THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Reconfigurable NoC and Processors Tolerant to Permanent Faults

ALIRAD MALEK

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

Reconfigurable NoC and Processors Tolerant to Permanent Faults
Alirad Malek

Technical report 129L
ISSN 1652-876X
Department of Computer Science and Engineering

Division of Computer Engineering
Chalmers University of Technology
SE-412 96 GÖTEBORG, Sweden
Phone: +46 (0)31-772 10 00

Author e-mail: `aliradm@chalmers.se`

Printed by Chalmers Reproservice
GÖTEBORG, Sweden 2015

Reconfigurable NoC and Processors Tolerant to Permanent Faults

Alirad Malek

Division of Computer Engineering, Chalmers University of Technology

ABSTRACT

Advances in semiconductor industry have led to reduced transistor dimensions and increased device density, but inevitably they have compromised the reliability of modern computing systems. In this thesis, we address the reliability problem by exploiting hardware reconfiguration for tolerating permanent faults. Processing components in a system-on-chip are divided into smaller Substitutable Units (SUs) and reconfigurable interconnects are used to isolate defective SUs and connect spare units to create a fault-free component. Furthermore, employing fine-grain logic for instantiating a functionally equivalent unit is another reconfiguration option considered. Based on these approaches, the first part of this thesis presents a probabilistic analysis of reconfigurable designs for calculating the average number of constructable components at different fault densities. Considering the area overheads of reconfigurability, we evaluate the resilience of various reconfigurable designs with different granularities (SU sizes). Concisely, the results reveal that the combination of fine and coarse-grain reconfiguration offers up to 3× more fault-tolerance compared to component redundancy. Performing a design-space exploration to find the most efficient granularity mix shows that different fault densities require different granularities of substitutable units to maximize fault-tolerance. Moreover, we explored the performance effects of pipelining the reconfigurable interconnects in adaptive processors and observed that the operating frequency and execution time of pipelined design is roughly 2.5× and 2× better than the design with non-pipelined interconnects, respectively. In the second part of this thesis, we describe RQNoC, a service-oriented Network-on-Chip (NoC) resilient to permanent faults. We characterize the network resources based on the particular service they support and, when faulty, bypass them allowing the respective traffic class to be redirected. We propose service merging (SMerge) and service detouring (SDetour) as the two service redirection schemes. Different RQNoC configurations are implemented and evaluated in terms of performance, area, power consumption and fault tolerance. Concisely, the evaluation results show that compared to the baseline network, SMerge requires 51% more area and 27% more power and has a 9% slower clock but maintains at least 90% of the network connectivity even in presence of 32 permanent network faults.

Keywords: Reconfigurable Hardware, Fault Tolerance, Permanent Faults, Networks-on-Chip, Adaptive processors

Acknowledgments

This thesis marks another checkpoint in my life. It seems an appropriate place and time for me to express my gratitude and appreciation to all those people who have supported me during these years.

First of all, I would like to express my deepest gratitude to my advisor Yiannis Sourdis for his generous guidance and inspiring patience. Thank you for giving me the opportunity to work with you.

I would sincerely appreciate Georgi Gaydadjiev, Per Stenström and Philippas Tsigas for their helpful suggestions during follow-up meetings and valuable inputs. Special thanks to Sally McKee and Lars Svensson for their kind support and guidance during different phases of my research.

I would also like to thank professor Axel Jantsch for accepting our invitation to be the discussion leader for the presentation of this thesis.

Next, I like to thank all my friends and colleagues in the Department of Computer Science and Engineering: Risat, Vasileios, Miquel, Sven, Kasyab, Angelos, Negin, Bhavishya, Jacob, Madhavan, Behrooz, Fatemeh, Chloe, Petros, Alen, Dmitry, Pierre, Michal and anyone who, one way or another, helped me during this time. It is a great pleasure knowing you guys. In particular I would like to thank my great office mates Stavros and Ahsen for the pleasant working environment and their kind-hearted support during my research. I learned a lot from you guys.

I would like to express my sincere gratitude to our kind administrative staff in the department, Eva Axelsson, Tiina Rankanen, Marianne Pleen-Schreiber, Peter Helander and Rune Ljungbjörn for their help and support.

This work has been partly funded by the European Union's 7^{th} Framework Programme, under project DeSyRe: on-Demand System Reliability. I am very grateful to all our partners in the DeSyRe projects, specially our colleagues in Erasmus Medical Center: Christos Strydis, Georgios Smaragdos, Robert Seepers, Sebastian Isaza and in Recore systems: Gerard Rauwerda and Yifan He.

Many thanks to my friends Kamyar, Hessam, Dorreh, Nojan, Behnaz, Homayoun, Mitra, Mohammad and all other friends having whom is a blessing for me.

I would like to thank my parents and my sisters for their limitless support and chances they've given me over the years. Thanks Mom. Thanks Dad.

Finally I would like to thank Naghmeh for her love and patience over all these years and the joy she brings to my life. None of this would have been possible without you. Thank you.

Alirad Malek
Göteborg, April 2015

Preface

This dissertation is for the degree of Licentiate of engineering. The Licentiate degree is a Swedish degree halfway between the Master of Science (MSc) and Doctor of Philosophy(Ph.D.).

Parts of the contributions presented in this thesis have previously been published in the following manuscripts.

▷ **Alirad Malek**, Stavros Tzilis, Danish Anis Khan, Ioannis Sourdis, Georgios Smaragdos, Christos Strydis, "A probabilistic analysis of resilient reconfigurable designs", accepted in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, October, 2014, pp.141-146.

▷ Ioannis Sourdis, Danish Anis Khan, **Alirad Malek**, Stavros Tzilis, Georgios Smaragdos, Christos Strydis, "Resilient CMPs with Mixed-grain Reconfigurability", accepted in IEEE Micro, January, 2015

▷ Georgios Smaragdos, Danish Anis Khan, Ioannis Sourdis, Christos Strydis, **Alirad Malek**, Stavros Tzilis, "A Dependable Coarse-grain Reconfigurable Multicore Array", in 21st Reconfigurable Architectures Workshop (RAW'14), 2014

Parts of the contributions presented in this thesis are included in the following manuscripts that are under review.

▷ **Alirad Malek**, Ioannis Sourdis, Stavros Tzilis, Yifan He, Gerard Rauwerda, "RQNoC: a Resilient QoS NoC with Service Redirection", submitted, 2015

▷ Georgios Smaragdos, Danish Anis Khan, **Alirad Malek**, Stavros Tzilis, Ioannis Sourdis, Christos Strydis, "FlexPipes: A Fault-Tolerant CMP architecture based on resource sharing", submitted, 2015

Other publication not directly related to this thesis are as follows.

▷ I. Sourdis, C. Strydis, A. Armato, C.S. Bouganis, B. Falsafi, G.N. Gaydadjiev, S. Isaza, **A. Malek**, R. Mariani, D.K. Pradhan, G. Rauwerda, R.M. Seepers, R.A. Shafik, G. Smaragdos, D. Theodoropoulos, S. Tzilis, M. Vavouras, S. Pagliarini, and D. Pnevmatikatos, "DeSyRe: On-Demand Adaptive and Reconfigurable Fault-Tolerant

SoCs ", accepted in 10th Int'l Symp. on Applied Reconfigurable Computing (ARC), 2014

- ⊳ I. Sourdis, C. Strydis, A. Armato, C.-S. Bouganis, B. Falsafi, G. Gaydadjiev, S. Isaza, **A. Malek**, R. Mariani, D.N. Pnevmatikatos, D.K Pradhan, G. Rauwerda, R. Seepers, R.K. Shafik, K. Sunesen, D. Theodoropoulos, S. Tzilis, M. Vavouras, "DeSyRe: on-Demand System Reliability", accepted in Elsevier Microprocessors and Microsystems, Special Issue on European Projects in Embedded System Design, November 2013
- ⊳ I. Sourdis, C. Strydis, C.-S. Bouganis, B. Falsafi, G.Gaydadjiev, **Alirad Malek**, R. Mariani, D. Pnevmatikatos, D.K. Pradhan, G. Rauwerda, K. Sunesen, S. Tzilis, "The DeSyRe project: on-Demand System Reliability ", accepted in 15th EUROMICRO Conference on Digital System Design (DSD), September 2012

Contents

List of Figures

1 Introduction

After many decades, Moore's law still holds as semiconductor industries are still able to deliver 2× of devices on a chip in every new technology generation, emerging almost in every 18 months. Aggressive scaling of transistor size to increase the device density has been a prominent factor in designs of recent computing system such as Multi-core and Many-core systems. On the other hand, shrinking device dimensions, hinders the overall reliability of systems [5]. Digital circuits are becoming more susceptible to manufacturing defects and more vulnerable to wear-out phenomena which will manifest themselves as intermittent or in more severe cases as permanent faults. This issue urges the need for new designs and approaches to improve yield and systems fault-free lifetime. In addition, considering the advances of embedded systems in mission-critical domains such as medical and automotive, the reliability issue has become an ever growing challenge for designers and needs to be addressed with various techniques. In general, modern system-on-chips (SoCs) can be equipped with various fault-tolerant techniques at different levels, e.g., processors and interconnects architecture, which altogether aim to increase the overall reliability of the system.

This introductory chapter presents a brief description of problems in hand and gives an overview of the works presented in this thesis. The remainder of this chapter is organized as follows: Section 1.1 describes hardware reconfiguration as a prevalent method for tolerating permanent faults. Section 1.2 succinctly defines the problems addressed in this thesis work. Section 1.3 describes the objectives and contributions of this thesis work and finally section 1.4 provides an outline for the rest of this manuscript.

1.1 Hardware Reconfiguration

An undesirable consequence of technology scaling is degradation of reliability. In modern system-on-chips comprising of many components, e.g., processing elements, memory units and interconnects, a single fault might hinder the correct functionality of the chip for a short time or permanently. This section gives an overview of various classes of faults in digital circuits, ways to distinguish between different classes and how to cope with each of them. Furthermore, hardware reconfiguration is introduced as a technique for tolerating permanent faults. Finally, hardware reconfiguration is briefly explained in the context of the research efforts presented in this thesis.

In a conventionally used categorization, faults are classified into *transient*, *intermittent* and *permanent* faults [6]. Transient faults are most commonly distinguished by their randomness and short-lifetime. Prevalent ways to cope with transient faults are temporal and spatial redundancy techniques. An intermittent fault, by definition, is a fault which repeats at intervals, normally causes burst errors and can be fixed by replacing the corrupted device [35]. Similar to transient faults, temporal and spatial redundancy can be used to tolerate this class of faults. Intermittent faults are often related to wear-out phenomena and might eventually lead to a permanent fault. The main causes of permanent faults are either manufacturing defects or aging. These faults are mainly addressed by using spatial redundancy [35].

In addition to common redundancy schemes, hardware reconfiguration can be employed to tolerate permanent faults, by either repairing the damaged component or by isolating and replacing them with correctly functioning ones. Hardware components can be equipped with interconnects and glue logic which makes reconfiguration feasible. Reconfiguration at coarse-grain (CG) level is normally employed by either exploiting the inherent redundancy inside a component or by introducing spares. On the other hand, it is possible to have fine-grain (FG) reconfiguration by introducing an FPGA-like reconfigurable fabric/substrate which offers more flexibility, i.e., any desired substitutable unit can be instantiated using a pre-designed and stored bit-stream.

In this thesis, we first focus on adaptive processors in a chip multi-processor (CMP)

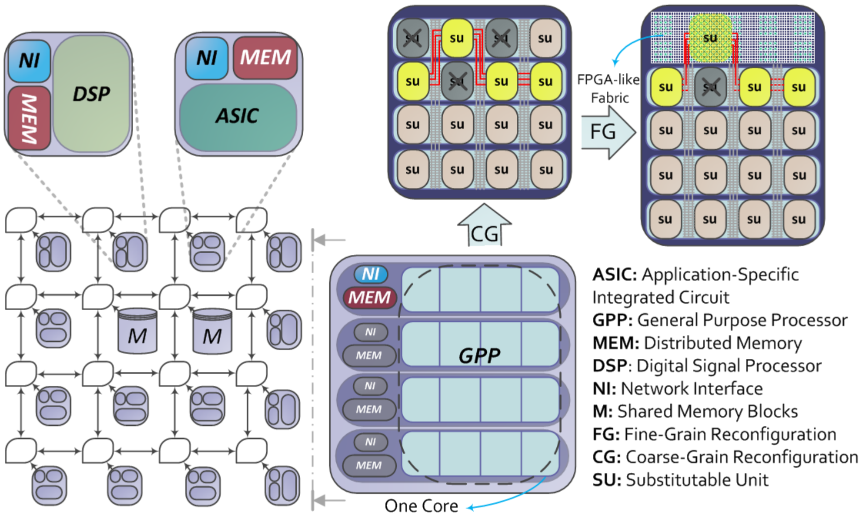

NI
MEM
DSP
NI
MEM
ASIC
M
M
su
FPGA-like Fabric
FG
CG
NI
MEM
GPP
One Core
ASIC: Application-Specific Integrated Circuit
GPP: General Purpose Processor
MEM: Distributed Memory
DSP: Digital Signal Processor
NI: Network Interface
M: Shared Memory Blocks
FG: Fine-Grain Reconfiguration
CG: Coarse-Grain Reconfiguration
SU: Substitutable Unit

Figure 1.1: *Adaptive processor in a chip-multiprocessor where permanent faults can be tolerated by using coarse-grain and fine-grain reconfiguration.*

where the architecture of processors is modified to support CG and CG+FG reconfigurations. To this end, the processor is divided into smaller substitutable units which are connected to each other using reconfigurable interconnects. In CG reconfiguration, when a permanent fault is detected on a processor, the affected unit is isolated and other fault-free units of the processor will be used as spares for the faulty processors on the CMP. When the option of FG reconfiguration is available, it is possible to instantiate the affected substitutable unit on the FG fabric and use reconfigurable interconnects to replace the faulty unit with the newly configured one. In general, using the fine-grain reconfiguration offers more flexibility and fault-tolerance but suffers from higher overheads compared to the coarse-grain option. Figure 1.1 shows the two option of coarse-grain and fine-grain reconfiguration for an array of four adaptive processors.

In the second part of this thesis, reconfiguration is employed in a service-oriented network-on-chip where the inherent redundancy inside the hardware architecture is exploited to mitigate permanent faults. With this goal, different reconfiguration techniques are explored: the functionality of the NoC is sustained by redirecting traffic classes either through alternative inter-router paths or intra-router data-paths. Figure 1.2(a) shows an example of redirecting traffic classes at the routers data-path[1]. Figure 1.2(b) is an example of redirecting traffic classes at routing level. In this example a

[1]The figure does not show all implementation details.

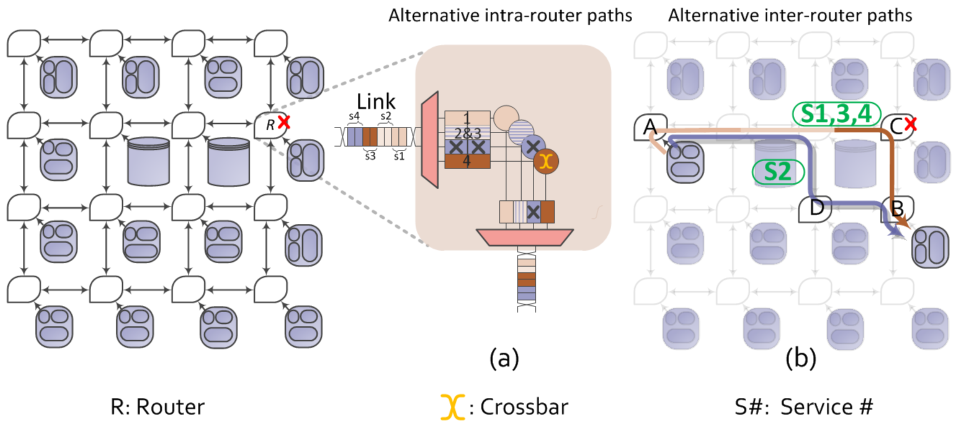


Figure 1.2: *Tolerating Permanent faults in a service-oriented NoC.*

fault is detected at service 2 of node *C* and node *D* is used to provide an alternative path for traffic of the affected service.

The main incentive of both parts of this thesis is based on the fact that hardware reconfiguration can be beneficial in the design of reliable systems where due to the application requirements, components must show high fault tolerance level. However, it should be noted that there is a fundamental trade-off between offered resilience and incurred hardware and performance overheads. In essence, these overheads are inevitable and are a major challenge in the design of reconfigurable hardware. In this thesis, we first study an adaptive processor array and then we explore alternative reconfiguration schemes inside a service-oriented NoC architecture. The end goal of both efforts is to find an efficient design point with desirable reliability and with acceptable overheads, i.e., performance, frequency, power consumption and silicon area.

1.2 Problem Statement

Shrinking transistor dimensions has magnified the importance of failure factors, e.g., process variation, manufacturing defects, transistors infant mortality and gate-oxide wearout [7]. As a result, fault-tolerant techniques have become important parts of recent system architectures. There is a broad spectrum of techniques for tolerating different fault classes, ranging from hardware to software and for different systems components, e.g., processing elements, on/off-chip interconnects. In this thesis, our focus is on hardware reconfiguration as a method for tolerating permanent faults and the issues regarding utilizing this technique in (i) an adaptive processor architecture and (ii) a service-oriented NoC architecture. Hardware reconfiguration can be used as way to work around permanent faults and improve systems lifetime, but potentially it might degrade performance, increase required silicon area or exacerbate power consumption. This section draws the problem statement for this thesis in the context of the two aforementioned topics and explains research questions behind performed tasks.

1.2.1 Tolerating Permanent Faults in the Processors

Hardware reconfiguration can be used in the context of CMPs, where adaptive processors take up a new configuration based on available non-faulty resources. We consider a CMP with multiple identical processors, i.e., simple MIPS-like in-order RISC, as our use-case. The architecture is modified to support coarse and fine-grain reconfiguration, here referred to as mixed-grain, for tolerating permanent faults. Whether or not fine- and/or coarse-grain reconfiguration is employed, is a design choice which depends on the desired level of fault-tolerance and accepted overhead. Thus, two important questions that arise are how to measure and compare: achieved fault-tolerance and incurred overheads for the two reconfiguration alternatives. Evidently, these questions need to be addressed considering different fault densities and also how they associate with the granularity of substitute units. Consequently, another open question is which granularity mix will maximize fault-tolerance of a system, considering the overheads in each case. In both reconfiguration alternatives, reconfigurable interconnects are essential to allow components parts to be substitutable, but the imposed wire-delays after reconfiguration will affect the systems performance. Based on this fact, another issue which needs to be addressed is the performance impact of pipelining the reconfigurable interconnects versus frequency scaling, i.e., reducing operating frequency to comply with increased critical-path delay. In summary, the research questions (RQs) which motivated the first part of this thesis are as follows:

- **RQ1** How can we measure systems fault-tolerance when using fine- and/or coarse-grain reconfiguration in an adaptive processor array in different fault densities?
- **RQ2** What are the overheads of each reconfiguration alternative?
- **RQ3** Which granularity mix is the most efficient considering fault densities and incurred overheads?
- **RQ4** What is the performance impact of pipelining interconnects versus frequency scaling?

1.2.2 Tolerating Permanent Faults in the NoC

The correct functionality of a SoC is crucially dependent on having reliable interconnects between system components. In modern SoC designs, NoCs have become the prominent choice for on-chip communication backbones due to their regular and robust structures. Several approaches have been proposed to improve the resilience of NoC architectural components, e.g., routers, links, network interfaces (NIs). On the other hand, the diversity of transmitted messages between on-chip components has raised the demand for NoCs which can support different traffic classes with different attributes. Thus, service-oriented NoCs that support certain quality of service (QoS), e.g., latency and throughput, are gaining more and more attention. Considering these essential requirements for modern NoCs, the focus of the second part of this thesis is to design a service-oriented NoC where different fault-tolerant schemes improve the resilience to permanent faults. To this end, we need to understand how we can employ hardware reconfiguration to harvest the redundancy inside the service-oriented NoC and improve reliability, while being minimally intrusive, i.e., maintaining the required level of QoS a much as possible. Different fault-tolerant schemes can be practically realized in a service-based NoC, each offering different reliability level. Then, inevitably, another open question is how beneficial is each one of these possible fault-tolerant techniques in presence of different number of network faults. To summarize, the second part of this thesis tries to answer following research questions:

- **RQ1** How can hardware reconfiguration be employed to design fault-tolerant methods in a service-oriented NoC?
- **RQ2** How can these methods be implemented, maintaining the QoS of the whole system as much as possible?
- **RQ3** What is the reliability level for different proposed methods in different fault densities?

1.3 Thesis Objectives

In this thesis, we focus on flexible/reconfigurable SoC designs tolerant to permanent faults and study their trade-offs between reliability and overheads. The main objectives of this thesis are the following:

- **Analyze the fault tolerance of an adaptive processor architecture.**

  One of the goals is to evaluate the reliability and overheads of different design points for an adaptive processor array at different fault densities. Each design point has different granularity of substitutable units and reconfiguration alternatives, i.e., CG or CG+FG. In particular, we aim at:

  - **Analytically calculating the fault tolerance of an adaptive processor array.**

  We shall find an analytical method for calculating the probability of constructing a particular number of fault-free components via different reconfiguration options, i.e., CG and CG+FG, given the fault density. Moreover, we shall find a method to calculate the probability of guaranteed certain number of fault-free components. Accomplishing this task is crucial in our attempt to perform our design-space exploration.

  - **Evaluating the performance impact of pipelining reconfigurable interconnects.**

  As a part of our work, we investigate possible performance benefits of pipelining reconfigurable interconnects which connect stages of adaptive processors.

  - **Performing a Design Space exploration for finding the most efficient granularity mix of adaptive processors.**

  Finally, using the obtained analytical calculations and overhead values from our use-case, we evaluate various reconfiguration scenarios, measuring the average number of fault-free components in given silicon area with the goal of finding efficient configurations for different fault densities.

- **Design and implementation of fault tolerant techniques to mitigate permanent faults in a service-oriented NoC.**

  In the second part of this thesis, we proposed different fault-tolerant schemes in a service-oriented NoC, by leveraging intra-router and inter-router redundancy. To this end, different hardware techniques are designed and implemented, increasing tolerance of the network to permanent faults. More precisely, we aim at:

  - **Modifying the architecture for service level reconfiguration.**

  The architecture of the baseline service-oriented NoC is modified to support different service redirection alternatives. Service redirection can be carried out at the routers data-path, here referred to as Service Merge, or at the routing level, here referred to as Service Detour.

– **Evaluating proposed fault-tolerant techniques in terms of performance, power and area.**

The above alternatives are then evaluated in terms of network performance, i.e., latency (in cycles), throughput (flits/cycle/node) and also the percentage of successful packet transmission. Moreover, area, power and frequency overheads are reported by synthesizing the RTL implementations of different techniques.

– **Evaluating attained reliability level.**

Finally, in order to study the level of fault-tolerance, analytical models of our techniques are created where network connectivity is measured under different fault densities.

The above are addressed in the thesis as outlined below. In the conclusions we describe how this thesis has contributed towards the above objectives.

1.4 Thesis Overview

This thesis consist of two main parts: the first part offers a probabilistic analysis of reconfigurable designs in terms of reliability and overheads and is covered in chapter 2. In the second part, covered in chapter 3, we introduce a hardware solution for increasing the resilience to permanent faults inside a service-oriented network-on-chip. Finally Chapter 4 presents the overall conclusions of the presented work, alongside proposed research directions. More precisely, the remainder of the thesis is organized as follow:

In chapter 2, a probabilistic analysis of hardware reconfigurability is performed, employing an adaptive processor architecture as our use-case. We describe different forms of reconfiguration and identify the required architectural modifications of our use-case processor. Furthermore, we offer a probabilistic method to calculate the fault tolerance of such designs. Different designs of the adaptive processor are then evaluated in terms of area, performance and power. Moreover, the performance impact of pipelining reconfigurable interconnects is explored. Finally, based on the results of evaluation and the probabilistic calculation, a comprehensive design space exploration is presented.

In chapter 3, hardware reconfiguration is used in the context of a NoC architecture to add inter-router and intra-router traffic redirection mechanisms, thus providing tolerance to permanent faults. We first introduce our baseline service-oriented NoC and its specifications. Afterwards, we introduce our proposed NoC architecture where different fault-tolerant techniques are described to improve NoC reliability in the presence of permanent faults at the links and the routers. More specifically, we present the Service Merge and Service Detour schemes as the main contributions of our work. Our techniques are subsequently compared with each other using a cycle-accurate simulator. Further, in order to evaluate the reliability of different schemes, an analytical study is performed to calculate the network connectivity in presence of different number of faults.

Finally, chapter 4 presents the concluding remarks. The chapter summarizes the content of this thesis, outlines contributions and findings and finally discusses possible future research directions.

2 A Probabilistic Analysis of Resilient Reconfigurable Designs

As technology scales chips become less reliable. Shrinking device features make circuits more susceptible to permanent faults due to manufacturing defects and wear-out phenomena [5]. Even a single permanent fault can damage an entire chip lowering production yields or jeopardizing availability when it appears in field.

The wide use of embedded systems in various application domains and the fact that such systems tend to become more complex and advanced as time passes, makes reliability and availability an even more pressing issue. Moreover, there is a number of mission-critical applications whereby faults are unacceptable; e.g. in the space, automotive, and medical domains. Such applications require high fault tolerance to deal with the increasing number of faults in emerging technologies.

In this chapter, we focus on System-on-Chip (SoC) recovery from permanent faults – simply denoted in here as *faults* – aiming at increased system availability and fault tolerance at high fault densities. One general approach to fault tolerance relies on dividing a design into basic blocks identical to each other, called *Substitutable Units* (SUs), whereupon so-called *sparing* strategies are employed: A faulty block of a system can be substituted by a spare (functioning) one. Clearly, this strategy requires redundancy of components and reconfigurability to replace damaged parts so as to keep the system functional.

Previous approaches divide a chip into smaller redundant SUs that can be isolated and replaced, if damaged, by spare identical parts. Such redundancy may appear in various granularities. In a *multiprocessor SoC*, core-redundancy is the most common choice [1,26]. Alternatively, smaller parts, such as pipeline stages and functional units, can be used as spares to repair a failing component [18,33,37,39]. At the other end of the design space, fine-grain logic (e.g., gate-level redundancy) can be used: for instance, Field-Programmable Gate Arrays (FPGAs) tolerate permanent faults by changing the configuration and/or the placement of a design [8]. The particular granularity (size of SU) chosen in a design introduces a trade-off between availability and efficiency. Finer granularities are more resilient to faults, but also less efficient in terms of area, performance and power consumption. Coarser redundancy tolerates fewer faults, but incurs lower overheads to a design.

This chapter presents a probabilistic analysis of the fault tolerance offered by hardware reconfigurability. Considering a generic design of components divided to SUs, we analytically estimate the resilience of different granularities. Thereby, we determine the most fault-tolerant reconfigurable design choice for a particular fault density. Concisely, the main contributions of this work are the following:

- We analytically calculate the probability of constructing a certain number of fault-free components via reconfiguration having as an input parameter the fault density.

- Then, we measure the area overheads of reconfigurability using a reconfigurable RISC processor with substitutable parts as a use-case component.

- We evaluate various granularities of reconfigurable substrates measuring the average number of fault-free components as well as the probability of delivering a minimum number of fault-free components in a given silicon area. In so doing, we identify the most fault-tolerant designs for a particular fault density.

In the remainder of this chapter, related work is presented first in Section 2.1, then Section 2.2 describes reconfigurable design alternatives for repairing faulty components. Section 2.3 details the design of the particular reconfigurable processor used in this study. Section 2.4 offers a probabilistic analysis of fault-tolerance in reconfigurable designs while Section 2.5 evaluates the fault-tolerance of different design points. Finally, in Section 2.6 overall conclusions are drawn.

2.1 Related Work

In the past, a large number of design techniques have been introduced for tolerating permanent faults. They consider different SU sizes ranging from entire components, e.g. micro-processors, to fine-grain logic and have different fault tolerance and design overheads.

Existing commercial systems offering high availability, such as the IBM z-series [11] and the HP NonStop systems, [3], provide redundant cores and dual modular redundancy (DMR) to combat permanent faults. LaFrieda et al. showed that DMR of dynamically coupled cores improves fault-tolerance [26]. Disabling [38] and isolating [1] permanently faulty cores can sustain degraded performance, while dark silicon makes it possible to have additional spare components [20]. Moreover, partly damaged cores can potentially be used with degraded functionality by allowing affected threads to migrate to other cores [34], or even to be rescued using advanced diagnostics and voltage/frequency tuning to reverse some of the wear-out phenomena [43].

In order to deal with the increasing number of permanent faults, designers turned to finer granularities. The SUs may be functional units [39] or pipeline stages [18, 33, 37]. In the latter case, processors — mostly simple RISCs — are modified to support dynamic replacement of stages.

Finer granularities further increase the fault tolerance of a design. A processor data-path can be protected by adding spare-bits [31], while a control-path can use field-programmable control logic to tolerate faulty states [46]. Finally, FPGAs and Programmable Logic Arrays (PLA) can be used as the hardware substrate of a fault tolerant SoC [8].

Following a passive fault-avoidance approach (disabling [38], isolating [1]) is inherently inefficient as it sacrifices availability when permanent faults occur. On the other hand, fault tolerance through sparing, replacing or repairing faulty components requires flexible wiring and multiplexing of the SUs, which not only adds significant delays to the design, but increases area and power consumption.

In general, larger SU sizes incur lower overheads, but also offer lower flexibility and hence lower fault tolerance. To the best of our knowledge, this is the first study to find the reconfigurable-design choices that maximize component availability for a given technology and a particular fault density. We explore a design space of various granularities of SUs using either coarse- or fine-grain reconfigurability, or a mix of both and analyze how component availability scales in different fault-densities.

2.2 Reconfigurable Designs for Tolerating Permanent Faults

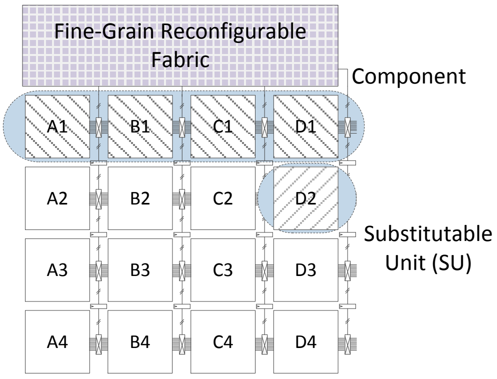
We discuss next the two options of reconfigurability, studied in this chapter, to combat permanent faults. A component can be partitioned into SUs that are interconnected with reconfigurable interconnects to allow one to substitute another; we call this coarse-grain reconfigurability. A second option is using fine-grain, FPGA-like, reconfigurable logic, shared among components and used to replace damaged parts when lacking identical spares; this is denoted as fine-grain reconfigurability. A faulty SU can be replaced by either an identical spare unit, presumably taken from a neighboring unused/faulty component, or instantiated in the fine-grain reconfigurable logic. We consider that a reconfigurable design may offer either one of the two reconfiguration options (coarse- or fine-grain) or both at different sizes of SUs (granularities). A generic example of a reconfigurable design that offers both coarse- and fine-grain redundancy is illustrated in Figure 2.1. The partitioning of the design is depicted as an array. Each column comprises identical, interchangeable, SUs, and each row represents a component in its fault-free configuration. The block at the top represents the shared fine-grain logic. A functioning component can be constructed connecting a single undamaged cell from each column; in case there is no fault-free cell for a particular column, the fine-grain block can be used as a wild card to replace it.

The granularity of a design affects its area overheads. The largest SU of a component defines the minimum size of the fine-grain block, consequently, it is more efficient to partition a component into units of similar size. Moreover, a larger number of smaller SUs requires more wiring, but also needs a smaller fine-grain block.

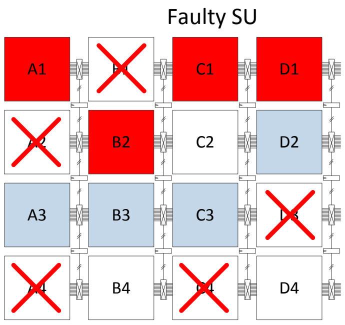
2.3 A Reconfigurable adaptive RISC processor

In order to conduct our design-space exploration and retrieve real measurements for the area overheads of reconfigurability, we consider as a use-case a coarse-grain reconfigurable RISC architecture with substitutable parts, previously described in [41]. Moreover the architecture is modified to additionally support fine-grain reconfiguration [42].

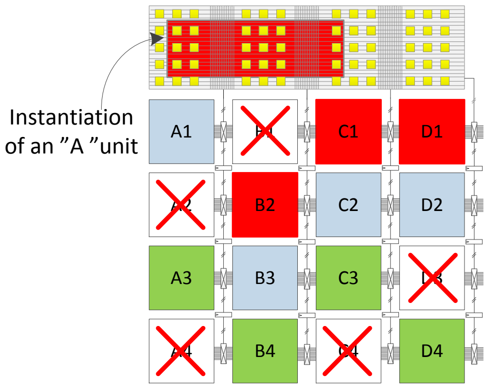
Our use case is a 32-bit, in-order RISC processor with four pipeline stages: Instruction Fetch (IF), Decode (DC), Execute (EX), and Memory (ME). The processor has a local instruction- and data-memory (32Kbyte each) and a 16-entry register file (RF).



(a) Four components with 4 Substitutable units (SUs) each and a fine grain logic block.



(b) Example of CG reconfiguration constructing 2 fault-free components: component-1 (A1, B2, C1, D1) and component-2 (A3, B3, C3, D2).



(c) Example of CG and FG reconfiguration constructing 3 fault-free components: component-1 (A{FG}, B2, C1, D1), component-2 (A1, B3, C2, D2), and component-3 (A3, B4, C3, D4).

Figure 2.1: *A design of four identical components with coarse (CG) and fine-grain (FG) reconfigurability.*

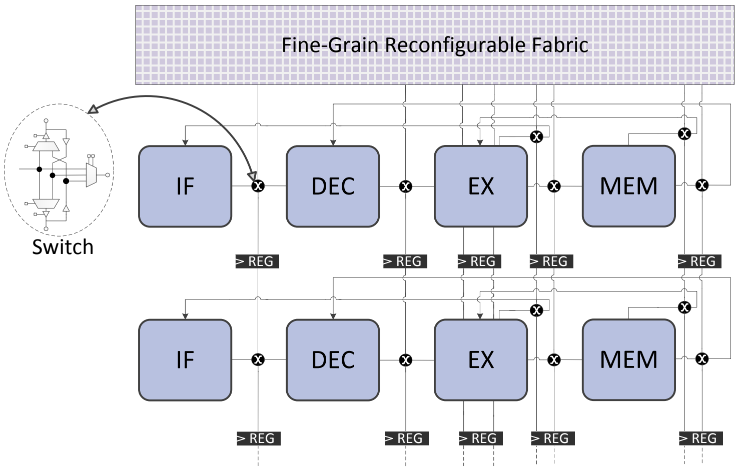


Figure 2.2: *An adaptive processors array with substitutable stages.*

The architecture offers decoupled pipeline stages, which can be substituted by any other identical stage or by fine-grain reconfigurable logic. This is achieved using reconfigurable interconnects. Interconnects that traverse across different processors are expected to be used infrequently. We therefore implement them as bidirectional interconnects and use switches with tri-states to go across different components of the same processor. Compared to unidirectional interconnects, this halves the wiring resources needed, but adds some extra delay to the switches.

In order to minimize the additional delay introduced by the reconfigurable interconnects and switches, interconnects that span across the processor boundaries are also pipelined, as depicted in Figure 2.2. Thereby, sharing stages from neighboring cores introduces additional stages to a pipeline. As a consequence, we need to support a variable pipeline depth in different configurations, but operating frequency scales better with the number of components [41]. The direct effect of this is that the architecture must adapt and operate correctly while remaining oblivious to the number of (extra) stages added to its pipeline; that is, reconfigurability needs to be transparent to the application level guaranteeing binary compatibility among different processor configurations. The performance impact of pipelining the interconnects that connect the SUs of processors is studied in section 2.5.2.

This design requirement calls for several micro-architectural changes in the data flow and control flow of the processor. Allowing a variable number of stages per processor and eliminating global control directly influences the hazard resolution in a typical RISC architecture. Briefly, these issues have been addressed in the proposed architecture, described in [41], as follows:

1. Data-hazard resolution: As in between the original four stages (in the fault-free configuration) new empty pipeline stages may be inserted when substituting faulty parts, the data-hazard resolution cannot be supported with traditional bypassing mechanisms. Instead, we store the produced, yet uncommitted, results in bypass buffers at the stages they are produced and may possibly be reused. These bypass buffers are located in the EX and MEM stages.

2. Control-hazard resolution, Global stalls, pipeline flushing: All three aspects are supported via pipeline flushing in a distributed way. Similar to the StageNet [18], instructions carry an extra bit, read at the IF stage, indicating the flow they belong to. This bit is compared with a similar instruction-flow bit stored at the EX stage. A mismatch prevents the instruction from being executed. In order to flush the pipeline, the bit stored at the EX stage is inverted (e.g. after a branch mis-prediction identified at the EX stage) and all the subsequent instructions are flushed until the equivalent Instruction-flow bit stored at the IF stage is updated.

The architecture presented in [41] is further modified to support fine-grain replacement of a faulty part. This needs the following additions [42]: When implemented in the fine-grain logic, the EX stage requires substantially more area and has longer delay than any other SU of our processor. To reduce this imbalance, we further partition the EX stage into three concurrent sub-units, each roughly containing a chunk of the ALU. Thereby we keep the size of the SUs balanced – as the EX stage occupies more area than the IF, DC or MEM – and consequently reduces the area needed for the fine-grain logic. A second modification is performed to improve the speed of the processor when part of the EX stage is implemented in the fine-grain logic. We add the option to further pipeline the EX stage (breaking the ALU logic in two stages) when part of it is instantiated in the fine-grain logic.

2.4 Probabilistic Analysis of Reconfigurability

Considering the generic design described in Section 2.2, we next present a probabilistic analysis used in our evaluation to assess the fault-tolerance of various design options. In particular, we derive the probability of having exactly M fault-free components out of N ($P_{ff}(N, M)$) as well as the probability of having at least M fault-free components out of N ($P_{\geq}(N,M)$) for different numbers of faults in area A for two different design approaches: (i) coarse-grain reconfigurability[1] and (ii) a mix of coarse- and fine-grain reconfigurability which offers the option of a faulty SU to be replaced either by an identical spare unit (coarse-grain replacement) or by fine-grain logic.

Assuming that an area A is divided into N SUs of equal size and that there are k faults in total in the area with a uniform random distribution, then the probability of a SU to have exactly i out of the k faults is:

$$P_{f=i}(k,N) = \binom{k}{i} \times \left(\frac{1}{N}\right)^{i} \times (1-\frac{1}{N})^{k-i} \tag{2.1}$$

That is, there are $\binom{k}{i}$ combinations of having exactly i out of k faults located in one SU (and exactly $k-i$ faults located in the rest of area A) and the probability of each case is equal to $\left(\frac{1}{N}\right)^{i} \times (1-\frac{1}{N})^{k-i}$.

Then, the probability of a SU to be faulty, $P_{sf}(k,N)$, is equal to the sum of probabilities of having $1,2,...,k$ faults:

$$P_{sf}(k,N) = \sum_{i=1}^{k} P_{f=i}(k,N) \tag{2.2}$$

Considering that the P_{sf}, for a specific number of faults[2] k in the area, of a single SU is known based on the equation 2.2, it is possible to calculate the probability of having exactly M fault-free SUs out of N, denoted as $P_{ff}(N,M)$ [21]:

$$P_{ff}(N,M) = \binom{N}{M} \times P_{sf}^{N-M} \times (1-P_{sf})^{M} \tag{2.3}$$

where $\binom{N}{M}$ enumerates possible ways of having *exactly* M non-faulty substitutable units out of N and $P_{sf}^{N-M} \times (1-P_{sf})^{M}$ is the combined probability of having $(N-M)$ faulty and M non-faulty SUs. It is also possible to expand this formula to get the probability of having *at least* M non-faulty SUs out of N for a specific P_{sf}, here indicated as $P_{\geq}$:

$$P_{\geq}(N,M) = \sum_{i=M}^{N} P_{ff}(N,i) \tag{2.4}$$

[1]Component redundancy is an extreme case of coarse-grain reconfigurability where the entire component is a SU.

[2]Henceforth, k is omitted from the equations, we consider that the analysis is continued for a specific k and repeated for the range of faults considered in the evaluation, e.g. 0 to 20 faults.

which is the sum of probabilities of having $i = \{M, \cdots, N\}$ fault-free SUs [21].

In a design with coarse-grain reconfigurability, each component is divided into a specific number of SUs. Considering the example illustrated in Figure 2.1 which depicts four components each of which are divided into four SUs, it can be observed that for having a fault-free (working) component, there should be at least one fault-free SU at each column available. Therefore, the number of fault-free components that can be constructed is defined by the minimum number of fault-free SUs in each column. For calculating the probability of having a specific number of fault-free components, we first calculate the probability of having at least M fault-free SUs out of N in one column, which is derived by equation 2.4:

$$P_{\geq}^{col}(N, M) = P_{\geq}(N, M) \tag{2.5}$$

Expanding this formula over the total number of columns will result in the probability of having at least M SUs in each column, which is equal to the probability of having at least M fault-free components available in a coarse-grain (cg) reconfigurable design:

$$P_{\geq}^{cg}(N, M) = \left(P_{\geq}^{col}(N, M)\right)^{c} \tag{2.6}$$

The exponent c is the total number of columns, which also defines the coarse-grain granularity[3]. In order to find the probability of having exactly M fault-free components in this case, we exclude from the probability of having at least M fault-free components ($P_{\geq}^{cg}(N, M)$) the probability of having at least $(M+1)$ fault-free components ($P_{\geq}^{cg}(N, M+1)$):

$$P_{ff}^{cg}(N, M) = P_{\geq}^{cg}(N, M) - P_{\geq}^{cg}(N, M+1) \tag{2.7}$$

As mentioned in the previous sections, fine-grain logic can also be used in order to instantiate different SUs and increase the resilience in the presence of permanent faults. We can calculate the probability of having a specific number of fault-free components when both fine-grain and coarse-grain reconfigurability is employed by modifying the probabilities for coarse-grain designs derived above.

Figure 2.1 can be used again as an example that depicts a design with four components, each divided into four SUs, having both fine- and coarse-grain reconfigurability. When only coarse-grain replacement is used, having two faulty SUs on

[3] The number of SUs in a component.

one column and at most one in the remaining columns limits the number of fault-free components to two. In a coarse-grain design the above event is included in the probability of having exactly two fault-free components. Introducing fine-grain logic to replace one faulty SU, rescues one additional component in the above example. Consequently, this event (two faulty units at one column and at most one in the remaining columns) should be included in the list of events which are counted under the probability of having three fault-free components and should be removed from the events that are considered under the probability of having two fault-free components.

Finding events that should be removed or added to the “coarse-grain" probability depends on the number of SUs that can fit inside the fine-grain logic. In this work we only consider the case where the fine-grain can be used to replace exactly one faulty SU. As all events are independent, it is possible to separately compute their probability and append it to the $P_{ff}^{cg}(N,M)$ of equation 2.7. $P_{append}^{+}(N,M)$ denotes the probability of events originally considered in the probability of having $(M-1)$ fault-free components but when using fine-grain logic are becoming part of the probability for M fault-free components. Similarly, $P_{append}^{-}(N,M)$ is the probability of events originally counted in $P_{ff}^{cg}(N,M)$, but using fine-grain replacement moves them to the probability of having $(M+1)$ fault-free components.

$$P_{append}^{+}(N,M) = \binom{c}{1} \times [P_{ff}(N,M-1)]^{1} \times [P_{\geq}(N,M)]^{c-1} \tag{2.8}$$

where c is the number of columns, $\binom{c}{1}$ enumerates all possible choices of one column out of c, $[P_{ff}(N,M-1)]^{1}$ is the probability of having exactly $(M-1)$ fault-free SUs out of N in one column and $[P_{\geq}(N,M)]^{c-1}$ is the probability of having at least M fault-free SUs in $(c-1)$ columns. Similarly, $P_{append}^{-}(N,M)$ can be calculated:

$$P_{append}^{-}(N,M) = \binom{c}{1} \times [P_{ff}(N,M)]^{1} \times [P_{\geq}(N,M+1)]^{c-1} \tag{2.9}$$

where $[P_{ff}(N,M)]^{1} \times [P_{\geq}(N,M+1)]^{c-1}$ is the probability of having exactly M fault-free SUs in one column and at least $(M+1)$ fault-free SUs in $(c-1)$ columns. Then, the probability of having exactly M fault-free components out of N for a design which uses fine-grain logic in addition to coarse-grain replacement is:

$$\begin{aligned} P_{ff}^{cg+fg}(N,M) =& P_{ff}^{cg}(N,M) \\ &+ P_{append}^{+}(N,M) - P_{append}^{-}(N,M) \end{aligned} \tag{2.10}$$

Similar to the previous cases, in order to find the probability of having at least M fault-free components out of N in a design with both fine and coarse-grain reconfigurability, it is possible to add all probabilities of having exactly $M, M+1, ..., N$ fault-free components:

$$P_{\geq}^{cg+fg}(N,M) = \sum_{i=M}^{N} P_{ff}^{cg+fg}(N,i) \tag{2.11}$$

The probabilities of having at least M fault-free components out of N in the different design approaches considered ($P_{\geq}^{cg}(N,M)$, $P_{\geq}^{cg+fg}(N,M)$) are used in the evaluation section to measure the probability of a design to guarantee a particular availability of components.

The probabilities of having exactly M fault-free components out of N in the considered design alternatives ($P_{ff}^{cg}(N,M)$, $P_{ff}^{cg+fg}(N,M)$) are used to evaluate the average number of fault-free components in a given area as described below. For a specific number of faults, k, the above probability ($P_{ff}^{j}(N,M)$, where j is "cg" or "$cg+fg$") is used to calculate the average number of fault-free components, $\overline{x}$, as the weighted average of the individual probabilities:

$$\overline{x} = \sum_{i=1}^{N} P_{ff}^{j}(N,i) \times i \tag{2.12}$$

where N is total number of components.

2.5 Evaluation

In this section, we first measure the reconfigurability overheads based on our use-case RISC processor. Moreover, the effect of pipelining reconfiguration interconnects is explored based on post place and router measurements. Finally we evaluate the fault tolerance of various reconfigurable designs using our probabilistic analysis.

2.5.1 Reconfigurability Overheads

Our use-case component is implemented using 65nm STM technology for the ASIC parts (an array of RISC processor and reconfigurable interconnects) and the Xilinx Virtex-5 65nm FPGA substrate for the fine-grain logic [42]. The area overhead of the fine-grain logic required to implement a SU is roughly 6× the area of the same unit implemented in ASIC[4] technology. The area overhead of the reconfigurable interconnects is measured considering a coarse-grain reconfigurable array of four processors, each divided in four SUs (pipeline stages); in that design reconfigurable interconnects add 12.8% more area to each processor [41]. In order to estimate the overhead of reconfigurable interconnects for coarser and finer granularities[5], we use Rent's rule [27] to estimate the number of interconnects per SU, as follows:

$$W = K \times (N_p)^{\beta} \tag{2.13}$$

W, is the estimated number of interconnects for a component, K is the average number of interconnects per SU, N_p is the total number of SUs in the component and β is the Rent's exponent which is normally $0.5 \leq \beta \leq 0.7$. In this study, we consider $\beta = 0.5$, which is a typical value for a microprocessor design. The estimated overheads are used in order to define the number of components which can be accommodated in a fixed area for each case of this analysis.

It should be noted that the reconfigurable RISC processor presented in Section 2.3, operates at 450 MHz when considering only coarse-grain redundancy. This is 10% lower compared to the same baseline processor before the micro-architectural modifications required to make its stages interchangeable[6]. Finally, the slowest SU when instantiated in fine-grain logic operates at 200 MHz, which limits any processor configuration using the fine-grain reconfigurable part to the same frequency.

2.5.2 Evaluating the Effect of Pipelining the Reconfigurable interconnects

This section explores the performance impact of using pipelined interconnects versus non-pipelined ones. We first describe our experimental setup, used benchmarks and designs under study. Subsequently, the results are presented and evaluated and a comparison is performed. Pipelining maintains a high frequency but increases the number of stages in an adaptive processor, thereby potentially increasing the number of cycles for executing a program.

For our experiments, we acquired an RTL implementation of the processor using a high level description language (Lisa 2.0) through the Synopsys Processor and Compiler

[4]In order to keep the area requirements of the fine-grain logic low, we consider that memory blocks, pipeline registers and buffers are implemented in ASIC, which is more area-efficient than in an FPGA implementation. Consequently, even when using a fine-grain block, only logic is mapped to FPGA logic cells.

[5]Number of SUs a component is divided into.

[6]The original baseline RISC processor has an operating frequency of 500 MHz

Designer tool. Different designs are then synthesized using Cadence RTL compiler and place & route is performed using Cadence SOC encounter.

Our evaluation is carried out by employing EEMBC CoreMark and Telebench 1.1[7] benchmarks. Furthermore, a set of small custom benchmarks is created with the main purpose of evaluating the processor under extreme cases. These benchmarks are small C-codes, running in loops and are briefly described in the following [40].

- **Function-Argument Heavy:** An application with large number of instructions prone to data hazards between memory operation and arithmetic.
- **Heavy Read-After-Write Conflict:** An application with large number of instructions prone to read after write (RAW) hazards.
- **Heavy Branch-Mis-predict:** An application with large number of non-taken branches.
- **Normal-C for-loop:** An application with typical C for-loop.

Our evaluation is performed for variants of our adaptive processor with coarse-grain (CG) reconfiguration. In addition to the baseline processor, other defined cases are as follows.

- **CG w/o switches:** The adaptive processor without switches for reconfiguration. This case is used study the effect of micro-architectural changes inside the processor.
- **CG w/o registers:** The adaptive processor, being part of a 4-core CMP, without registers on the reconfigurable interconnects. This case is used to study the benefits of pipelined interconnects.
- **CG w/ registers every 2 cores:** In this case, Considering the adaptive processor in a 4-core CMP, reconfigurable interconnects are pipelined at every other processor. This design is an intermediate case between two extremes of having fully pipelined and no-pipelined interconnects.
- **CG fault-free:** The adaptive processor, being part of a 4-core CMP, considering fault-free scenario. Interconnects are fully pipelined. This case is illustrated in Figure 2.3 a.

[7]The FFT benchmark is not included as it requires floating point operations, which the under study processors do not currently support.

- **CG 2 extra stages:** The adaptive processor, being part of a 4-core CMP with fully pipelined interconnects, with 2 extra stages, imposed by reconfiguration. The two extra stages are result of using one stage of a neighboring core to construct a processor as shown in Figure2.3 b.

- **CG 6 extra stages:** The adaptive processor, being part of a 4-core CMP with fully pipelined interconnects, with 6 extra stages, imposed by reconfiguration. The six extra stages are result of using one stage of the farthest core to construct a processor as shown in Figure 2.3 c.

- **CG 12 extra stages:** The adaptive processor, being part of a 4-core CMP with fully pipelined interconnects, with 12 extra stages, imposed by reconfiguration. The twelve extra stages are result of having an extreme case where every stage is connected to the farthest next stage to construct one processor as shown in Figure 2.3 d.

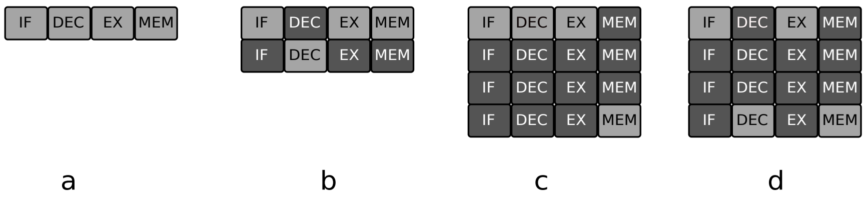


Figure 2.3: *Four different configurations of the adaptive processor, being part of a 4-core CMP with fully pipelined interconnects: a. fault-free configuration, b. replacing the DEC stage with the one from the neighboring processor creates two extra stages, c. replacing the MEM stage with the one in the farthest processor creates 6 extra (empty) stages, d. Worst case scenario where the DEC and MEM stages are replaced with the ones three processors away creating 12 extra (empty) stages.*

It should be mentioned that the overall efficiency of the CMP depends on the occurrence rate of various configurations. The configurations studied here are enforced by the density and location of faults and thus it is possible to calculate the probability of each of these cases to happen. With the fault density of one fault per core, 38%, 60%, and 2%, of the configurations in a CG design have zero, 1-4 and 5-15 extra stages, respectively.

Table 2.1 presents the operating frequency of the baseline and four of the explained cases. Whilst the clock period of the baseline design is measured at 2.5ns, the micro-architectural changes required for coarse-grain reconfiguration introduce 16.4% performance degradation with clock period of 2.91ns. The adaptive processor with no faults incurs a 41% overhead compared to the baseline with a cycle time of 2.5ns. The non-pipelined version of the adaptive processor shows worst performance degradation and fastest possible clock cycles is measured to be 8.53ns, that is 3.4× longer than the baseline and 2.9× compared to design without switches. Finally the design with pipeline registers at very other processors can operate at clock period of 7.1ns.

Table 2.1: *Place and Route Timing measurements for our CMP*

Design	Clock Period (ns)	Overhead	
		vs Baseline	vs Adaptive Processor w/o switches
Baseline	2.5	-	-
Adaptive Processor w/o switches	2.91	16.4%	-
Adaptive Processor (fault-free)	3.53	41%	21.3%
Adaptive Processor w/o registers	8.53	241%	193%
Adaptive Processor w/ registers every 2 cores	7.1	184%	143%

Figure 2.4 shows the measured execution cycles of the baseline, adaptive processor with no faults and adaptive processor with 2, 6 and 12 extra stages for the EEMBC benchmarks. The adaptive processor under fault-free condition maintains 100% of the baseline performance. However, adding 2, 6 and 12 extra stages increases the required execution cycles by 18%, 37% and 100%, respectively. We performed the same evaluation using our custom benchmarks. As depicted in Figure 2.5, the adaptive processor – in the absence of faults – introduces a small overhead in terms of execution cycles compared to the baseline. The overhead is more noticeable for the Argument-Heavy benchmark where we have about 7.5% increase that can be explained by frequent use of flush/reload mechanism in this benchmark. Adaptive processor with 2 extra stages exhibit acceptable performance with around 20% overhead in average. Furthermore, the processors with the two extreme cases of having 6 and 12 extra stages, increase the number of execution cycles by 1.8× and 2.5×, respectively.

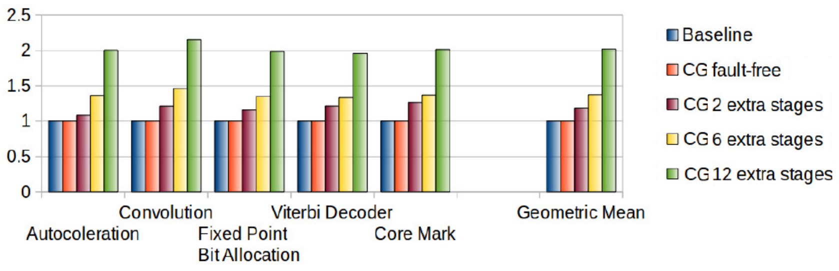


Figure 2.4: *Execution cycles for each case using EEMBC benchmarks, normalized by the baseline cycle count.*

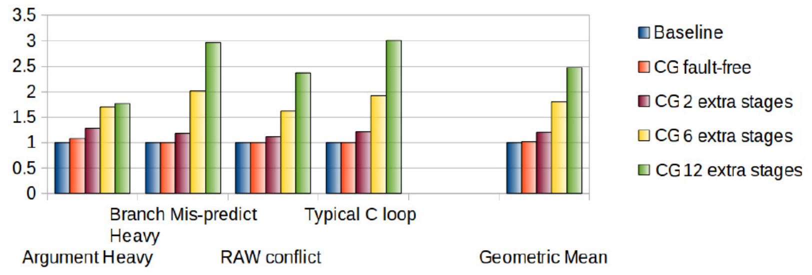


Figure 2.5: *Execution cycles for each case using custom benchmark, normalized by the baseline cycle count*

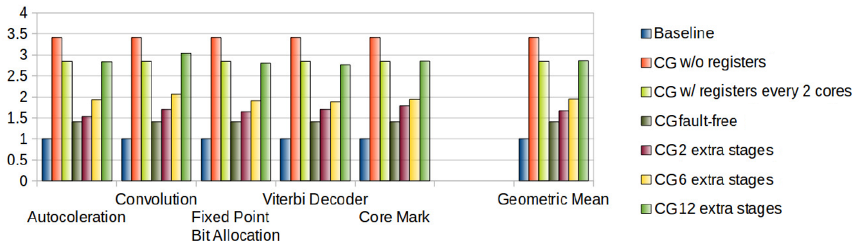


Figure 2.6: *Execution time for EEMBC benchmarks normalized to baseline values.*

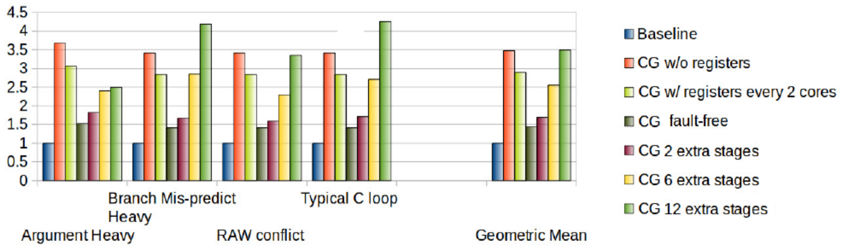


Figure 2.7: *Execution time for each custom benchmark normalized to baseline values.*

In the next step, by associating the measured operating frequencies with execution cycles, we find the overall execution time of each case. Figures 2.6, depict the results for EEMBC benchmarks. The results clearly indicate that the adaptive processor without pipelined interconnects suffers from longest execution time which is almost 3.5× of the baseline. Using registers at every other core just slightly mitigates the execution time to nearly 3× of the baseline. Furthermore, in presence of no fault, the fully pipelined adaptive processor increases the execution time by 41%, due to expansion of critical path. In case with 2, 6 and 12 extra stages, the execution time compared to the baseline increases by roughly 1.7×, 1.9× and 2.8×, respectively. Using our custom benchmarks show almost the same results for the execution time as depicted in Figures 2.7. The overhead of execution time for the designs with 6 and 12 extra stages degrades to 2.6× and 3.5× of the baseline, respectively.

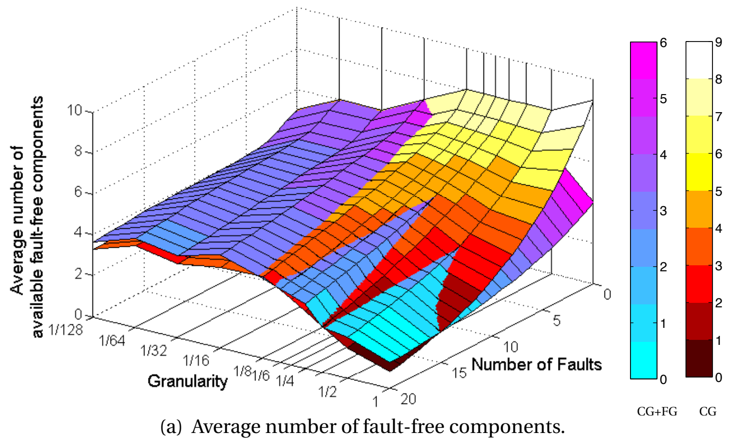
In summary, our comparison using post place and route results showed that even for a 4-core CMP design the wire delay between the stages of processors adds significant delay, degrading the operating frequency 3.5× compared to the baseline design. Moreover, our experiments showed that the processor with non-pipelined reconfigurable interconnects incurs 3.5× overhead in terms of execution time compared to the baseline. The results for the processor with fully-pipelined interconnects showed that even in the extreme case of having 12 extra stages, the performance is better than non-pipelined version.

2.5.3 Fault-tolerance of Reconfigurable Designs

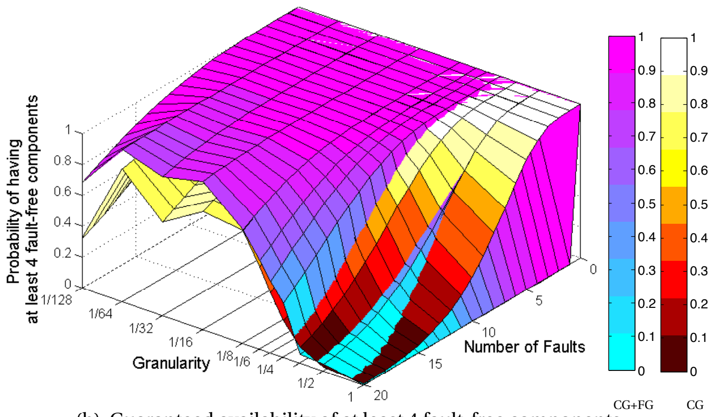
We take into account the above area overheads for each reconfigurable design and evaluate their fault tolerance. We consider designs that offer only coarse-grain (CG) reconfiguration or both coarse and fine-grain (CG+FG), varying their granularity (SU size). All designs occupy roughly the same area[8] and therefore, due to their particular resource overheads, each design fits a different number of components in the fault-free case. We consider a fault density that causes up to 20 faults in the silicon area. This translates, even for our small use-case microprocessor, to a probability of a single transistor to fail to be up to 2.2×10^{-6}.

Figure 2.8 depicts the average number of fault-free components that can be constructed in each design at a particular fault density (number of faults), as well as the probability of a design to deliver at least four fault-free components.

[8]The silicon area constraint in our design-space exploration is equal to that of 9 baseline components. A baseline component is a component with granularity equal to 1 (the entire component is considered as one SU).



(a) Average number of fault-free components.



(b) Guaranteed availability of at least 4 fault-free components.

Figure 2.8: *Average availability and probability of guaranteed availability (4 fault-free components) of coarse-grain (CG) and coarse+fine-grain (CG+FG) reconfigurable designs with various granularities at different fault densities. Granularity is the fraction of a component that constitutes a SU. All designs use area smaller or equal to 9 baseline components; that is a component with granularity equal to 1.*

The average number of available components is measured as explained in Section 2.4 based on equation 2.12 by analytically calculating the probability of constructing correctly functioning components for the given reconfiguration options offered by each design. Figure 2.8(a) shows that for low number of faults (≤ 2), component redundancy provides a higher number of fault-free components, as the area overheads of reconfigurability are not capitalized yet. For higher number of faults (≤ 8), coarse-grain reconfigurability of granularity $\frac{1}{8}$ (CG($\frac{1}{8}$)) maximizes availability of components. Adding fine-grain logic at smaller SU sizes (CG+FG($\frac{1}{16}$)) is up to 13.5% better for fault densities beyond that point. Even at high number of faults, CG+FG($\frac{1}{16}$) provides almost five available components, tolerating over 3× and 2× more faults than component-redundancy (CG(1)) and other CG points, respectively.

Guaranteed availability is measured using equations 2.6 and 2.11 as the probability of a design to deliver at least 4 fault-free components at a particular fault density. As shown in Figure 2.8(b), the probability for the CG+FG($\frac{1}{16}$) design does not drop below 99% for up to 20 faults. On the contrary, component redundancy (CG(1)) is below 90% and 50% beyond 6 and 10 faults, respectively. Finally, the best CG granularity crosses the threshold of 90% for more than 17 faults.

2.6 Conclusions

A probabilistic analysis was presented for estimating the number of fault-free components that can be constructed in various reconfigurable designs at the presence of permanent faults. Adding fine-grain logic in a design can increase availability, tolerating 3× more faults than mere component redundancy. Different fault densities require different granularities of substitutable component-parts in order to maximize fault-tolerance. Component redundancy is better for a low number of faults. As the number of faults increases, coarse-grain and mixed-grain reconfigurability (of granularity $\frac{1}{8}$ and $\frac{1}{16}$, respectively) provide the best availability.

Moreover, the effect of pipelining the reconfiguration interconnects was explored, in terms of operating frequency and execution time. The results showed that in the processor with no pipelined interconnects the frequency and execution both increase by almost 3.5×, compared to the baseline, while adding pipeline registers shows significantly better performance where for the fault-free case the overhead of operating frequency and execution time are 1.41× and 1.7×.

3 RQNoC: a Resilient QoS NoC with Service Redirection

Technology scales to feature sizes of a few nanometers bringing great opportunities and new challenges to the design of future computing systems. On one hand, designers can use billions of available transistors to fit multiple cores on a single chip and build more sophisticated embedded systems. On the other hand, shrinking of transistors increases variability and affects the reliability of circuits. On-chip resources become more susceptible to manufacturing defects and wearout phenomena, which may damage permanently a circuit and render an entire chip unusable. In turn, the increasing number of permanent faults can reduce production yield or cause failures during the lifetime of a system.

The wide use of more advanced embedded systems in various domains, such as medical, space and automotive, makes reliability and availability even more pressing issues. In particular, safety-critical systems with low or no accessibility for replacing damaged parts require efficient techniques for self-repair and fault tolerance to deal with the increasing number of faults in shrinking technology nodes.

In a multicore system, a faulty processor can be isolated using runtime mechanisms to redistribute its workload to other available cores. However, tolerating permanent faults at the interconnects can be less simple. A Network-on-Chip (NoC) shares its resources with every component on a chip. It interconnects multiple processing elements, memory blocks and IO and therefore constitutes a single point of failure for an entire System-on-Chip (SoC). A fault-tolerant NoC architecture is therefore critical for the design of a fault-tolerant SoC.

The NoC of an advanced embedded system needs to support multiple traffic classes with certain quality of service (QoS) requirements, such as latency and throughput. This is often achieved using virtual channels and priorities among different classes of traffic related, for instance, to control messages or data-exchange. However, many systems — especially in the embedded domain — require their traffic classes to interfere as little as possible with each other in order to best fit their particular QoS constraints. In the past years, several service-oriented NoCs address this design objective by statically allocating their resources to different traffic classes (services). For example, QNoC provides four separate router data-paths each supporting different services [4], moreover, Tilera's iMesh is composed of five separate NoCs entirely isolating each traffic class of their tiled architecture [47].

This work addresses a single challenge pertaining to the design of future reliable multicore systems: the *tolerance of permanent faults on a service-oriented NoC*. We categorize network resources based on the service they support and provide mechanisms to bypass them when faulty, allowing a traffic class to be redirected. Traffic redirection can be performed either by modifying its routing or alternatively by using a router data-path dedicated to a different service. Network resources that are common to all services, such as the links and the router control, employ additional mechanisms for fault tolerance.

Concisely, the main contributions of this work are the following:

- We extend previous detour techniques, allowing the network to selectively detour traffic per service, rather than detouring all packets, thereby avoiding faulty network parts and achieving significant improvement in the fault-tolerance of a NoC.
- Within a router, multiple services can be merged, still preserving their priorities, in order to bypass the damaged data-path of one or multiple services.
- Faulty wires at the links, that are common for all services, can be tolerated extending an existing shifting mechanism with additional spare wires.

- We measure the area and power overheads of the above techniques, evaluate their impact on the performance and QoS of each service and analyze their fault tolerance.

In the remainder of this chapter, first related work is presented in Section 3.1. The description of the baseline service-oriented NoC used in this study follows in Section 3.2. Then, Section 3.3 details the proposed mechanisms for tolerating permanent faults on the NoC. Section 3.4 evaluates the fault tolerance of our approach, measures its area, power and performance overheads, and compares with related work. Finally, in Section 3.5 overall conclusions are drawn.

3.1 Related Work

In the past a plethora of techniques, some of them summarized in [35], have been suggested to combat permanent faults in Networks-on-Chip. They are applied at different levels of a NoC design: ranging from entire network regions (entire routers or links) down to individual wires of a link, a single buffer or crossbar of a router. Some techniques modify the routing of packets to bypass faulty links and routers, others divide routers or links to smaller parts and isolate, spare them or share them to tolerate permanent faults. In general, the finer the granularity of the considered fault-model the better the fault-tolerance, but the higher the design overhead. Bellow, a brief overview of such techniques is presented elaborating more on the techniques most relevant to our approach.

Many approaches modify the routing algorithm of a NoC to be dynamically adaptive and avoid faulty links or routers [2, 13, 30]. Faults on links or routers are broadcasted to all nodes in order to update their routing tables and avoid damaged network parts [2]. Packets are sent across multiple non-intersecting paths to ensure that they are delivered correctly [30]. Feng et al. introduced deflection routing where the packets are routed based on a defect map of the neighboring links and switches [13]. In general, methods that adapt the routing according to the faulty components of the network introduced low area and power overheads (~5% in [30]), however they provide limited fault-tolerance. On the other hand, a routing algorithm based on reinforcement-learning improves fault-tolerance but suffers from high area and power overheads (90% and 120%, respectively) [13].

Other techniques re-route packets without modifying the actual routing algorithm. Instead, they perform a detour selecting an intermediate destination for packets that would normally need to pass through some faulty links or routers. Although detour does not change the routing algorithm, deadlocks need to be reconsidered in case packets are not stored entirely in the intermediate destination. ReliNoC performs detour using a Logic-based Distributed Routing table proposed by [36], [22]. Routing reconfiguration through intermediate destinations uses the turn model to avoid deadlocks in [14]. Han and Fu have also described a detouring technique for a 2D mesh with XY-routing and avoid deadlocks considering the turn-model [19]. Furthermore, Vitkovskiy et al. proposed a technique where detour is handled locally at network regions that suffer from faulty resources; a packet entering such region is redirected through an alternative path [45]. In general, detour introduces low area and power overheads (less than 10%), but the performance decreases rapidly with increasing number of faults, since even partially faulty components are omitted from network resources.

Besides rerouting of packets, faulty network parts can be bypassed in hardware. An alternative path from an input port to the output is employed to bypass a faulty router in [25]. A faulty crossbar is bypassed in a similar manner in Vicis [15]. Finally, a single input buffer can be bypassed in RoCo [23]. In order to contain a fault in the router, RoCo additionally splits the router in two distinct parts responsible for either row or column traversal of packets.

Evripidou et al. exploited the redundancy offered by Virtual Channels (VC) and proposed renaming of the VC buffers [10]. They modified the input ports of a router to be able to avoid faulty buffers and redirect traffic classes through the remaining available VCs. ReliNoC [22], considers a NoC that has two separate physical networks one for traffic that requires some QoS and one for normal traffic. ReliNoC considers that the two physical networks have interchangeable links and routers. In essence, a faulty link or router of a physical network can be replaced by the equivalent part of the second network. As a consequence, upon the occurrence of permanent faults some links or routers need to accommodate both traffic classes.

At the router level, Vicis reduces the impact of faulty input and output ports using port swapping [15]. This technique allows to pair on-demand input and output ports of neighboring routers in order to increase the connectivity of the network at the presence of faults. Furthermore, Liu et al. partition links and router data-path in four slices, then, only the fault-free slices are used in a time-division multiplexing manner [29]. Besides the performance overhead, slicing has a significant area cost (about 65%).

The above approach by Liu et al. allows, among others, to use partly faulty links. Another approach for protecting links is proposed by Lehntonen et al.; they use spare wires at the links to replace faulty ones [28]. Moreover, Vitkovskiy et al. used shifting and retransmission of flits to communicate correctly data across partially faulty links [45]. In doing so, they can tolerate up to 50% faulty wires on a link, suffering however, the performance overhead of one retransmission. Both approaches have high area cost (about 75% and 30%, respectively).

Finally, BulletProof [7] router introduced by Constantinides et al. described a method to automatically inject sparing logic to the netlist of a router and be able to replace faulty circuitry [7]. However, this method has a substantial area overhead of up to 3.4×.

Besides ReliNoC, none of the above techniques is explicitly addressing fault-tolerance on a service-oriented NoC nor analyzes the affect of faults to the performance of individual traffic classes. Even ReliNoC considers only two classes of traffic (one for QoS and one for best-effort), not discussing scalability to a higher number of services. Supporting only one QoS traffic class together with best-effort traffic is expected to simplify arbitration and flow control, although these aspects of ReliNoC are not elaborated [22]. Moreover, Virtual Channel renaming has some similarities with the proposed Service Merge [10], however, it can be applied to a NoC with VCs rather than a service oriented NoC with separate router data-paths such as the QNoC. As a consequence, VC renaming protects only the VC buffers rather than an entire router data-path. It further increases the buffer complexity and size significantly, while priorities between physical buffers are fixed. On the contrary, this work offers a complete solution for tolerating permanent faults on a service-oriented NoC, which is scalable to multiple traffic classes, presenting flow control and credit handling modifications to support service redirection.

3.2 The QNoC Architecture

We use the QNoC architecture as our basis to apply our fault-tolerant techniques [4]. QNoC can be positioned between two extreme approaches for providing service-oriented interconnects. The first one is using different virtual channels for different traffic classes. At the other end of the spectrum, iMesh uses separate physical networks per service [47]. iMesh may not increase significantly the area cost of the routers compared to a network with virtual channels and wiring of separate links per service may be cheap in current technologies, however, the power consumption of a NoC with multiple links is expected to be increased. QNoC offers separate router data-paths per service however links are shared between all services and are allocated using priority arbitration at the output ports. Gilbert et al. showed that a NoC architecture with shared links and separate router data-paths per service can be more efficient in terms of area, power and performance compared to using virtual channels [16].

Table 3.1: *Specification of four considered traffic classes*

Traffic Classes	**Traffic Classes**	**Average Packet Size [flits]**	**Proirity**
Signaling (Control signals, interrupts)	5% (Very Low)	5 (Very small)	Very High
Real-Time data (Streaming)	15% (Medium)	20 (Large)	High
Single Read/Write	10% (Low)	10 (Medium)	Low
Block Read/Write	70% (High)	20 (Large)	Very Low

QNoC is a packet switched network composed of routers with service flow control interconnected on a 2-D mesh topology. It uses XY routing with multi-class wormhole routing [4]. It supports four distinct services (SVCs) to accommodate equal number of traffic classes for *signaling* (control signals, interrupts), *real-time* data, *single read/write* (short data access) and *block read/write* (large data access) as shown in Table 3.1. A router connects to each neighbor with two 40-bit links (32-bit data, 8-bit control), one per direction. It further provides four local connections, one per SVC, through a Network Interface (NI) to the local component(s). A hop-by-hop credit based flow control is used; each router sends credits per SVC to its neighbors corresponding to available input-buffer slots. XY routing guarantees deadlock freeness. Starvation is avoided by setting boundaries for maximum traffic in each SVC, i.e. services of high priority are set to have a low upper bound of traffic load and vice versa [4].

As illustrated in Figure 3.1, a QNoC router has three stages: input, switching, output. At the input stage, flits are sent to the corresponding input buffer based on their SVC. In case of a header flit, the output port is computed at the Routing Computation (RC) unit and stored for the remaining flits of the same packet in the Current Routing Table (CRT). Subsequently, flits are switched from the input buffer to the selected output port buffer, through a crossbar (one per SVC) according to the CRT information. When multiple packets of a single service, located at different input ports, compete for the same crossbar output, a *round-robin* mechanism is used for fair arbitration (FA). At the output, the output buffer stores flits ready to be sent. The SVCs are then multiplexed and access the output port based on their priority and their available credits. In summary, each SVC has a separate data-path in the router and the only logic shared between them is related to the priority arbiter at the output. In our implementation, flits are 40-bits, input buffers store 2 flits and output buffers store 1 flit.

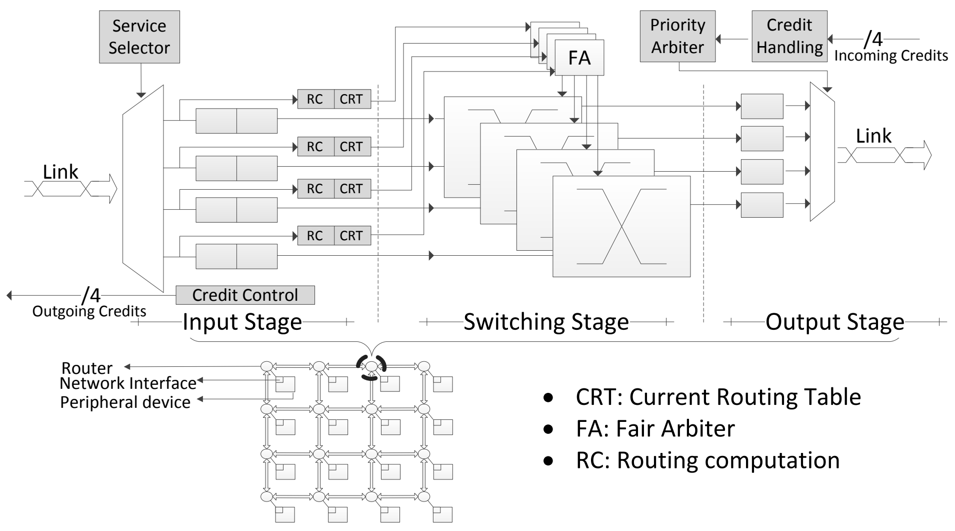


Figure 3.1: *The architecture of QNoC.*

3.3 RQNoC: a Resilient QoS NoC

Tolerating permanent faults in a service-oriented NoC can be achieved considering each SVC separately. Faulty SVC resources can be bypassed either allowing the respective traffic class to be redirected using an alternative path of the same service, called *Service Detour (SDetour)* or alternatively redirect it through the router data-path of another service, called *Service Merge (SMerge)*. Each of the two options have their advantages and disadvantages. Service detour prevents services to interfere with each other, but increases the latency of traffic that belongs to the faulty service. Service merge has a lower overhead as the packet route remains the same, however, it allows services to share the same router data-path and consequently affect each other's performance. In order to prevent in an RQNoC with SMerge traffic classes with hard QoS requirements to share router-resources with other traffic, we further introduce Service Renaming, which allows dynamic assignment of router data-paths to a particular traffic class. To provide a complete fault-tolerant solution, the remaining network resources that are common for all services need to be protected, too. Links can tolerate faults by adding to a shifting technique spare wires, thereby increasing the number of faulty wires they can sustain. Finally, the common control of a router[1] is protected using Triple Modular Redundancy (TMR). Next, we describe separately each proposed technique.

[1] Practically the service selector in the input ports and the priority arbiter in the output ports.

3.3.1 SDetour: Service Detour

Detour is a well-known technique for mitigating permanent faults at links and routers. It allows to bypass faulty network regions without however modifying the routing algorithm. This simplifies the design compared to adaptive routing techniques for fault-tolerance. We describe first the general (global) detour mechanism applied in the proposed RQNoC and subsequently explain how it is selectively used per service.

Packets that, according to the routing algorithm, need to traverse a path through damaged links or routers, can still reach their destination taking a detour through an intermediate node. Such cases are detected at the source NI using a detour table, which is computed based on the defect map of the NoC. Before a packet is sent to its destination, the detour table is consulted to check whether there is a detour to be taken. In such case an additional *detour-header* flit will be added before the original header flit indicating the intermediate node as the first destination. When the detour header flit reaches the input port of the intermediate destination, it is discarded and the subsequent original packet header is used in the remaining path of the packet to its final destination. This approach can scale to multiple intermediate destinations as multiple header flits can be added to a packet and one-by-one be discarded at each intermediate node.

There might be several candidates to be used as intermediate nodes. We choose the one that minimizes the path and avoids deadlocks. XY routing is deadlock-free, however detoured packets are not entirely stored-and-forwarded at the intermediate node, for performance reasons, and therefore may create dependency loops. Such loops are avoided considering a turn model. As depicted in Figure 3.2, the additional turns introduced by detour should be the ones of West-First turn model, a subset of which is XY routing. As a consequence, intermediate nodes are selected so that the packets use only the turns allowed by the turn model. The routing algorithm (in our case XY) and the restrictions posed by the turn model may not allow a destination to be reached although it may still be connected to the rest of the network. To exemplify, Figure 3.4 shows that a packet from any node (X,Y), where Y≤2, cannot be sent to (0,0) when the vertical link between (0,3) and (0,2) is broken, although the node is still connected to the rest of the network; that is because the North-West turn is not allowed by the turn model. This introduces a disadvantage compared to the subsequent SMerge technique described in the next subsection. It should be noted that deadlocks due to detour could be avoided by storing the entire packet at the NI of the intermediate node before forwarding it to its final destination, however this would significantly increase communication latency.

In our approach faults are detected in the granularity of a service (router data path

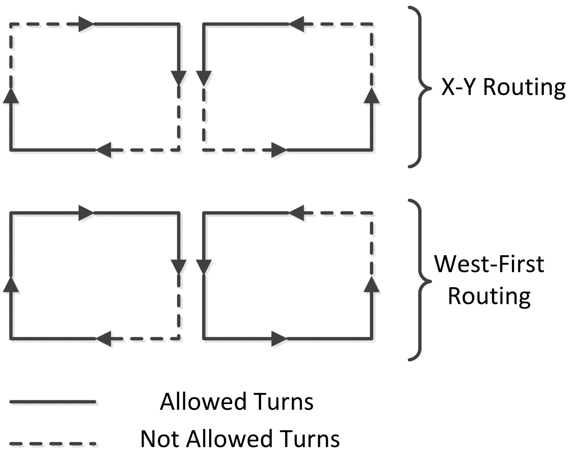


Figure 3.2: *Possible turns in X-Y routing and West-First turn model.*

dedicated to a particular SVC). Consequently, upon the detection of a fault, only the traffic class belonging to the faulty SVC needs to be detoured, the remaining traffic can still use the original paths. This is achieved by modifying the detour table at the NI to store intermediate destinations per service. Figure 3.3 illustrates an example of service detour as described above. Packets sent from node (0,3) to (1,1) are routed through node (1,3) using XY. In case the data path of service 3 in router (1,3) is faulty, then service 2 is detoured through (0,2). The traffic of the remaining services still enjoy the original route. Service detour does not introduce any additional turns. As described in the previous paragraphs intermediate nodes are selected according to the West-first turn model.

In the worst case, SDetour may double the number of hops a packet needs to take from its source to a destination and consequently may double worst case packet latency. This is taken into account for traffic classes with hard packet latency requirements as follows: an intermediate node is selected that does not exceed the maximum allowed number of hops of a particular traffic class. In case that is not possible then the system should need seek other mitigation techniques described below, or discard (if possible) the respective destination node in order to avoid system failure.

3.3.2 SMerge: Service Merge

The redundant data-paths per service in a QNoC router can be exploited for fault-tolerance. As opposed to service detour, this approach allows a router with a faulty service to still be used by the respective traffic class redirecting (merging) it to the

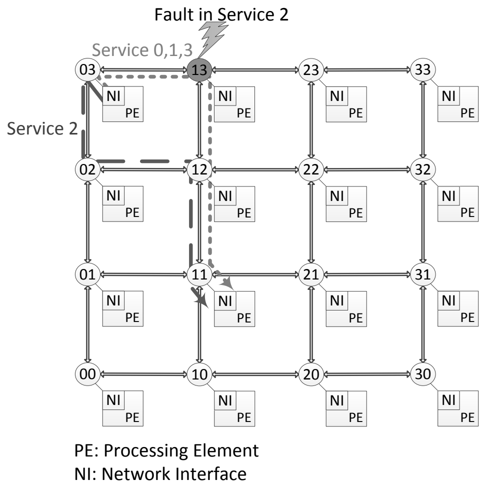


Figure 3.3: *Basic concept of SVC detour.*

data-path of another service. Each service data-path in the proposed RQNoC router is able to host the traffic of any other service. Such data-path is called host and the services it hosts are called guests. In general, an RQNoC router can be configured to host any combination of guests in each service data-path. In the extreme case, a single data-path may host all four services.

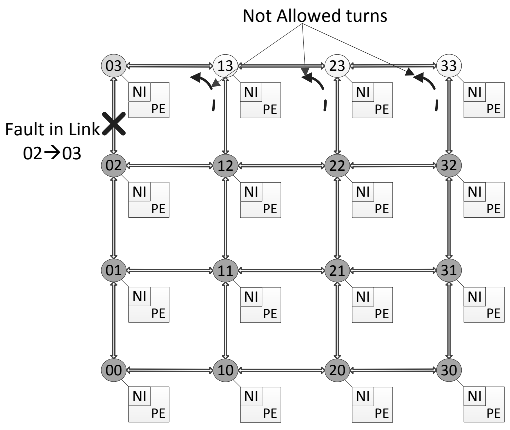


Figure 3.4: *Example of a fault not mitigated by detour.*

Router modifications

In order to support SMerge, the architecture of the original QNoC router needs to be modified as follows. A router needs to be modified at the *input port* to be able to merge packets of different services to a single data-path and at the *output port* to send them out separately based on their original SVCs. In addition, each *neighboring router* needs to be modified in order to handle correctly the credits received. Different credits correspond to the input buffers of different SVCs, consequently, credits of faulty SVCs should be ignored. The configuration of a router, indicating which services are merged through which data-paths, is encoded in a configuration array (CA). In the following, these modifications are detailed further using the example of Figure 3.5. In this example, two service merge cases are shown: (i) in router-1, SVC2 uses data-path of SVC3 and (ii) in router-2 SVC1 uses the resources of SVC2.

At the *input port* of the router, each arriving flit is sent through the SVC multiplexer to its corresponding SVC data-path, in particular to its input buffer. In the RQNoC, the select of the SVC multiplexer is configurable (through CA) in order to allow the service merge. Moreover, in order to keep track of the original SVC of the flits and be able to distinguish them at the output, each flit is tagged with its original SVC-id. This adds 2 extra bits in the data-path.

After flits are guided to a particular SVC data-path and stored to the corresponding input buffer, they will be switched to the correct output port as described in the QNoC: RC computes the output port when the header flit arrives and updates the CRT which is used to route the upcoming flits of the packet. Each SVC data path has its own crossbar which uses fair arbitration. In our current implementation, even when multiple traffic classes are hosted in the same SVC data-path, switching is still performed with fair arbitration.

At the *output port*, the flits are stored in the output buffer and wait for the permission to be sent out. A flit is sent out based on its priority and the available credits in the corresponding credit counter. For service merge, the *priority arbiter* should be modified. Originally the id of the output buffers (each corresponding to a service data path) was used to grant access to the link, however, in the RQNoC flits of different traffic classes may share the same data-path. Consequently, in order to maintain priority arbitration between the original services (even when sharing the same data-path) we modified the logic of the arbitration decision to be based on the original SVC-id appended to each flit.

Additional changes are required so that neighboring routers can send packets to

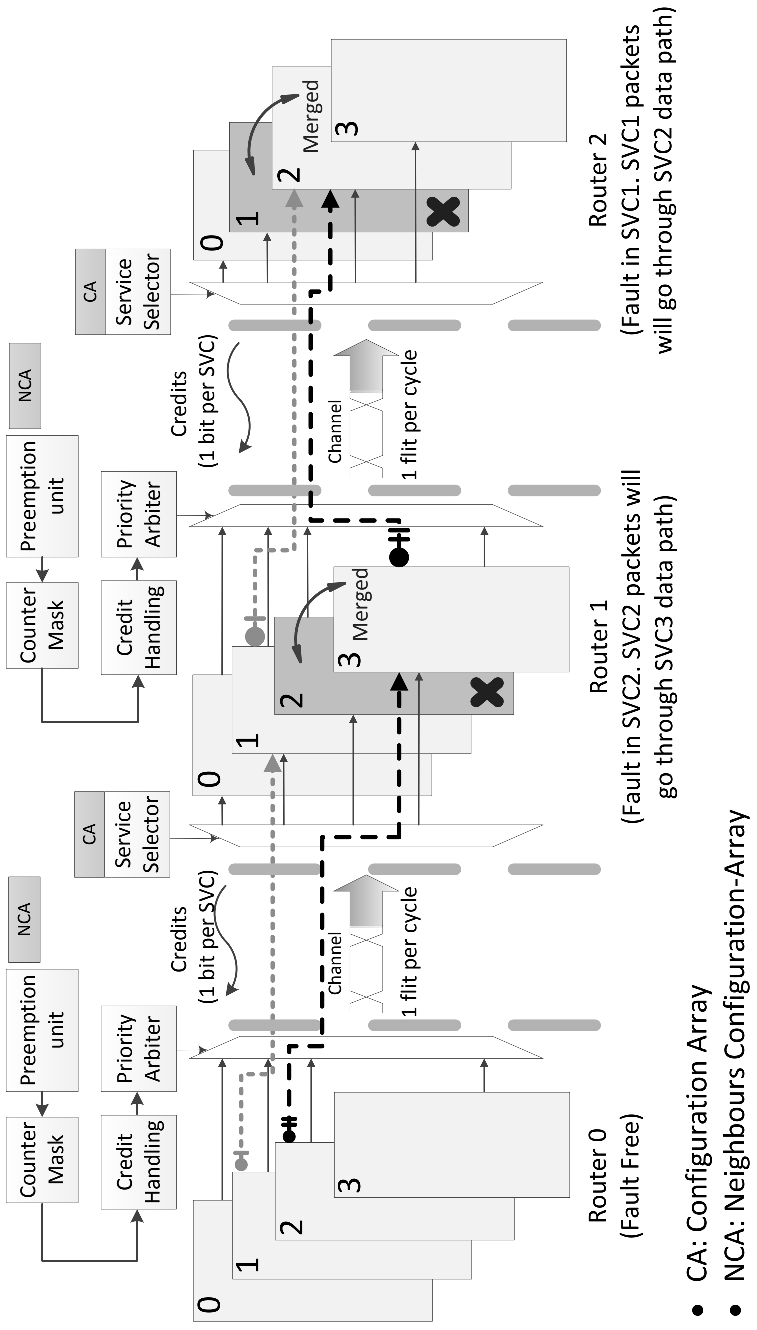

Counter Mask
Preemption unit
NCA
Credit Handling
Priority Arbiter
CA
Service Selector
Credits (1 bit per SVC)
Channel
1 flit per cycle
0
1
2
3
Merged
Router 0 (Fault Free)
Router 1 (Fault in SVC2. SVC2 packets will go through SVC3 data path)
Router 2 (Fault in SVC1. SVC1 packets will go through SVC2 data path)
• CA: Configuration Array
• NCA: Neighbours Configuration-Array

Figure 3.5: *An example of service merge.*

a faulty router (for correct credit handling). Neighboring routers should adapt to the configuration of the faulty router, discarding credits that belong to faulty paths and taking into account which traffic classes are merged to a specific SVC data-path. In order to support the above, the *credit handling* module is changed and two new modules are introduced at the output port, namely: the *Preemption unit* and *Counter-mask*.

In credit-based flow control, each SVC has a dedicated credit counter at each output port which keeps track of the credits received from the immediate neighboring router and used in a service. Each counter indicates the available SVC input-buffer space of the next router. The credit counter corresponding to a faulty SVC data-path of a neighboring router should be masked. This is achieved by the *counter mask* module. In the example of Figure 3.5, the credit counter of SVC2 in router-0 will be ignored and the counter mask module will redirect the priority arbiter to look at the credit counter of SVC3 instead; this is because in router-1 traffic of SVC2 is redirected to the data-path of SVC3. Traffic of SVC3 will still use its original credit counter.

Credit Handler increments and decrements the credit counters based on the credits received and the flits sent. When multiple traffic classes are merged to the same SVC data-path the credit handler needs to consider that as follows: received credits that correspond to a faulty SVC data-path are ignored, when sending a flit that belongs to a redirected SVC, the counter of the host SVC should be decremented. For example, in router-0 sending a flit of SVC2 will decrement the credit counter of SVC3, as in router-1 it is mapped to the SVC3 data-path, in addition credits that belong to SVC2 will have no effect in the flow control.

Finally, it should be noted that flits of different services cannot be interleaved when sharing the same SVC data-path. That is due to a packet-integrity check performed at the input port, which ensures that flits of a packet come in order. Then, mixing packets of different services at the same SVC data-path requires an additional mechanism to ensure that. The *Preemption unit* module, guarantees that a neighboring router will send to a faulty router all flits of a packet that belongs to a specific traffic class, before selecting flits of other class that shares the same SVC data-path. In the example of Figure 3.5, router-0 will send to router-1 an entire packet of SVC2 and then select flits of SVC3 as both traffics are mapped to SVC3 in router-1.

To exemplify credit handling in RQNoC with service merge we discuss in more detail the connection between router-1 and router-2 of Figure 3.5. SVC2 in router-1 is faulty and therefore the traffic classes of service 2 and 3 share the data-path of SVC3. SVC1 in router-2 is faulty which forces traffic classes of service 1 and 2 to share the data-path of SVC2. Figure 3.6 shows in detail how the credit counters in router-1 are associated with each traffic class. Each counter is associated with one SVC in the next router, in this case router-2. Consequently, the counter-1 in router-1 is ignored as SVC-1 in router-2 is faulty. Then, in router-1, Packets of service 2 (which are hosted in data-path SVC3) and packets of service 1 in SVC1 will consult counter-2 as in the next router they both share SVC2. In addition, packets of service 3 and packets of service 1 will consult counters 3 and 1, respectively, as they use their original data-paths in router-2. In summary, the credit counter associated with a traffic class is determined by the NCA, while the data-path where the packets of a class are located is determined by the CA.

Service renaming

An RQNoC with SMerge allows different traffic classes to share router resources. This may not be acceptable for some high priority traffic classes such as SVC0 (Signaling) and SVC1 (Real-time data). In order to overcome this problem the proposed RQNoC router architecture allows the router data-paths to be dynamically assigned to a traffic class via Service renaming. In practice, this is supported by the credit handling mechanism, which reassigns the credit counters, and by the service selector at the input ports.

Service merge policy

A faulty service data-path is merged with the data-path of the next lowest priority. This ensures that higher priority services are not affected. At the output port arbitration of merged traffic is performed considering the original packet priorities. The only exception to this policy is when the lowest (SVC3) or the highest (SVC0) priority traffic is faulty. In the first case, SVC3 is merged with the next available higher priority service data-path. Even so, arbitration in the output port is performed considering the original packet priorities. In the latter case, when faulty the highest priority SVC0 should not be merged for QoS reasons. It is then renamed to an available router data-path and lower priority SVCs are merged together. For an RQNoC that requires its highest priority traffic class to be isolated, each router should have at least 2 out of its 4 data-paths available or otherwise an SDetour should be taken as explained below.

3.3.3 Combining SDetour and SMerge

An RQNoC may use either SDetour or SMerge to mitigate faults or combine them both to maximize fault tolerance. In the latter case, a fault is first attempted to be tolerated with SMerge and if this is not possible then SDetour is employed. It would be inefficient to apply the two techniques in the opposite order. Applying SDetour first to bypass a faulty service on a router would prevent the local port from sending traffic of that service and therefore reduce network connectivity from the first fault. On the contrary, applying first SMerge allows the packets of a faulty service to be injected through another SVC data-path. All three alternative RQNoC designs provide fault tolerant links as described next and protect common router control logic with TMR.

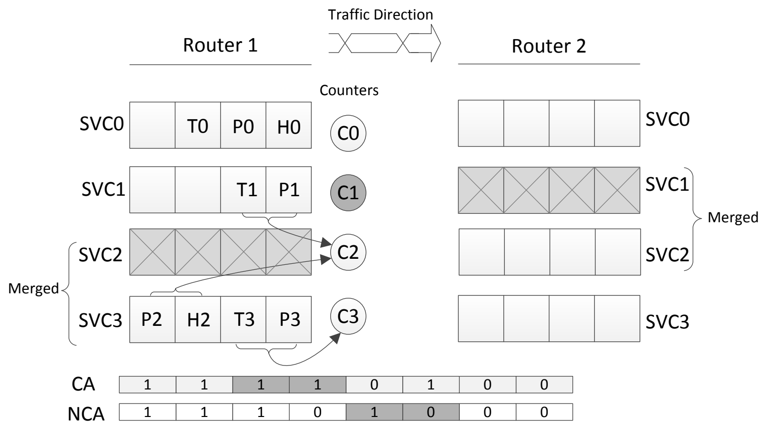


Figure 3.6: *Credit handling example in a RQNoC router with Service Merge.*

Configuration arrays

At each port of the router an 8-bit configuration-array (CA) is stored. Service merging decisions are made off-line and the information about the current status of each SVC is stored in the CA at each router. Figure 3.7 illustrates the default format of the CA. Each 2-bits of CA are dedicated to one SVC ,i.e. bits 7-5 for SVC3, 6-4 SVC2, 3-2 SVC1, 1-0 SVC0. The value of each 2-bits indicates the SVC data-path that host the corresponding traffic class. For instance, a CA configuration of 11101000 indicates that SVC0 and SVC3 use their own resources, SVC1 and SVC2 use the SVC2 resources. Each router is also aware of the CA of its neighboring routers in order to send correctly packets to faulty routers. We call this *Neighbors Configuration-Array* (NCA). The NCA is a copy of the CA of the immediate neighboring router.

3.3.4 Resilient Links

A single faulty wire may render an entire link unusable. In order to tolerate faulty wires at a link and use with a degraded bandwidth, Vitkovskiy et al. suggested shifting and

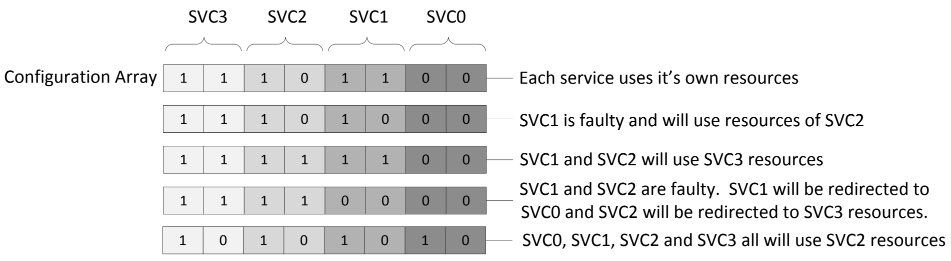

SVC3
SVC2
SVC1
SVC0
Configuration Array
1 1 1 0 1 1 0 0
Each service uses it's own resources
1 1 1 0 1 0 0 0
SVC1 is faulty and will use resources of SVC2
1 1 1 1 1 1 0 0
SVC1 and SVC2 will use SVC3 resources
1 1 1 1 0 0 0 0
SVC1 and SVC2 are faulty. SVC1 will be redirected to SVC0 and SVC2 will be redirected to SVC3 resources.
1 0 1 0 1 0 1 0
SVC0, SVC1, SVC2 and SVC3 all will use SVC2 resources

Figure 3.7: *Configuration Array.*

retransmission of flits traversing a partially faulty link [45]. Thereby, the flit data will be able to be sent across the fault-free wires after multiple retransmissions. The number of retransmission required depends on the number and location of faulty wires. A flit needs to be shifted by one bit and retransmitted to tolerate one faulty wire and in the best case up to 50% of the wires being faulty; this reduces the link bandwidth to half. Consecutive fault wires have a more severe impact on performance. In general, *N* consecutive faulty wires require *N* shifting and retransmissions of the same flit substantially affecting the bandwidth of the link. In order to reduce the performance overhead of the shifting method we added two spare wires to each link. In effect, this allows to tolerate two faulty wires before degrading link bandwidth at all.

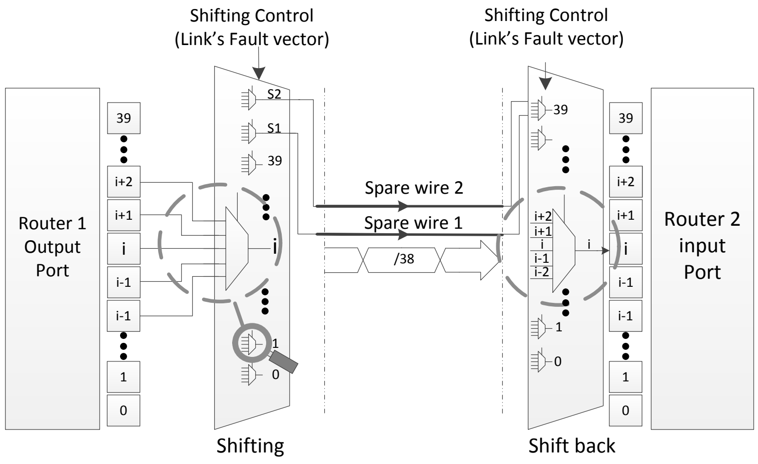


Figure 3.8: *Two spare wires and a shifting mechanism can tolerate up to 2 faults without compromising link performance. More faults are tolerated by shifting and retransmission at reduced link bandwidth.*

Figure 3.8, shows the design of the links used in our approach providing spare wires and shifting. At every output port of a router, the 40 bits of a link provide input to 42 5-to-1 multiplexers in order to support shifting. In particular, the i^{th} multiplexer receives input from bits $i-2, i-1, i, i+1, i+2$ and the multiplexers corresponding to the spare wires have the same inputs as the 39^{th} and 40^{th} multiplexer. At the other end of the link 5-to-1 multiplexers are used to shift back transmitted data to their original form. When retransmission, the output buffer is controlled to keep the transmitted flits and the input buffer is controlled to overwrite the respective bits of the flit. We do not allow more than one retransmission, as this would need additional hardware changes in order to shift the previously shifted values, requiring the insertion of registers at the two ends of the links.

3.3.5 Fault Model, Diagnosis and Reconfiguration

From the above techniques it is evident that our fault model distinguishes faulty resources – due to manufacturing defects or aging faults – in the following granularity: single router data-path (dedicated to one service), router control, and individual link wires. Such faults are permanent, manifested first as (repeated) transient faults which subsequently trigger an online diagnosis test. Consequently, detection mechanisms are required to determine whether each one of the above network parts are faulty.

RQNoC packets are protected with Error-Detection Code (EDC) which are checked at each input and output port of a router, similar to [17, 24]. This allows us to locate the particular link or router data-path where the fault occurred. Each EDC checker in addition provides a simple mechanism to keep track of multiple faults using a simple Finite State Machine (FSM), similar to [24]. When multiple repeated faults are detected on a link or a router data-path, then a test packet is generated by the NIs of the respective and a neighboring router(s)[2]. The neighboring routers are informed to send test packets via dedicated wires. A router in test mode will have a fixed control directing the test packets to the correct output port as it cannot rely on the test packet. Test packets for links and router data-paths are constructed considering the test vectors of [45] in order to diagnose particular faulty wires on a link. The same test pattern is also used for testing the data-paths. During testing, the same test packets are replicated at the demultiplexer of the input port to all four service data-paths; thereby the control logic which is repeated per service (RC, CRT, FA) can be tested. Finally, faults at the common parts of the router control are detected via TMR and an FSM is used to detect repeated faults. Then, a

[2]A faulty link requires a test packet from the neighboring NI and a faulty router data-path from the NIs of all neighbors as well as the local NI towards the output port that reported multiple detected faults.

consistently faulty copy of the control is disabled; at a second permanent control fault the entire router is considered faulty.

All three techniques presented above require a way to modify the configuration of the network. Service detour requires the detour table at the NI to be updated according to the status of network resources. Service Merge needs the configuration arrays of the faulty router and its neighbors to be updated. Also the link configuration needs to be updated in order to bypass the faulty wires. This is performed via control packets generated and broadcasted at the NI receiving the test packets. In cases where this is not possible (the faulty router cannot communicate the results of a test), a time out of the neighbors that generated the test packets will declare the router under test faulty and update their configuration accordingly.

3.4 Evaluation

In this section, we evaluate three variations of the proposed RQNoC. We measure networks performance and fault tolerance, in terms of network connectivity preserved in the presence of faults, all contributing to the QoS provided by the networks. Moreover, we retrieve post-synthesis results analyzing the overheads of our techniques in the operating frequency, power consumption and area of the QNoC. The RQNoC designs are compared to the baseline QNoC and a QNoC with global detour.

3.4.1 Implementation Results

We have implemented in RTL three 4x4 RQNoC 2D mesh designs that provide (i) service detour (RQNoC SDetour), (ii) Service Merge (RQNoC SMerge), or (iii) both (RQNoC SMerge+SDetour). We further implemented a baseline 4x4 QNoC 2D mesh (QNoC baseline) and a QNoC providing global detour[3] (QNoC GDetour). All designs, except the baseline, tolerate permanent faults at the links and router-control as described in Section 3.3.

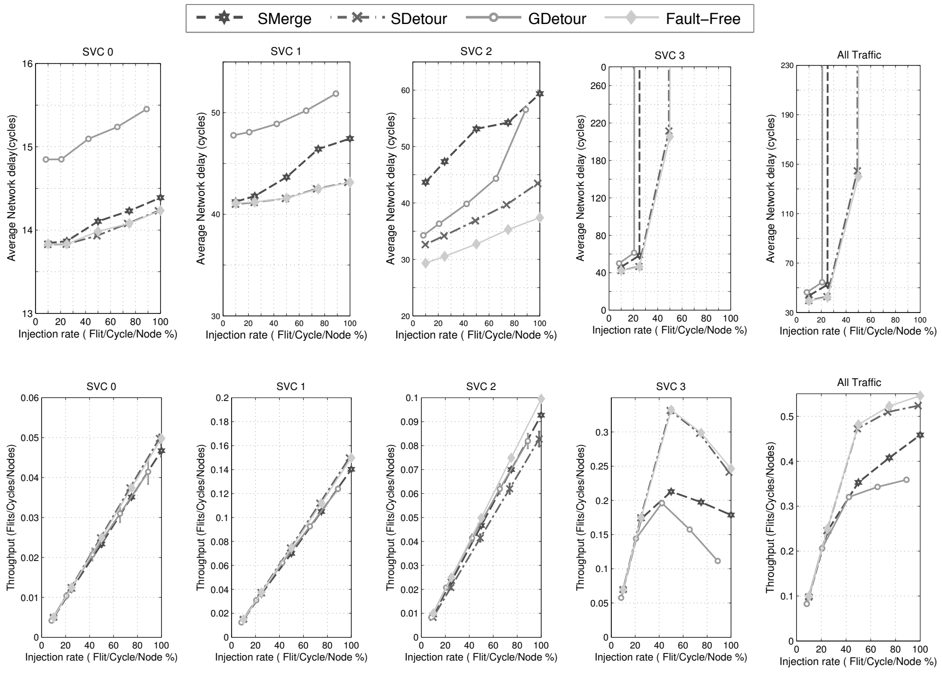
The designs are implemented using ST 65nm low-power library in Synopsis Design compiler. Power consumption is measured by simulating the designs netlists for 1ms. Table 3.2 summarizes the overheads in operating frequency, power consumption and silicon resources of the three alternative RQNoCs versus the baseline QNoC. The QNoC GDetour has the same area and frequency with the RQNoC SDetour and is therefore omitted from the table. Employing Service detour increases the area by 24.3%; $\frac{2}{5}$ of it

[3]All services are redirected the same way.

Table 3.2: *Implementation results of the QNoC and RQNoC designs.*

Design	**Power (Watt)**	**Area (μm^2)**	**Frequency (MHz)**
Baseline QNoC	1.63	946k	1100
RQNoC SMerge	2.06 (+26.6%)	1428k (+50.8%)	1000 (-9.1%)
RQNoC SDetour (also QNoC GDetour)	1.74(+6.8%)	1117k (+24.3%)	1100
RQNoC SMerge + SDetour	2.14 (+31.6%)	1476k (+55.8%)	1000 (-9.1%)

is due to the fault tolerant links, $\frac{2}{5}$ due to the triplication of the common router control parts, and the remaining $\frac{1}{5}$ due to the changes in flit handling for the additional detour header. SDetour power consumption is increased by 6.8% and its operating frequency is equal to the baseline. SMerge introduces a higher area overhead of 50.8% primarily due to the changes in the control needed for merging services and to a smaller extend due to a few data-path additions for tagging packets of different services. SMerge power consumption increases compared to the baseline by 26.6% and its clock rate decreases by 9.1% due to the complexity of the router control. Finally, an RQNoC integrating both SDetour and SMerge requires 55.8% additional resources mainly due to resources and consumes 31.6 more power, having 9.1% slower clock.



Packet completion rate

SVC0

- ✡ -	-×-	-o-
100%	100%	82%

SVC1

- ✡ -	-×-	-o-
100%	100%	82%

SVC2

- ✡ -	-×-	-o-
100%	82%	82%

SVC3

- ✡ -	-×-	-o-
100%	100%	82%

All Traffic

- ✡ -	-×-	-o-
100%	99%	82%

Standard deviation (σ), Maximum and Mean values for the packet latency (cycles)

SVC0

	- ✡ -	-×-	-o-	-◆-
σ*	8.8	4.5	5.7	4.5
Max*	48.5	27.5	31.5	27.5
Mean*	14.4	14.2	15.5	13.9

* At the highest injection rate

SVC1

	- ✡ -	-×-	-o-	-◆-
σ*	38.8	20.4	30.9	20.4
Max*	193.5	104.5	144.5	104.5
Mean*	47.5	43.3	51.9	43.3

SVC2

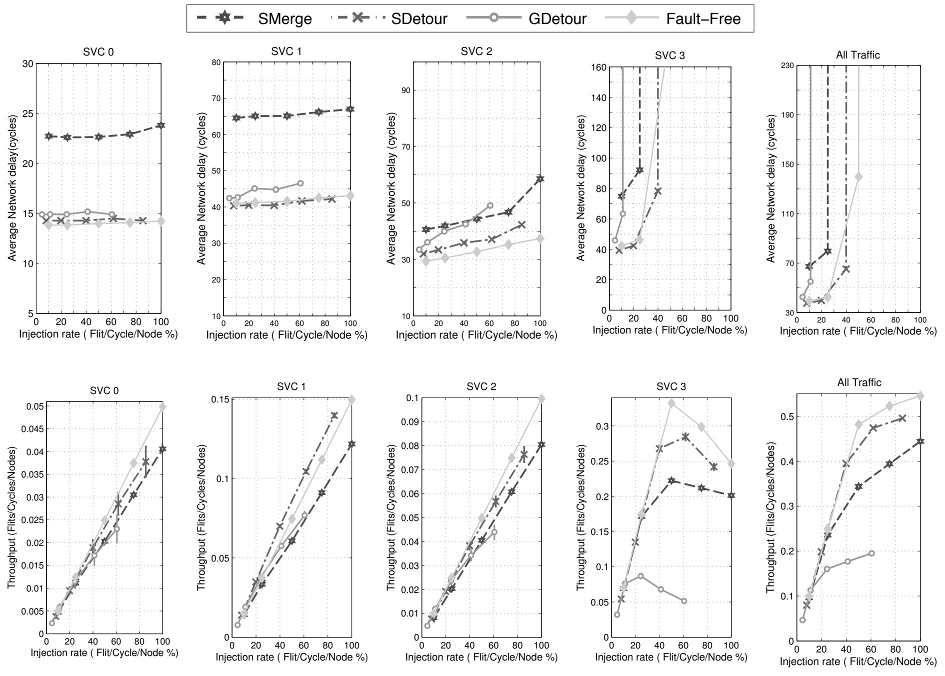
	- ✡ -	-×-	-o-	-◆-
σ*	41.1	25.7	45.2	20.0
Max*	180.5	119.5	190.5	96.5
Mean*	59.4	43.3	55.7	37.5

SVC3

	- ✡ -	-×-	-o-	-◆-
σ**	60.0	25.3	44.1	25.1
Max**	273.5	122.5	193.5	121.5
Mean**	58.4	46.7	61.2	46.6

** At the highest injection rate before saturation.

Figure 3.9: *Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 1 permanent fault.*



Packet completion rate

SVC0

- ✡ -	-×-	-○-
100%	75%	47%

SVC1

- ✡ -	-×-	-○-
100%	93%	47%

SVC2

- ✡ -	-×-	-○-
100%	76%	47%

SVC3

- ✡ -	-×-	-○-
100%	76%	47%

All Traffic

- ✡ -	-×-	-○-
100%	79%	47%

Standard deviation (σ), Maximum and Mean values for the packet latency (cycles)

SVC0

	- ✡ -	-×-	-○-	-◆-
σ*	13.6	6.2	6.9	4.5
Max*	64.5	32.5	35.5	27.5
Mean*	23.9	14.6	14.9	13.9

* At the highest injection rate

SVC1

	- ✡ -	-×-	-○-	-◆-
σ*	36.5	19.9	30.2	20.4
Max*	183.5	101.5	136.5	104.5
Mean*	74.8	42.3	46.7	43.3

SVC2

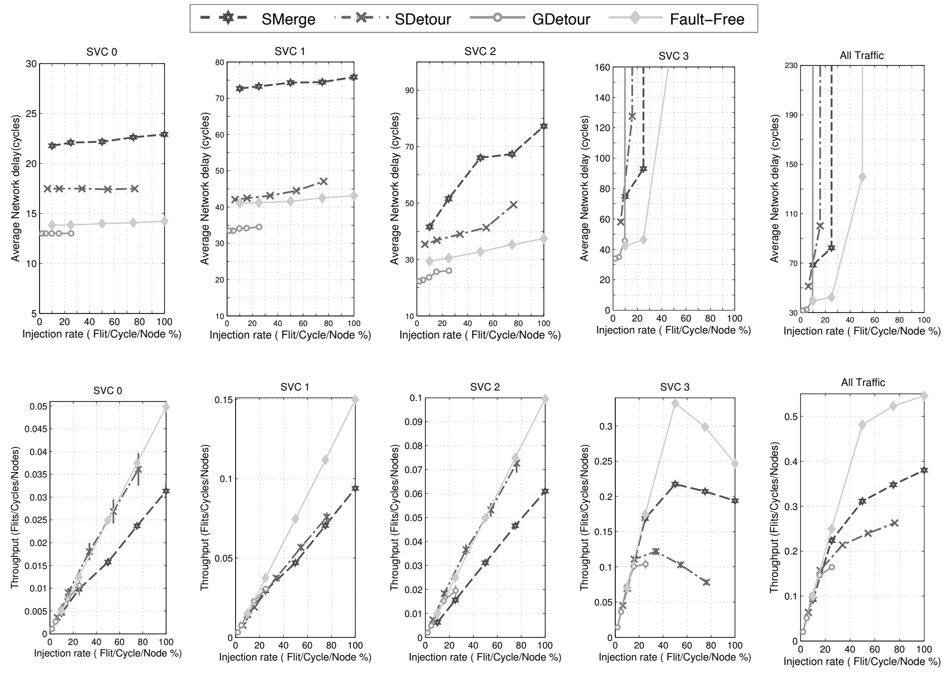
	- ✡ -	-×-	-○-	-◆-
σ*	47.0	27.0	176.0	20.0
Max*	194.5	122.5	643.5	96.5
Mean*	54.5	42.4	115.7	37.5

SVC3

	- ✡ -	-×-	-○-	-◆-
σ**	58.1	22.1	49.6	25.1
Max**	265.5	108.5	207.5	121.5
Mean**	91.6	42.5	59.0	46.6

** At the highest injection rate before saturation.

Figure 3.10: *Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 4 permanent faults.*



Packet completion rate

SVC0

–✶–	–×–	–○–
100%	71%	20%

SVC1

–✶–	–×–	–○–
100%	50%	20%

SVC2

–✶–	–×–	–○–
100%	73%	20%

SVC3

–✶–	–×–	–○–
100%	53%	20%

All Traffic

–✶–	–×–	–○–
100%	60%	20%

Standard deviation (σ), Maximum and Mean values for the packet latency (cycles)

SVC0

	–✶–	–×–	–○–	–◆–
σ*	11.3	8.2	4.4	4.5
Max*	55.5	41.5	25.5	27.5
Mean*	22.5	17.7	12.4	13.9

* At the highest injection rate

SVC1

	–✶–	–×–	–○–	–◆–
σ*	29.1	27.1	15.7	20.4
Max*	153.5	127.5	81.5	104.5
Mean*	66.7	46.1	34.6	43.3

SVC2

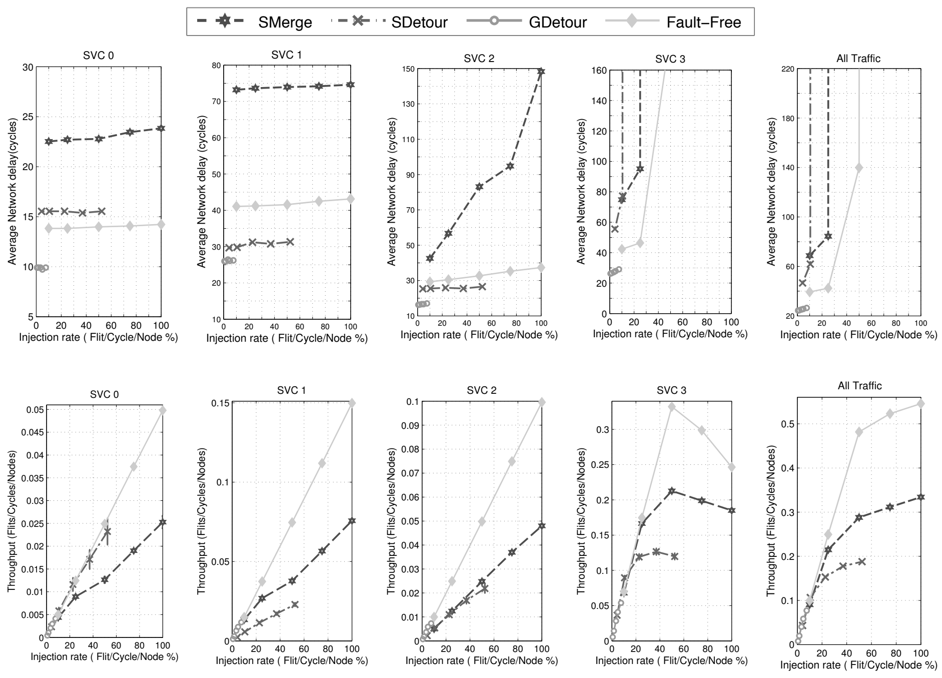
	–✶–	–×–	–○–	–◆–
σ*	61.5	36.5	15.6	20.0
Max*	258.5	157.5	73.5	96.5
Mean*	74.3	48.6	27.1	37.5

SVC3

	–✶–	–×–	–○–	–◆–
σ**	60.2	203.6	17.8	25.1
Max**	272.5	734.5	88.5	121.5
Mean**	92.4	124.7	35.7	46.6

** At the highest injection rate before saturation.

Figure 3.11: *Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 8 permanent faults.*



Packet completion rate

SVC0

✱	×	○
100%	45%	8%

SVC1

✱	×	○
100%	34%	8%

SVC2

✱	×	○
100%	21%	8%

SVC3

✱	×	○
100%	63%	8%

All Traffic

✱	×	○
100%	54%	8%

Standard deviation (σ), Maximum and Mean values for the packet latency (cycles)

SVC0

	✱	×	○	◆
σ*	8.2	6.9	1.8	4.5
Max*	45.5	35.5	13.5	27.5
Mean*	20.8	16.0	10.1	13.9

* At the highest injection rate

SVC1

	✱	×	○	◆
σ*	38.7	11.7	5.7	20.4
Max*	188.5	66.5	37.5	104.5
Mean*	72.5	31.5	26.3	43.3

SVC2

	✱	×	○	◆
σ*	155.1	12.4	5.4	20.0
Max*	606.5	63.5	33.5	96.5
Mean*	141.9	27.1	17.3	37.5

SVC3

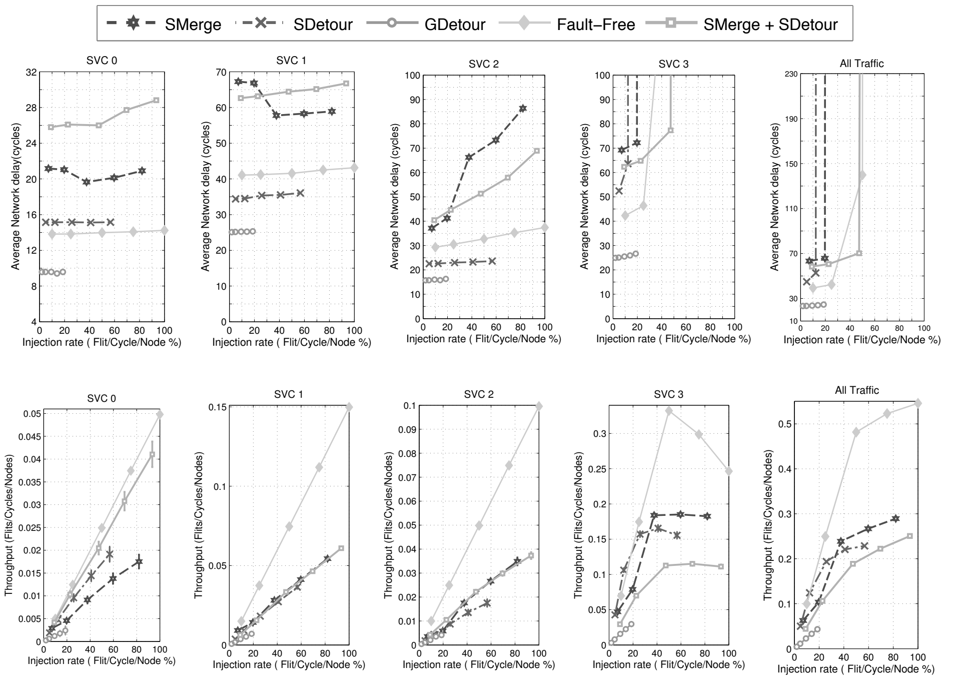
	✱	×	○	◆
σ**	64.0	81.7	6.7	25.1
Max**	286.5	321.5	46.5	121.5
Mean**	94.7	77.2	27.0	46.6

** At the highest injection rate before saturation.

Figure 3.12: *Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 16 permanent faults.*

3.4.2 Performance results

We evaluate the above 4x4 RQNoCs and QNoCs in terms of performance, measuring latency (in cycles), throughput (flits per cycle per node), and also the percentage of successfully received packets. The above indicate the preserved QoS of RQNoC in the presence of faults compared to the fault-free case. We consider that SVC0 traffic requires to always have separate router resources in order to avoid protocol-level deadlocks [32] and therefore apply service renaming to achieve this when SMerge is used. This has as a



Packet completion rate

SVC0

✱	×	○	□
65%	38%	4%	82%

SVC1

✱	×	○	□
66%	24%	4%	75%

SVC2

✱	×	○	□
70%	17%	4%	84%

SVC3

✱	×	○	□
76%	53%	4%	82%

All Traffic

✱	×	○	□
73%	45%	4%	83%

Standard deviation (σ), Maximum and Mean values for the packet latency (cycles)

SVC0

	✱	×	○	◆	□
σ*	1.8	3.5	1.8	4.5	15.8
Max*	13.5	21.5	13.5	27.5	73.5
Mean*	10.1	11.9	10.1	13.9	26.5

* At the highest injection rate

SVC1

	✱	×	○	◆	□
σ*	5.7	11.0	5.7	20.4	32.5
Max*	37.5	63.5	37.5	104.5	164.5
Mean*	26.3	30.7	26.3	43.3	67.2

SVC2

	✱	×	○	◆	□
σ*	5.4	8.9	5.4	20.0	65.4
Max*	33.5	48.5	33.5	96.5	269.5
Mean*	17.3	22.0	17.3	37.5	74.1

SVC3

	✱	×	○	◆	□
σ**	49.0	74.9	6.7	25.1	29.4
Max**	42.5	297.5	46.5	121.5	152.5
Mean**	78.8	73.3	27.0	46.6	65.0

** At the highest injection rate before saturation.

Figure 3.13: *Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 32 permanent faults.*

consequence each router to require at least two undamaged data-paths before it fails.

Our experiments have been performed simulating the RTL of the network designs for 1.5 million cycles each, having a warmup phase of 20 thousand cycles. Synthetic uniform random traffic is injected with up to one flit/cycle/node injection rate distributed among the four SVCs as described in Table 3.1.

We perform experiments injecting to the networks 0, 1, 4, 8, 16 and 32 faults, the location of which is randomly selected considering the silicon area of each network part. For each of the above fault densities, multiple runs have been performed[4] and the average results are plotted in Figures 3.9 to 3.13. In addition, standard deviation of throughput and packet latency, as well as maximum packet latency are presented. It is worth noting that, all networks have the same performance when fault free, which is also depicted in the figures for comparison.

We first measure the percentage of the packets successfully delivered to provide a better insight of the actual network load and a first estimate of fault tolerance[5]. The RQNoC SMerge+SDetour improves fault tolerance compared to RQNoC SMerge only in the case of 32 faults and therefore its performance is depicted only in Figure 3.13. In these experiments, RQNoC SMerge delivers successfully 100% of the injected packets for up to 16 network faults and about 70% for a network with 32 faults as some links or routers are entirely damaged. RQNoC SMerge+SDetour is able to mitigate some of the above faults increasing the successfully delivered packets to 83% as shown in in Figure 3.13. RQNoC SDetour follows in fault-tolerance delivering 99% to 45% of the injected packets for 1 to 32 faults. Finally, GDetour treats a partially faulty router as entirely damaged preventing traffic that would otherwise use it from being sent, even if it belongs to a fault-free SVC. As a consequence, QNoC GDetour accommodates 82% down to only 4% of the packets sent.

The performance of each network is only fair to be measured considering an injection rate after excluding the packets dropped at the network interface[6]. In so doing, we consider the actual load of the networks. The injection rate in Figures 3.9 to 3.13 refers to

[4]We have performed 4 runs for each particular number of faults (fault density), injecting faults to different (randomly selected) network locations and present their average results. We recognize that a particular technique may have different performance when injecting a specific set of faults, however exhaustive fault injection and network simulation would be practically impossible. Even so, our experiments show a clear trend in the performance and fault-tolerance of the described techniques, allowing us to derive useful conclusions about their efficiency.

[5]Fault tolerance is more accurately evaluated in the next Section 3.4.3 where half a million fault injections have been performed to measure the connectivity of the different networks. Even so, the percentages of successfully received packets presented in this subsection are quite close to the analysis of Section 3.4.3 and in most cases within a 10% margin from the estimated mean network connectivity.

[6]These are packets that require to pass through faulty network resources which cannot be repaired or avoided. As a consequence, these packets are dropped before injected in the network.

the total traffic load of a network, a fraction of which belongs to each service as indicated in Table 3.1.

In general, packet latency remains more stable, as the injection rate increases, for higher priority services and saturates only for the lowest priority service (SVC3). A similar observation holds for throughput which scales (almost) linearly with the injection rate for higher priority services.

RQNoC SDetour incurs low performance overheads in low fault densities and high priority packets. However, as the number of faults increases and routing paths get longer, SDetour becomes inefficient and this is mostly evident in low priority services. More precisely, in networks with up to 4 faults and in SVC0 and 1, SDetour has similar performance with the fault free case, for SVC 2 average packet latency increases 1-15%, while SVC3 saturates at the same point with a fault-free network. Throughput is similar or even better (in SVC1) than fault free as shown in Figure 3.10 as detouring packets contributes to load balancing and avoiding congestion. At higher number of faults, SDetour starts having a significant performance decrease. For 8 to 32 faults, SDetour maintains about 50% of the overall fault-free throughput and saturates at 10-15% injection rate versus 50% of the fault free case. Finally, in all fault densities, SDetour performs well for SVC 0 and 1 showing that longer paths may have a smaller impact in the performance of high priority services.

SMerge incurs a significant latency overhead in most cases. Allowing different traffic classes to share, besides the network links, the same router data-paths increases the overall latency, even if priority arbitration is maintained in the output ports. An additional reason for the lower SMerge performance is that flits of different packets cannot be interleaved in data-paths shared by multiple services as they would if they had their own data-path. Packet latency increases more for lower priority traffic and higher fault densities. In particular, for one network fault latency increases by 1%, 10%, and 100% for SVC 0, 1 and 2, respectively, while SVC3 saturates at 25% of injection rate versus 50% in a fault free network. Throughput drops notably only in SVC3 to about 70-80% of the fault free. For 4 faults, SMerge has about 50% to 100% higher latency and maintains 80% of the fault free throughput. At 8, and 16 network faults, latency increases by 1.7-2×, 1.6-3.8×, respectively, while for 32 faults latency slightly reduces to and 1.5-2.4× of the fault free due to higher rate of dropped packets. In these fault densities, the saturation point is at 20% of the injection rate versus 50% in the fault free case and throughput is about 50%-70% of a fault-free network. It is worth noting that for 32 faults the latency of SVC0 and SVC1 reduces after the saturation point of the network (injection rate 20%), as shown in Figure 3.13; lower priority traffic gets congested and hence high priority packets have lower competition for the shared resources.

An RQNoC with SMerge and SDetour can improve fault tolerance in networks with 32 faults delivering 83% of the generated packets. The combination of sharing resources among services (SMerge) and using longer paths (SDetour)increases the packet latency for high priority SVCs, as shown in Figure 3.13. However, for the larger, low priority traffic-load, using different paths (SDetour) implicitly avoids congestion points and reduces latency. On the contrary, throughput is better than SMerge for high priority traffic and worse for lower priority. This is explained by the fact that sharing NoC resources favors higher priority packets, which get a larger share of the links and router data-paths. Overall, latency is 1.5-2× higher than the fault free QNoC (or RQNoC) and throughput is similar to fault free for SVC0 and drops to about 50% for the other services.

In most cases, GDetour substantially increases packet latency although throughput is close to the fault-free case. To exemplify, GDetour latency doubles for some services even with a single network fault and reaches saturation point at 10% injection rate with 4 faults. Beyond these fault densities we cannot derive useful conclusions about GDetour as the percentage of successfully transmitted packets is below 20%.

In addition to the average latency and throughput, Figures 3.9 to 3.13 depict the maximum packet latency and standard deviation of latency and throughput[7]. In general, maximum latency and standard deviation of latency and throughput suffer similar overheads with their respective average values. There are however some exceptions. Maximum latency and its standard deviation double in SMerge RQNoC with a single permanent fault although the overhead in mean packet latency is substantially lower. A similar observation is made for the combined SMerge and SDetour for 32 faults as well as in some cases of the GDetour.

In summary, tolerating permanent faults in the RQNoC incurs (in some cases significant) overheads, but it offers better network connectivity. Allowing a minimum number of network nodes to remain connected can be more critical than performance for avoiding system failure. Besides, many systems are designed to offer graceful degradation of performance and/or functionality in the presence of permanent faults [44]. Despite the RQNoC performance overheads only the lowest priority service saturates as it does so in the fault-free case, too; this shows that traffic classes still enjoy QoS even when parts of the NoC are damaged.

[7] In most cases, standard deviation of throughput is too small to be observable in the plots.

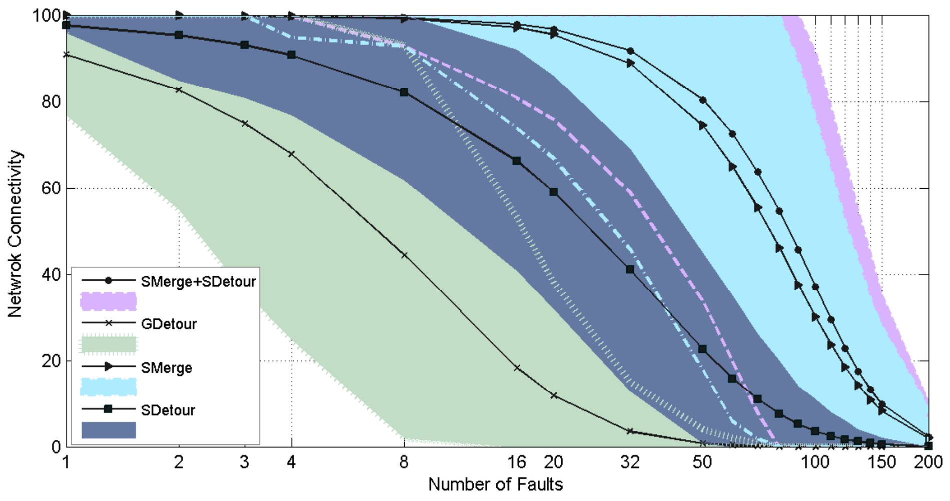


Figure 3.14: *Mean and* $\pm 3\sigma$ *range of connectivity values for the proposed RQNoC designs at different fault densities.*

3.4.3 Network connectivity and Fault-tolerance

We present next a more accurate evaluation of fault-tolerance and network connectivity; which directly relates to NoC and SoC QoS as system failure may depend to a minimum number of connected nodes, besides minimum performance. This is achieved creating network models for each RQNoC design and performing multiple fault-injections for different fault densities. In each case, our models return the network connectivity defined as the percentage of available paths in the network out of the total number of source to destination pairs. In this part the following 4x4 2D mesh network are evaluated: SMerge+SDetour, SMerge, SDetour, and GDetour. We considered fault densities ranging from 1 to 200 network faults. For each fault density, half a million different fault injections have been performed. Faults are injected randomly considering the probability of failure for different network parts based on their area.

Figure 3.14 depicts, for different fault densities (number of faults), the mean network connectivity as well as the range of values that are up to 3 standard deviations ($\pm 3\sigma$) away from the mean. SMerge+SDetour provides the highest network connectivity, which is above 99% for up to 8 network faults, 92% for 32 faults, 80% for 50 faults and 37% 100 faults. SMerge alone can support a similar connectivity preserving 99.3%, 89%, 75% and 30% of the network paths for 8, 32, 50 and 100 faults, respectively. SDetour follows with 82%, 41%, 20% and 3.6% connectivity for the same fault densities. Finally, GDetour shows poor fault tolerance maintaining only 44% of the network connectivity at 8 network faults, 3.5% for 32 faults and below 1% for more than 50 faults.

Figure 3.15 illustrates the probability of a network to offer 100%, 75%, 50% and 25% connectivity at a particular fault density. This is important for networks that are expected to guarantee a particular quality of service. SMerge and SMerge+SDetour offer similar fault-tolerance having above 80% probability to offer a fully connected network even with 20 permanent faults. The probability of full connectivity is significantly reduced for SDetour and GDetour as the turn model used for the routing cannot always connect all node pairs. However, as the connectivity requirements reduce to 75%, 50% and 25%, SDetour performs better than GDetour. Even so, SMerge and SMerge+SDetour can deliver at fault densities 2-3× higher the same probability as SDetour for a particular network connectivity.

3.4.4 Comparison

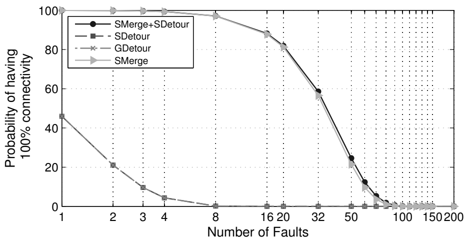
Providing a complete and detailed comparison of RQNoC with related work is extremely difficult as related techniques are applied to different networks with different network sizes, baseline router architectures, or even topologies. In addition, various different metrics are used to evaluate NoC fault tolerance and its overheads. Even so, we attempt next to compare with related techniques to the degree that is possible, retrieving RQNoC results that match other networks parameters and metrics.

Compare to FoN [12], Cost-based [24] and FTDR-H [13] RQNoC has lower overheads and maintains a larger fraction of its performance in the presence of faults. In particular, FoN, Cost-based and FTDR-H have area overheads of 1.35×, 2.85× and 2.58×, respectively, compared to their baseline, while RQNoC[8] requires 1.58× more resources. Their power overheads, measured in mWatts/MHz, are 4×, 17×, and 9× that of their baseline, respectively, while RQNoC increases its power consumption by only 1.44×. Moreover, FoN, Cost-based and FTDR-H maintain 50-25% of their throughput; at the same fault rates RQNoC maintains 81% to 66% of its throughput.

ReliNoC [22] increases its area requirements by 15% versus 58% in RQNoC. However, the percent of fully connected ReliNoC networks at 20 to 100 faults ranges from 90% to 30%; for the same network size (8x8 mesh) RQNoC is able to deliver 99%-70% at the same number of faults delivering substantially higher network connectivity and fault tolerance.

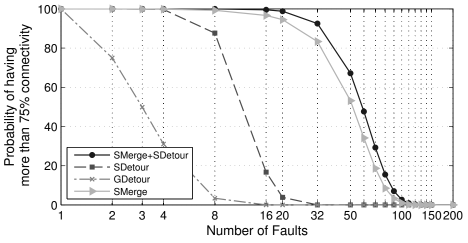
Bulletproof [7] has a very high area overhead of 3.42× compared to its baseline; that

[8] In all comparisons, we consider the RQNoC with both SMerge and SDetour.



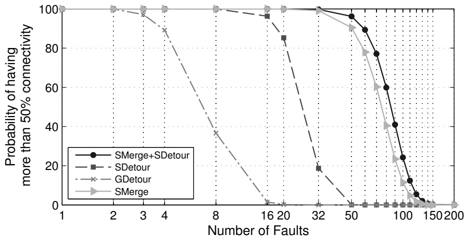
(a)

Probability of having full network connectivity under different fault densities.



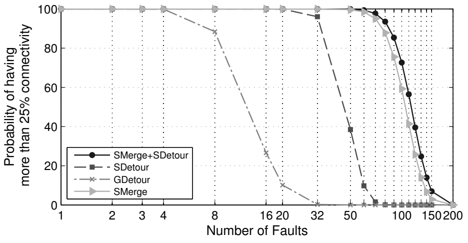
(b)

Probability of having at least 75% network connectivity under different fault densities.



(c)

Probability of having at least 50% network connectivity under different fault densities.



(d)

Probability of having at least 25% network connectivity under different fault densities.

Figure 3.15: *Probability of different RQNoC networks to deliver a particular network connectivity under different fault densities.*

is 4× higher overhead than the one of RQNoC. Reliability is measured in Bulletproof as the mean number of defects that cause a router failure. Bulletproof can sustain up to 38 defects before a router fails, while an RQNoC router can tolerate 6. We can observe a clear trade-off between area and reliability in the two approaches, which is based on the granularity of the considered fault model. An RQNoC router is divided in 7 distinct parts, as opposed to over 200 partitions in the Bulletproof router, having significantly lower area overhead but tolerating fewer faults. Fault densities of tens of permanent faults in a silicon area that a single NoC router occupies is quite unlikely even in future emerging technologies and therefore we consider that a coarser fault model granularity is preferable.

Vicis [9, 15] requires 51% more area than its baseline (versus 58% for RQNoC) and is able to keep about 90% and 80% of its routers connected at fault rates of 2.5 and 5 faults per router, respectively. At the same fault rates RQNoC keeps 85% and 55% of its routers connected. Although Vicis appears to be more reliable than RQNoC, it uses a torus topology which is inherently more fault tolerant than the RQNoC 2D-mesh topology.

In RoCo, a 8×8 [23] 2D-mesh network with 4 faults offers only 85% packet completion. RQNoC is significantly more reliable as it achieves above 99% packet completion even with 16 faults in a 4 times smaller network (4×4 2D-mesh). Area, performance and power overheads are combined into a single metric rather than presented separately and therefore we cannot compare them.

Finally, Virtual channel renaming has very low area and power overheads (5.3% and 3.7%, respectively) and low performance costs at low fault rates (similar to RQNoC) [10]. However, it requires each physical buffer to fit two flits per virtual channel; for a router that supports 4 VCs with 4 physical buffers that means 8 flits per buffer, which is 4× larger than the minimum buffer size of a regular VC-based network. Such area (and power) costs are possibly not reflected in the above overheads. Moreover, VC renaming protects only the buffers of a router, while packet priorities are only fixed to each physical buffer.

3.5 Conclusion

This part of this thesis work described RQNoC, a solution for tolerating permanent faults on a Network-on-Chip that supports multiple services. Service Detour and Service Merge were proposed to redirect traffic and bypass the faulty resources of a particular service. On one hand, Service detour employs alternative paths to support a faulty service, on the other hand, Service Merge shares a router data-path between multiple traffic classes. The two approaches were evaluated in terms of implementation costs, network performance per traffic class and fault tolerance. SDetour has lower cycle time, area and power costs compared to SMerge as it requires fewer hardware modifications. It also has lower network performance overheads in low fault rates, but becomes inefficient as the number of faults increases, substantially reducing its packet completion rate in the network. SMerge requires roughly 50% more resources than our baseline QNoC network and has about a quarter of higher power consumption. SMerge network latency may double and its throughput can be reduced to 50% compared to a fault-free case, but it is able to tolerate substantially more faults. SMerge preserves 90% of the network connectivity even with 32 network faults, which is more than double compared to SDetour. Combining SMerge with SDetour further improves fault tolerance increasing connectivity by up to 5%.

4 Conclusions

In this thesis, hardware reconfigurability for tolerating permanent faults was explored in the context of (i) adaptive processors and (ii) service-oriented NoCs. It has been indicated that hardware designs are becoming more susceptible to permanent faults as a consequence of decrease in the transistors feature size. In the first part of this thesis, a probabilistic analysis of adaptive processors was presented, tolerant to permanent faults. We defined different hardware reconfiguration granularities and evaluated each one by utilizing an adaptive RISC processor as our use-case. We explored the advantages of pipelining reconfigurable interconnects by measuring the achieved execution time and comparing it to the design with non-pipelined wires. We performed a comprehensive design space exploration in order to find the most efficient granularity mix. In the second part of the thesis, we introduced RQNoC, a service-oriented Network-on-chip resilient to permanent faults at both routers and links. The architecture of the RQNoC was described and different fault-tolerance techniques are explained and evaluated in terms of reliability as well as in terms of hardware and performance overheads.

This chapter summarizes the content of this thesis, outlines its contributions and propose future research directions. The rest of this chapter is organized as follows.

Section 4.1 briefly summarizes the content of this thesis. Section 4.2 presents the main findings and contributions of this thesis. Finally, section 4.3 provides a guideline for possible future works.

4.1 Summary

In this thesis, our research objectives were first defined and subsequently addressed in two distinct parts where the core idea of both was to employ hardware flexibility in order to offer tolerance to permanent faults.

In the first part, covered in chapter 2, a probabilistic analysis was presented for estimating the number of fault-free components that can be constructed in various reconfigurable designs in presence of permanent faults. We used adaptive processors as our use-case and explored different designs with possibility of coarse-grain or mixed-grain reconfiguration. We explored the performance effects of using pipeline registers on reconfigurable interconnects and finally, in order to find the most efficient granularity mix, a design space exploration was performed. Our analysis showed that employing fine-grain logic in addition to the coarse-grain reconfiguration, i.e., mixed-grain, can increase availability, tolerating 3× more faults than mere component redundancy. Furthermore, the design-space exploration revealed that different fault densities require different granularities of substitutable units to maximize fault-tolerance. With growing number of faults, coarse-grain with granularity of $\frac{1}{8}$ and mixed-grain with granularity of $\frac{1}{16}$ offered the best availability figures. Finally, our results showed that in adaptive processors with no pipelined interconnects the frequency and execution time both increased by almost 3.5×, compared to the baseline. Adding pipeline registers showed significantly better performance, where for the fault-free case the overhead of operating frequency and execution time were 1.41× and 1.7× compared to the baseline.

In the second part of this thesis, covered in chapter 3, we described RQNoC, a service-oriented Network-on-Chip (NoC) resilient to permanent faults. We characterized the network resources based on the particular service they support and when faulty bypass them allowing the respective traffic class to be redirected. We proposed two alternatives for service redirection, each having different advantages and disadvantages. The first one, Service Detour, used longer alternative paths through resources of the same service to bypass faulty network parts, keeping traffic classes isolated. The second approach, Service Merge, used resources of other services providing shorter paths but allowing traffic classes to interfere with each other. The remaining network resources that are common for all services employed additional mechanisms for tolerating faults. Links tolerated faults using additional spare wires combined with a flit-shifting mechanism and the router control was protected with Triple-Modular-Redundancy (TMR). The proposed RQNoC network designs were implemented in 65nm technology and evaluated in terms of performance, area, power consumption and fault tolerance. Service Detour required 25% more area and consumed 7% more power compared to a baseline network, not tolerant to faults; its packet latency and throughput were close to the fault free performance at low fault densities, but fault-tolerance and performance dropped substantially above 4 network faults. Service Merge required 51% more area and 27% more power than the baseline and had a 9% slower clock. Compared to a fault free network, a Service Merge RQNoC with up to 32 faults had increased packet latency up to 1.5-2.4× and reduced throughput to 70% or 50%. However, it delivered substantially better fault-tolerance having a mean network connectivity of above 90% even with 32 network faults versus 41% of a Service Detour network. Combining Serve Merge and Service Detour improved fault tolerance further sustaining a higher number of network faults and reduced packet latency.

The following section identifies contributions of this thesis for the two aforementioned parts.

4.2 Contributions

This thesis addressed several issues regarding hardware reconfiguration for tolerating permanent faults in a SoC. The first part of the thesis dealt with tolerating permanent faults on the processing components of a SoC. The main contributions of this part of the thesis were the following:

- We described a method to analytically calculate the probability of constructing a certain number of fault-free components via reconfiguration, having as an input parameter the fault density. We performed our analysis for two alternatives of processor adaptation, namely, coarse-grain reconfiguration and mixed-grain reconfiguration.

- We measured the area overheads of reconfigurability for the adaptive processors considered as our use-case.

- We explored the performance impact of pipelining the reconfigurable wires that connect the Substitutable Units (SUs) of components and compared it to a design without pipelined wires.

- We performed a design space exploration and evaluated various granularities of reconfigurable substrates. We calculated for various fault densities the average number of fault-free components in an adaptive processor array as well as the probability of delivering a minimum number of fault-free components in a given silicon area. Thereby, we determined the most efficient granularity mix for our approach.

In the second part of the thesis, our focus was on tolerating permanent faults in the (service-oriented) NoC part of a SoC. To this end, we presented RQNoC, a resilient network architecture with four different traffic classes. Concisely, the main contributions of this part were as follows.

- We extended a previously known detour technique to the service level, allowing the network to selectively detour traffic per service, rather than detouring all packets.
- We modified router architecture in order to offer the option to merge multiple services, maintaining the priorities, in order to avoid the damaged data-path of one or multiple services.
- We modified links between routers to provide fault-tolerance by employing a spare-wire and shifting mechanism.
- We measured the area and power overheads of the above techniques, evaluated their impact on the performance and QoS of each service and analyzed their fault tolerance.

4.3 Proposed Research directions

During the time of this research, we have come across many interesting research challenges to study further. In this section, we list some of them:

- Exploiting the SMerge scheme for energy efficiency is another compelling topic. The RQNoC can be configured to operate in a degraded mode in order to preserve power by employing merging scheme.
- Designing a fault detection and dynamic reconfiguration mechanism compatible with RQNoC could be another topic. To this end, a fast on-line testing mechanism with granularity of services is needed. Moreover, a runtime management for generating configuration-bits based on the detected faults and subsequently reconfiguring the NoC is required to have a complete setup.

- Different configurations of the adaptive processors array, as well as of the RQNoC, may have different probability of failure. One interesting research opportunity can be to characterize the different configuration options in terms of Probability of Failure per Hour (PFH) and consider them at runtime, increasing system safety.
- Finally, it would be interesting to explore different policies for merging/detouring services based on performance requirements.

Bibliography

[1] Nidhi Aggarwal et al. Configurable isolation: building high availability systems with commodity multi-core processors. In *ISCA*, 2007.

[2] Muhammad Ali, Michael Welzl, and Sven Hessler. A fault tolerant mechanism for handling permanent and transient failures in a network on chip. In *Information Technology, 2007. ITNG'07. Fourth International Conference on*, pages 1027–1032. IEEE, 2007.

[3] David Bernick, Bill Bruckert, Paul Del Vigna, David Garcia, Robert Jardine, Jim Klecka, and Jim Smullen. Nonstop advanced architecture. In *DSN*, 2005.

[4] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Qnoc: Qos architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2):105–128, 2004.

[5] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10 – 16, 2005.

[6] Cristian Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE micro*, 23(4):14–19, 2003.

[7] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin, and Michael Orshansky. Bulletproof: A defect-tolerant cmp switch architecture. In *In Proceedings of the 12th International Symposium on High Performance Computer Architecture*, pages 3–14, 2006.

[8] A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *Design Test of Computers, IEEE*, 22(4):306–315, 2005.

[9] Andrew DeOrio, David Fick, Valeria Bertacco, Dennis Sylvester, David Blaauw, Jin Hu, and Gregory Chen. A reliable routing architecture and algorithm for nocs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(5):726–739, 2012.

[10] Marios Evripidou, Chrysostomos Nicopoulos, Vassos Soteriou, and Jongman Kim. Virtualizing virtual channels for increased network-on-chip robustness and upgradeability. In *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pages 21–26. IEEE, 2012.

[11] M.L. Fair et al. Reliability, availability, and serviceability (ras) of the ibm eserver z990. *IBM Jour. of Research & Development*, 48(3-4):519–534, 2004.

[12] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, and Minxuan Zhang. Fon: Fault-on-neighbor aware routing algorithm for networks-on-chip. In *SOC Conference (SOCC), 2010 IEEE International*, pages 441–446. IEEE, 2010.

[13] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Minxuan Zhang, and Zuocheng Xing. Addressing transient and permanent faults in noc with efficient fault-tolerant deflection router. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(6):1053–1066, 2013.

[14] David Fick, Andrew DeOrio, Gregory Chen, Valeria Bertacco, Dennis Sylvester, and David Blaauw. A highly resilient routing algorithm for fault-tolerant nocs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 21–26. European Design and Automation Association, 2009.

[15] David Fick, Andrew DeOrio, Jin Hu, Valeria Bertacco, David Blaauw, and Dennis Sylvester. Vicis: a reliable network for unreliable silicon. In *Proceedings of the 46th Annual Design Automation Conference*, pages 812–817. ACM, 2009.

[16] F Gilabert, María Engracia Gómez, Simone Medardoni, and Davide Bertozzi. Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 165–172. IEEE Computer Society, 2010.

[17] Cristian Grecu, Andre Ivanov, Res Saleh, Egor S Sogomonyan, and Partha Pratim Pande. On-line fault detection and location for noc interconnects. In *On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International*, pages 6–pp. IEEE, 2006.

[18] S. Gupta, Shuguang Feng, A. Ansari, and S. Mahlke. Stagenet: A reconfigurable fabric for constructing dependable cmps. *IEEE Trans. on Computers*, 60(1):5–19, 2011.

[19] Yinhe Han and Binzhang Fu. A new fault-tolerant routing based on turn model. In *Proceedings of the 3rd Workshop on Diagnosttic Services in Network-on-Chips, DSNOC'09*, pages 102–103. IEEE Computer Society, 2009.

[20] Nikos Hardavellas et al. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[21] Scott Hauck and Andre DeHon. *Reconfigurable computing: the theory and practice of FPGA-based computation.* Morgan Kaufmann, 2010.

[22] Mohammad Reza Kakoee, Valeria Bertacco, and Luca Benini. Relinoc: A reliable network for priority-based on-chip communication. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[23] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Vijaykrishnan Narayanan, Mazin S Yousif, and Chita R Das. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 4–15. IEEE Computer Society, 2006.

[24] Adán Kohler, Gert Schley, and Martin Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(6):883–896, 2010.

[25] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, and Timothy Mark Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *Proceedings of the second ACM/IEEE international symposium on networks-on-chip*, pages 13–22. IEEE Computer Society, 2008.

[26] C. LaFrieda et al. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *DSN*, pages 317–326, 2007.

[27] B.S. Landman and R.L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. on Computers*, 20(12):1469–1479, 1971.

[28] Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. Online reconfigurable self-timed links for fault tolerant noc. *VLSI design*, 2007, 2007.

[29] Cheng Liu, Lei Zhang, Yinhe Han, and Xiaowei Li. A resilient on-chip router design through data path salvaging. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 437–442. IEEE Press, 2011.

[30] Srinivasan Murali, David Atienza, Luca Benini, and Giovanni De Michel. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of the 43rd annual Design Automation Conference*, pages 845–848. ACM, 2006.

[31] David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. Resilient high-performance processors with spare ribs. *IEEE Micro*, 33(4):26–34, 2013.

[32] Li-Shiuan Peh and Natalie Enright Jerger. *On-Chip Networks*. Morgan and Claypool Publishers, 1st edition, 2009.

[33] Andrea Pellegrini, Joseph L. Greathouse, and Valeria Bertacco. Viper: virtual pipelines for enhanced reliability. In *ISCA '12*, pages 344–355, 2012.

[34] Michael D. Powell, Arijit Biswas, Shantanu Gupta, and Shubhendu S. Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *ISCA*, pages 93–104, 2009.

[35] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. Methods for fault tolerance in networks-on-chip. *ACM Computing Surveys (CSUR)*, 46(1):8, 2013.

[36] Samuel Rodrigo, Jose Flich, Antoni Roca, Simone Medardoni, Davide Bertozzi, J Camacho, Federico Silla, and Jose Duato. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pages 25–32. IEEE, 2010.

[37] Bogdan F. Romanescu and Daniel J. Sorin. Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults. In *PACT*, pages 43–51, 2008.

[38] S. Shamshiri and Kwang-Ting Cheng. Modeling yield, cost, and quality of a spare-enhanced multicore chip. *IEEE Trans. on Computers*, 60(9):1246–1259, 2011.

[39] Smitha Shyam, Kypros Constantinides, Sujay Phadke, Valeria Bertacco, and Todd Austin. Ultra low-cost defect protection for microprocessor pipelines. In *ASPLOS XII*, pages 73–82, 2006.

[40] Georgios Smaragdos, Danish Anis Khan, Alirad Malek, Stavros Tzilis, Ioannis Sourdis, and Christos Strydis. Flexpipes: A fault-tolerant cmp architecture based on resource sharing. *ACM Transactions on Reconfigurable Technology and Systems*, -(-):–, 2014.

[41] Georgios Smaragdos, Danish Anis Khan, Ioannis Sourdis, Christos Strydis, Alirad Malek, and Stavros Tzilis. A dependable coarse-grain reconfigurable multicore array. In *21st Reconfigurable Architectures Workshop (RAW'14)*, 2014.

[42] Ioannis Sourdis, Danish Khan, Alirad Malek, Stavros Tzilis, Georgios Smaragdos, and Christos Strydis. Resilient cmps with mixed-grain reconfigurability.

[43] D. Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *Design Test of Computers, IEEE*, 23(6):484–490, 2006.

[44] Stavros Tzilis and Ioannis Sourdis. A runtime manager for gracefully degrading socs. In *in Int'l Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2014.

[45] Arseniy Vitkovskiy, Vassos Soteriou, and Chrysostomos Nicopoulos. Dynamic fault-tolerant routing algorithm for networks-on-chip based on localised detouring paths. *Computers & Digital Techniques, IET*, 7(2), 2013.

[46] I. Wagner, V. Bertacco, and T. Austin. Shielding against design flaws with field repairable control logic. In *DAC*, pages 344–347, 2006.

[47] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.