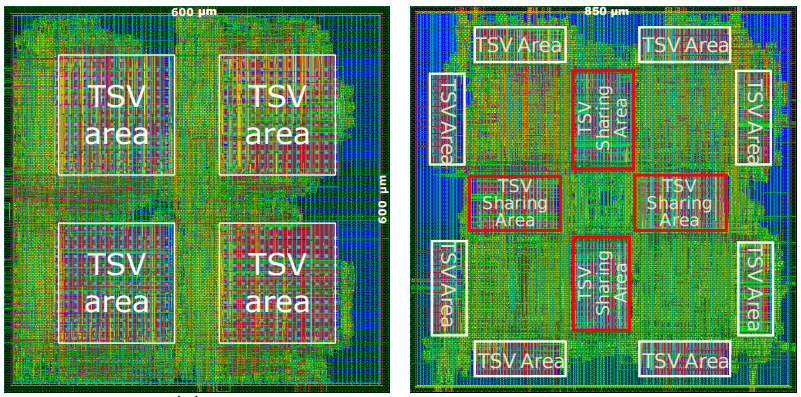

Figure 7.16: Layout of a 2 × 2 layer: (a) 3D-FETO; (b) 3D-FETO + TSV Fault-Tolerance.

7.4.3 Performance Evaluation

Latency Evaluation

In this experiment, we evaluate the performance of the proposed architecture in terms of Average packet Latency (APL) over various benchmark programs and defect-rates. The simulation results are shown in Fig. 7.17 (a). From this graph, we notice that with a 0% of defect-rate, the system's tolerance has similar performance in comparison to the baseline system.

When we increase the defect-rates in the proposed system, it has demonstrated additional impacts on APL. At a 1% fault-rate using Matrix, Uniform, Transpose, and Hotspot 10% bench-

marks, the system increases the APL by 83.24%, 64.46%, 11.30% and 66.55%, respectively. These high impacts are due to the occurrence of bottlenecks inside the network. Because all vertical connections are utilized, *Virtual TSVs* have caused congestions by sharing the TSV between two routers. The serialization is already a bottleneck technique. These bottlenecks effects are even higher at a 30% of defect-rate where the APL can be over 3 times the 0% case in the synthetic benchmarks.

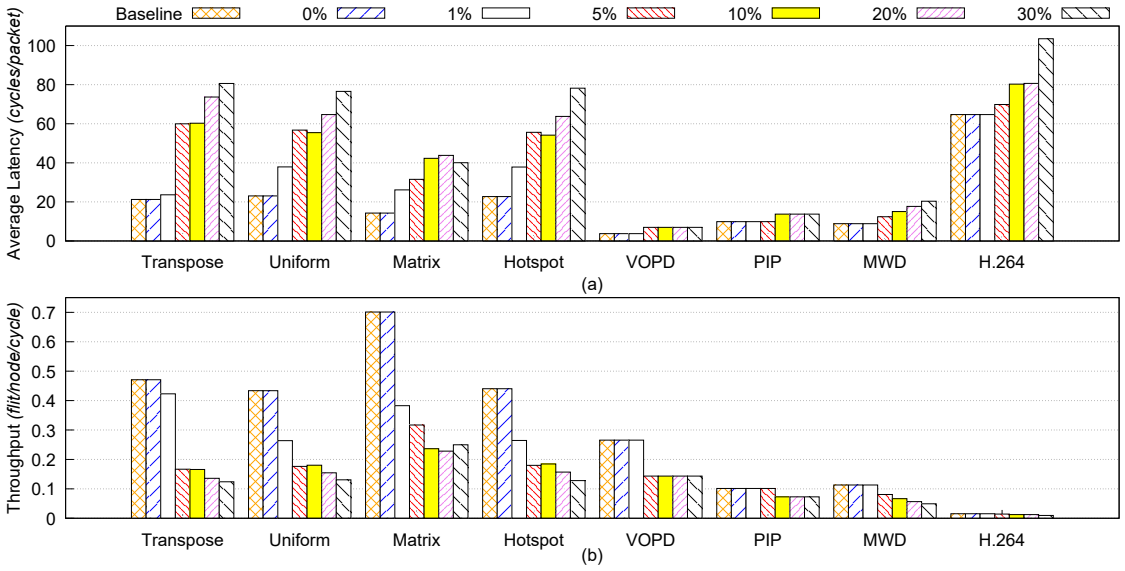

Figure 7.17: Evaluation result: (a) Average Packet Latency; (b) Throughput.

With H.264, PIP, MWD and VOPD benchmarks, the APL incrementation are significantly reduced due to the low utilization of TSV. We can observe the identical performance of the VOPD benchmark from a 1% to a 30% defect-rates. With the PIP benchmark, the system under 1% defect-rate has similar performance to 0% thanks to the optimization process which disables the unused clusters. With the MWD and H.264 benchmarks, the impact on APL is gradually increased when increasing the defect-rate. Even at a 30% of defect-rate, the APL values of MWD and H.264 are increased by 129.91% and 60.04%, respectively. Because there is no optimized routing technique for these benchmarks, the bottleneck effect is expected to happen. Although there are significant impacts on latency, the system has proven to work without major issues in all benchmarks.

Throughput Evaluation

Figure 7.17 (b) depicts the throughput evaluation with different benchmarks. At 0% defect-rate, the proposed system's throughput is similar to that of the baseline. When defects are injected into the system, we can observe some degradation in throughput caused by the bottleneck effects on the system. Similar to APL, the throughput degradation on realistic traffic benchmarks (VOPD, H.264, MWD and PIP) are significantly better than the synthetic ones. The system at a 20% defect-rate provides a decreased throughput by 71.17%, 64.36%, 67.44% and 64.37% for Transpose, Uniform, Matrix, and Hotspot 10%, respectively. At the same defect-rate, VOPD, MWD, PIP and H.264 have 46.03%, 50.04%, 28.17%, and 19.79% of throughput degradation. This lower impact is caused by the low utilization of vertical connection rate and the optimization process. The throughput of realistic benchmarks are naturally smaller than the synthetic ones because of the specific tasks order of execution that was observed in the task graphs [162, 163].

Although there is a considerable degradation in the throughput evaluation, the system still maintains over 0.1 *flit/node/cycle* in the highly stressed benchmarks, even at extremely high defect-rates.

7.5 Conclusion

In this chapter, the dissertation approach on *Design for Reliability* was presented. In summary, this work aims to complement the three first design stages: *Specification*, *Preliminary Design* and *Detail Design*. The previous chapter has presented the fault-tolerance techniques and the assessment methodologies. By merging them into the *Design for Reliability* graph, this chapter tried to present a complete and comprehensive design flow. The adopted design method incorporates all the fault-tolerant architectures and algorithms along the reliability assessment models to present a highly reliable 3D-NoC design,

8

Conclusion and Future Work

Nowadays, technology has become an essential part of our life, not only in industry or academic research, but also in consumers' daily life. In fact, with the continuous technology scaling and system miniaturization, it is now possible to implement a full system capable of performing various tasks (e.g., computation, control, memory storage, etc.) on a very small chip area that can be embedded and used anywhere and anytime. Despite the huge steps that technology has witnessed, an extensive load of research have been undertaken throughout the past decades to make these systems much more faster, power efficient and durable. Among the numerous proposed solutions, *3D Integrated-Circuits* (3D-ICs) have been considered as one of the pillars of future systems. This is because they offer less power consumption and delay, improves the density and allows heterogeneous stacking. From the system perspective, 3D-stacked ICs can integrate a larger number of cores per chip. On the other hand, increasing the number of integrated

cores arises another major concern for future ICs. In fact, connecting such a large number of cores requires better scalability, power consumption and latency. In order to solve the interconnection challenges, *3D-Network-on-Chip* (3D-NoCs) is widely considered as the backbone of future on-chip communication infrastructures.

Despite its benefits, 3D-NoCs are predicted to face several reliability issues. As it is the case for existing 2D-ICs, both soft errors and hard faults are the major challenges for highly reliable systems. Moreover, by using *Through-Silicon-Vias* (TSVs) as the interlayer medium, 3D-NoCs also encounter the TSV defect problem. All of these fault types give has significants impact on the reliability of the system from manufacturing (yield) to the run-time (time to failure). Since 3D-NoCs are predicted as the future communication paradigm, it has become primordial to properly address the potential risk of faults and present efficient methods to work around their presence. Besides proposing fault-tolerance methods, one of the most critical challenges for designers is to understand the efficiency of the methods in terms of reliability. Not after the system manufacturing or design; but, it should be in the early design stages. Therefore, reliability assessment has become a required step in the *Design for Reliability* flow.

Starting from the aforementioned facts, the ultimate goal of this research was to propose a comprehensive set of faults-tolerant architectures and algorithms to tackle the presence of soft errors, hard faults, and TSV defects in 3D-NoC systems, and 3D-ICs in general. In addition, a reliability assessment platform was presented to help designers analyze the fault-tolerance mechanism and avoid any risk of expensive redesign in terms of time and cost. Consequently, this dissertation offers a complete and encompassing design flow that covers the most critical challenges for future highly reliable systems.

A soft error resilient method was presented to deal with the occurrence of soft errors on the pipeline stages. By adding redundant calculation and ensuring the transmission FSM, the system can detect soft errors. For the correction, another redundancy is required for using TMR. The proposed method was successfully implemented and has obtained a maximum of 33% (1/3) of error-rate while ensuring graceful degradation in terms of performance, area cost and power consumption. With the help of an existing Error Correction Code, previous hard fault tolerant techniques, and a light-weigh detection, diagnosis and recovery mechanism, the firstly proposed system (called 3D-FETO) was able to deal with both soft errors and hard faults.

After ensuring the NoC fault-tolerance within layers, the inter-layer communication was considered. In fact, cluster defects are considered as the major source of failure in TSV-based 3D-ICs which cannot be efficiently dealt by using redundancy or coding. To solve these issues, a scalable management scheme is proposed. Without requiring any redundancy, the proposed architecture and algorithm provide smart architectural and TSV mapping solutions. The final results show an extremely high amount of working connections (nearly 100%) even with high defect-rate (up to 50% defected cluster). Although the performance has shown a degradation with high defect-rate, it still reasonable seeing its high fault-tolerance capacity. For instance, when evaluating the proposed system with realistic traffic patterns, the system performance is still maintained, even with a 1% or 5% of defect-rate. Finally, with the aid of the aforementioned soft error and hard fault tolerant techniques, the final and complete system, named TSV-3D-FETO can handle all types of defects while maintaining a graceful performance degradation.

In order to help designers understand the efficiency of any fault tolerant technique employed in a given NoC system, a platform for reliability assessment is presented. Based on the *Dividing and Conquer* concept and *Markov state model*, a complex NoC design can be split for analysis and later summarized for final reliability results. A new dedicated parameter is also presented to measure the efficiency of fault-tolerant techniques. For validation, a *netlist-based* simulation was also performed and compared with the proposed assessment platform. The final results showed a reasonable accuracy of the reliability assessment which can be obtained at an extremely short amount of time. Moreover, an in-depth analysis of the proposed fault-tolerance techniques was undertaken.

In summary, this dissertation has shown a complete and comprehensive set of fault-tolerant architectures, algorithms, and reliability assessment for 3D-NoC systems. The final system can handle multiple types of errors/faults/defects, and its performance, area cost, power consumption, and reliability efficiency can be precisely analyzed. In addition, this work has completed the final layout of the design which showed a solid flow of *Design for Reliability*: specification, analytical analysis, and design stage. Moreover, the presented approach is proposed for general 3D-IC design and can be applied to any system or any type of NoC.

Despite the completed work has been provided, 3D-ICs and 3D-NoCs still have several issues to be addressed. As classified in the related works, there are different approaches on fault-tolerance

(e.g., physical-layer or system-layer solutions) which can be combined to obtain a better reliability. Since 3D-NoCs is only the communication infrastructure, highly reliable systems must require reliable PEs which also need to be addressed. Furthermore, fault-tolerance is still developed independently without defining any standards. In fact, this work has briefly shown one of the very first fault-tolerant designs for multiple types of fault. By standardizing the fault-tolerance, designers can select and adjust the fault-tolerant techniques to optimize their requirements.

Besides the possible future works for 3D-NoC fault-tolerance, 3D-ICs still have several issues to address which also make impacts on reliability. For example, heat dissipation and power density is a critical issue for 3D-ICs. Research on 3D-ICs is also trying to reduce the TSV size or even using monolithic 3D structures which open new challenges.

List of Publications

Refereed Journals

1. **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, "A Low-overhead Soft-Hard Fault Tolerant Architecture, Design, and Management Scheme for Reliable High-performance Many-core 3D-NoC Systems", The Journal of Supercomputing, Volume 73, Issue 6, pp 2705–2729, June 2017.

2. **Khanh N. Dang**, Akram Ben Ahmed, Xuan-Tu Tran, Yuichi Okuyama and Abderazek Ben Abdallah, "A Comprehensive Reliability Assessment of Fault-Resilient Network-on-Chip Using Analytical Model", IEEE Transactions on Very Large Scale Integration Systems, *(in press)*.

Refereed Refereed International conferences

1. **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, "Reliability Assessment and Quantitative Evaluation of Soft-Error Resilient 3D Network-on-Chip Systems", The IEEE 25th Asian Test Symposium (ATS), pp. 161-166, Hiroshima, Japan, November 21-24, 2016.

2. **Khanh N. Dang**, Yuichi Okuyama, and Abderazek Ben Abdallah, "Soft-error resilient network-on-chip for safety-critical applications", The 2016 International Conference on IC Design and Technology (ICICDT), pp. 1-4, Ho Chi Minh City, Vietnam, June 27-29, 2016.

3. **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama, Abderazek Ben Abdallah, and Xuan-Tu Tran, "Soft-error resilient 3d network-on-chip router", The 2015 IEEE 7th International Conference on Awareness Science and Technology (iCAST), pp. 84-90 Qinhuangdao, China, September 22-24, 2015.

4. **Khanh N. Dang**, The H. Vu, Yuichi Okuyama and Abderazek Ben Abdallah, "Fault Analysis and an Efficient Reliability Assessment for Network-on-Chip Systems", The 14th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, October 30 - November 03, 2017, *(accepted)*.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–116, April 1965.

[2] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[3] T. Ray. (2015, May) Xilinx, altera: "dark silicon" makes their wares essential, says pac crest. [Online]. Available: http://blogs.barrons.com/techtraderdaily/2015/05/19/xilinx-altera-dark-silicon-makes-their-wares-essential-says-pac-crest/

[4] J. X. Chen, "The evolution of computing: Alphago," *Computing in Science Engineering*, vol. 18, no. 4, pp. 4–7, July 2016.

[5] C. Batten. (2014) Energy-efficient parallel computer architecture. Cornell University.

[6] M. D. Hill, S. Adve, L. Ceze, M. J. Irwin, D. Kaeli, M. Martonosi, J. Torrellas, T. F. Wenisch, D. Wood, and K. Yelick, "21st century computer architecture," *arXiv preprint arXiv:1609.06756*, 2016.

[7] L. Ceze, M. D. Hill, and T. F. Wenisch, "Arch2030: A vision of computer architecture research over the next 15 years," *arXiv preprint arXiv:1612.03182*, 2016.

[8] ITRS, "2011 Edition Update Process Integration, Devices, and Structures," The International Technology Roadmap for Semiconductor, Tech. Rep., 2011, http://www.itrs2.net(accessed 16.06.16).

[9] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas, "A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*. IEEE, 2016, pp. 1–2.

[10] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Power challenges may end the multicore era," *Communications of the ACM*, vol. 56, no. 2, pp. 93–102, 2013.

[11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[12] A. Ben Abdallah and S. Masahiro, "Basic Network-on-Chip Interconnection for Future Gigascale MCSoCs Applications: Communication and Computation Orthogonalization," in *Proc. of the Symposium on Science, Society, and Technology (JASSST2006)*, 2006, pp. 1–7.

[13] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[14] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob *et al.*, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS," in *IEEE International Solid-State Circuits Conference, 2007.Digest of Technical Papers.* IEEE, 2007, pp. 98–589.

[15] ITRS, "2005 Edition Interconnect," The International Technology Roadmap for Semiconductor, Tech. Rep., 2005, http://www.itrs2.net(accessed 16.06.16).

[16] Ping Chao and Lavi Lev, "Down to the wire – requirements for nanometer design implementation," http://www.eetimes.com/document.asp?doc_id=1205898, 2002, (accessed 16.06.16).

[17] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," in *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*, ser. SLIP '04. New York, NY, USA: ACM, 2004, pp. 7–13. [Online]. Available: http://doi.acm.org/10.1145/966747.966750

[18] ITRS, "2012 Edition Update Process Integration, Devices, and Structures," The International Technology Roadmap for Semiconductor, Tech. Rep., 2012, http://www.itrs2.net(accessed 16.06.16).

[19] T. Kagami, H. Matsutani, M. Koibuchi, Y. Take, T. Kuroda, and H. Amano, "Efficient 3-d bus architectures for inductive-coupling thruchip interfaces," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 24, no. 2, pp. 493–506, 2016.

[20] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, "Wireless noc as interconnection backbone for multicore chips: Promises and challenges," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 228–239, 2012.

[21] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, no. 5545, pp. 1317–1320, 2001.

[22] H. Li and K. Banerjee, "High-frequency analysis of carbon nanotube interconnects and implications for on-chip inductor design," *IEEE Transactions on Electron Devices*, vol. 56, no. 10, pp. 2202–2214, Oct 2009.

[23] J. S. Levy, A. Gondarenko, M. A. Foster, A. C. Turner-Foster, A. L. Gaeta, and M. Lipson, "Cmos-compatible multiple-wavelength oscillator for on-chip optical interconnects," *Nature photonics*, vol. 4, no. 1, pp. 37–40, 2010.

[24] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 602–633, 2001.

[25] G. Smith, L. Smith, S. Hosali, and S. Arkalgud, "Yield considerations in the choice of 3D technology," in *International Symposium on Semiconductor Manufacturing (ISSM)*. IEEE, 2007, pp. 1–3.

[26] N. Srivastava, H. Li, F. Kreupl, and K. Banerjee, "On the applicability of single-walled carbon nanotubes as vlsi interconnects," *IEEE Transactions on Nanotechnology*, vol. 8, no. 4, pp. 542–559, July 2009.

[27] S. A. Panth, K. Samadi, Y. Du, and S. K. Lim, "Design and cad methodologies for low power gate-level monolithic 3d ics," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 171–176.

[28] R. Ishihara, J. Derakhshandeh, M. T. Mofrad, T. Chen, N. Golshani, and C. Beenakker, "Monolithic 3D-ICs with single grain Si thin film transistors," *Solid-State Electronics*, vol. 71, pp. 80–87, 2012.

[29] A. Ben Ahmed, A. Ben Abdallah, and K. Kuroda, "Architecture and Design of Efficient 3D Network-on-Chip (3D NoC) for Custom Multicore SoC," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, Nov 2010, pp. 67–73.

[30] F. Ye and K. Chakrabarty, "TSV open defects in 3D integrated circuits: Characterization, test, and optimal spare allocation," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 1024–1030.

[31] J. Zhao, Q. Zou, and Y. Xie, "Overview of 3D Architecture Design Opportunities and Techniques," *IEEE Design & Test*, vol. PP, no. 99, pp. 2168–2356, July 2015.

[32] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J. Ahn, D. Kwon, J. H. Kim, J.-W. Lee *et al.*, "8Gb 3D DDR3 DRAM using through-silicon-via technology," in *IEEE International Solid-State Circuits Conference-Digest of Technical Papers (ISSCC)*. IEEE, 2009, pp. 130–131.

[33] T. Zhang, Y. Zhan, and S. Sapatnekar, "Temperature-aware routing in 3D ICs," in *Asia and South Pacific Conference on Design Automation*, Jan 2006, pp. 309–314.

[34] Y. J. Park, M. Zeng, B. seok Lee, J.-A. Lee, S. G. Kang, and C. H. Kim, "Thermal Analysis for 3D Multi-core Processors with Dynamic Frequency Scaling," in *IEEE/ACIS 9th International Conference on Computer and Information Science (ICIS)*, Aug 2010, pp. 69–74.

[35] A. Eghbal, P. M. Yaghini, N. Bagherzadeh, and M. Khayambashi, "Analytical Fault Tolerance Assessment and Metrics for TSV-based 3D Network-on-Chip," *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3591–3604, December 2015.

[36] 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 *IEEE International Reliability Physics Symposium (IRPS)*, April 2011, pp. 3F.4.1–3F.4.6.

[37] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A highly resilient routing algorithm for fault-tolerant NoCs," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 21–26.

[38] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Reliability Modeling and Management in Dynamic Microprocessor-based Systems," in *Proceedings of the 43rd Annual Design Automation Conference*, 2006, pp. 1057–1060.

[39] ITRS, "2012 Edition Update Process Integration, Devices, and Structures," The International Technology Roadmap for Semiconductor, Tech. Rep., 2012, http://www.itrs2.net(accessed 16.06.16).

[40] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *2011 International Reliability Physics Symposium*, April 2011, pp. 5B.4.1–5B.4.7.

[41] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in networks-on-chip," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 8, 2013.

[42] D. Crowe and A. Feinberg, *Design for reliability*. CRC press, 2001, vol. 11.

[43] I. S. C. Committee, *IEEE Guide for Selecting and Using Reliability Predictions. IEEE 1413.1*, A. Ickowicz, Ed. IEEE, 2002.

[44] T. Lehtonen, P. Liljeberg, and J. Plosila, "Online reconfigurable self-timed links for fault tolerant NoC," *VLSI design*, vol. 2007, pp. 1–13, 2007.

[45] S. F. Al-Sarawi, D. Abbott, and P. D. Franzon, "A review of 3-d packaging technology," *IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part B*, vol. 21, no. 1, pp. 2–14, 1998.

[46] H. Yoshikawa, A. Kawasaki, Tomoaki, Iiduka, Y. Nishimura, K. Tanida, K. Akiyama, M. Sekiguchi, M. Matsuo, S. Fukuchi, and K. Takahashi, "Chip Scale Camera Module (CSCM) using Through-Silicon-Via (TSV)," in *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2009, pp. 476–477,477a.

[47] J. Ahn *et al.*, "7.1 A 1/4-inch 8Mpixel CMOS image sensor with 3D backside-illuminated 1.12 um pixel with front-side deep-trench isolation and vertical transfer gate," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 124–125.

[48] V. Suntharalingam, R. Berger, S. Clark, J. Knecht, A. Messier, K. Newcomb, D. Rathman, R. Slattery, A. Soares, C. Stevenson *et al.*, "A 4-side tileable back illuminated 3d-integrated mpixel cmos image sensor," in *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*. IEEE, 2009, pp. 38–39.

[49] K. Ishida, T. Yasufuku, S. Miyamoto, H. Nakai, M. Takamiya, T. Sakurai, and K. Takeuchi, "A 1.8 V 30nJ adaptive program-voltage (20V) generator for 3D-integrated NAND flash SSD," in *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*. IEEE, 2009, pp. 238–239.

[50] M. Saen, K. Osada, Y. Okuma, K. Niitsu, Y. Shimazaki, Y. Sugimori, Y. Kohama, K. Kasuga, I. Nonomura, N. Irie *et al.*, "3-d system integration of processor and multi-stacked srams using inductive-coupling link," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 856–862, 2010.

[51] M. Karnezos, "3d packaging: Where all technologies come together," in *Electronics Manufacturing Technology Symposium, 2004. IEEE/CPMT/SEMI 29th International*. IEEE, 2004, pp. 64–67.

[52] J. Miettinen, M. Mantysalo, K. Kaija, and E. O. Ristolainen, "System design issues for 3d system-in-package (sip)," in *Electronic Components and Technology Conference, 2004. Proceedings. 54th*, vol. 1. IEEE, 2004, pp. 610–615.

[53] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon, "Demystifying 3d ics: The pros and cons of going vertical," *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 498–510, 2005.

[54] E. Culurciello and A. G. Andreou, "Capacitive inter-chip data and power transfer for 3-d vlsi," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 12, pp. 1348–1352, 2006.

[55] C. Fenouillet-Beranger, P. Batude, L. Brunet, V. Mazzocchi, C. M. V. Lu, F. Deprat, J. Micout, M. P. Samson, B. Previtali, P. Besombes, N. Rambal, F. Andrieu, O. Billoint, M. Brocard, S. Thuries, G. Cibrario, and M. Vinet, "Recent advances in 3d vlsi integration," in *2016 International Conference on IC Design and Technology (ICICDT)*, June 2016, pp. 1–4.

[56] G. Van der Plas, P. Limaye, I. Loi, A. Mercha, H. Oprins, C. Torregiani, S. Thijs, D. Linten, M. Stucchi, G. Katti *et al.*, "Design issues and considerations for low-cost 3-D TSV IC technology," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 293–307, 2011.

[57] D. Triyoso, T. Dao, T. Kropewnicki, F. Martinez, R. Noble, and M. Hamilton, "Progress and challenges of tungsten-filled through-silicon via," in *IEEE International Conference on Integrated Circuit Design and Technology*, 2010.

[58] T. Dao, D. H. Triyoso, R. Mora, T. Kropewnicki, B. Griesbach, D. Booker, M. Petras, and V. Adams, "Thermo-mechanical stress characterization of tungsten-fill through-silicon-via," in *International Symposium on VLSI Design Automation and Test (VLSI-DAT)*. IEEE, 2010, pp. 7–10.

[59] M. J. Wolf, T. Dretschkow, B. Wunderle, N. Jurgensen, G. Engelmann, O. Ehrmann, A. Uhlig, B. Michel, and H. Reichl, "High aspect ratio TSV copper filling with different seed layers," in *Electronic Components and Technology Conference*, 2008, pp. 563–570.

[60] G. Katti, A. Mercha, J. Van Olmen, C. Huyghebaert, A. Jourdain, M. Stucchi, M. Rakowski, I. Debusschere, P. Soussan, W. Dehaene *et al.*, "3D stacked ICs using Cu TSVs and die to wafer hybrid collective bonding," in *IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2009, pp. 1–4.

[61] M. Koyanagi, T. Fukushima, and T. Tanaka, "High-density through silicon vias for 3-D LSIs," *Proceedings of the IEEE*, vol. 97, no. 1, pp. 49–59, 2009.

[62] M. H. Jabbar and D. Houzet. 3D Architecture Implementation: A Survey. [Online]. Available: http://www.design-reuse.com/articles/29484/3d-architecture-implementation-survey.html

[63] 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*, vol. 31, no. 5, pp. 726–739, 2012.

[64] 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 *The Twelfth International Symposium on High-Performance Computer Architecture*. IEEE, 2006, pp. 5–16.

[65] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 818–831, June 2005.

[66] L. Jiang, Q. Xu, and B. Eklow, "On effective through-silicon via repair for 3-d-stacked ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 559–571, 2013.

[67] J. Cong and Y. Zhang, "Thermal via planning for 3-D ICs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2005, pp. 745–752.

[68] A. Ghaleshahi *et al.*, "Temperature-Gradient Based Burn-In for 3D Stacked ICs," in *The 12th Swedish System-on-Chip Conference*, 2013.

[69] S.-K. Ryu, K.-H. Lu, X. Zhang, J.-H. Im, P. S. Ho, and R. Huang, "Impact of Near-Surface Thermal Stresses on Interfacial Reliability of Through-Silicon Vias for 3-D Interconnects," *Device and Materials Reliability, IEEE Transactions on*, vol. 11, no. 1, pp. 35–43, March 2011.

[70] C. Selvanayagam, J. Lau, X. Zhang, S. Seah, K. Vaidyanathan, and T. Chai, "Nonlinear Thermal Stress/Strain Analyses of Copper Filled TSV (Through Silicon Via) and Their Flip-Chip Microbumps," *IEEE Transactions on Advanced Packaging*, vol. 32, no. 4, pp. 720–728, Nov 2009.

[71] A. Trigg, L. H. Yu, Y. Z. Xiong, S. J. Lin, C. C. Kuo, N. Khan, T. K. Hwa, and G. Shan, "Design for reliability in via middle and via last 3-D chipstacks incorporating TSVs," in *12th Electronics Packaging Technology Conference (EPTC)*, Dec 2010, pp. 328–332.

[72] G. Katti, M. Stucchi, K. De Meyer, and W. Dehaene, "Electrical modeling and characterization of through silicon via for three-dimensional ICs," *IEEE Transactions on Electron Devices*, vol. 57, no. 1, pp. 256–262, 2010.

[73] M. Cho, C. Liu, D. H. Kim, S. K. Lim, and S. Mukhopadhyay, "Design method and test structure to characterize and repair TSV defect induced signal degradation in 3D system," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 694–697.

[74] Y. Zhao, S. Khursheed, and B. Al-Hashimi, "Online Fault Tolerance Technique for TSV-Based 3-D-IC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 8, pp. 1567–1571, 2015.

[75] A. Topol, D. La Tulipe, L. Shi, S. Alam, D. Frank, S. Steen, J. Vichiconti, D. Posillico, M. Cobb, S. Medd *et al.*, "Enabling soi-based assembly technology for three-dimensional (3d) integrated circuits (ics)," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. IEEE, 2005, pp. 352–355.

[76] B. Swinnen, W. Ruythooren, P. De Moor, L. Bogaerts, L. Carbonell, K. De Munck, B. Eyckens, S. Stoukatch, D. S. Tezcan, Z. Tokei *et al.*, "3D integration by Cu-Cu thermo-compression bonding of extremely thinned bulk-Si die containing 10 μm pitch through-Si vias," in *Electron Devices Meeting, 2006. IEDM'06. International*. IEEE, 2006, pp. 1–4.

[77] N. Miyakawa, E. Hashimoto, T. Maebashi, N. Nakamura, Y. Sacho, S. Nakayama, and S. Toyoda, "Multilayer stacking technology using wafer-to-wafer stacked method," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 4, no. 4, p. 20, 2008.

[78] N. Miyakawa, "A 3D prototyping chip based on a wafer-level stacking technology," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*. IEEE, 2009, pp. 416–420.

[79] D. Malta, C. Gregory, D. Temple, T. Knutson, C. Wang, T. Richardson, Y. Zhang, and R. Rhoades, "Integrated process for defect-free copper plating and chemical-mechanical polishing of through-silicon vias for 3d interconnects," in *2010 Proceedings 60th Electronic Components and Technology Conference (ECTC)*, June 2010, pp. 1769–1775.

[80] L. Jiang, F. Ye, Q. Xu, K. Chakrabarty, and B. Eklow, "On effective and efficient in-field TSV repair for stacked 3D ICs," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 74.

[81] Y. Zhao, S. Khursheed, and B. M. Al-Hashimi, "Cost-effective TSV grouping for yield improvement of 3D-ICs," in *Asian Test Symposium (ATS)*. IEEE, 2011, pp. 201–206.

[82] I. Loi, S. Mitra, T. H. Lee, S. Fujita, and L. Benini, "A low-overhead fault tolerance scheme for TSV-based 3D network on chip links," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2008, pp. 598–602.

[83] T. Naito, T. Ishida, T. Onoduka, M. Nishigoori, T. Nakayama, Y. Ueno, Y. Ishimoto, A. Suzuki, W. Chung, R. Madurawe *et al.*, "World's first monolithic 3d-fpga with tft sram over 90nm 9 layer cu cmos," in *VLSI Technology (VLSIT), 2010 Symposium on*. IEEE, 2010, pp. 219–220.

[84] P. L. Yang and M. Marek-Sadowska, "A fast, fully verifiable, and hardware predictable asic design methodology," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 364–367.

[85] M. Lin and A. El Gamal, "A routing fabric for monolithically stacked 3d-fpga," in *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*. ACM, 2007, pp. 3–12.

[86] Xilinx Inc. (2017, March) All programmable 3d ics. [Online]. Available: https://www.xilinx.com/products/silicon-devices/3dic.html

[87] V. F. Pavlidis and E. G. Friedman, *Three-dimensional integrated circuit design*. Morgan Kaufmann, 2010.

[88] Altera Inc. Enabling next-generation platforms using intel's 3d system-in-package technology. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01251-enabling-nextgen-with-3d-system-in-package.pdf

[89] G. H. Loh, Y. Xie, and B. Black, "Processor design in 3d die-stacking technologies," *IEEE Micro*, vol. 27, no. 3, pp. 31–48, May 2007.

[90] Y. Kohama, Y. Sugimori, S. Saito, Y. Hasegawa, T. Sano, K. Kasuga, Y. Yoshida, K. Niitsu, N. Miura, H. Amano, and T. Kuroda, "A scalable 3d processor by homogeneous chip stacking with inductive-coupling link," in *2009 Symposium on VLSI Circuits*, June 2009, pp. 94–95.

[91] B. Vaidyanathan, W.-L. Hung, F. Wang, Y. Xie, V. Narayanan, and M. J. Irwin, "Architecting microprocessor components in 3d design space," in *20th International Conference on VLSI Design, 2007. Held Jointly with 6th International Conference on Embedded Systems.* IEEE, 2007, pp. 103–108.

[92] M. Taouil, M. Masadeh, S. Hamdioui, and E. J. Marinissen, "Post-Bond Interconnect Test and Diagnosis for 3D Memory Stacked on Logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1860–1872, November 2015.

[93] ——, "Interconnect test for 3D stacked memory-on-logic," in *Proceedings of the conference on Design, Automation & Test in Europe*, 2014, p. 126.

[94] A. Ben Ahmed and A. Ben Abdallah, "Architecture and design of high-throughput, low-latency, and fault-tolerant routing algorithm for 3D-network-on-chip (3D-NoC)," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1507–1532, 2013.

[95] K. Manna, V. S. S. Teja, S. Chattopadhyay, and I. Sengupta, "Tsv placement and core mapping for 3d mesh based network-on-chip design using extended kernighan-lin partitioning," in *2015 IEEE Computer Society Annual Symposium on VLSI*, July 2015, pp. 392–397.

[96] A.-M. Rahmani, P. Liljeberg, J. Plosila, and H. Tenhunen, "Bbvc-3d-noc: an efficient 3d noc architecture using bidirectional bisynchronous vertical channels," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*. IEEE, 2010, pp. 452–453.

[97] D. G. Raheja and L. J. Gullo, *Design for reliability*. John Wiley & Sons, 2012.

[98] A. B. Ahmed and A. B. 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, 2016.

[99] S. Lin, D. Costello, and M. Miller, "Automatic-repeat-request error-control schemes," *IEEE Communications Magazine*, vol. 22, no. 12, pp. 5–17, 1984.

[100] Q. Yu and P. Ampadu, "Transient and permanent error co-management method for reliable networks-on-chip," in *Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2010, pp. 145–154.

[101] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*. IEEE, 2003, pp. 7–18.

[102] Q. Yu, M. Zhang, and P. Ampadu, "Addressing network-on-chip router transient errors with inherent information redundancy," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 4, pp. 105:1–105:21, 2013.

[103] K. N. Dang, M. Meyer, Y. Okuyama, X.-T. Tran, and A. Ben Abdallah, "A soft-error resilient 3d network-on-chip router," in *IEEE 7th International Conference on Awareness Science and Technology (iCAST)*, Sept 2015, pp. 84–90.

[104] J. U. Knickerbocker, P. S. Andry, B. Dang, R. R. Horton, M. J. Interrante, C. S. Patel, R. J. Polastre, K. Sakuma, R. Sirdeshmukh, E. J. Sprogis *et al.*, "Three-dimensional silicon integration," *IBM Journal of Research and Development*, vol. 52, no. 6, pp. 553–569, 2008.

[105] M. Laisne, K. Arabi, and T. Petrov, "Systems and methods utilizing redundancy in semiconductor chip interconnects," 2013, uS Patent 8,384,417.

[106] A.-C. Hsieh and T. Hwang, "TSV redundancy: architecture and design issues in 3-D IC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 4, pp. 711–722, 2012.

[107] Y. J. Hwang, J. H. Lee, and T. H. Han, "3d network-on-chip system communication using minimum number of tsvs," in *ICT Convergence (ICTC), 2011 International Conference on*. IEEE, 2011, pp. 517–522.

[108] A. Kologeski, C. Concatto, D. Matos, D. Grehs, T. Motta, F. Almeida, F. L. Kastensmidt, A. Susin, and R. Reis, "Combining fault tolerance and serialization effort to improve yield in 3d networks-on-chip," in *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, Dec 2013, pp. 125–128.

[109] United States of America: Department of Defense, *Military Handbook: Reliability Prediction of Electronic Equipment: MIL-HDBK-217F*. United States of America: Department of Defense, 1991.

[110] M. Zhang and N. R. Shanbhag, "Soft-error-rate-analysis (SERA) methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2140–2155, 2006.

[111] J. B. Bernstein, M. Gurfinkel, X. Li, J. Walters, Y. Shapira, and M. Talmor, "Electronic circuit reliability modeling," *Microelectronics Reliability*, vol. 46, no. 12, pp. 1957–1979, 2006.

[112] A. Y. Yamamoto and C. Ababei, "Unified reliability estimation and management of NoC based chip multiprocessors," *Microprocessors and Microsystems*, vol. 38, no. 1, pp. 53–63, 2014.

[113] K. Aisopos, C.-H. O. Chen, and L.-S. Peh, "Enabling system-level modeling of variation-induced faults in networks-on-chips," in *Proceedings of the 48th Design Automation Conference*. ACM, 2011, pp. 930–935.

[114] W. Najjar and J.-L. Gaudiot, "Network resilience: A measure of network fault tolerance," *IEEE Transactions on Computers*, vol. 39, no. 2, pp. 174–181, 1990.

[115] J. Chen, G. Wang, C. Lin, T. Wang, and G. Wang, "Probabilistic analysis on mesh network fault tolerance," *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 100–110, 2007.

[116] A. Dalirsani, M. Hosseinabady, and Z. Navabi, "An analytical model for reliability evaluation of NoC architectures," in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*. IEEE, 2007, pp. 49–56.

[117] 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 *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2012, pp. 60–71.

[118] R. Parikh and V. Bertacco, "Formally Enhanced Runtime Verification to Ensure NoC Functional Correctness," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. ACM, 2011, pp. 410–419.

[119] 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, no. 4, pp. 527–540, 2010.

[120] S. Shamshiri and K.-T. Cheng, "Yield and cost analysis of a reliable NoC," in *27th IEEE VLSI Test Symposium (VTS'09)*. IEEE, 2009, pp. 173–178.

[121] C. Hernández, F. Silla, V. Santonja, and J. Duato, "Dealing with variability in NoC links," in *2nd Workshop on Diagnostic Services in Network-on-Chips*, June 2008, pp. 4–10.

[122] A. Ben Ahmed and A. Ben Abdallah, "Graceful deadlock-free fault-tolerant routing algorithm for 3D Network-on-Chip architectures," *Journal of Parallel and Distributed Computing*, vol. 74, no. 4, pp. 2229–2240, 2014.

[123] A. B. Ahmed, "High-throughput architecture and routing algorithms towards the design of reliable mesh-based many-core network-on-chip systems," Ph.D. dissertation, Graduate School of Computer Science and Engineering, University of Aizu, March 2015.

[124] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[125] F. P. Mathur and A. Avižienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair," in *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, ser. AFIPS '70 (Spring). New York, NY, USA: ACM, 1970, pp. 375–383. [Online]. Available: http://doi.acm.org/10.1145/1476936.1476994

[126] S. Shamshiri, A.-A. Ghofrani, and K.-T. Cheng, "End-to-end error correction and online diagnosis for on-chip networks," in *IEEE International Test Conference (ITC)*. IEEE, 2011, pp. 1–10.

[127] K. C. J. Chen, C. H. Chao, and A. Y. A. Wu, "Thermal-Aware 3D Network-On-Chip (3D NoC) Designs: Routing Algorithms and Thermal Managements," *IEEE Circuits and Systems Magazine*, vol. 15, no. 4, pp. 45–69, Fourthquarter 2015.

[128] L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "A crosstalk aware interconnect with variable cycle transmission," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 1. IEEE, 2004, pp. 102–107.

[129] R. W. Berger, D. Bayles, R. Brown, S. Doyle, A. Kazemzadeh, K. Knowles, D. Moser, J. Rodgers, B. Saari, D. Stanley *et al.*, "The rad750/sup tm/-a radiation hardened power-pc/sup tm/processor for high performance spaceborne applications," in *Aerospace Conference, 2001, IEEE Proceedings.*, vol. 5. IEEE, 2001, pp. 2263–2272.

[130] K. Hass, R. Treece, and A. Giddings, "A radiation-hardened 16/32-bit microprocessor," *IEEE Transactions on Nuclear Science*, vol. 36, no. 6, pp. 2252–2257, 1989.

[131] D. K. Ravindan, "Structural fault-tolerance on the noc circuit level," Institut fur Technische Informatik, Universitat Stuttgart, Tech. Rep., June 2009.

[132] H. T. Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-level soft error estimation method," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 365–381, 2005.

[133] A. Simevski, R. Kraemer, and M. Krstic, "Automated integration of fault injection into the ASIC design flow," in *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. IEEE, 2013, pp. 255–260.

[134] M. L. Shooman, *Reliability of computer systems and networks: fault tolerance, analysis, and design*. John Wiley & Sons, 2003.

[135] R. Cressent, P. David, V. Idasiak, and F. Kratz, "Designing the database for a reliability aware Model-Based System Engineering process," *Reliability Engineering & System Safety*, vol. 111, pp. 171–182, 2013.

[136] J. A. Rivers, M. S. Gupta, J. Shin, P. N. Kudva, and P. Bose, "Error tolerance in server class processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 30, pp. 945–959, 2011.

[137] H. Elmiligi, A. A. Morgan, M. W. El-Kharashi, and F. Gebalis, "A reliability-aware design methodology for networks-on-chip applications," in *Design & Technology of Integrated Systems in Nanoscal Era, 2009. DTIS'09. 4th International Conference on*. IEEE, 2009, pp. 107–112.

[138] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 51–56.

[139] J. B. Bowles, "A survey of reliability-prediction procedures for microelectronic devices," *IEEE Transactions on Reliability*, vol. 41, no. 1, pp. 2–12, 1992.

[140] J. Shin, V. Zyuban, Z. Hu, J. A. Rivers, and P. Bose, "A framework for architecture-level lifetime reliability modeling," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 534–543.

[141] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A model of process variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, 2008.

[142] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: A methodology and some applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, 1990.

[143] K. N. Dang, M. Meyer, Y. Okuyama, and A. B. Abdallah, "Reliability Assessment and Quantitative Evaluation of Soft-Error Resilient 3D Network-on-Chip Systems," in *IEEE 25th Asian Test Symposium*. IEEE, 2016, pp. 161–166.

[144] T. Lehtonen, P. Liljeberg, and J. Plosila, "Fault tolerance analysis of NoC architectures," in *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 2007, pp. 361–364.

[145] H. Zimmer and A. Jantsch, "A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2003, pp. 188–193.

[146] J. Kim, D. Park, C. Nicopoulos, N. Vijaykrishnan, and C. R. Das, "Design and analysis of an noc architecture from performance, reliability and energy perspective," in *Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*. ACM, 2005, pp. 173–182.

[147] A. Savino, S. Di Carlo, G. Politano, A. Benso, A. Bosio, and G. Di Natale, "Statistical reliability estimation of microprocessor-based systems," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1521–1534, 2012.

[148] M. Khayambashi, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh, "Analytical reliability analysis of 3D NoC under TSV failure," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 11, no. 4, p. 43, 2015.

[149] A. Ben Ahmed and A. Ben Abdallah, "LA-XYZ: low latency, high throughput look-ahead routing algorithm for 3D network-on-chip (3D-NoC) architecture," in *IEEE 6th International Symposium on Embedded Multicore Socs (MCSoC)*. IEEE, September 2012, pp. 167–174.

[150] A. B. Ahmed and A. B. Abdallah, "Low-overhead Routing Algorithm for 3D Network-on-Chip," in *2012 Third International Conference on Networking and Computing (ICNC)*, December 2012, pp. 23–32.

[151] S. Pasricha and Y. Zou, "A low overhead fault tolerant routing scheme for 3D Networks-on-Chip," in *12th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2011, pp. 1–8.

[152] K. N. Dang, Y. Okuyama, and A. B. Abdallah1, "Soft-error resilient Network-on-Chip for safety-critical applications," in *2016 International Conference on IC Design and Technology (ICICDT)*, June 2016, pp. 1–4.

[153] K. N. Dang, M. Meyer, Y. Okuyama, and A. B. Abdallah, "A low-overhead soft–hard fault-tolerant architecture, design and management scheme for reliable high-performance many-core 3D-NoC systems," *The Journal of Supercomputing*, pp. 1–25, 2017.

[154] M.-Y. Hsiao, "A class of optimal minimum odd-weight-column sec-ded codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.

[155] NCSU Electronic Design Automation. FreePDK3D45 3D-IC process design kit. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK3D45:Contents

[156] NanGate Inc. Nangate Open Cell Library 45 nm. [Online]. Available: http://www.nangate.com/

[157] A. A. Chien and J. H. Kim, "Planar-adaptive routing: low-cost adaptive networks for multiprocessors," *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 91–123, 1995.

[158] R. Sivaram, "Queuing delays for uniform and nonuniform traffic patterns in a MIN," *ACM SIGSIM Simulation Digest*, vol. 22, no. 1, pp. 17–27, 1992.

[159] P. Chen, K. Dai, D. Wu, J. Rao, and X. Zou, "The parallel algorithm implementation of matrix multiplication based on ESCA," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2010, pp. 1091–1094.

[160] A. S. Zekri and S. G. Sedukhin, "The general matrix multiply-add operation on 2D torus," in *20th International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2006, pp. 8–16.

[161] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[162] A.-M. Rahmani, K. R. Vaddina, K. Latif, P. Liljeberg, J. Plosila, and H. Tenhunen, "High-performance and fault-tolerant 3D noc-bus hybrid architecture using arb-net-based adaptive monitoring platform," *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 734–747, 2014.

[163] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, 2005.

[164] K. Manna, S. Chattopadhyay, and I. Sengupta, "Through silicon via placement and mapping strategy for 3d mesh based network-on-chip," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2014, pp. 1–6.

[165] M. Tsai, A. Klooz, A. Leonard, J. Appel, and P. Franzon, "Through silicon via (TSV) defect/pinhole self test circuit for 3D-IC," in *IEEE International Conference on 3D System Integration*. IEEE, 2009, pp. 1–8.

[166] Y.-J. Huang and J.-F. Li, "Built-in self-repair scheme for the TSVs in 3-D ICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 10, pp. 1600–1613, 2012.

[167] Z. Qian and C. Y. Tsui, "A thermal-aware application specific routing algorithm for network-on-chip design," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, Jan 2011, pp. 449–454.

[168] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application specific routing algorithms for networks on chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 316–330, March 2009.

[169] Y. Ghidini, M. Moreira, L. Brahm, T. Webber, N. Calazans, and C. Marcon, "Lasio 3D NoC vertical links serialization: Evaluation of latency and buffer occupancy," in *26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Sept 2013, pp. 1–6.

[170] A. Bobbio and K. S. Trivedi, "An Aggregation Technique for the Transient Analysis of Stiff Markov Chains," *IEEE Transactions on Computers*, vol. C-35, no. 9, pp. 803–814, September 1986.

[171] R. W. Butler and S. C. Johnson, *Techniques for modeling the reliability of fault-tolerant systems with the Markov state-space approach*. NASA, 1995.

[172] J. A. Cheatham, J. M. Emmert, and S. Baumgart, "A survey of fault tolerant methodologies for FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 2, pp. 501–533, 2006.

[173] J. P. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 875–884, 1976.

[174] G. Van Rossum, "An introduction to Python for UNIX/C programmers," *Proc. of the NLUUG najaarsconferentie. Dutch UNIX users group*, 1993.

[175] M. Vasic, "Physical design of a 3d router: reducing the number of vertical connections and enabling asynchronous operation," Master's thesis, Delft University of Technology, 2014.

[176] Y. Tanaka, "Architecture and design of an efficient router for oasis 3d network-on-chip system," The University of Aizu, Tech. Rep., 2015. [Online]. Available: http://web-ext.u-aizu.ac.jp/~benab/publications/treport/YukiTanaka-TR2015.pdf

[177] A. B. Abdallah, M. Nakamura, A. Ben, M. M. Ahmed, and Y. Okuyama, "Fault-tolerant router for highly-reliable many-core 3d-noc systems," in *The 3rd International Scientific Conference on Engineering and Applied Sciences (ISCEAS 2015)*, 2015.

[178] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media, 2012.

[179] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration," in *Proceedings of the conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 423–428.

[180] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 33–42.

[181] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 469–480.

[182] E. J. Marinissen, "Testing TSV-based three-dimensional stacked ICs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 1689–1694.

[183] ——, "Challenges and emerging solutions in testing TSV-based 2 1/2D-and 3D-stacked ICs," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 1277–1282.

A Benchmarks

This appendix depicts the details of the used benchmarks in this dissertation. The list and details of the benchmarks are described in Table A.1.

Table A.1: Description of benchmarks.

Benchmark	Description	Citation
Transpose	Each node (a,b,c) in a network with (X,Y,Z) sends packets to node (X-a, Y-b, Z-c). See Algorithm A.1 for details.	[157]
Uniform	Each node in a network sends packets to all nodes. See Algorithm A.2 for details.	[158]
Matrix-multiplication	Performs C=A*B. Matrix A is stored in layer-1, is sent to layer-2 which has matrix B. The final values are accumulated in layer-3 as matrix C. See Algorithm A.3 for details.	[159, 160]
Hotspot 10%	Each node in a network sends packets to all nodes. X (X=1 or 2 or more) nodes have additional 10% amount of traffic. See Algorithm A.4 for details.	[161]
Realistic Traffic Pattern	Generate from task graphs which provide the connections (e.g: node A→B) and the traffic (e.g: 100 packets). See Algorithm A.5 for details.	[162, 163]

A.1 Synthetic Traffic

Algorithm A.1 shows the pseudo-code of transpose benchmark. For each node, its data is sent to the reversed index node.

Algorithm A.2 shows the pseudo-code of uniform benchmark. For each node, its data is sent to every nodes in the network.

Algorithm A.3 shows the pseudo-code of matrix benchmark. Matrix A is stored in one layer and its values are sent to another layer which consists of Matrix B. In the Matrix-B layer, data is multiplied and sent to the final layer where data are accumulated to obtain the final value.

Algorithm A.4 shows the pseudo-code of hotspot benchmark. The hostpot benchmark is similar to uniform benchmark but the hotspot nodes will have extra percentages of data.

Algorithm A.1: Transpose Algorithm.

```
   // Network
   Input: Network(X, Y, Z)
   // Amount of data for each communication
   Input: D
   // Communication set
   Output: C = {c_i : (source → destination, amount of data)}
1  foreach node (a,b,c) in Network(X, Y, Z) do
2      add ((a, b, c) → (X − a − 1, Y − b − 1, Z − c − 1), D packets) to C
3  return C
```

Algorithm A.2: Uniform Algorithm.

```
   // Network
   Input: Network(X, Y, Z)
   // Amount of data for each communication
   Input: D
   // Communication set
   Output: C = {c_i : (source → destination, amount of data)}
1  foreach node (a,b,c) in Network(X, Y, Z) do
2      foreach node (m,n,p) in Network(X, Y, Z) do
3          add ((a, b, c) → (m, n, p), D packets) to C
4  return C
```

A.2 Realistic Traffic Pattern

The realistic traffic pattern benchmark requires task graph and task map in order to perform. Figure A.1 and A.2 show the task graph and task map for H.264 encoder. Figure A.3, A.4, and A.5 show the task graph and task map of VOPD, MWD and PIP, respectively.

Algorithm A.3: Matrix-multiplication Algorithm.

Input: $layerA(n,n), layerB(n,n), layerC(n,n)$,
Input: $A(n,n), B(n,n)$
Output: $C(n,n)$

```
1  foreach node (i,j) in layerA(n,n) do
2      send A(i,j) → layerB(j,i)
3  foreach node (i,j) in layerB(n,n) do
4      receive A(j,i)
5      R(i,j) = A(j,i) × B(i,j)
6      foreach k in 1:n do
7          send R(i,j) → layerC(i,k)
8  foreach node (i,j) in layerC(n,n) do
9      foreach k in 1:n do
10         send C(i,j) = C(i,j) + R(k,i)
11 return C(n,n) from layerC(n,n)
```

Algorithm A.4: Hotspot Algorithm.

```
// Network
Input: Network(X,Y,Z)
// Amount of data for each communication
Input: D
// Extra percentage of hotspot node
Input: E
// Communication set
Output: C = {c_i : (source → destination, amount of data)}
1  foreach node (a,b,c) in Network(X,Y,Z) do
2      foreach node (m,n,p) in Network(X,Y,Z) do
3          if node (m,n,p) is hotspot node then
4              add ((a,b,c) → (m,n,p), (D+D*E/100) packets) to C
5          else
6              add ((a,b,c) → (m,n,p), D packets) to C
7  return C
```

Algorithm A.5: Realistic Benchmark Algorithm.

```
Input: Network(X,Y,Z)
// Communication set
Input: C = {c_i : (source → destination, D, O)}
1  ProgramCounter = 0;
2  foreach node (i,j,k) in Network(X,Y,Z) do
3      foreach c_i in C do
4          if c_i(source) == (i,j,k) and ProgramCounter == O then
5              send (i,j,k) → c_i(destination) with c_i(D) packets.
6          if c_i(destination) == (i,j,k) and ProgramCounter == O then
7              receive c_i(D) packets.
8  if all destinations completedly receive their own c_i(D) packets then
9      ProgramCounter++;
```

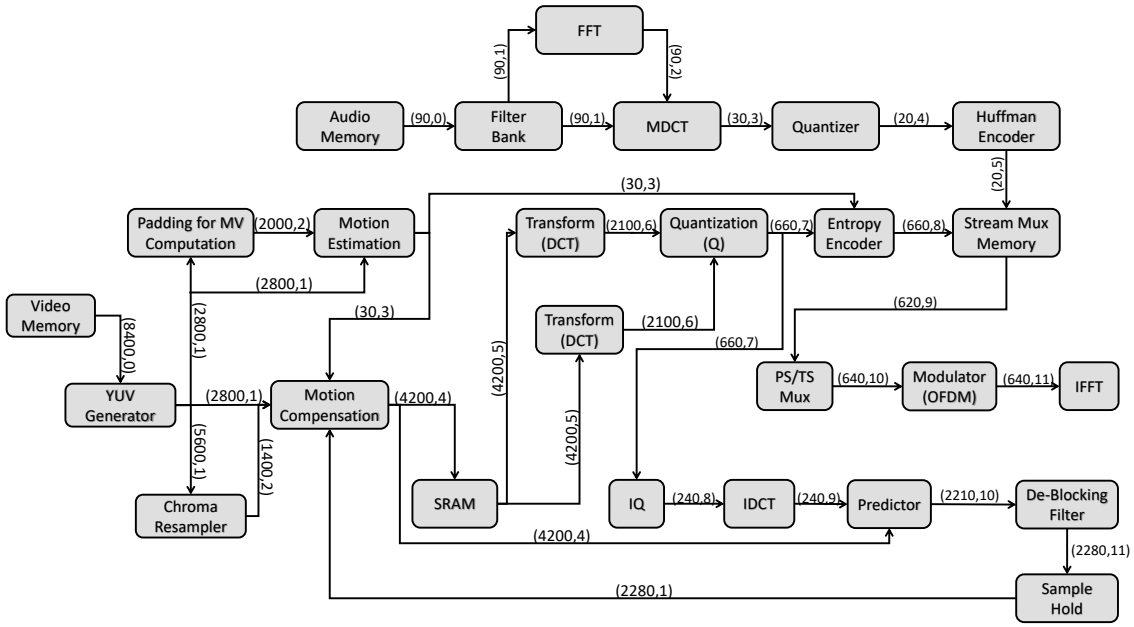

Figure A.1: H.264 Task Graph.

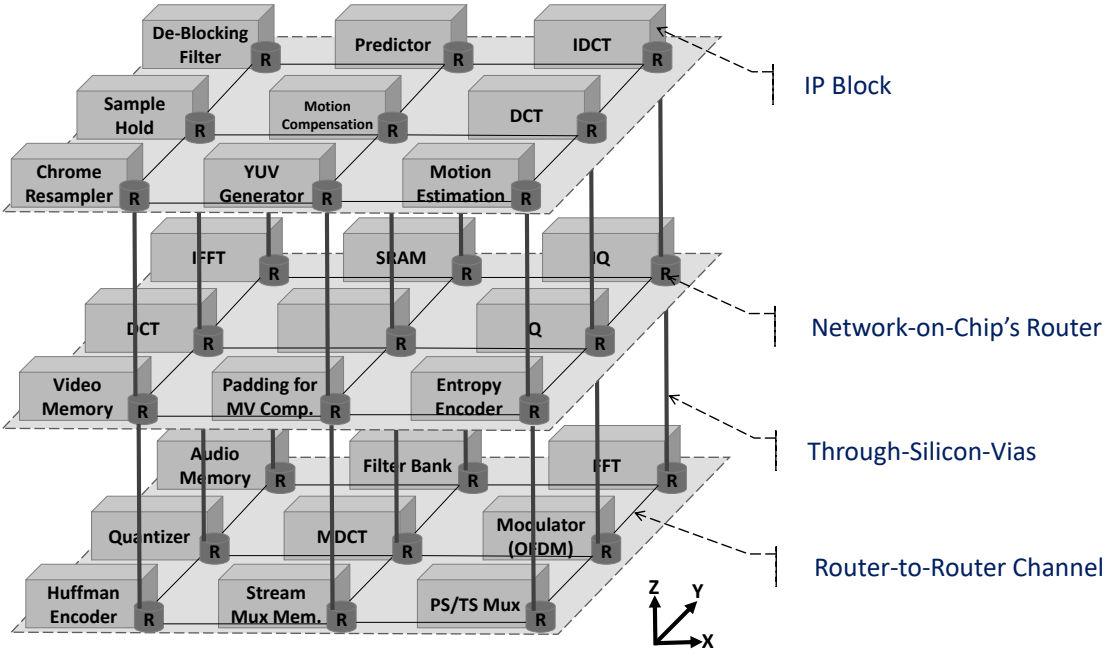

Figure A.2: H.264 Task Map.

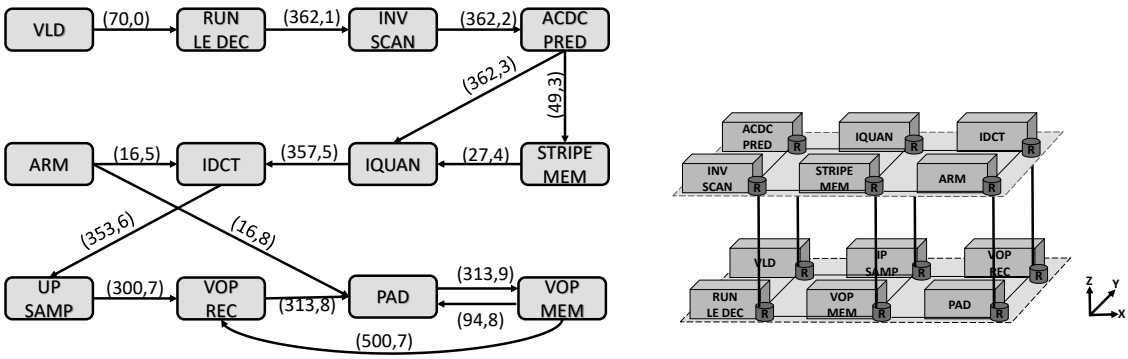

Figure A.3: VOPD Task Map.

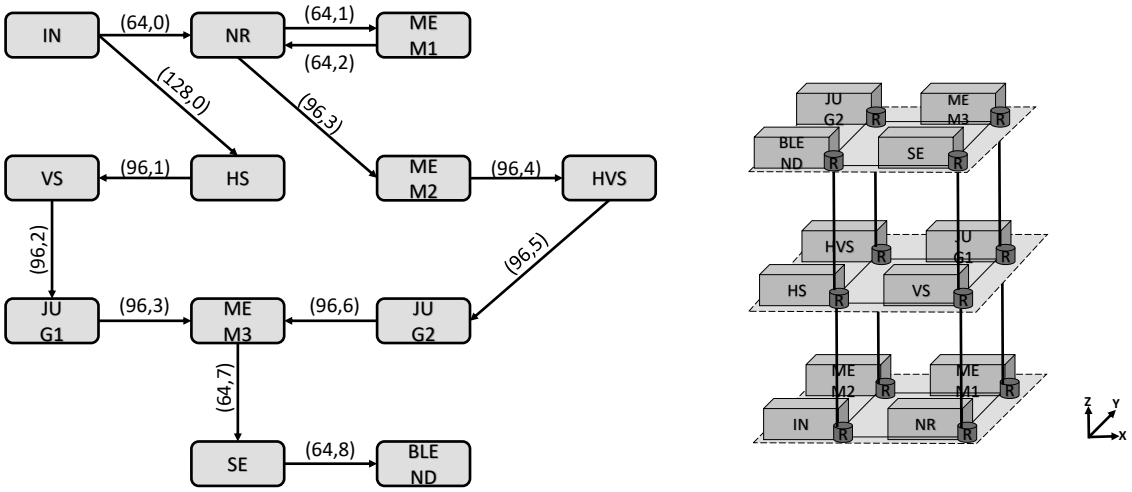

Figure A.4: MWD Task Map.

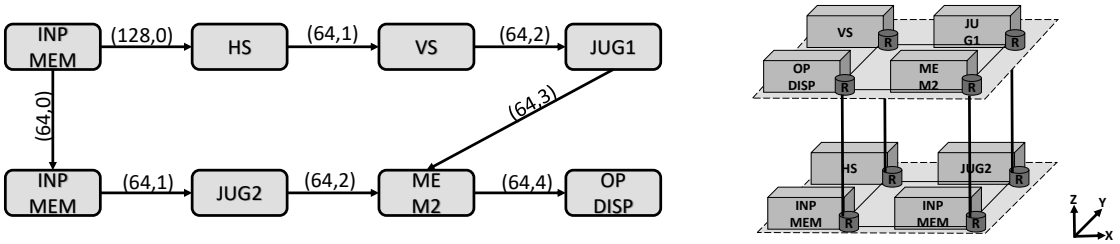

Figure A.5: PIP Task Map.

B
TSV Router

The following is the source code of TSV router.

```
`ifndef VCS
 `include "defines.v"
`endif

module router_TSV (

  input                                     clk,
  input                                     reset,
  input [`L2NET_SIZE-1:0]                   xaddr,
  input [`L2NET_SIZE-1:0]                   yaddr,
  input [`L2NET_SIZE-1:0]                   zaddr,

  input [(`WIDTH*`NOUT)-1:0]                data_in,
  input [(36)-1:0]                          faulty_in,
  input [`NOUT-1:0]                         stop_in,
  input [`NOUT-1:0]                         arq_in,

  input [2:0]                               UP_serial_part_in,
  input [2:0]                               DOWN_serial_part_in,
  input [1:0]                               UP_sync_bits_in,
  input [1:0]                               DOWN_sync_bits_in,

  output [2:0]                              UP_serial_part_out,
  output [2:0]                              DOWN_serial_part_out,
  output [1:0]                              UP_sync_bits_out,
  output [1:0]                              DOWN_sync_bits_out,

  input [`NOUT-1:0]                         f_RAB,
  input [`NOUT-1:0]                         f_XBAR,
  input [`NOUT-1:0]                         f_LINK,
  input                                     f_trigger,
  input                                     f_reset,
  input  [2*`TSV_CLUSTER-1:0]               f_TSV,

  output [`DEBUG_WIDTH+3*`L2NET_SIZE-1:0]   f_flag,

  input                                     EnableUpdate,
  input  [`WEIGHT*`TSV_CLUSTER-1:0]         NeighborWeight,
  input  [2*`TSV_CLUSTER-1:0]               NeighborTSVStatus_Up,
  input  [`TSV_CLUSTER-1:0]                 NeighborRequestLink_Up,
  input  [2*`TSV_CLUSTER-1:0]               NeighborTSVStatus_Down,
```

```
42  input  [`TSV_CLUSTER-1:0]                    NeighborRequestLink_Down,
43
44  input  [2*`TSV_CLUSTER-1:0]                  NeighborTSVVirtualState_Up,
45  input  [`TSV_CLUSTER-1:0]                    NeighborRequestVirtual_Up,
46  input  [2*`TSV_CLUSTER-1:0]                  NeighborTSVVirtualState_Down,
47  input  [`TSV_CLUSTER-1:0]                    NeighborRequestVirtual_Down,
48
49  output  [`WIDTH-1:0]                         Current2Neighbor_up_in,
50  output  [`WIDTH-1:0]                         Current2Neighbor_down_in,
51  output  [`WIDTH-1:0]                         Current2Neighbor_up_out,
52  output  [`WIDTH-1:0]                         Current2Neighbor_down_out,
53
54  input  [`WIDTH-1:0]                          Neighbor2Current_up_in,
55  input  [`WIDTH-1:0]                          Neighbor2Current_down_in,
56  input  [`WIDTH-1:0]                          Neighbor2Current_up_out,
57  input  [`WIDTH-1:0]                          Neighbor2Current_down_out,
58
59  output [`WEIGHT-1:0]                         CurrentWeight,
60  output [2*`TSV_CLUSTER-1:0]                  CurrentStatus_Up,
61  output [`TSV_CLUSTER-1:0]                    RequestLink_Up,
62  output [2*`TSV_CLUSTER-1:0]                  CurrentStatus_Down,
63  output [`TSV_CLUSTER-1:0]                    RequestLink_Down,
64
65  input  [2*`TSV_CLUSTER-1:0]                  NeighborCStatus_Up,
66  input  [2*`TSV_CLUSTER-1:0]                  NeighborCStatus_Down,
67  output [2*`TSV_CLUSTER-1:0]                  CurrentCStatus_Up,
68  output [2*`TSV_CLUSTER-1:0]                  CurrentCStatus_Down,
69
70  output [`TSV_CLUSTER-1:0]                    RequestVirtual_Up,
71  output [2*`TSV_CLUSTER-1:0]                  TSVVirtualState_Up,
72  output [`TSV_CLUSTER-1:0]                    RequestVirtual_Down,
73  output [2*`TSV_CLUSTER-1:0]                  TSVVirtualState_Down,
74
75  input  [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_up_sync_in,
76  output [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_up_sync_out,
77  input  [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_down_sync_in,
78  output [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_down_sync_out,
79
80  output [(`WIDTH*`NOUT)-1:0]                  data_out,
81  output [6*`NOUT-1:0]                         faulty_out,
82  output [`NOUT-1:0]                           stop_out,
83  output [`NOUT-1:0]                           arq_out);
84
85  wire   [(`WIDTH*`NOUT)-1:0]                  curr_data_in;
86  wire   [(36)-1:0]                            curr_faulty_in;
87  wire   [`NOUT-1:0]                           curr_stop_in;
88  wire   [`NOUT-1:0]                           curr_arq_in;
89
90  wire   [(`WIDTH*`NOUT)-1:0]                  curr_data_out;
91  wire   [6*`NOUT-1:0]                         curr_faulty_out;
92  wire   [`NOUT-1:0]                           curr_stop_out;
93  wire   [`NOUT-1:0]                           curr_arq_out;
94
95  wire   [(`WIDTH)-1:0]                        tsv_data_up_in_I,tsv_data_up_in_O;
96  wire   [(`WIDTH)-1:0]                        tsv_data_down_in_I,tsv_data_down_in_O;
97  wire   [(`WIDTH)-1:0]                        tsv_data_up_out_I,tsv_data_up_out_O;
98  wire   [(`WIDTH)-1:0]                        tsv_data_down_out_I,tsv_data_down_out_O;
99
100 wire   [`WEIGHT-1:0]                         CurrentWeightX;
101
102 wire   [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_up_in;
103 wire   [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_up_out;
104 wire   [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_down_in;
105 wire   [(`TSV_CLUSTER+2)*`TSV_CLUSTER-1:0]   config_down_out;
106
107 wire                                         TSVinUsage_Up, TSVinUsage_Down;
108 wire                                         TSVReq_Up, TSVReq_Down;
109 wire                                         UpGrant, DownGrant;
110
```

```
111  wire   [1:0]                                UpConnectionStatus;
112  wire   [1:0]                                DownConnectionStatus;
113
114  assign CurrentCStatus_Up   = UpConnectionStatus;
115  assign CurrentCStatus_Down = DownConnectionStatus;
116
117  assign config_up_sync_out = config_up_out;
118  assign config_up_in       = config_up_sync_in;
119
120  router
121  router_ori (
122    .clk                  (clk                ),
123    .reset                (reset              ),
124    .data_in              (curr_data_in       ),
125    .data_out             (curr_data_out      ),
126    .stop_in              (curr_stop_in       ),
127    .stop_out             (curr_stop_out      ),
128    .faulty_in            (curr_faulty_in     ),
129    .faulty_out           (curr_faulty_out    ),
130    .f_RAB                (f_RAB              ),
131    .f_XBAR               (f_XBAR             ),
132    .f_LINK               (f_LINK             ),
133    .f_trigger            (f_trigger          ),
134    .f_flag               (f_flag             ),
135    .f_reset              (f_reset            ),
136    .arq_out              (curr_arq_out       ),
137    .arq_in               (curr_arq_in        ),
138    .UP_serial_part_in    (UP_serial_part_in  ),
139    .UP_sync_bits_in      (UP_sync_bits_in    ),
140    .DOWN_serial_part_in  (DOWN_serial_part_in ),
141    .DOWN_sync_bits_in    (DOWN_sync_bits_in  ),
142    .UP_serial_part_out   (UP_serial_part_out ),
143    .UP_sync_bits_out     (UP_sync_bits_out   ),
144    .DOWN_serial_part_out (DOWN_serial_part_out),
145    .DOWN_sync_bits_out   (DOWN_sync_bits_out ),
146    .UpReq                (TSVReq_Up          ),
147    .DownReq              (TSVReq_Down        ),
148    .UpGrant              (UpGrant            ),
149    .DownGrant            (DownGrant          ),
150    .UpInUsage            (TSVinUsage_Up      ),
151    .DownInUsage          (TSVinUsage_Down    ),
152    .UpConnectionStatus   (UpConnectionStatus ),
153    .DownConnectionStatus (DownConnectionStatus),
154    .xaddr                (xaddr              ),
155    .yaddr                (yaddr              ),
156    .zaddr                (zaddr              ));
157
158  TSV_cntrl TSV_cntrl_up(
159    .clk                     (clk                       ),
160    .reset                   (reset                     ),
161    .xaddr                   (xaddr                     ),
162    .yaddr                   (yaddr                     ),
163    .zaddr                   (zaddr                     ),
164    .EnableUpdate            (EnableUpdate              ),
165    .f_TSV                   (f_TSV[`TSV_CLUSTER-1:0]   ),
166    .NeighborWeight          (NeighborWeight            ),
167    .NeighborRequestLink     (NeighborRequestLink_Up    ),
168    .NeighborTSVStatus       (NeighborTSVStatus_Up      ),
169    .NeighborCStatus         (NeighborCStatus_Up        ),
170    .CurrentWeight           (CurrentWeight             ),
171    .CurrentStatus           (CurrentStatus_Up          ),
172    .RequestLink             (RequestLink_Up            ),
173    .ConnectionStatus        (UpConnectionStatus        ),
174    .RequestVirtual          (RequestVirtual_Up         ),
175    .NextTSVVirtualState     (TSVVirtualState_Up        ),
176    .NeighborTSVVirtualState (NeighborTSVVirtualState_Up),
177    .NeighborRequestVirtual  (NeighborRequestVirtual_Up ),
178    .serial_part             (UP_serial_part_out        ),
179    .TSVinUsage              (TSVinUsage_Up             ),
```

```
  .TSVReq                  (TSVReq_Up                ),
  .TSVGrant                (UpGrant                  ),
  .config_out              (config_up_out            ));

TSV_cntrl TSV_cntrl_down(
  .clk                     (clk                                ),
  .reset                   (reset                              ),
  .xaddr                   (xaddr                              ),
  .yaddr                   (yaddr                              ),
  .zaddr                   (zaddr                              ),
  .EnableUpdate            (EnableUpdate                       ),
  .f_TSV                   (f_TSV[2*`TSV_CLUSTER-1:`TSV_CLUSTER]),
  .NeighborWeight          (NeighborWeight                     ),
  .NeighborRequestLink     (NeighborRequestLink_Down           ),
  .NeighborTSVStatus       (NeighborTSVStatus_Down             ),
  .CurrentWeight           (CurrentWeightX                     ),
  .NeighborCStatus         (NeighborCStatus_Down               ),
  .CurrentStatus           (CurrentStatus_Down                 ),
  .RequestLink             (RequestLink_Down                   ),
  .ConnectionStatus        (DownConnectionStatus               ),
  .RequestVirtual          (RequestVirtual_Down                ),
  .NextTSVVirtualState     (TSVVirtualState_Down               ),
  .NeighborTSVVirtualState (NeighborTSVVirtualState_Down       ),
  .NeighborRequestVirtual  (NeighborRequestVirtual_Down        ),
  .TSVinUsage              (TSVinUsage_Down                    ),
  .serial_part             (DOWN_serial_part_out               ),
  .TSVReq                  (TSVReq_Down                        ),
  .TSVGrant                (DownGrant                          ),
  .config_out              (config_down_out                    ));

assign config_down_sync_out = config_down_out;
assign config_down_in       = config_down_sync_in;

assign curr_data_in[(`WIDTH*5-1):0] = data_in[(`WIDTH*5-1):0];
assign curr_faulty_in               = faulty_in;
assign curr_stop_in                 = stop_in;
assign curr_arq_in                  = arq_in;

assign data_out[(`WIDTH*5-1):0] = curr_data_out[(`WIDTH*5-1):0];
assign faulty_out               = curr_faulty_out;
assign stop_out                 = curr_stop_out;
assign arq_out                  = curr_arq_out;

genvar x,y;
generate for (x=0; x<`WIDTH; x=x+1) begin:UD_port
  TSV tsv_input_up    (.i(tsv_data_up_in_I[x]), .o(tsv_data_up_in_O[x]));
  TSV tsv_input_down  (.i(tsv_data_down_in_I[x]), .o(tsv_data_down_in_O[x]));
  TSV tsv_output_up   (.i(tsv_data_up_out_I[x]), .o(tsv_data_up_out_O[x]));
  TSV tsv_output_down (.i(tsv_data_down_out_I[x]), .o(tsv_data_down_out_O[x]));
end
endgenerate

assign tsv_data_up_in_I   = data_in[`WIDTH*6-1:`WIDTH*5];
assign tsv_data_down_in_I = data_in[`WIDTH*7-1:`WIDTH*6];

assign data_out[`WIDTH*6-1:`WIDTH*5] = tsv_data_up_out_O;
assign data_out[`WIDTH*7-1:`WIDTH*6] = tsv_data_down_out_O;

genvar cluster;
generate for (cluster = 0; cluster <`TSV_CLUSTER; cluster=cluster+1) begin: TSV_CLUSTER_CONFIG
  tristate_gates_in up_in (
    .config_reg       (config_up_in[(cluster+1)*(`TSV_CLUSTER+2)-1:cluster*(`TSV_CLUSTER+2)]      ),
    .datfTSV          (tsv_data_up_in_O[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)]              ),
    .datfNeigh        (Neighbor2Current_up_in                                                     ),
    .current_dat_out  (curr_data_in[(`WIDTH*5+(cluster+1)*`WIDTH/4-1):(`WIDTH*5+cluster*`WIDTH/4)]),
    .neighbor_dat_out (Current2Neighbor_up_in[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)]        )
    );
  tristate_gates_in down_in (
    .config_reg       (config_down_in[(cluster+1)*(`TSV_CLUSTER+2)-1:cluster*(`TSV_CLUSTER+2)]    ),
```

```
      .datfTSV          (tsv_data_down_in_O[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)]           ),
      .datfNeigh        (Neighbor2Current_down_in                                                  ),
      .current_dat_out  (curr_data_in[(`WIDTH*6+(cluster+1)*`WIDTH/4-1):(`WIDTH*6+cluster*`WIDTH/4)]),
      .neighbor_dat_out (Current2Neighbor_down_in[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)]     )
      );
    tristate_gates_out  #(cluster) up_out (
      .config_reg       (config_up_out[(cluster+1)*(`TSV_CLUSTER+2)-1:cluster*(`TSV_CLUSTER+2)]     ),
      .current_dat_in   (curr_data_out[(`WIDTH*5+(cluster+1)*`WIDTH/4-1):(`WIDTH*5+cluster*`WIDTH/4)]),
      .neighbor_dat_in  (Neighbor2Current_up_out[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)]       ),
      .dat2Neigh        (Current2Neighbor_up_out                                                   ),
      .dat2TSV          (tsv_data_up_out_I[`WIDTH*(cluster+1)/4-1:`WIDTH*cluster/4]                )
      );
    tristate_gates_out  #(cluster) down_out (
      .config_reg       (config_down_out[(cluster+1)*(`TSV_CLUSTER+2)-1:cluster*(`TSV_CLUSTER+2)]   ),
      .current_dat_in   (curr_data_out[(`WIDTH*6+(cluster+1)*`WIDTH/4-1):(`WIDTH*6+cluster*`WIDTH/4)]),
      .neighbor_dat_in  (Neighbor2Current_down_out[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)]     ),
      .dat2Neigh        (Current2Neighbor_down_out                                                 ),
      .dat2TSV          (tsv_data_down_out_I[`WIDTH*(cluster+1)/4-1:`WIDTH*cluster/4]              )
      );

    assign Current2Neighbor_up_out[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)] = (config_up_out[cluster
     +2+0*(`TSV_CLUSTER+2)] == 1'b0 &&
      config_up_out[cluster+2+1*(`TSV_CLUSTER+2)] == 1'b0 && config_up_out[cluster+2+2*(`TSV_CLUSTER+2)]
     == 1'b0 &&  config_up_out[cluster+2+3*(`TSV_CLUSTER+2)] == 1'b0)? 0:{`WIDTH/4{1'bZ}};
    assign Current2Neighbor_down_out[((cluster+1)*`WIDTH/4-1):(cluster*`WIDTH/4)] = (config_down_out[
     cluster+2+0*(`TSV_CLUSTER+2)] == 1'b0 &&
      config_down_out[cluster+2+1*(`TSV_CLUSTER+2)] == 1'b0 && config_down_out[cluster+2+2*(`TSV_CLUSTER
     +2)] == 1'b0 &&  config_down_out[cluster+2+3*(`TSV_CLUSTER+2)] == 1'b0)? 0:{`WIDTH/4{1'bZ}};
  end
  endgenerate
endmodule
```

Listing B.1: TSV Router.