TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

IA70LT

René Pihlak 178247IASM

EFFICIENCY IMPLICATIONS OF TRANSIENT FAULT MITIGATION

Master's Thesis

Supervisor:	Jaan Raik
	PhD
Co-Supervisor:	Karl Janson
	MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

IA70LT
René Pihlak 178247IASM

AJUTISTE RIKETE HALDUSE MÕJU EFEKTIIVSUSELE

Magistritöö

Juhendaja:	Jaan Raik
	PhD
Kaasjuhendaja:	Karl Janson
	MSc

Tallinn 2019

Author’s declaration of originality

Abstract

EFFICIENCY IMPLICATIONS OF TRANSIENT FAULT MITIGATION

Electronic circuits, especially those deployed in harsh environments such as space, are susceptible to transient faults caused by radiation.

In this thesis, two online fault mitigation methods—AWAIT1 and AWAIT2—are proposed for mitigating transient faults on links between modules. Both methods are designed in Register-Transfer Level (RTL) using VHDL.

While AWAIT1 only offers mitigation against faults that do not occur critically close to the rising edge of the next clock, however, it is lightweight and can be used together with other methods. In contrast, AWAIT2 does not have such limitations. AWAIT2 can handle all Single Event Transient (SET) faults. The experimental results show that the AWAIT methods continue to operate even under harsh condition such as 80 million faults per second.

Even-though both AWAIT mechanisms are implemented using Hop-By-Hop (HBH) principle, the area overhead was 18.1% for AWAIT2 and only 5.1% for AWAIT1 compared to the baseline design. In case of AWAIT2 critical-path delay had a slight increase: 7.8%. Meanwhile, power consumption overhead of AWAIT1 increased 2.83%, while AWAIT2 had only 1.88% increase.

This thesis is written in English and is 50 pages long, including 4 chapters, 32 figures, and 7 tables.

Annotatsioon

AJUTISTE RIKETE HALDUSE MÕJU EFEKTIIVSUSELE

Digitaalseadmetel, mida kasutatakse kosmoses, esineb ajutisi rikkeid, mille peamiseks põhjuseks on kosmiline kiirgus. Kuivõrd missioonikriitiliste reaalajasüsteemide puhul ei saa (ajutisi) rikkeid ignoreerida, siis käesolevas töös esitatakse kaks ajutiste rikete mõju minimeerimise meetodit: AWAIT1 ja AWAIT2. Mõlemad meetodid on disainitud registerülekande tasemel (*Register-Transfer Level*) kasutades VHDL'i.

Nii AWAIT1 kui ka AWAIT2 võimaldavad ajutisi rikkeid tuvastada ja elimineerida. Mõlemad AWAIT meetodid peatavad rikke tuvastamisel rikete eest kaitstava mooduli töö ajutiselt. Kui rikke mõju kaob, siis mooduli töö jätkub. Rikke tuvastamiseks on kasutusel paarsuskontroll. Seetõttu suudavad mõlemad AWAIT meetodid süsteemi kaitsta kõigi tuvastatud üksikrikete eest.

Paraku AWAIT1 toimib ainult teatud kindlate ajutiste rikete puhul. Kui rike tekib liiga lähedal järgmise takti tõusvale servale, siis AWAIT1 ei jõua õigeaegselt tuvastada riket. Selle probleemi lahenduseks on esitatud AWAIT2, mis hilistab rikke tuvastust ühe takti võrra.

Eksperimentide tulemuste kohaselt toimivad mõlemad AWAIT meetodid isegi väga ohtlikes keskkondades. Mõlemad meetodid töötasid isegi siis, kui sekundis esines 80 miljonit riket.

AWAIT1 meetod vajab ainult 5.1% ja AWAIT2 18.1% rohkem pindala, kui baasmudel. Kuigi AWAIT2 on kulukam ja suurem kui AWAIT1, siis see lisa kulu tagab kõrgendatud usaldusväärsuse. Samas elektri energia kasutuselt on AWAIT2 ökonoomsem. AWAIT1 vajab 2.83% rohkem energiat, kui baas mudel; AWAIT2 ainult 1.88%. Kriitilise tee hilistumine oli AWAIT2 puhul 7.8%. AWAIT1 puhul hilistumist ei olnud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 50 leheküljel, 4 peatükki, 32 joonist, 7 tabelit.

List of abbreviations and terms

BF	Bridging Fault
BIST	Built-In Self-Test
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
E2E	End-To-End
ECC	Error Correction Code
EM	Electromigration
FIR	Fault Injection Rate
F-RET	Flit-Based Retransmission
FIFO	First In, First Out
HBH	Hop-By-Hop
IP	Intellectual property
LBDR	Logic-Based Distributed Routing
MBU	Multi-Bit Upset

MET	Multi-Event Transient
NBTI	Negative Bias Temperature Instability
NoC	Network-on-Chip
PIR	Packet Injection Rate
P-RET	Packet-Based Retransmission
P2P	Point-to-Point
PE	Processing Element
PMOS	P-Type Metal-Oxide-Semiconductor
RMA	Return Merchandise Authorization
RR	Relaxed Retransmission
RTL	Register-Transfer Level
S-A-F	Store-and-Forward
SAF	Stuck-At Fault
SE	Soft Error
SEE	Single Event Effect
SEL	Single Event Latch-Up
SET	Single Event Transient
SEU	Single Event Upset
SoC	System on Chip
TDDB	Time-Dependent Dielectric Breakdown

TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
TSMC	Taiwan Semiconductor Manufacturing Company

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Modern economy largely relies on electronics. Thus electronics have a significant monetary impact on most industries [1]. Many traditional services and industries—from legal services such as verifying signatures to entertainment content production such as motion pictures—are moving from analog to digital domain. Moreover, many electronic circuits are expected to operate in new untested and/or harsh environments such as in space, e.g., for asteroid exploration [2]. Therefore, research into reliability of electronics based systems continues to stay a relevant topic.

For a number of years, [3] relates, products that have been impacted by transient faults have been often returned to suppliers as Return Merchandise Authorization (RMA) under warranty, thus incurring costs to replace and diagnose the products. However, often even after thorough analysis, these products are deemed to be not defective because some of these returns are caused by Single Event Upsets (SEUs). Applying fault mitigation methods would diminish the effect of SEUs, thus increasing dependability through improved reliability.

While dependability spans over a range of topics, this thesis focuses on reliability improvements by proposing two lightweight designs of fault mitigation methods addressing non-destructive Single Event Transient (SET) faults on links between modules such as Network-on-Chip (NoC) routers. These two designs—also referred to as “AWAIT1” and “AWAIT2”—were published, respectively, in [4] and [5], to increase reliability of realtime systems.

The fault mitigation mechanisms of AWAIT1 and AWAIT2 are designed to operate online in harsh environments with high rate of transient faults. Thus these designs are suitable—but not limited—to environments such as space, etc.

1.2 Contributions

The main contribution of both—AWAIT1 and AWAIT2—mechanisms is an improved yet lightweight fault mitigation method to harden digital designs against SET faults. This increased reliability is achieved with relatively small (area, critical-path delay, and power consumption) overhead when compared to the baseline design without any fault mitigation mechanisms. Despite of low overhead, the AWAIT1 and AWAIT2 fault mitigation mechanisms are capable of handling SET faults while relying on a simple parity checker for fault detection mechanism.

Both mechanisms are designed in Register-Transfer Level (RTL) VHDL and are open-source [6].

The AWAIT mechanisms take advantage of the fact that SET faults are non-permanent. Thus using so-called Relaxed Retransmission (RR), the affected modules are designed to halt operation until the effect of the SET fault disappears. The proposed approaches cannot mitigate the effect of permanent faults, however mitigation of permanent faults is out of the scope of the current thesis, and it is assumed that the system has other mechanisms such as Built-In Self-Test (BIST), reconfigurability, etc. to detect and mitigate the effects of permanent faults.

The AWAIT1 mechanism has two limitations. Firstly, new input data cannot be processed before previous data is cleared; and, secondly, there is a critical zone near the rising clock edge. Both of these limitations are addressed in AWAIT2.

1.3 Structure

The rest of the thesis is structured as described below.

Section 2 describes the background and main concepts relevant to the fault mitigation of SET faults. This includes Section 2.2 on fault types, Section 2.2.2 on fault mitigation methods, and Section 2.1 on NoC architecture since the experimental results are obtained using NoC with Bonfire routers [7]. In the following section, Section 3, related works are discussed providing a literature review of closely related works. Next, in Section 4 the two main contributions of this thesis—fault mitigation mechanisms AWAIT1 and AWAIT2 for inter-component links of digital systems—are presented together with experimental results. In addition, possible future research possibilities to improve the proposed mechanisms are discussed. Finally, Section 5 provides a summary and conclusions of the work.

The intermediary results were presented in conferences. The conference papers covering the AWAIT1 and AWAIT2 methods are included in Appendices 1 and 2.

2 Background

Technological advancement is a result of modernization in all domains—behavioral, structural and physical domain—and abstraction levels (see Table 1). For example, new architectures such as NoC has been introduced, and transistor size continues to shrink roughly following Moore's Law [8] (see [9] for evolving definition of Moore's Law). While some of these advancements provide new challenges, others provide new solutions as well.

Table 1. Design hierarchy [10].

Level	**DOMAINS**		
	Behavioral	**Structural**	**Physical**
System	Communicating Processes	Processors, Memories, Switches	Cabinets, Cables
Algorithm	Input-Output	Memory, Ports, Processors	Board Floorplan
Register-Transfer	Register Transfers	ALUs, Regs, Muxes, Bus	ICs, Macro Cells
Logic	Logic Equations	Gates, Flip flops	Standard Cell Layout
Circuit	Network Equations	Transistors, Connections	Transistor Layout

There is an increased focus on dependability related topics in order to mitigate faults associated with new technologies (e.g., see [11], [12]), especially when applied in systems that are required to be ultra-reliable, safety-critical, mission-critical, long-lived and/or highly-available (e.g., see [1]), for example, artificial satellites.

Section 2.1 discusses NoC *versus* system bus. Having background knowledge about NoC is necessary, since in this thesis the proposed fault mitigation mechanisms are applied on a NoC to obtain experimental results. Section 2.2 provides background information on how to improve dependability through mitigation of transient faults.

2.1 Network-on-Chip

One of the key elements in traditional single core Central Processing Units (CPUs) is the ***system bus***. However, as many or multiple (reusable) Intellectual properties (IPs) [13] (e.g., Processing Elements (PEs)) are increasingly used in Systems on Chip (SoCs), the traditional system bus becomes a bottleneck for the system as a whole [14]. Thus, SoCs need better alternatives for internal communications. One of such possible replacements for the system bus is ***Network-on-Chip (NoC)*** [15]–[17].

2.1.1 Network-on-Chip architecture: overview

There are numerous variations of NoC architectures. For example, NoCs can have different **topologies** [18] such as:

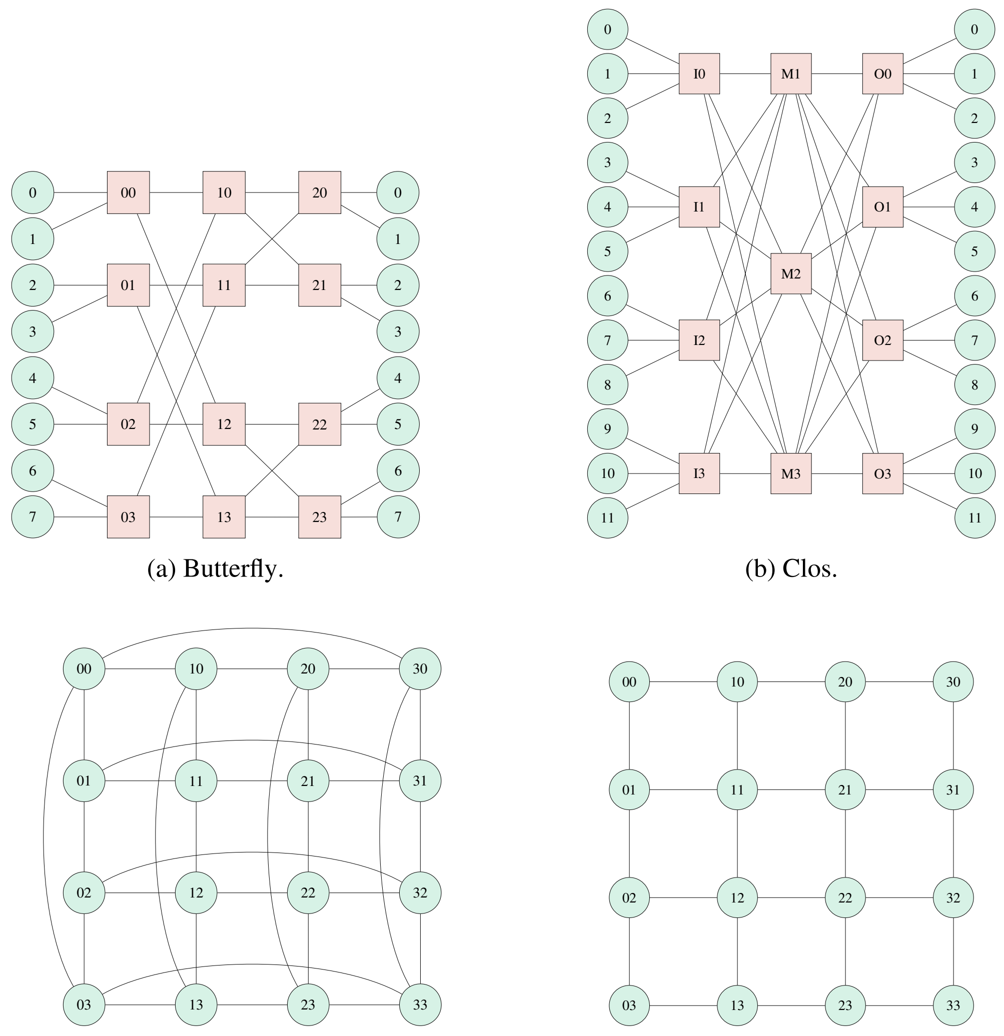
- butterfly (see Figure 1a),
- clos (see Figure 1b),
- torus (see Figure 1c),
- mesh (see Figure 1d) and
- irregular topology.

The topologies presented in Figure 1 are 2D topologies. However, for example [19] discusses the use of 3D NoC topology.

There are three main types of **routing algorithms**: 1) deterministic, 2) oblivious and 3) adaptive [18]. These routing algorithms can be implemented using different routing mechanisms, e.g., 1) **table lookup** or 2) **logic based** routing mechanisms [20] (such as Logic-Based Distributed Routing (LBDR) mechanism [21]).

LBDR has advantages over lookup table based routing mechanisms:

> "[LBDR] is scalable compared to table-based routing in NoCs. Furthermore, LBDR describes the topology and the routing algorithm in a 2D NoC in terms of a fixed number of configuration bits, i.e., connectivity and routing bits. This makes it possible to use the connectivity bits for the indication of links in the 4 main directions as healthy or faulty, by setting the corresponding connectivity bit to zero (faulty) or one (healthy). Routing algorithm re-configuration (if necessary) can be done by changing the routing bits." [7]



(a) Butterfly. (b) Clos.

(c) Torus. (d) Mesh.

Figure 1. Popular interconnection network topologies (adapted from [18]).

Routing inside of a NoC router can be executed using different methods such as **turn models**. For example, [22] provides a comprehensive overview of turn models (see also [23]).

While there are several **switching techniques**, the two best known are *Store-and-Forward (S-A-F)* and *wormhole* switching. The main difference between these two techniques is that S-A-F uses packet transmission while wormhole switching uses flit based transmission. This implies that S-A-F has to store the whole packet in the router before the packet can be transmitted, while wormhole switching sets up a route when a head flit arrives, passes subsequent flits using this route until a tail flit is transmitted. As a result the NoC routers need less input buffers, thus reducing the area overhead.

The **flow control** between routers can be achieved, for example, by using *acknowledgment-based* (handshake) or by *credit-based* flow control. In case of acknowledgment-based flow control, the two routers first have to ‘negotiate’ if transmission can occur. This increases latency. In contrast, in case of credit-based flow control the upstream router keeps count how many input buffers of the downstream router are currently available. Thus providing latency improvement since the 'negotiation' process is eliminated.

2.1.2 System bus *versus* Network-on-Chip

To fully appreciate the importance of NoC in the modern SoC design, one first has to understand the functioning of system bus and also the shortcomings of it. Using the BBC microcomputer as an example, one can illustrate the essence of system bus. A definition of system bus can be found in [24]:

> "A *bus* is simply a number of electrical links connected in parallel to several [modules]."

In other words, system bus is a communication tool or device that enables communication of data between the CPU and/or other modules such as memory. Therefore the system bus is an important part for the functioning of CPU, since the CPU needs data to process and the data is provided to the CPU by the system bus. In order to better understand how this data transfer is performed, the system bus should be examined more closely. While often the system bus is considered as one single unit, in fact, it consists of three buses [24]:

> "The communication protocols which enable this transfer of data to take place are set up by the control, address and data buses."

The functions of these three buses (see Figure 2) within the system bus are described by [25] as:

> "The data bus moves data from main memory to the CPU registers (and vice versa). The address bus holds the address of the data that the data bus is currently accessing. The control bus carries the necessary control signals that specify how the information transfer is to take place."

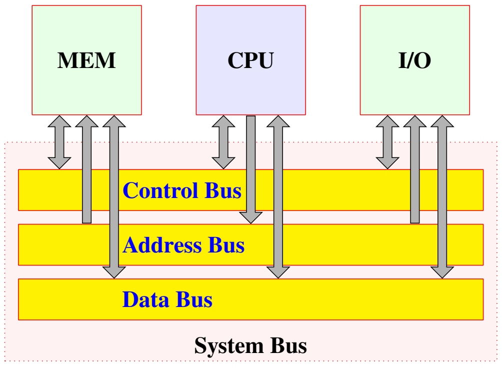


Figure 2. Example of a single system bus [25].

A more realistic graphical representation of system bus is provided, for example, by BBC Microcomputer (see Figure 3) where the bus is marked with wide arrow.

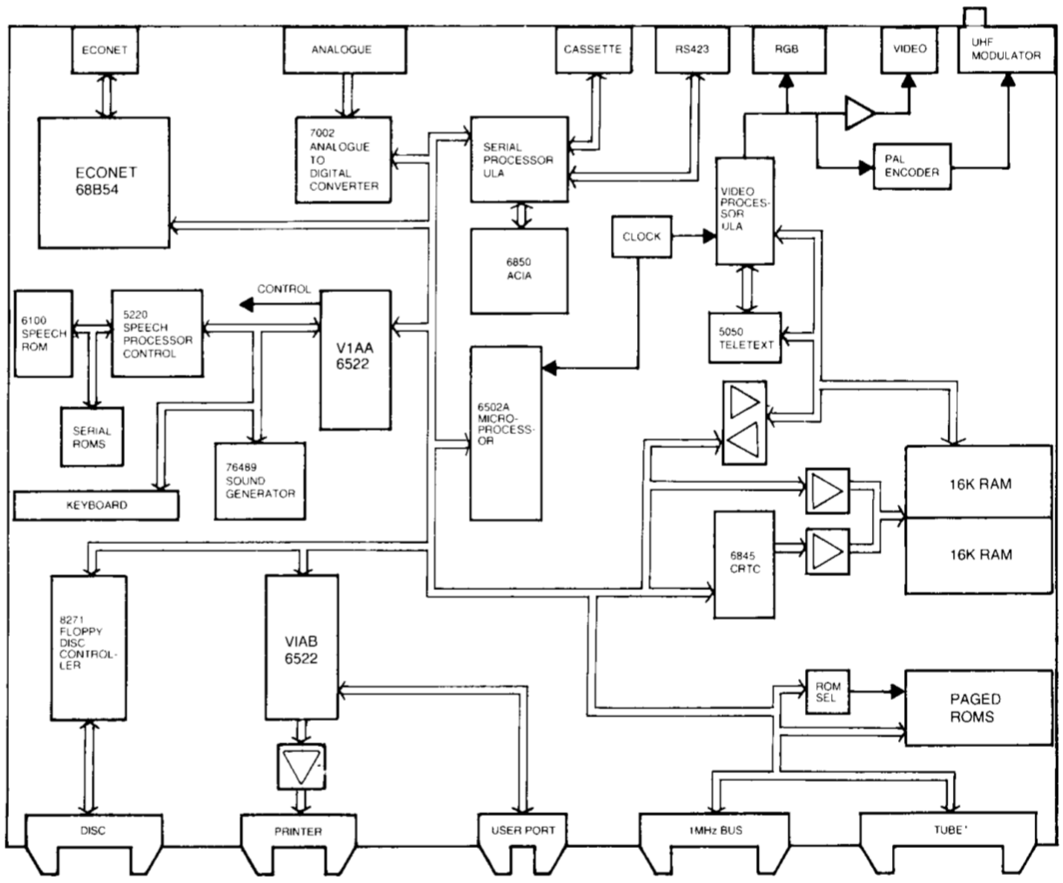


Figure 3. Bus for BBC Microcomputer [24].

In contrast to system bus, NoCs are, according to [16], multi-hop interconnection networks using packet switching that are integrated onto a SoC, where modules (e.g., PEs) access the network by means of Point-to-Point (P2P) interfaces, and have their packets forwarded to destinations through a number of hops (e.g., see Figure 4).

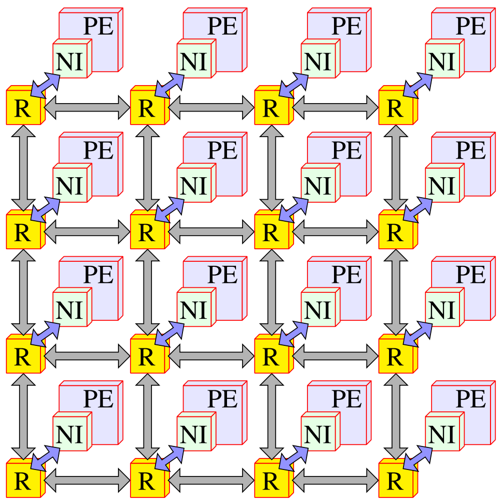


Figure 4. Illustration of a 4×4 Network-on-Chip.

The differences between bus and NoC are discussed in [26] (see Table 2). Neither of the two designs (neither system bus nor NoC) have complete superiority over the other. Therefore, both designs are still used depending on which design is more suitable for a given task. However, in case of designs consisting of many and multiple IPs, there is an increasing trend to migrate away from the bus to interconnected networks such as NoCs.

Table 2. Comparison of bus and Network-on-Chip [26].

Bus Pros & Cons		**NoC Pros & Cons**	
Every module attached adds parasitic capacitance, thus degrading electrical performance.	✗	✓	Only one-way P2P links are used, thus local performance is not degraded when modules are added to the design.
Bus timing is difficult in a deep submicron process.	✗	✓	P2P links can be pipelined.
The arbitration delay grows with the number of masters.	✗	✓	Routing decisions can be distributed.
The bus arbiter is instance-specific.	✗	✓	Same router for all network sizes.
Testability is problematic & slow.	✗	✓	Locally placed dedicated BIST is fast and can offer good test coverage.
Bandwidth is limited and shared by all attached modules.	✗	✓	Aggregated bandwidth scales with the network size.
Bus latency is wire-speed once arbiter has granted control.	✓	✗	Internal network contention may cause increase in latency.
Any bus is almost directly compatible with most available IPs, including software running on CPUs.	✓	✗	Bus-oriented IPs need smart wrappers. Software needs clean synchronization in multiprocessor systems.
The concepts are simple, standardized and well understood.	✓	✗	System designers need reeducation for new concepts, because it is a relatively new and changing design.

One of the key reasons to favour NoC over system bus is to minimize links/buses that connect IPs to each other. Equation 1 shows the relationship between the number of links connecting IPs in full mesh topology and the number of IPs (n).

$$\begin{array}{c}\texttt{number of links}\\ \texttt{(for a bus in full mesh topology)}\end{array} = \begin{cases} 0 & : \{n \in \mathbb{N} | n \leq 1\} \\ \dfrac{n \times (n-1)}{2} & : \{n \in \mathbb{N} | n > 1\} \end{cases} \qquad (1)$$

Meanwhile, Equations 2 and 3 show the relationship between the number of links and the number of nodes in case of two different NoC configurations. Equation 2 applies for nodes arranged in two rows but not necessarily always in even numbers. This configuration requires less links than having IPs ordered in a square format (see Equation 3), however average latency will grow faster than in case of a square NoC (see also Figure 5).

$$\begin{array}{c}\texttt{number of links}\\ \texttt{(for a two row NoC)}\end{array} = \begin{cases} 0 & : \{n \in \mathbb{N} | n = 0\} \\ n-1 & : \{n \in \mathbb{N} | 1 \leq n \leq 3\} \\ \frac{(n-4)\times 3}{2} + 4 & : \{n \in \mathbb{N} | n \geq 4 \wedge n = 2 \times \mathbb{N}\} \\ \frac{(n-5)\times 3}{2} + 5 & : \{n \in \mathbb{N} | n \geq 4 \wedge n = 2 \times \mathbb{N} + 1\} \end{cases} \tag{2}$$

$$\begin{array}{c}\texttt{number of links}\\ \texttt{(for a square NoC)}\end{array} = \begin{cases} 0 & : \{n \in \mathbb{N} | n = 0\} \\ (n-\sqrt{n})\times 2 & : \{n \in \mathbb{N} | n \geq 1 \wedge \sqrt{n} \in \mathbb{N}\} \end{cases} \tag{3}$$

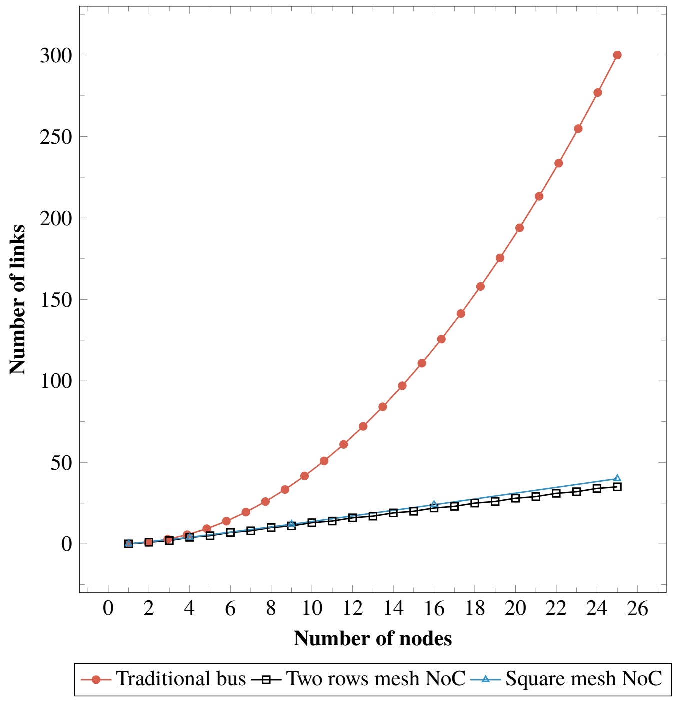


Figure 5. Number of additional links/buses to connect as the number of nodes grows.

Figure 5 presents a graphical illustration of the effect of adding IPs to the system: links grow exponentially if IPs are arranged as fully connected mesh using buses, while using NoC based system, the growth is, roughly, linear.

2.1.3 Bonfire: Network-on-Chip router

The fault mitigation methods discussed in the current thesis are implemented on the open-source Bonfire NoC router [6], [7] (see Figure 6). The Bonfire router is a wormhole switching router for 2D (mesh) NoC networks. In order to demonstrate the operation of the fault mitigation methods, a baseline version of Bonfire NoC router is used without implementing any fault tolerance methanisms.

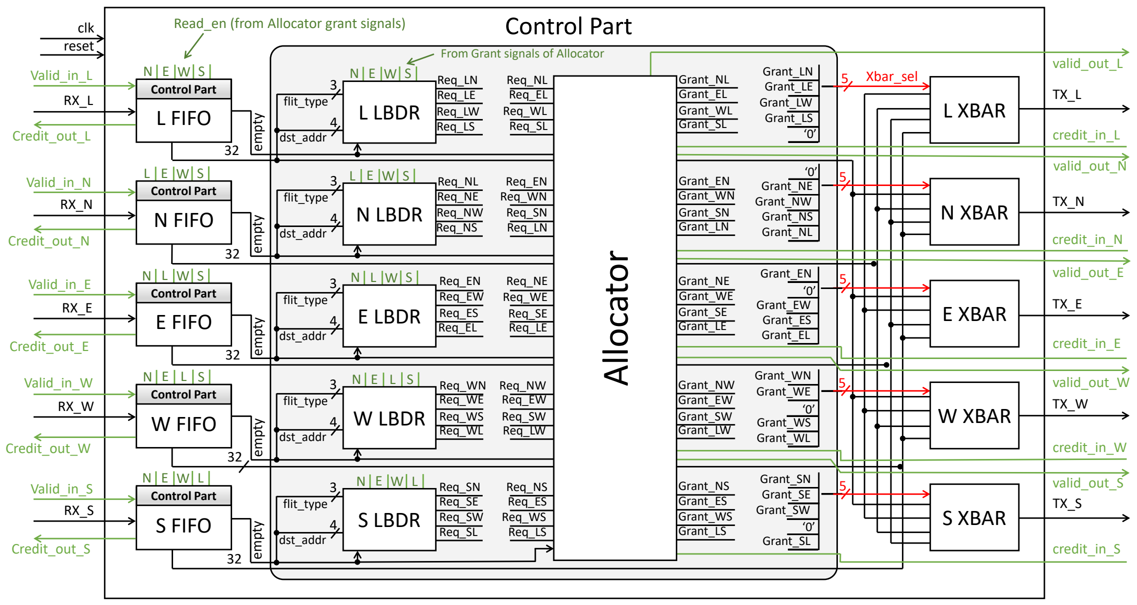


Figure 6. Architecture of the Bonfire Network-on-Chip router [6].

The Bonfire router is a 32-bit flit router. It consists of input buffers, LBDR based routing computation units, a switch allocator and crossbar switches. Each input buffer is implemented as a 4-flit circular buffer First In, First Out (FIFO). In the Bonfire router, the switch allocator prioritizes multiple requests to the same output port based on Round-Robin policy [27].

2.2 Faults and dependability

The occurrence of faults can have a negative impact on reliability and thus also on dependability of systems. When a fault occurs, it can manifest as an error and lead to a failure [28], thus reducing the dependability. In this context, **failure** is a system state where the perceived operation of the system differs from the correct operation [1]. Dependability related terms are arranged as an illustrative tree in Figure 7.

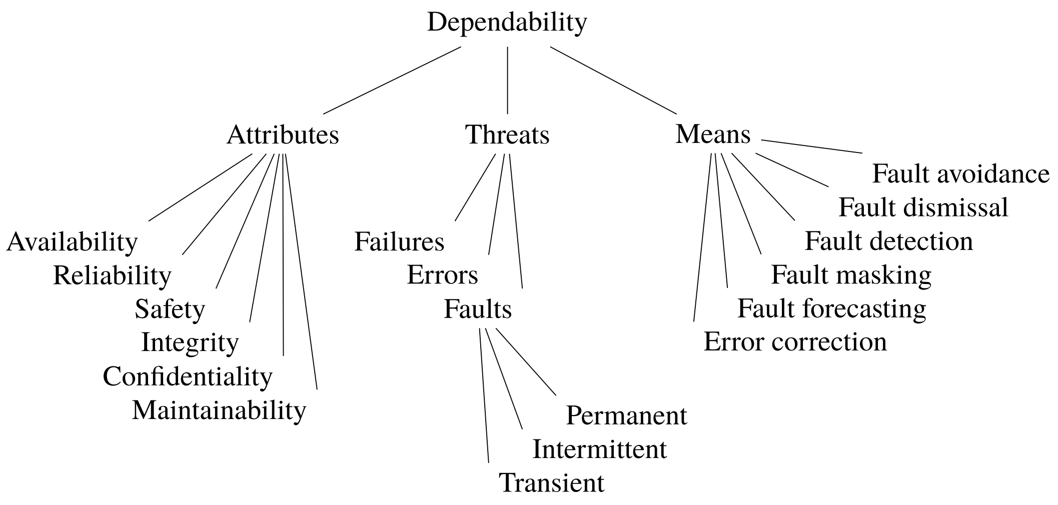


Figure 7. Dependability related terms (adopted from [1]).

According to [29], the approaches to dependability can be divided into two main categories:

- fault avoidance, and
- fault tolerance (and/or fault mitigation).

Fault avoidance is the process of taking actions that assure that faults do not occur. Thus, (partial) fault avoidance, according to [30], can be achieved by using more reliable parts, using higher quality manufacturing procedures, etc. However, full fault avoidance is unlikely to be possible, and/or it would be prohibitively expensive.

Fault tolerance is "the process of masking or detecting errors and taking corrective actions to avoid failure" [30]. Hence, faults are assumed to take place (at least sometimes), but the system prevents that those faults become failures. Thus, the system functions correctly even in the presence of faults.

Both—fault avoidance and fault tolerance—can be combined to assure that the designed system meets the required dependability level. However, as argued above, full fault avoidance and/or full fault tolerance is either unlikely and/or prohibitively expensive. Therefore this thesis focuses on the aspects of fault mitigation. Hence, the goal is to improve reliability instead of designing an ideal fault free system. Thus, Section 2.2.1 discusses different types of faults and Section 2.2.2 discusses fault detection and mitigation techniques.

2.2.1 Fault classification

> "A fault is the adjudged or hypothesized cause of an error. An active fault produces an error, while a dormant fault does not produce an error. The term "fault" actually denotes an anomalous physical condition in the system, that could be caused by various sources such as manufacturing problem, fatigue, external disturbance, design flaw…" [1]

According to [31], there are two main causes of faults in digital circuits that are deployed in space (see also Figure 8):

- **Aging**:
  - Time-Dependent Dielectric Breakdown (TDDB)
  - Electromigration (EM)
- **Radiation**:
  - Total Ionizing Dose (TID)
  - Single Event Effect (SEE):
    - Non-destructive
    - Destructive

In this context, **aging** is a temporal degradation of circuits significantly affecting the lifetime and the performance of a device. One of the most important aging mechanisms, according to [32], is Negative Bias Temperature Instability (NBTI), which increases the threshold voltage of P-Type Metal-Oxide-Semiconductor (PMOS) and causes the degradation of circuit performance. See [12] for more detailed discussion on TDDB and EM.

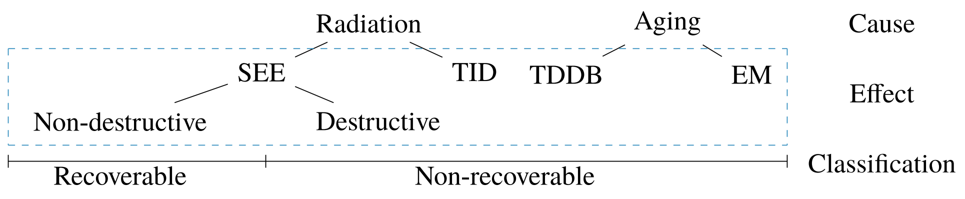


Figure 8. Cause based classification of faults in space [31].

While, according to [3], [31], SEE is a measurable effect caused by an ionizing particle (i.e., **radiation**) as it passes through the semiconductor material, the TID is “a cumulative long term ionizing damage mostly due to protons and electrons” [31].

While aging is an important topic, it is outside of the scope of current thesis as the current thesis focuses on 1) non-destructive (i.e., recoverable) SET faults caused by radiation and on 2) mitigation of those faults. However, it should be noted that aging can increase the rate of SEU [32].

Following the classification was proposed in [28], faults can be divided in three categories based on the fault duration and on the location of fault:

- permanent faults,
- intermittent faults, and
- transient faults.

Both permanent and intermittent faults have a fixed location, however intermittent faults occur intermittently. In contrast, transient faults are relatively short-lived faults that occur randomly both in time and in location, thus making it difficult to predict when and where a (transient) fault might occur because they are usually caused by environmental factors such as radiation.

Detecting **permanent faults**, such as Stuck-at Faults (SAFs) and Bridging Faults (BFs), is a non-trivial task, however, there is a large body of research done on this topic (e.g., see [33]–[35]). For example, BISTs can be used for detecting permanent faults offline, thus it is not suitable for online fault detection. A review of diagnosis techniques for **intermittent faults** in dynamic systems is provided in [36].

Transient faults could be considered to be faults that can lead to Soft Errors (SEs), i.e., an error where a signal value or datum is not correct, however, the circuit is not permanently damaged. Thus cold booting would typically make the system to recover from SEs. In the literature, transient faults are often equated with SEUs:

> "In space high-energy neutrons generated from the interaction of cosmic rays with the atmosphere are the main source of incident radiation. Neutrons cannot cause direct ionization, but the by-products of nuclear reactions with the silicon generate ionizing particles that cause [SEUs]"…SEUs "are mostly induced by alpha particles emitted from radioactive impurities in materials such as packaging, solder bumps and by highly ionizing secondary particles produced from the reaction of both thermal and high-energy neutrons with component materials." [3].

In general, SEEs can be divided into following categories [3]:

- Single Event Upset (SEU),
- Multi-Bit Upset (MBU),
- Single Event Transient (SET),
- Single Event Latch-Up (SEL).

SETs will become SEUs if they are latched. While SEUs and Multi-Bit Upsets (MBUs) are considered non-destructive, Single Event Latch-Ups (SELs) can lead to permanent faults.

2.2.2 Fault mitigation methods

Since neither is it possible to avoid all faults in harsh environments, nor is it usually economically feasible to build a fully fault tolerant system, a middle ground is often required when deploying fault mitigation techniques. However, since those techniques and methods have different costs and benefits, an analysis must be performed.

On the topic of handling faults in the system, [1], [30] discusses the following approaches to fault mitigation:

- fault dismissal,
- fault detection,
- fault masking,
- fault forecasting and
- error correction.

Any fault mitigation approach will include, as discussed in [30], at least one or more approaches from the above mentioned list.

Insignificant faults can be ignored or dismissed in case of **fault dismissal**. Depending on the usage of the data, in some cases faults can be ignored. For example, if a TV set used for home entertainment occasionally and briefly displays some random pixel in a slightly different shade of blue than intended, then this fault/error can probably be ignored.

However, if there is a danger that the fault can lead to failure(s), then other approaches are required, such as fault masking, which require redundancy in the form of additional [28]:

- hardware,
- software,
- information and/or
- time.

For example, [30] uses error detection instead of **fault detection**. However error detection can, in some cases, take place too late and/or too far from the origin of the fault. A fault might manifest itself to the outside world as an error many clock cycles (and, in the worst case, days or more) later. Thus detecting errors prevents applying timely remedies. Moreover, the error might manifest in a module different from the fault location. Thus making it difficult to both localize the fault and to prevent the fault propagation to other modules of the system. For example, parity checker could be used as a lightweight fault detection mechanism for data path (e.g., see [4], [5]). However, much more complicated checker designs have also been implemented [7], [37].

Some fault mitigation techniques rely on fault detection; on the other hand, for some techniques fault detection is not required (e.g., fault mitigation techniques based on fault masking such as Triple Modular Redundancy (TMR)).

The goal of **fault masking** is described in [30]:

> "Fault masking aims at blocking the progress of faults into errors. So, the faults in the system are not fixed but their effects are masked so that no errors are generated that can be observed by the outside world."

A well-known example of fault masking is TMR. In case of TMR, modules are triplicated and the correct output value is assumed to be the value that at least two modules have. For more detailed discussion on the merits of TMR, see Section 2.3.

Error correction can be implemented in many ways, however, Hamming codes and modifications of Hamming code are widely used. For example [30] divides error correction methods into following methods:

- off-line methods:
  - module replacing (with new module)
  - adding additional modules (that fix the faults of original module)
  - soft module fixing (by reprogramming)
  - hard module fixing (by offline diagnostics and fixing);
- on-line methods:
  - automatic repairing/adapting (self-test with automatic reconfiguration)
  - automatic error fixing (e.g., by retransmission)
  - using error correction codes (e.g., Hamming codes).

While Error Correction Codes (ECCs) such as Hamming codes are elegant as a concept, these methods have a significant data and area overhead (see also section 3.1).

A generic **fault forecasting** can provide important insight, e.g., where are the single points of failure, which modules are not shielded from interference, how many faults are expected to occur, etc. However, forecasting the time and location of SET faults is, probably, waste of resources because SET faults by definition are random.

2.3 Discussion on methods

Some faults can be dismissed, especially **dormant faults** that do not cause errors. For example, particle can hit a transistor in a module that is not currently used or is already scheduled for reboot. **Active faults**, however, which would lead to an error should be dealt with if, for example, data integrity is crucial. Thus fault detection and mitigation is needed for active faults. The focus of this thesis is on the active SET faults.

Fault detection could be performed either offline or online. For example, BIST could be used for offline fault detection. However, BIST is more suitable for detecting permanent faults and, moreover, it cannot be used for online fault detection.

TMR could be used for fault masking (see Figure 9). Conceptually the TMR is a simple approach, however, the area overhead of TMR exceeds 200%, since in addition to triplicating the design also a majority voter is added. Moreover, the majority voter becomes a single point of failure. In addition, also power consumption is approximately triplicated.

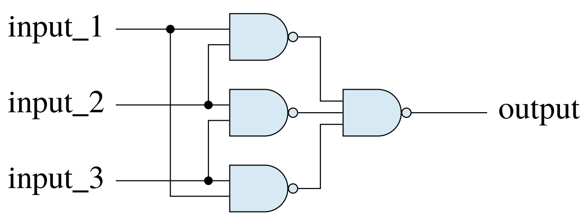


Figure 9. Simple Triple Modular Redundancy from NAND gates.

The TMR can be extended to n-modular redundancy (see Figure 10). While fault coverage increases, also area overhead increases and, similar to TMR, the majority voter becomes a single point of failure.

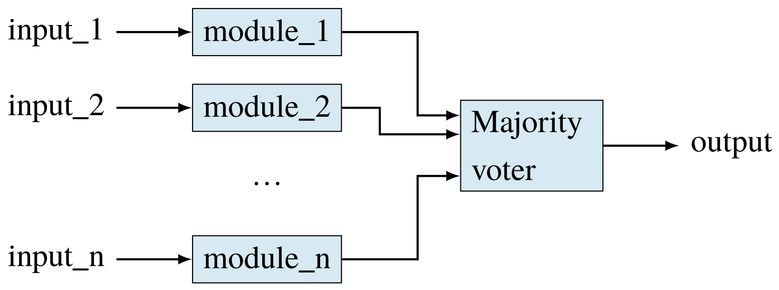


Figure 10. n-modular redundancy.

Thus the aim of current thesis is to provide the same level of fault coverage as TMR, but with substantially smaller area overhead.

3 Literature review of related works

There are several works focusing on obtaining data integrity by applying fault mitigation methods on the physical links in Networks-on-Chip (NoCs). Based on the location where the fault mitigation method is implemented, the principles described in these works can be broadly divided into two categories (e.g., see [4], [5]):

- End-To-End (E2E), and
- Hop-By-Hop (HBH) principle.

These two principles—End-To-End (E2E) and Hop-By-Hop (HBH)—have very different effect on the latency and area overhead. Since methods based on the E2E principle only need fault mitigation methods in the end-points, these methods require less area overhead than methods based on the HBH principle where fault mitigation is implemented not only in the endpoints but also in the intermittent points (i.e., NoC routers). However, since methods based on the E2E principle rely on data retransmission that is triggered only at the endpoint, the methods based on the HBH principle usually have lower latency overhead as in case of the E2E since the entire packet needs to arrive at the destination first. Therefore, when deciding which principle to use, this trade-off of area overhead for latency overhead must be considered.

In case of NoC based SoCs, the methods utilizing the HBH principle have an additional advantage over E2E. If a fault were to occur in the address part of the packet, then the packet might never reach the endpoint and the data would be lost when using methods based on the E2E principle or, in the worse case, lead to system-wide congestion. Therefore, in this thesis only HBH based fault mitigation approaches are considered.

Within the two—E2E and HBH—principles the methods can be further divided into the following categories:

- error correction (see Section 3.1),
- retransmission (see Section 3.2), and
- Relaxed Retransmission (RR) [38] methods (see Section 3.3).

For an overview of related works, see Table 3. Cells marked with red color indicate sub-optimal approach and/or result.

Table 3. Comparison of related works [4].

Approach	Used Method	HBH/E2E	No Extra Buffers	Targeted Fault Model	Correction Latency
[7]	P-RET	E2E	✓	SETs + 50% of METs	$\gg$ 1 clk
[39]	ECC	E2E	✓	SETs	N.A.
[40]	P-RET	E2E	✗	SETs	$\gg$ 1 clk
[41]	P-RET	E2E	✓	SETs	$\gg$ 1 clk
[42]	P-RET	E2E	✓	SETs + METs	$\gg$ 1 clk
[43]	P-RET	E2E	✗	SETs + METs	$\gg$ 1 clk
[44]	ECC	HBH	✗	SETs	N.A.
[45]	ECC	HBH	✗	SETs	N.A.
[46]	ECC	HBH	✗	SETs	$\gg$ 1 clk
[47]	ECC	HBH	✗	SETs + METs	N.A.
[48]	P-RET	HBH	✗	SETs + Some DET	N.A.
[49]	F-RET	HBH	✗	SETs + METs	3 clks
[50]	F-RET	HBH	✓	SETs	N.A.
[38]	RR + ECC + EDC	HBH	✗	SETs + METs (Prediction)	2 clks
[4]	**AWAIT1**	**HBH**	✓	**SETs + 50% of METs**	**0–1 clk**
[5]	**AWAIT2**	**HBH**	✓	**SETs + 50% of METs**	**1 clk**

3.1 Error correction methods

Methods based on Error Correction Codes (ECCs) have low latency, because errors are corrected online and interruptions or retransmissions are not required. Since data integrity is important for NoC based systems, methods based on ECC could be used in NoCs. For example, works such as [44]–[46] have provided solutions for correcting SET faults using ECCs. Moreover, [47] has applied ECC techniques to cover Multi-Event Transient (MET) faults.

ECC based methods can be implemented either as E2E [44]–[47] or HBH [39]. Regardless of chosen principle (E2E or HBH), the ECC based methods have one disadvantage: overhead, i.e., both area and data overhead. For example, implementing Hamming code on NoC router links would require approximately 11 times more additional area compared to parity checker; similarly the number of data bits that the basic version of Hamming code would require is eight times more than that of parity checker [51]. Therefore ECC based methods would have an increase in area overhead even for correcting SET faults and in addition the number of bits used for data transmission is reduced by eight.

3.2 Data retransmission methods

Data retransmission methods resend data when fault is detected. Implementing retransmission methods relies on assumption that incorrupt data is stored and can be retrieved or regenerated. While data retransmission can also be divided according to E2E and HBH principles, there is an additional element to consider: data can be retransmitted as full packages (Packet-Based Retransmission (P-RET)) (see Section 3.2.1) or as flits (Flit-Based Retransmission (F-RET)), i.e., chunks of data (see Section 3.2.2), instead of full packets.

3.2.1 Packet retransmission

Especially in case of S-A-F, packet transmission requires that the whole packet is stored in a module before it is transmitted to the next module. This has several drawbacks:

- increased number of router buffers,
- all packets must have the same (maximum) size.

Several papers, including [7], [40]–[43] have proposed data retransmission methods based on E2E principle. However, methods based on E2E principle have a high latency overhead. The increased latency is due to the fact that fault/error detection is implemented only in the endpoint. Since NoCs are inherently multi-hop systems, then initializing data retransmission would require accessing data that was sent from the initial module several clock cycles ago. Thus retransmission would make the data arrive several clock cycles later. Moreover, if there are data dependencies, retransmitting or rolling back one packet might require rolling other packets too, thus the latency penalty might apply on more than one packet. Consider, for example, video transmission: if video frame $\texttt{frame}_t$ has to be retransmitted, then also frame $\texttt{frame}_{t+1}$ has to be retransmitted to assure, that the sequence of frames (especially when using compressed video codecs) is unaltered. Thus even non-faulty packets might have to be dropped and retransmitted.

Therefore, a packet retransmission method using E2E principle packet would require a distributed packet dropping mechanism in order to prevent a network-wide failure. This makes packet retransmission using E2E principle a less feasible option compared to flit based methods.

Consequently, methods based on E2E principle would also have area overhead penalty because packets would have to be stored for several clock cycles because they might be needed for data retransmission. Thus, each initial data sending module would require additional input buffers (see Equation 4).

$$\begin{matrix}\texttt{additional}\\ \texttt{buffers}\\ \texttt{in bits}\end{matrix} = \max\begin{pmatrix}\texttt{transmission}\\ \texttt{length in}\\ \texttt{clock cycles}\end{pmatrix} \times \max\begin{pmatrix}\texttt{packet}\\ \texttt{size}\\ \texttt{in bits}\end{pmatrix} \tag{4}$$

The issue with increased latency is addressed by [48] by proposing a HBH based P-RET that uses S-A-F switching method. While S-A-F method alleviates the latency overhead associated with E2E based systems, the S-A-F method has its own adverse impact on latency and area overhead due to additional input buffers at each NoC router for storing the entire packet before transmission compared to flit level switching, such as wormhole switching.

These limitations of packet based retransmission are addressed when using Flit-Based Retransmission (F-RET) (see Section 3.2.2).

3.2.2 Flit retransmission

The problems associated with packet transmission and S-A-F can be alleviated by using wormhole switching method which is flit based. For example, [49], [50] propose HBH based methods of F-RET. While HBH based F-RET methods have lower latency overhead compared to based on E2E principle, and lower area overhead compared to HBH based P-RET, nevertheless, those HBH based F-RET methods require additional input buffers for storing the incoming flits.

3.3 Relaxed data retransmission methods

Relaxed Retransmission (RR) [38] intends to alleviate shortcomings of ECC (see Section 3.1) and data retransmission (both packet and flit) based methods (see Section 3.2). Instead of correcting the fault/error as in case of ECC based methods, the Relaxed Retransmission (RR) takes advantage of the nature of transient faults: these faults are short-lived and their effects vanish over time. Thus the RR method proposes waiting until the transient faults disappear and continuing transmission after this pause. This guarantees very fast recovery in case of faults.

While the RR method was proposed by [38], the approach in the paper had several drawbacks, including predicting the occurrence of transient faults, which by definition are random, thus not predictable.

The AWAIT methods (see Sections 4.2 and 4.3) propose an alternative RR method for SET fault mitigation. While the methods proposed in this thesis and the method in [38] all rely on the transient nature of SETs, these methods are unrelated and the term RR is used here only to refer to the generic approach.

4 Methodology and experimental results

4.1 Analysis of reliability of baseline design

Experiments were conducted to provide evidence that fault mitigation mechanism (such as AWAIT1) is needed. These experiments were conducted on a 4×4 2D NoC using Bonfire routers (hereafter "*baseline*") which, as explained in Section 2.1.3, utilize wormhole switching using 4-flit FIFO buffers without any fault mitigation mechanisms. The duration of each experiment was 100 thousand clock cycles. These experiments were repeated using multiple different Fault Injection Rate (FIR) and Packet Injection Rate (PIR) values (see Figures 11, 12 and 13).

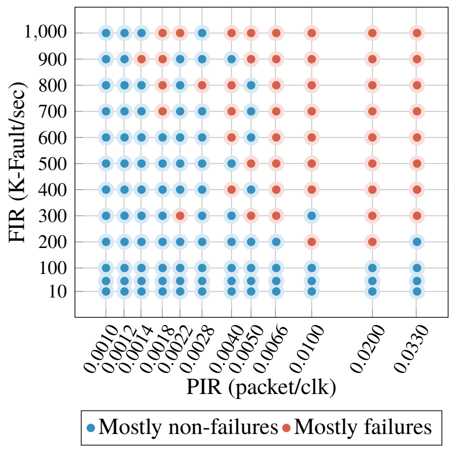


Figure 11. Effect of faults with duration of 10% of clock period during 100,000 clock cycles [4].

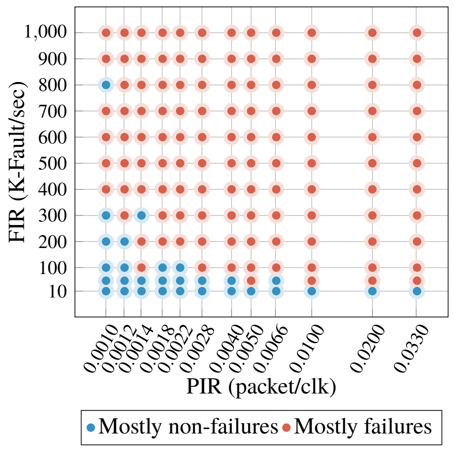


Figure 12. Effect of faults with duration of 100% of clock period during 100,000 clock cycles [4].

In these figures, each data point is 'best out of five' experiments. In other words, a data point is considered as a 'non-failure' (depicted in blue color) if most of the five experi-

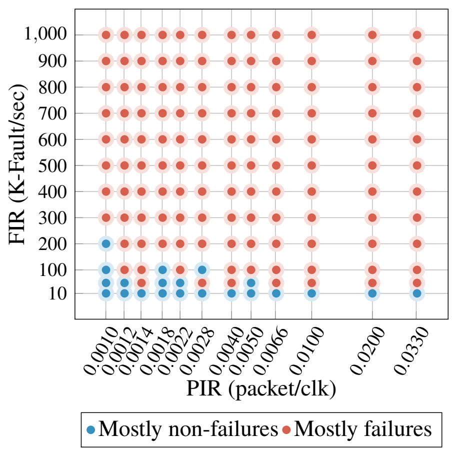


Figure 13. Effect of faults with duration of 200% of clock period during 100,000 clock cycles [4].

ments were failure free, while it was considered to be a ‘failure’ (depicted in red color) if most of the five experiments led to failure. Thus at least three 100 thousand clock cycle experiments had to lead to failure for a data point to be considered as ‘failed’. Failure in this context mean that not all transmitted packets arrived to destination.

Based on the results, the hypothesis that failures increase when PIR and FIR increase cannot be rejected. At very low PIR values (see Figure 11), it appears that the system is not starting to fail even when FIR increases (and *vice versa*). This sounds at first contradicting the hypothesis, however, if either FIR or PIR is sufficiently low, then the probability that the fault becomes a SEU approaches zero. Moreover, the blue data points denote ‘mostly failure free’ operations instead of ‘failure free’ operations, since most of the five experiments did not encounter failures.

The results of these experiments failed to reject the hypothesis that increase in fault duration increases the probability of a fault becoming a failure (see Figures 11, 12 and 13).

Hence, if the system is expected to operate without failures, then the baseline model without any fault mitigation mechanism cannot provide the required level of dependability under harsh conditions (such as space, etc).

Therefore, in order to improve the reliability of the baseline model, this thesis proposes AWAIT1 (see Section 4.2). However, AWAIT1 has limitations. Thus, for systems with more stringent reliability requirements, this thesis proposes a second alternative: AWAIT2 (see Section 4.3).

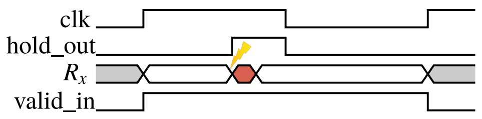
4.2 AWAIT1: a lightweight fault mitigation mechanism

A lightweight online fault mitigation mechanism—**AWAIT1**—is proposed in the current thesis. It was also published in [4] (see Appendix 1). The goal of AWAIT1 mechanism is to address SET faults on links between modules such as NoC routers (hereafter *modules*) to increase reliability. In broad terms, the AWAIT1 fault mitigation mechanism is a Relaxed Retransmission (RR) type fault mitigation mechanism. AWAIT1 is implemented using parity for fault detection in order to achieve low area overhead (however other fault detection mechanisms can be used instead or along the parity checker). Checker design and optimization, however, is out of the scope of this thesis. AWAIT1 was implemented on the baseline Bonfire NoC router (see Section 2.1.3 for description).

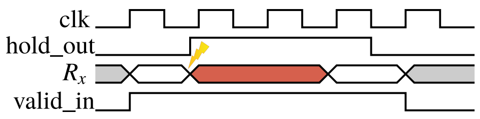
4.2.1 AWAIT1 mechanism

The AWAIT1 fault mitigation approach relies on the transient nature of SETs. However, in comparison with [38], the AWAIT1 achieves lower latency. Moreover, compared to [50] which uses HBH based F-RET, the AWAIT1 method does not require additional buffers.

When the downstream module detects a SET fault, a signal is sent to the upstream module to 'pause' the current operation and hold the incorrupt value at the output. Once the effect of a SET fault disappears on the module link, the downstream module receives the incorrupt value (in red) and signals the upstream module to continue normal operations (see Figure 14). Moreover, the 'pause' signal propagates to other modules along the route of the packet.



(a) Transient fault with duration of 10% of clock period.



(b) Transient fault with duration of 200% of clock period.

Figure 14. Pausing the system in case of transient faults.

Thus, in order to use the RR method to mitigate SET faults, three steps are required for AWAIT1:

1. The downstream module must detect the presence of fault(s).
2. The downstream module must stop sampling data.
3. The upstream module must keep the transmitted data on the link until it is certain that the downstream module has received the correct, fault-free data.

The main idea behind AWAIT1 is that the upstream module holds values in outputs (and in input buffers, if necessary) constant until the downstream module signals that the data was received successfully. A simplified version of the AWAIT1 mechanism is illustrated by Figure 15.

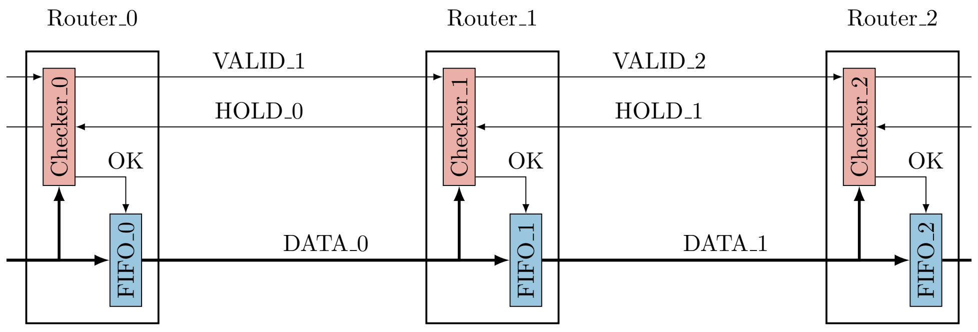


Figure 15. AWAIT1 simplified design.

The middle module (marked as '`Router_1`' in Figure 15) only updates the FIFO buffer ('`FIFO_1`') if the downstream module ('`Router_2`') does not send '`HOLD`' ('`HOLD_1`') signal, the upstream module ('`Router_0`') sends a '`VALID`' signal ('`VALID_1`') and '`Checker_1`' does not detect a fault in data ('`DATA_0`'), thus sending '`OK`' signal to the FIFO (see Equation 5).

$$\mathtt{OK} = \mathtt{VALID} \bigwedge \overline{\left(\mathtt{FAULT} \bigvee \mathtt{HOLD}\right)} \tag{5}$$

Since fault did not occur, the middle module pulls the '`HOLD`' signal low and the upstream module can update its FIFO if and only if new uncorrupted input data is available.

However, in case '`Checker_1`' detects a fault, the the middle module signals both the upstream ('`Router_0`') and the downstream ('`Router_2`') modules. Firstly, the '`VALID_2`' is pulled to low, thus the downstream module does not update its FIFO. Secondly, the upstream module receives '`HOLD_0`' signal and therefore keeps its output until the middle module signals successful transmission by pulling '`HOLD_0`' to low.

A more detailed schematic of the AWAIT1 is presented in Figure 16, where Bonfire routers are used as examples. Changes to the baseline design are marked with red for modules ('Hold generator' and 'Parity checker') and with blue for individual elements. It can be seen that the required changes are minimal.

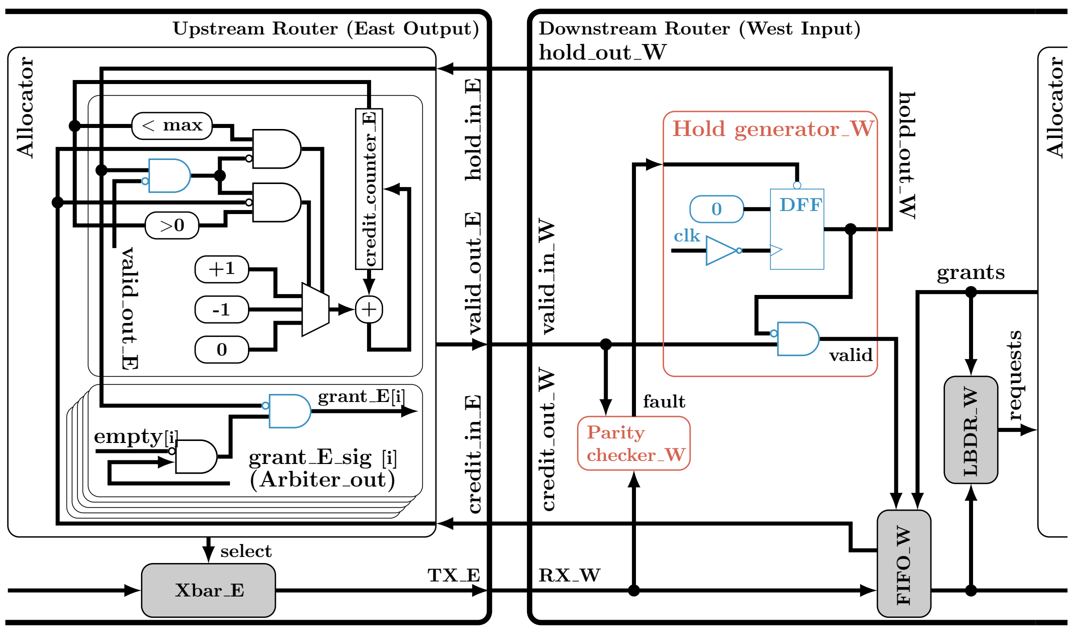


Figure 16. AWAIT1 applied to Bonfire router [4].

The AWAIT1 method relies on a single parity bit for fault detection, however other fault detection methods could be used instead or together with the parity checking. Using a single parity bit enables detection and correction of all single bit faults on links between modules. While, for example, Hamming code provides a single bit fault correction, it would require eight times more data overhead [4] per flit and about ten times more area overhead for 32-bit links compared to parity checker for single bit fault detection.

4.2.2 AWAIT1 experimental results

A set of experiments were conducted to demonstrate that a simple fault mitigation mechanism (such as AWAIT1) can provide significant dependability increase.

To validate the effectiveness of AWAIT1 mechanism, the design was simulated using ModelSim [52] and uniformly distributed random faults were injected into the signals of a 4×4 2D NoC with Bonfire routers utilizing AWAIT1 mechanism by setting signal values to incorrect values and freezing those values for a duration of a modeled transient fault. In order to measure the latency overhead, 1500 experiments were conducted. The experiments used three variables: fault duration, PIR and FIR. While in these experiments the fault duration had only three levels: 10%, 100% and 200% of clock cycle; the FIR, for example, ranged from 0 to 80 million faults per second (see Figures 17, 18 and 19). Every data point is an average of five experiments. Each of the three figures also includes a fault free baseline reference (marked as orange lines at $FIR = 0$).

Not only did the system operate without failures (see Sections 4.2.3, 4.3 and 4.4 for critique of the AWAIT1 mechanism), AWAIT1 induced only marginal increase in latency (see, especially, Figure 17).

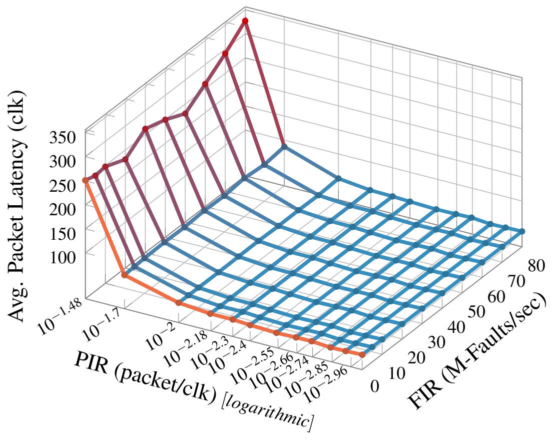


Figure 17. AWAIT1: Average packet latency at fault duration of 10% of clock period under different Packet Injection Rate and Fault Injection Rate values [4].

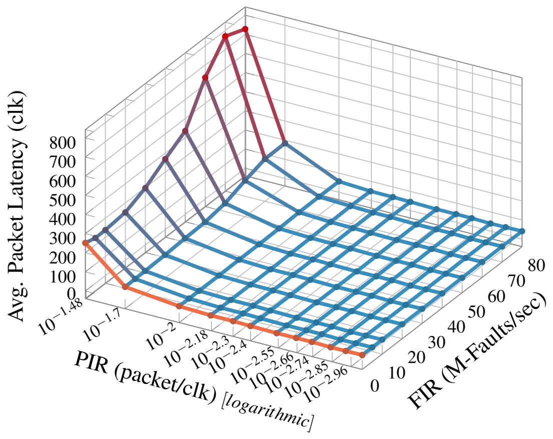


Figure 18. AWAIT1: Average packet latency at fault duration of 100% of clock period under different Packet Injection Rate and Fault Injection Rate values [4].

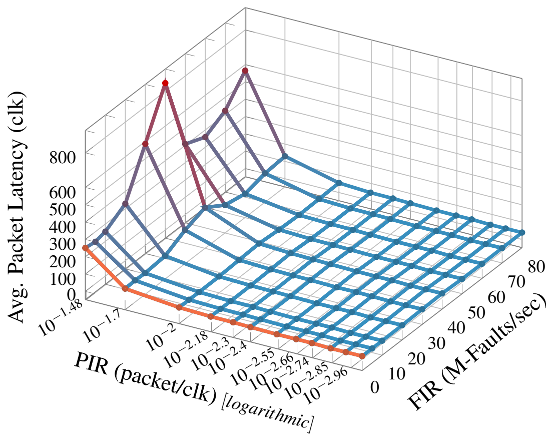


Figure 19. AWAIT1: Average packet latency at fault duration of 200% of clock period under different Packet Injection Rate and Fault Injection Rate values [4].

Both—the baseline NoC without fault mitigation and NoC with AWAIT1 mechanism—are implemented in RTL in VHDL. When these two designs are synthesized in Synopsys Design Compiler [53] using Taiwan Semiconductor Manufacturing Company (TSMC) 40 nm Complementary Metal-Oxide-Semiconductor (CMOS) technology standard library, this

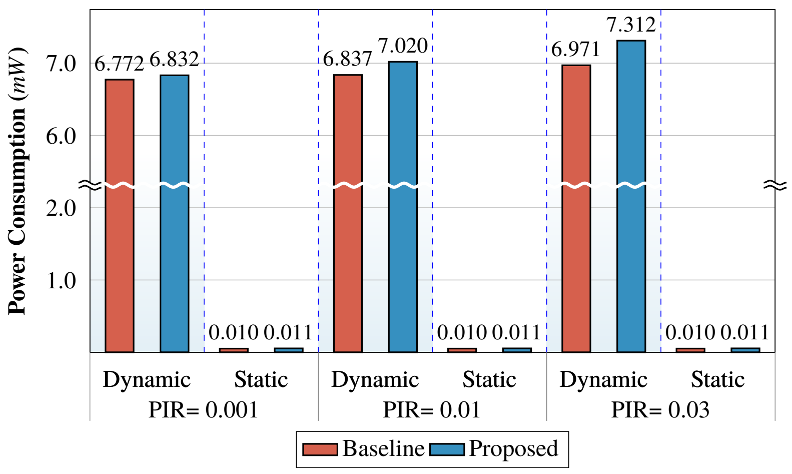


Figure 20. AWAIT1: Power consumption (in mW) of the proposed and baseline 4×4 Network-on-Chip under different Packet Injection Rate [4].

improved dependability is achieved with $\approx 5.1\%$ area overhead (see Table 4). Moreover, the critical-path delay was not increased (Table 4 shows, in fact, that critical-path delay decreased which is contra-intuitive at first, however, the additional logic elements might have made it possible for the synthesis tools to simplify the design. For example, two 'inversion' gates could be optimized out into a wire, thus reducing critical-path delay).

Table 4. AWAIT1: area and critical-path delay overhead [4].

	Area	**Critical-Path Delay**
Baseline design	8289.38 μm^2	1.96 ns
AWAIT1 module	8719.56 μm^2	1.94 ns
Overhead	$\approx 5.1\%$	$\approx 0.0\%$

Finally, power consumption overhead was obtained for both the baseline design and the Bonfire NoC with integrated AWAIT1 at three different PIR values (see Figure 20). The power consumption overhead of AWAIT1 mechanism is mostly due to dynamic power consumption and averaging roughly 2.89% increase (see Table 5).

Table 5. AWAIT1: Power consumption overhead.

PIR	Category	Dynamic	Static	Total
0.001	Baseline	6.772 *mW*	0.010 *mW*	6.782 *mW*
	AWAIT1	6.832 *mW*	0.011 *mW*	6.843 *mW*
	Change	0.87%	10.00%	0.90%
0.010	Baseline	6.837 *mW*	0.010 *mW*	6.847 *mW*
	AWAIT1	7.020 *mW*	0.011 *mW*	7.031 *mW*
	Change	2.68%	10.00%	2.69%
0.020	Baseline	6.971 *mW*	0.010 *mW*	6.981 *mW*
	AWAIT1	7.312 *mW*	0.011 *mW*	7.323 *mW*
	Change	4.90%	10.00%	2.83%
Average change		2.82%	10.00%	2.83%

4.2.3 AWAIT1: discussion

The AWAIT1 mechanism has two main drawbacks. One of the weaknesses of AWAIT1 is that the upstream module cannot process new input data, while the output has to stay unchanged while waiting for acknowledgment from the downstream module. The other, and the most severe limitation of AWAIT1, is that faults that occur very close to the rising clock edge (see Figure 21) cannot be detected before the faulty data is latched into register(s).

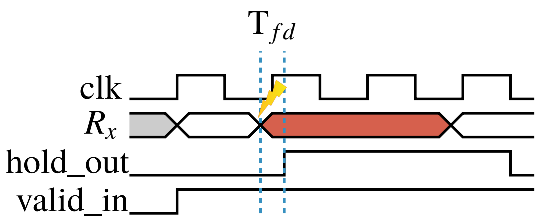


Figure 21. Fault detection delay.

In Equation 6, Δt denotes time of occurrence of a fault measured starting from the previous rising clock edge, T_{clk} denotes clock period and T_{fd} is the time necessary for the checker (in this case parity checker) to evaluate the input data. Thus, if the fault occurs too close to the next rising clock edge, the checker will fail to detect the fault (before the rising edge).

$$\begin{matrix}\texttt{AWAIT1} \\ \texttt{mitigation}\end{matrix} = \begin{cases} \texttt{success} & if\ \mathrm{T}_{clk} - \Delta t > \mathrm{T}_{fd} \\ \texttt{failure} & if\ \mathrm{T}_{clk} - \Delta t \leq \mathrm{T}_{fd} \end{cases} \tag{6}$$

Nevertheless, since the AWAIT1 mechanism has only a small overhead, it can be used to add reliability, especially, if the clock period is significantly longer than the critical path of parity checker. However, additional fault tolerance mechanisms, e.g., packet dropping, is needed to avoid network-wide congestion.

In order to address these drawbacks of AWAIT1, AWAIT2 was designed (see Section 4.3).

4.3 AWAIT2: optimized fault mitigation mechanism

As an improvement to the AWAIT1 mechanism, in this thesis an **AWAIT2** method is introduced. The main goal of AWAIT2 is to enable data rollback and retransmission without requiring additional registers. Similar to AWAIT1, the AWAIT2 method was implemented on a NoC with Bonfire NoC routers.

4.3.1 AWAIT2 mechanism

A simplified schematic for AWAIT2 is presented in Figure 22. The changes to the baseline design are marked in blue and red color. The main changes include a multiplexer, two additional buffers (REG_A and REG_B), and a checker (PARITY).

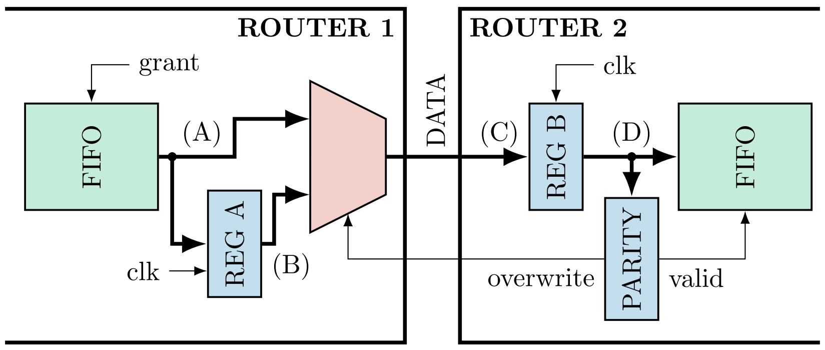


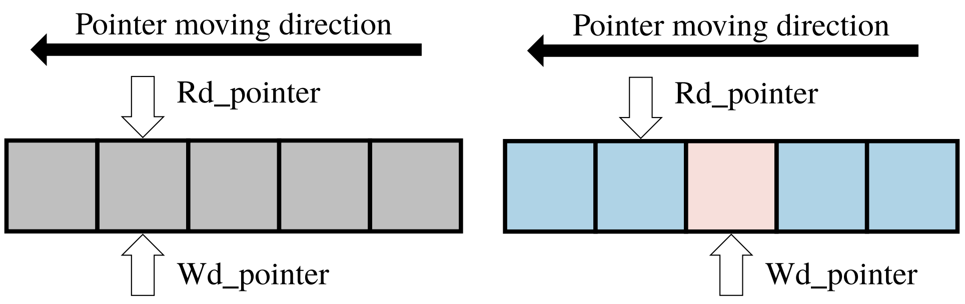
Figure 22. AWAIT2: schematic [5].

While AWAIT1 mechanism is designed to prevent that SET faults become latched into a register, the AWAIT2 method uses an input register (REG_B in Figure 22) to latch the fault and implements fault detection on the values of this register. The rationale is that if the transient fault is short-lived and not present at the rising clock edge, then the fault can be dismissed. Such a fault is irrelevant since it does not become latched into the REG_B.

However, if the transient fault is present at the rising clock edge, then the faulty value is latched into `REG_B`. In contrast to AWAIT1, the transient fault can be mitigated always, even if the fault occurred close to the rising clock edge, since AWAIT2 is designed to latch the fault into an input register, thus getting rid of the limition of AWAIT1.

In case a fault is detected, the downstream module (`Router_2`) requests retransmission by sending the `overwrite` signal (see Figure 22) to the upstream module (`Router_1`). The upstream module stores the previous value in a register (`REG_A`). If the `overwrite` signal is set then previous data is retransmitted, if it is not set then new data is transmitted.

While `REG_B` is in fact a normal input buffer already existing in the baseline design, the `REG_A` appears, at first glance, to be an additional register. However, the AWAIT2 mechanism does not require an additional register because it reuses the unused register that is part of the architecture of circular buffers (see Figure 23). Thus enabling data rollback without adding additional registers (see Figure 24).



(a) Empty FIFO.　(b) Full FIFO.

Figure 23. AWAIT2: circular buffer [5].

When RTL designs are synthesizing using TSMC 40 nm CMOS technology standard cell library in Synopsys Design Compiler, the optimized version of AWAIT2 requires approximately half of the area overhead of the non-optimized version (see Figure 24) with two additional buffers (`REG_A` and `REG_B`).

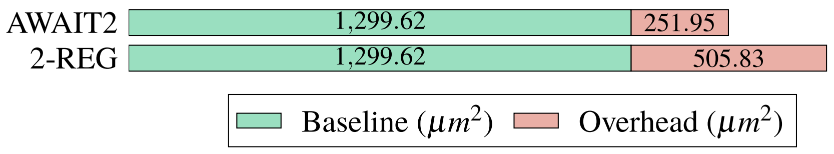


Figure 24. Comparison of AWAIT2 with and without optimization.

In other words, the previous value is stored in the FIFO slot with index of current read pointer minus one (`Rd_pointer` − 1). If the FIFO is empty (`Rd_pointer` = `Wr_pointer`) then the data at `Rd_pointer` − 1 is still stored there; and even when the FIFO is full

($\texttt{Rd\_pointer} - 1 = \texttt{Wr\_pointer}$) because writing is disabled until read pointer value is increased. Therefore, REG_A is also already present in the baseline design!

However, some changes are required (see Figure 25). These changes are marked in red color in the figure. Among these changes are changes necessary to add a parity checker. Also the read and write pointer logic is updated: the pointer is not incremented if a fault is detected. Moreover, one additional register is needed for both read and write pointers, i.e., the previous value.

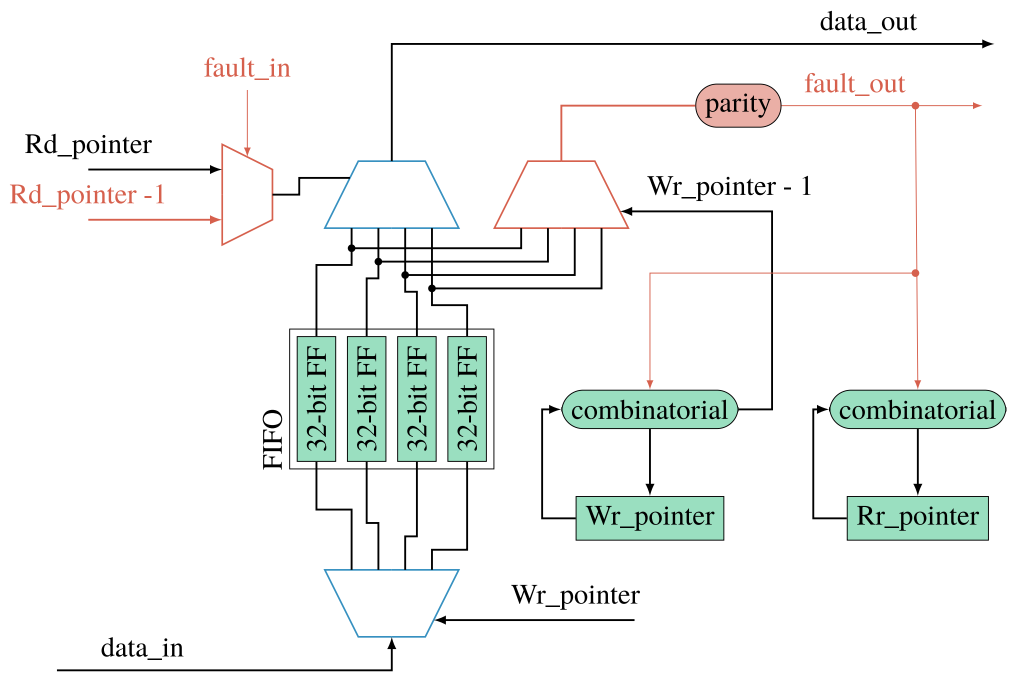


Figure 25. AWAIT2: ReUSE input buffer.

4.3.2 AWAIT2 experimental results

The experimental results were obtained by applying AWAIT2 on data links of a 4×4 2D Bonfire NoC (see Section 2.1.3). The system was tested by inserting SET faults at random time with varying settings such as duration.

The fault injection was performed in Modelsim simulation by forcing signals to take faulty values randomly using uniform distribution. Figures 26, 27 and 28 show how the fault free baseline experiments (marked with orange color lines in the figures at $FIR = 0$) compares to experiments of AWAIT2 with fault injection.

The three figures illustrate how latency of the system changes when parameters such as length of the SET faults, FIR and PIR are changed. Even under harsh conditions (80 million faults per second), the system continues to operate without failures. Only in case of long SET faults—100% and 200% of clock period—and high PIR when the network is already saturated is there noticeable increase in latency compared to the fault free baseline

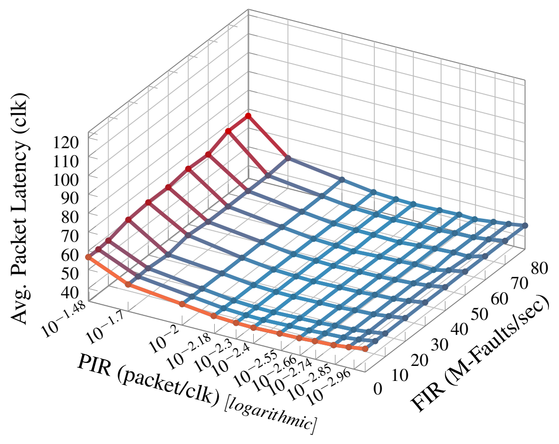


Figure 26. AWAIT2: Average packet latency at fault duration of 10% of clock period under different Packet Injection Rate and Fault Injection Rate values [5].

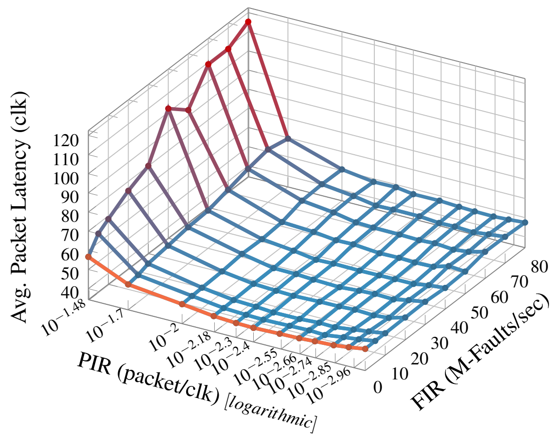


Figure 27. AWAIT2: Average packet latency at fault duration of 100% of clock period under different Packet Injection Rate and Fault Injection Rate values [5].

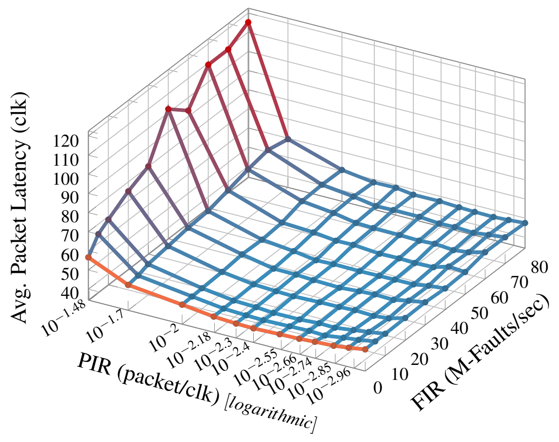


Figure 28. AWAIT2: Average packet latency at fault duration of 200% of clock period under different Packet Injection Rate and Fault Injection Rate values [5].

model. Thus the AWAIT2 shows promise to be applicable for deployment in, for example, space.

By implementing both the baseline and AWAIT2 version of 4×4 2D NoC with Bonfire routers for 400 MHz clock frequency in RTL in VHDL and synthesizing using TSMC

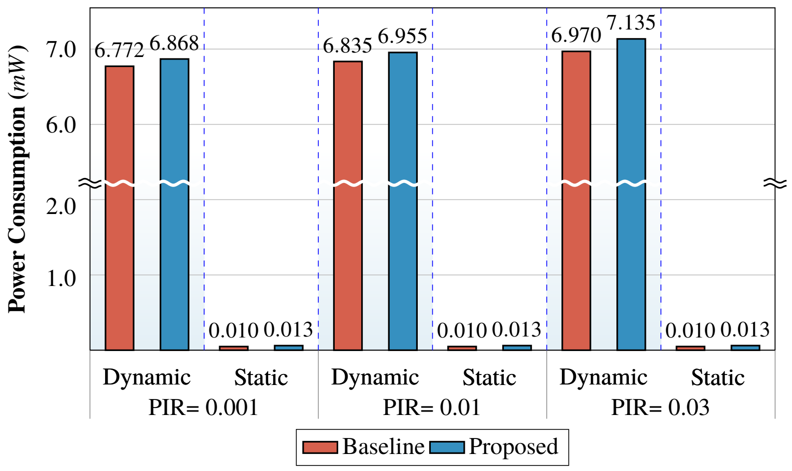


Figure 29. AWAIT2: Power consumption (in mW) of the proposed and baseline 4×4 Network-on-Chip under different Packet Injection Rate [5].

40 nm CMOS technology standard cell library in Synopsys Design Compiler, the results in Table 6 were obtained. AWAIT2 increases both area (18.1%) and critical-path delay (7.8%) overhead. Thus, when only comparing overhead measures, AWAIT1 is superior to AWAIT2. However, this increased overhead is used to overcome the drawbacks in AWAIT1. Morever, 18.1% area overhead is insignificant compared to 200% area overhead of TMR.

Figure 29 provides a comparison of power usage for AWAIT2 and baseline model. Both dynamic and static power usage of AWAIT2 is roughly equal to that of the baseline model. Thus the increase in power usage is minimal (see Table 7), while securing the system from SET faults.

Table 6. AWAIT2: area and critical-path delay overhead [5].

	Area	**Critical-Path Delay**
Baseline design	8276.45 μm^2	2.28 ns
AWAIT2 module	9777.26 μm^2	2.46 ns
Overhead	$\approx 18.1\%$	$\approx 7.8\%$

Table 7. AWAIT2: Power consumption overhead.

PIR	Category	Dynamic	Static	Total
0.001	Baseline	6.772 *mW*	0.010 *mW*	6.782 *mW*
	AWAIT2	6.868 *mW*	0.013 *mW*	6.881 *mW*
	Change	1.42%	30.00%	1.46%
0.010	Baseline	6.837 *mW*	0.010 *mW*	6.847 *mW*
	AWAIT2	6.955 *mW*	0.013 *mW*	6.968 *mW*
	Change	1.73%	30.00%	1.77%
0.020	Baseline	6.970 *mW*	0.010 *mW*	6.980 *mW*
	AWAIT2	7.135 *mW*	0.013 *mW*	7.148 *mW*
	Change	2.37%	30.00%	2.41%
Average change		1.84%	30.00%	1.88%

4.3.3 AWAIT2: discussion

In contrast to AWAIT1, the AWAIT2 method can detect all significant SET faults, i.e., SET faults that become SEU.

However, neither AWAIT1 nor AWAIT2 can handle permanent faults. Permanent faults were outside of the scope of current thesis. Nevertheless, it must be noted that in case of permanent faults AWAIT mechanisms would cause parts or, in the worse case scenario, whole system to stop operating. Thus additional fault tolerance mechanisms (e.g., packet dropping, reconfiguration of turn models) are needed.

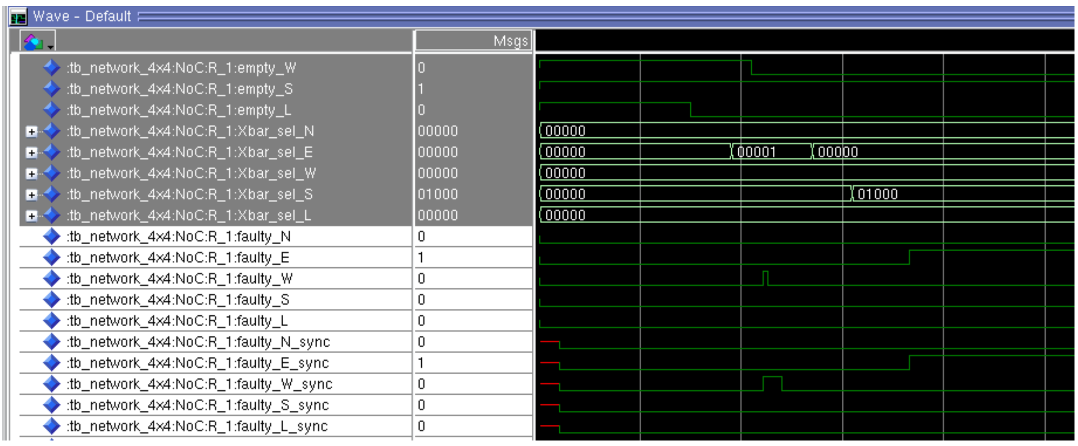
4.4 Additional tests on AWAIT1 and AWAIT2

It was theorized in [4] that the AWAIT1 mechanism would fail to mitigate faults if SET (or MET) faults happen close to the rising clock edge (see Equation 6). Therefore, additional experimental results were generated. These results are obtained using a modified version of AWAIT1 mechanism (see Figure 30). The line number four was altered by adding "`after PAR_DELAY`" to model critical-path delay of the parity checker.

```
err_check : process(RX, valid_in)
begin
  if valid_in='1' and XOR_REDUCE(RX) /= '0' then
    faulty <= '1' after PAR_DELAY; -- critical path delay;
  else
    faulty <= '0';
  end if;
end process; -- err_check
```

Figure 30. AWAIT1: Simulating critical-path of parity checker.

For example, simulating the modified AWAIT1 with a critical-path delay equal to 80% of clock cycle, the system failed even when using AWAIT1 fault mitigation mechanism (see Figure 31). Signal `faulty_E` becomes latched to 1 even though the faults are transient, thus the signal that fault was detected should become 0 when the effect of the fault disappears.



```
:tb_network_4x4:NoC:R_1:empty_W        0
:tb_network_4x4:NoC:R_1:empty_S        1
:tb_network_4x4:NoC:R_1:empty_L        0
:tb_network_4x4:NoC:R_1:Xbar_sel_N     00000
:tb_network_4x4:NoC:R_1:Xbar_sel_E     00000
:tb_network_4x4:NoC:R_1:Xbar_sel_W     00000
:tb_network_4x4:NoC:R_1:Xbar_sel_S     01000
:tb_network_4x4:NoC:R_1:Xbar_sel_L     00000
:tb_network_4x4:NoC:R_1:faulty_N       0
:tb_network_4x4:NoC:R_1:faulty_E       1
:tb_network_4x4:NoC:R_1:faulty_W       0
:tb_network_4x4:NoC:R_1:faulty_S       0
:tb_network_4x4:NoC:R_1:faulty_L       0
:tb_network_4x4:NoC:R_1:faulty_N_sync  0
:tb_network_4x4:NoC:R_1:faulty_E_sync  1
:tb_network_4x4:NoC:R_1:faulty_W_sync  0
:tb_network_4x4:NoC:R_1:faulty_S_sync  0
:tb_network_4x4:NoC:R_1:faulty_L_sync  0
```

Figure 31. Example of AWAIT1 mechanism failure.

Modifying the design of FIFO buffer in AWAIT2 (see lines 6 and 7 in Figure 32), the AWAIT2 mechanism continues to operate without failures.

```
1 process (written_data, write_pointer, write_pointer_prev)
2 begin
3   fault <= '0';
4   write_pointer_effective <= write_pointer;
5   if XOR_REDUCE(written_data) /= '0' then
6     fault <= '1' after PAR_DELAY; -- critical path delay;
7     write_pointer_effective <= write_pointer_prev after
         PAR_DELAY;
8   end if;
9 end process;
```

Figure 32. AWAIT2: Simulating critical-path of parity checker.

4.5 Discussion on AWAIT mechanisms and future research

AWAIT1 has a limitation since it is sensitive to the time when a fault occurs (see Section 4.3.1). If the fault occurs very close to the next rising clock edge, then AWAIT1 is unable to detect the fault on time and the faulty value will be latched into register and, thus, propagates to the next module. This problem manifests itself more severely when using higher clock rates.

In contrast, AWAIT2 delays fault detection and performs fault detection on latched value, thus using the fault/error latching as a feature. Therefore, the AWAIT2 mechanism addresses this short-falling of AWAIT1.

However, AWAIT2 requires that the module has an input buffer—namely circular input buffer. If the design implementing AWAIT2 does not have circular input buffer, then additional area overhead would be introduced by adding a buffer. Moreover, if the design uses "Mealy machine" [54] type of architecture, adding input buffers would increase latency. Thus more generic fault mitigation technique is needed.

The AWAIT methods have one serious limitation: these methods only mitigate transient faults. Occurrence of permanent faults would, in the worse case scenario, bring the whole system to a stop. Because the AWAIT mechanisms are designed to pause the route until the fault disappears, a permanent fault would make the whole route stop data transmission. Therefore, additional fault tolerance mechanisms are needed to drop packets or to reset the network, and to exclude the faulty node or link by updating turn models in routers.

The current methods are only used on securing links between modules. In some systems, such as CPU pipelines, the output data is a function of the input, thus the combinatorial logic of the module needs to be secured too. While this was outside of the scope of current thesis, future work ought to focus on protecting combinatorial logic from transient faults in, for example, a pipelined CPU.

5 Summary

Designing dependable system for harsh conditions (such as space) poses challenges, since the system cannot be economically serviced after deployment. This means that the systems have to be designed with special attention on reliability. This includes hardening the design against SET faults using fault mitigation methods.

A simple solution to do this would be using TMR as it requires relatively small changes to the design. However, TMR increases area overhead over 200%. Thus, other methods must be considered.

This thesis proposes using two online fault mitigation methods: AWAIT1 and AWAIT2. Both of these are Relaxed Retransmission (RR) type of fault mitigation methods and operate on Hop-By-Hop (HBH) basis. Thus data is verified at each module not just at the endpoints.

Firstly, AWAIT1 was introduced as an ultra-lightweight fault mitigation method. AWAIT1 can mitigate all SET faults (that do not occur in a critically close to the next rising clock edge) by utilizing simple parity checker for fault detection. The area, critical-path delay and power consumption overheads of AWAIT1 are, respectively, 5.1%, 0% and 2.83%.

However, AWAIT1 has two shortcomings: 1) new data is processed only after current data has been validated by next module, and 2) there is a critical time-frame before the next rising clock edge when fault mitigation cannot detect the fault. Even though, the AWAIT1 method cannot detect 100% of faults, it can be useful method to increase reliability, especially if combined with other existing fault tolerance methods, such as packet dropping.

To address the shortcomings of AWAIT1, an upgraded version—AWAIT2—was introduced. AWAIT2 mitigates all SET faults. Because AWAIT2 uses latched values for fault detection, it can detect all faults regardless of the time when the fault occurred. However, this increase in reliability comes at a cost: the area, critical-path delay and power consumption overheads of AWAIT2 compared to the baseline design are, respectively, 18.1%, 7.8% and 1.88%. While the area and critical-path delay measures are significantly higher than AWAIT1, the power consumption overhead has, in fact, decreased.

The experimental results show that AWAIT mechanisms mitigate faults even in severe conditions (80 million faults per second).

References

[1] A. Simevski, “Architectural framework for dynamically adaptable multiprocessors regarding aging, fault tolerance, performance and power consumption”, Brandenburgische Technische Universität Cottbus–Senftenberg, 2015.

[2] J. Kawaguchi, “The Hayabusa mission — its seven years flight”, in *2011 Symposium on VLSI Circuits - Digest of Technical Papers*, Jun. 2011, pp. 2–5.

[3] N. Bidokhti, “SEU concept to reality (allocation, prediction, mitigation)”, in *2010 Proceedings - Annual Reliability and Maintainability Symposium (RAMS)*, Jan. 2010, pp. 1–5. DOI: `10.1109/RAMS.2010.5448078`.

[4] K. Janson, R. Pihlak, S. P. Azad, B. Niazmand, G. Jervan, and J. Raik, “AWAIT: An ultra-lightweight soft-error mitigation mechanism for Network-on-Chip links”, in *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, Jul. 2018, pp. 1–6. DOI: `10.1109/ReCoSoC.2018.8449374`.

[5] ——, “Handling of SETs on NoC links by exploitation of inherent redundancy in circular input buffers”, in *2018 16th Biennial Baltic Electronics Conference (BEC)*, Oct. 2018, pp. 1–4. DOI: `10.1109/BEC.2018.8600989`.

[6] *AWAIT*, `https://github.com/Project-Bonfire/AWAIT`, 2018.

[7] S. P. Azad, B. Niazmand, K. Janson, N. George, A. S. Oyeniran, T. Putkaradze, A. Kaur, J. Raik, G. Jervan, R. Ubar, and T. Hollstein, “From online fault detection to fault management in Network-on-Chips: A ground-up approach”, in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, Apr. 2017, pp. 48–53. DOI: `10.1109/DDECS.2017.7934565`.

[8] G. E. Moore *et al.*, *Cramming more components onto integrated circuits*, 1965.

[9] E. P. DeBenedictis, “It’s time to redefine moore’s law again”, *Computer*, vol. 50, no. 2, pp. 72–75, Feb. 2017, ISSN: 0018-9162. DOI: `10.1109/MC.2017.34`.

[10] M. C. McFarland, A. C. Parker, and R. Camposano, “The high-level synthesis of digital systems”, *Proceedings of the IEEE*, vol. 78, no. 2, pp. 301–318, Feb. 1990, ISSN: 0018-9219. DOI: `10.1109/5.52214`.

[11] Y. Sun, C. Zhan, J. Guo, Y. Fu, G. Li, and J. Xia, “Localized thermal effect of sub-16nm finfet technologies and its impact on circuit reliability designs and methodologies”, in *2015 IEEE International Reliability Physics Symposium*, Apr. 2015, pp. 3D.2.1–3D.2.6. DOI: `10.1109/IRPS.2015.7112712`.

[12] F. Griggio, J. Palmer, F. Pan, N. Toledo, A. Schmitz, I. Tsameret, R. Kasim, G. Leatherman, J. Hicks, A. Madhavan, J. Shin, J. Steigerwald, A. Yeoh, and C. Auth, “Reliability of dual-damascene local interconnects featuring cobalt on 10 nm logic technology”, in *2018 IEEE International Reliability Physics Symposium (IRPS)*, Mar. 2018, 6E.3-1-6E.3-5. DOI: `10.1109/IRPS.2018.8353641`.

[13] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. P. Pande, C. Grecu, and A. Ivanov, “System-on-Chip: Reuse and integration”, *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1050–1069, Jun. 2006, ISSN: 0018-9219. DOI: `10.1109/JPROC.2006.873611`.

[14] F. Tang, P. Zhang, F. Li, and Z. Huang, “Design of distributional bus in control system”, in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, May 2015, pp. 2102–2107. DOI: `10.1109/CCDC.2015.7162268`.

[15] L. Benini and G. D. Micheli, “Networks on chips: a new SoC paradigm”, *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002, ISSN: 0018-9162. DOI: `10.1109/2.976921`.

[16] L. Benini and D. Bertozzi, “Network-on-Chip architectures and design methods”, *Computers and Digital Techniques, IEE Proceedings* -, vol. 152, pp. 261–272, Apr. 2005. DOI: `10.1049/ip-cdt:20045100`.

[17] W. Dally and B. Towles, “Route packets, not wires: On-Chip interconnection networks”, Aug. 2004. DOI: `10.1145/378239.379048`.

[18] W. Dally, B. Towles, and B. Patrick, *Principles and Practices of Interconnection Networks*, ser. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2004, ISBN: 9780122007514.

[19] V. F. Pavlidis and E. G. Friedman, “3-d topologies for Networks-on-Chip”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 10, pp. 1081–1090, Oct. 2007, ISSN: 1063-8210. DOI: `10.1109/TVLSI.2007.893649`.

[20] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks*, ser. THE MORGAN KAUFMANN SERIES IN. Elsevier Science, 2003, ISBN: 9781558608528.

[21] J. Flich and J. Duato, “Logic-based distributed routing for NoCs”, *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 13–16, Jan. 2008, ISSN: 1556-6056. DOI: `10.1109/L-CA.2007.16`.

[22] S. P. Azad, B. Niazmand, K. Janson, T. Kogge, J. Raik, G. Jervan, and T. Hollstein, “Comprehensive performance and robustness analysis of 2D turn models for Network-on-Chips”, in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4. DOI: `10.1109/ISCAS.2017.8050634`.

[23] *Turn-Model*, `http://turnmodel.pld.ttu.ee/`, 2017.

[24] A. C. Bray, A. C. Dickens, and M. A. Holmes, *The Advanced User Guide for the BBC Microcomputer*. Cambridge Microcomputer Centre, 1983.

[25] L. Null and J. Lobur, *The Essentials of Computer Organization and Architecture*. Jones & Bartlett Learning, 2010, ISBN: 9781449600068.

[26] T. Bjerregaard and S. Mahadevan, “A survey of research and practices of Network-on-Chip”, *ACM Comput. Surv.*, vol. 38, Jun. 2006. DOI: `10.1145/1132952.1132953`.

[27] and and, “Low power circuits for NoC-based SoC design”, in *2008 9th International Conference on Solid-State and Integrated-Circuit Technology*, Oct. 2008, pp. 2180–2183. DOI: `10.1109/ICSICT.2008.4735001`.

[28] I. Koren and C. Krishna, *Fault-Tolerant Systems*. Elsevier Science, 2010, ISBN: 9780080492681.

[29] J. Podivinsky, J. Lojda, O. Cekan, R. Panek, and Z. Kotasek, “Reliability analysis and improvement of FPGA-based robot controller”, in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug. 2017, pp. 337–344. DOI: `10.1109/DSD.2017.15`.

[30] H. Al-Asaad, “A simplified overview of handling faults in systems around us”, in *2016 IEEE AUTOTESTCON*, Sep. 2016, pp. 1–9. DOI: `10.1109/AUTEST.2016.7589642`.

[31] C. Bolchini and C. Sandionigi, “Fault classification for SRAM-based FPGAs in the space environment for fault mitigation”, *IEEE Embedded Systems Letters*, vol. 2, no. 4, pp. 107–110, Dec. 2010, ISSN: 1943-0663. DOI: `10.1109/LES.2010.2073441`.

[32] C. Y. H. Lin, R. H. .-.-. Huang, C. H. .-.-. Wen, and A. C. .-.-. Chang, “Aging-aware statistical soft-error-rate analysis for nano-scaled CMOS designs”, in *2013 International Symposium onVLSI Design, Automation, and Test (VLSI-DAT)*, Apr. 2013, pp. 1–4. DOI: `10.1109/VLDI-DAT.2013.6533854`.

[33] O. Golubeva, “Detection of hard-to-detect stuck-at faults and generation of their tests based on testability functions”, in *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, May 2018, pp. 1–5. DOI: `10.1109/AQTR.2018.8402703`.

[34] S. Hwang and R. Rajsuman, “Effectiveness of stuck-at test sets to detect bridging faults in iddq environment”, in *Proceedings of 1993 IEEE 2nd Asian Test Symposium (ATS)*, Nov. 1993, pp. 249–254. DOI: `10.1109/ATS.1993.398813`.

[35] I. Pomeranz, "Lfsr-based test generation for path delay faults", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 345–353, Feb. 2019, ISSN: 0278-0070. DOI: 10.1109/TCAD.2018.2812120.

[36] D. Zhou, Y. Zhao, Z. Wang, X. He, and M. Gao, "Review on diagnosis techniques for intermittent faults in dynamic systems", *IEEE Transactions on Industrial Electronics*, pp. 1–1, 2019, ISSN: 0278-0046. DOI: 10.1109/TIE.2019.2907500.

[37] R. Hariharan, T. Ghasempouri, B. Niazmand, and J. Raik, "From rtl liveness assertions to cost-effective hardware checkers", in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, Nov. 2018, pp. 1–6. DOI: 10.1109/DCIS.2018.8681487.

[38] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri, "Dynamic error mitigation in NoCs using intelligent prediction techniques", in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12. DOI: 10.1109/MICRO.2016.7783734.

[39] S. Shamshiri, A. Ghofrani, and K. T. Cheng, "End-to-end error correction and online diagnosis for on-chip networks", in *2011 IEEE International Test Conference*, Sep. 2011, pp. 1–10. DOI: 10.1109/TEST.2011.6139156.

[40] M. Ali, M. Welzl, S. Hessler, S. Hellebrand, and S. And, "An efficient fault tolerant mechanism to deal with permanent and transient failures in a network on chip", *International Journal of High Performance Systems Architecture*, vol. 1, Jan. 2007.

[41] G. Schley, N. Batzolis, and M. Radetzki, "Fault localizing End-to-End flow control protocol for Networks-on-Chip", in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb. 2013, pp. 454–461. DOI: 10.1109/PDP.2013.74.

[42] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst, "Designing Networks-on-Chip for high assurance real-time systems", in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan. 2017, pp. 185–194. DOI: 10.1109/PRDC.2017.32.

[43] E. Wachter, V. Fochi, F. Barreto, A. Amory, and F. Moraes, "A hierarchical and distributed fault tolerant proposal for NoC-based MPSoCs", *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2017, ISSN: 2168-6750. DOI: 10.1109/TETC.2016.2593640.

[44] S. Ogg, B. Al-Hashimi, and A. Yakovlev, "Asynchronous transient resilient links for NoC", in *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Atlanta, GA, USA: ACM, 2008, pp. 209–214, ISBN: 978-1-60558-470-6.

[45] A. P. Frantz, F. L. Kastensmidt, L. Carro, and E. Cota, "Dependable Network-on-Chip router able to simultaneously tolerate soft errors and crosstalk", in *2006 IEEE International Test Conference*, 2006, pp. 1–9. DOI: 10.1109/TEST.2006.297635.

[46] A. P. Frantz, M. Cassel, F. L. Kastensmidt, É. Cota, and L. Carro, "Crosstalk- and SEU-aware Networks on Chips", *IEEE Design Test of Computers*, vol. 24, no. 4, pp. 340–350, Jul. 2007, ISSN: 0740-7475. DOI: 10.1109/MDT.2007.128.

[47] X. Chen, Z. Lu, Y. Lei, Y. Wang, and S. Chen, "Multi-bit transient fault control for NoC links using 2D fault coding method", in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, Aug. 2016, pp. 1–8. DOI: 10.1109/NOCS.2016.7579328.

[48] A. Dutta and N. A. Touba, "Reliable network-on-chip using a low cost unequal error protection code", in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sep. 2007, pp. 3–11. DOI: 10.1109/DFT.2007.20.

[49] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant Network-on-Chip architectures", in *International Conference on Dependable Systems and Networks (DSN'06)*, Jun. 2006, pp. 93–104. DOI: 10.1109/DSN.2006.35.

[50] S. R. Naqvi, V. S. Veeravalli, and A. Steininger, "Protecting an asynchronous NoC against transient channel faults", in *2012 15th Euromicro Conference on Digital System Design*, Sep. 2012, pp. 264–271. DOI: `10.1109/DSD.2012.108`.

[51] S. P. Azad, B. Niazmand, K. Janson, N. George, A. S. Oyeniran, T. Putkaradze, A. Kaur, J. Raik, G. Jervan, R. Ubar, and T. Hollstein, "From online fault detection to fault management in Network-on-Chips: A ground-up approach", in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, Apr. 2017, pp. 48–53. DOI: `10.1109/DDECS.2017.7934565`.

[52] *ModelSim, Mentor Graphics*, `https://www.mentor.com/products/fv/modelsim/`, 2017.

[53] (1994). Synopsys design compiler.

[54] G. H. Mealy, "A method for synthesizing sequential circuits", *The Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

Appendix 1 – Published AWAIT1

K. Janson, R. Pihlak, S. P. Azad, B. Niazmand, G. Jervan and J. Raik, “*AWAIT: An Ultra-Lightweight Soft-Error Mitigation Mechanism for Network-on-Chip Links*,” 2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Lille, 2018, pp. 1–6. doi: 10.1109/ReCoSoC.2018.8449374

AWAIT: An Ultra-Lightweight Soft-Error Mitigation Mechanism for Network-on-Chip Links

Karl Janson, René Pihlak, Siavoosh Payandeh Azad, Behrad Niazmand, Gert Jervan, Jaan Raik
Department of Computer Systems, Tallinn University of Technology,
email: {karl.janson, rene.pihlak, siavoosh.azad, behrad.niazmand, gert.jervan, jaan.raik}@ttu.ee

***Abstract*—Networks-on-Chip have become a widely accepted communication paradigm for many-core Systems-on-Chip. However, with the ever-shrinking transistor size, the network's sensitivity to transient faults on the physical links cannot be ignored since even a single transient fault can lead to a network-wide congestion and a system failure. This paper proposes the AWAIT mechanism, an ultra-lightweight transient fault mitigation mechanism for Network-on-Chip links. The proposed mechanism covers all single event transients. The experimental results show that the AWAIT mechanism prevents network-wide failure even in harsh environments (up to 80 million random faults on links per second). The AWAIT mechanism is also scalable and imposes only 5.1% area overhead with very negligible critical path delay overhead.**

***Index Terms*—Network-on-Chip, Fault tolerance, Transient fault mitigation**

I. Introduction

The ever-decreasing transistor feature size has enabled the integration of large number of cores on a single die. However, the traditional bus-based interconnection infrastructure is not scalable enough and becomes the bottleneck in inter-core communication. Network-on-Chip (NoC) has emerged as a solution to this problem. However, the decreasing feature size also makes the system much more susceptible to both environmental faults (*e.g.*, faults caused by radiation) and also introduces new problems, such as Negative-Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) [1]. The aforementioned problems add the need for integrated reliability mechanisms. In this paper we concentrate on improving the reliability of the inter-router links.

This paper proposes AWAIT, a scalable mechanism which is able to handle an extensive amount of transient faults (see Section VI) by relying on a single parity bit for fault detection, together with only a few additional gates per link for transient fault mitigation. This approach enables the detection and correction of all single bit link faults while requiring eight times less data overhead per flit and 10 times less area for 32-bit links than Hamming code, which only provides single bit correction [2].

The AWAIT mechanism pauses system operation in the affected part of the system until the faults disappear and does not require retransmission of data. Applying this method would result in only a slight increase in latency in the presence of faults and in case of transient faults will always guarantee correct arrival of the data.

978-1-5386-3344-1/17/$31.00 ©2018 IEEE

This paper focuses only on Single Event Transient (SET) faults in NoC links (due to the use of a parity checker for fault detection). Moreover, it is important to note that the AWAIT mechanism is not limited to the usage of the parity checker but can be also combined with other fault detection (or correction) mechanisms, if needed. The behavior in the presence of permanent faults is out of the scope of this paper.

The rest of this paper is organized as follows. Section II provides a literature review. Sections III and IV provide details of the baseline router and an analysis of the NoC's behavior in different operation environments. Section V will provide details on the proposed AWAIT mechanism and Section VI will provide experimental results which illustrate the efficiency of the proposed mechanism. Finally, section VII will conclude the paper.

II. Literature Review

Many works have investigated the problem of transient fault mitigation for physical links in NoCs. However, most of the mechanisms proposed in the literature are very costly in terms of latency, area or data overhead. In general, all fault tolerance mechanisms used for NoC links can be categorized as End-to-End (E2E) or Hop-by-Hop (HBH) mechanisms, based on the fault detection and mitigation granularity. In HBH mechanisms, the packet/flit is examined every time the data is transferred from one router to another. However, in E2E mechanisms, a fault is detected only once the packet is ejected from the network. HBH mechanisms have usually lower latency than the E2E based mechanisms, since in E2E mechanisms the entire packet needs to be retransmitted in case a fault, which cannot be corrected in the endpoint, is detected.

In order to mitigate transient faults in physical links in a NoC, three main approaches exist in the literature:

A. Data Retransmission

1) Packet Retransmission (P-RET): Works such as [2], [6], [3], [4] and [5] have proposed methods based on End-to-End (E2E) packet retransmission. This approach suffers from high latency caused by the retransmission; once a faulty packet is detected, often by an Error Detecting Code (EDC), a NACK packet should be sent to the sender of the faulty packet for requesting a retransmission. During this process, the receiving node might need to also discard the non-faulty packets in order to preserve the packet order. It is important to note that such E2E packet retransmission mechanisms require

TABLE I: Comparison of related works

Approach	Used Method	HBH/E2E	No Extra Buffers	Targeted Fault Model	Correction Latency
[2]	P-RET	E2E	✓	SETs + 50% of METs	$\gg$ 1 clk
[3]	P-RET	E2E	✓	SETs	$\gg$ 1 clk
[4]	P-RET	E2E	✓	SETs + METs	$\gg$ 1 clk
[5]	P-RET	E2E	✗	SETs + METs	$\gg$ 1 clk
[6]	P-RET	E2E	✗	SETs	$\gg$ 1 clk
[7]	ECC	E2E	✓	SETs	N.A.
[8]	ECC	HBH	✗	SETs	N.A.
[9]	ECC	HBH	✗	SETs	N.A.
[10]	F-RET	HBH	✓	SETs	N.A.
[11]	F-RET	HBH	✗	SETs + METs	3 clks
[12]	RT + ECC + EDC	HBH	✗	SETs + METs (Prediction)	2 clks
[13]	P-RET	HBH	✗	SETs + Some DET	N.A.
[14]	ECC	HBH	✗	SETs + METs	N.A.
[15]	ECC	HBH	✗	SETs	$\gg$ 1 clk
Proposed	**RT**	**HBH**	✓	**SETs + 50% of METs**	**0-1 clk**

distributed packet dropping mechanisms in order to prevent a network-wide failure.

On the other hand, [13] has proposed HBH packet retransmission using store-and-forward switching. However, store-and-forward requires large input buffers since each router has to buffer the entire packet before forwarding it, which is very inefficient use of network buffers and chip area.

2) Flit retransmission (F-RET): In contrast to packet retransmission, works such as [11] and [10] have proposed HBH flit retransmission in case of receiving a faulty flit. These works provide much lower latency compared to E2E approaches but suffer from the need for additional retransmission buffers in the router.

B. Error Correction Code (ECC)

Another class of fault tolerance mechanisms for NoC links is Error Correction Codes (ECCs). Works such as [8], [9] and [15] have provided solutions for correcting single transient upsets, while [14] has used more sophisticated error correction techniques to cover multiple event transients. Similar to retransmission mechanisms, ECCs are also implemented either as HBH [8], [9], [14] and [15] or as E2E [7]. However, the main problem with those approaches is the large area overhead of such mechanisms while providing only limited fault coverage.

C. Relaxed Transmission (RT)

In contrast to above mentioned approaches which try to correct the fault, Relaxed Transmission (RT) uses the transient nature of the SETs and METs in order to mitigate them. The main idea of this approach is to inform one party in the transmission to wait until the faults disappear. A relaxed transmission mechanism has been proposed in [12], where the upstream router predicts the faults and informs the downstream router to delay data sampling. Even though the RT method is very efficient in mitigating faults, prediction of transient faults which have a random nature, will not be effective. Hence, in [12] also other methods have been used alongside RT.

This paper proposes an ultra-lightweight, HBH, relaxed transmission based transient fault mitigation approach called AWAIT which does not require any additional buffers. The proposed mechanism can handle transient faults immediately by pausing the operation of the faulty components without data loss and guarantees returning to normal, fault free operation in zero to one clock cycle after the fault disappears. This is achieved by utilizing a parity checker as the fault detection mechanism which allows to detect and mitigate all SETs. However, the proposed AWAIT mechanism is not limited to using parity checker but can be used together with any other fault detection mechanism. Since packet retransmission is not required, the AWAIT mechanism is also power efficient.

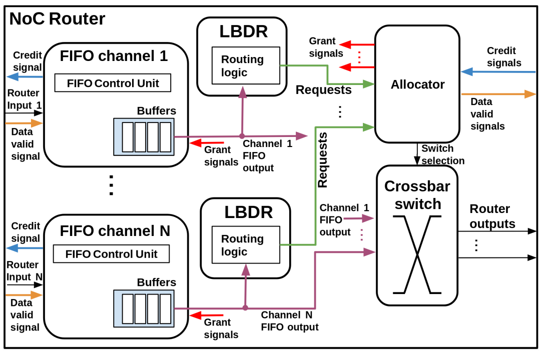


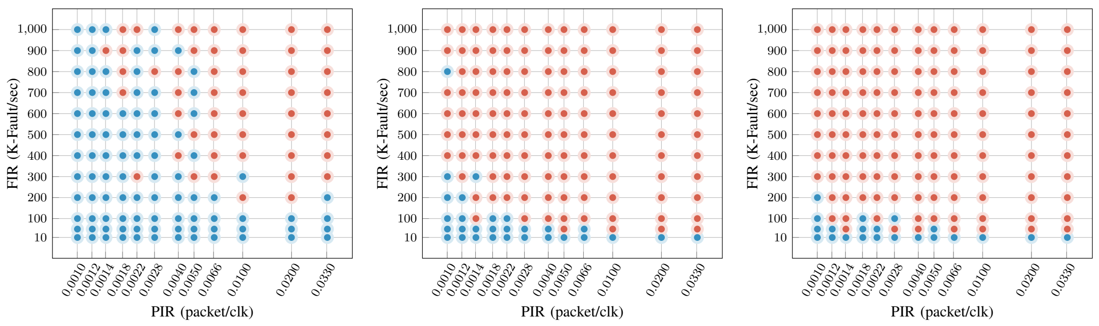
Fig. 1: Block diagram of the baseline Bonfire router

A side-by-side comparison of different fault tolerance mechanisms for physical links in NoCs proposed in the literature can be seen in Table I.

III. Bonfire Router

In order to prove the applicability of the AWAIT mechanism in a real design, it was implemented for inter-router links in a 4×4 NoC utilizing the Bonfire router [2]. An overview of the baseline Bonfire NoC router (without any fault tolerance mechanisms) can be seen in Fig. 1. The Bonfire router utilizes wormhole switching and 32-bit flit size.

The Bonfire router uses a credit-based flow control mechanism, where the transmitter includes a credit counter to keep track of the free slots in the receiver's input buffer. When



(a) Fault duration: 10% of clock period (b) Fault duration: 100% of clock period (c) Fault duration: 200% of clock period

Fig. 2: Effect of faults on a 4×4 2D-mesh baseline network during 100,000 clock cycles

initialized, the credit counter is set to equal to the number of slots in the receiver's input buffers. The value in the credit counter is decremented each time a flit is sent (and an additional slot in the receiver's input buffer gets occupied). When a slot frees up in the receiver's input buffer, the receiver will issue a credit signal to the transmitter, which causes the credit counter in the transmitter to increase. When credit counter reaches zero, the receiver's input buffer is full, and the transmitter will stop transmitting. If the value in the credit counter is larger than zero, data can be transmitted one flit per clock cycle.

Each input port of the Bonfire router consists of an input buffer, implemented as a First-In-First-Out (FIFO) and a routing computation unit which is implemented as Logic-Based Distributed Routing (LBDR) [16]. The main advantage of using LBDR compared to routing tables is its scalability. Additionally, since LBDR describes the network topology and routing algorithm using a fixed number of connectivity and routing bits, it is possible to easily mark the links as healthy or faulty, by disabling routing to faulty links. It is also easy to change the routing algorithm using the routing bits.

The output ports of the router are allocated by the allocator unit based on the routing decisions from LBDR. The allocator unit uses the Round-Robin policy for prioritizing multiple requests to the same output port. Finally, data is transferred from the input port to the output port using a crossbar switch.

IV. Network Failure Analysis

On-chip networks are very sensitive to faults on the network's physical links. A single faulty value stored in a register may lead to a network wide failure. We define a *network failure* as the situation were the entire network or part(s) of the network is congested (and the system cannot recover from it) due to inability to route a packet. Such a faulty value stored in a register can be one of the following:

- **Fault in the flit type:** might lead to a network-wide congestion
- **Fault in the destination address:** might lead to a network-wide congestion
- **Fault in other header information:** a fault in source address, packet length, etc. may cause problem at the application layer but has no effect on the network behavior, thus the effect of these faults can be ignored for the network.
- **Fault in the payload:** has no-effect on the network behavior. It might cause a problem at the application layer. The effect of these faults can be ignored for the network.

In this work, we have performed fault injection experiments to evaluate the effects of these faults on the network's behavior. The faults were randomly injected into the network links by temporarily forcing signals in the simulation to faulty values at fault rates ranging from ten thousand to one million faults per second. In order to see the effects of SETs with different lengths in the network, three different fault length scenarios were investigated where the fault length varied from 10% of the clock period to 100% and 200% of clock period (the fault effectively stayed on the link for two clock cycles). For this investigation, in total 1500 experiments were performed using random traffic pattern with different Packet Injection Rates (PIR) and Fault Injection Rates (FIR).

Fig. 2 depicts behavior of a 4×4, 2D-mesh network with wormhole switching using 4-flit FIFO buffers, without any fault tolerance mechanisms under different packet and fault injection rates (during a period of 100 thousand clock cycles). The fault injection rate is between ten thousand to one million faults per second. The red dots mark a network failure while blue dots depict the cases where the network successfully transmitted all of the injected packets. Each data point in this plot is based on 5 experiments with the same fault rate and packet injection rate but with different seeds for the random traffic generators. A network failure is declared only if at least 3 out of 5 experiments for that data point failed. As expected, with increasing packet injection rate and fault injection rate, the probability of a fault hitting a sensitive part of the packet increases. The experiments also show that the duration of fault's presence in the network has a great effect on the network behavior. This is due to the fact that faults

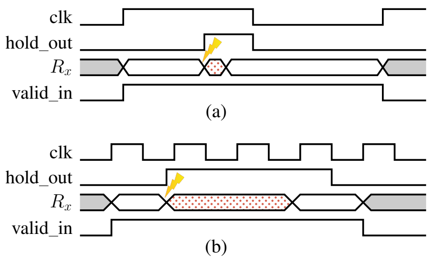


Fig. 3: Status of the signals during a) short (10% clk period) and b) long (200% clk period) transient fault

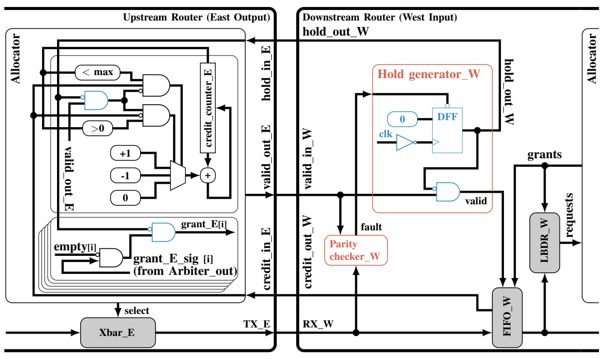


Fig. 4: Schematic of the AWAIT mechanism

with longer presence have higher chance of being registered in the FIFO buffers.

These experiments highlight the network's sensitivity to transient faults and the need for a scalable mitigation mechanism.

V. AWAIT Mechanism

In this paper AWAIT, a new light-weight fault mitigation mechanism is proposed to address transient faults on network links. The AWAIT mechanism operates on hop-by-hop (HBH) basis which enables fault correction with considerably lower latency when compared to end-to-end (E2E) packet retransmission. Also, unlike approaches like HBH flit/packet re-transmission, the proposed mechanism does not require additional buffers.

The AWAIT fault mitigation approach for network links is based on the transient nature of the SETs. The upstream router holds correct data when a transient fault occurs on the router link. Once the transient fault disappears, the data on the link comes back to its normal value and can be read by the downstream router's FIFO. In order to wait out the transient fault, three steps are required:

1) The downstream router should detect the presence of fault(s).
2) The downstream router should stop sampling data in case of a fault.
3) The upstream router must keep the transmitted data on the link until it is certain that the downstream router has received the correct, fault-free data.

In order to provide step one, a simple fault detection system – a parity checker – is used. Parity checker is especially useful since it can detect all single faults.

In order to support step two, the process of storing the data into downstream router's FIFO is stopped if a fault is detected. This is achieved by replacing the *valid_in* signal of the FIFO module with a new signal: $\texttt{valid} = (\texttt{valid\_in} \bigwedge \overline{\texttt{fault}})$.

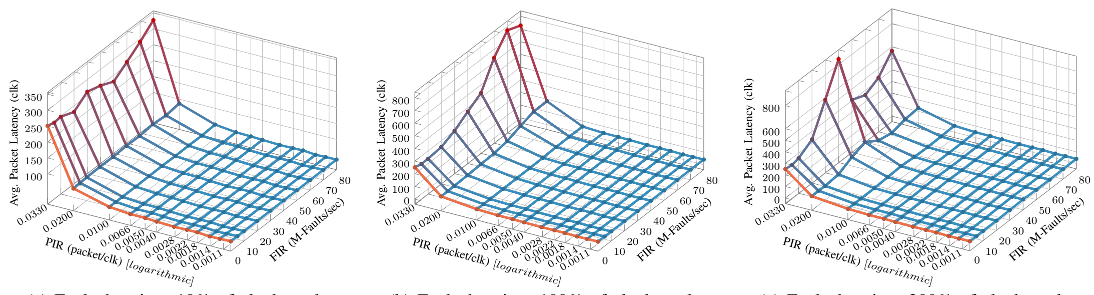
Finally, to support step three, a hold signal is propagated from the downstream router to the upstream router, informing it about the presence of a fault on the link. Upon receiving this signal, the upstream router's allocator unit halts transmission and maintains the value on the link. The AWAIT mechanism does not require any additional buffers in the system and is entirely governed via controlling of the grant signals. The *hold* signal can only be cleared once the fault has disappeared. It will be done on the next falling clock edge after the disappearance of the fault, enabling the system to register the correct data on the next rising clock edge.

Fig. 3 depicts the behavior of the mechanism for a short (10% of the clock period) and a long (200% of the clock period) transient fault, respectively. Rx signal in the figure represents the data receiving line of the router. The duration while the Rx signal had a faulty value is colored red. The *valid_in* signal describes the valid signal received from the upstream router (showing upstream router maintaining the value on the link). It is important to note that if the fault disappears before the falling edge of the system clock, the effects of the fault will be mitigated in the same clock cycle. However, if the fault disappears after the falling edge of the clock, the correct data will be latched one clock cycle later during the next falling clock edge.

Fig. 4 shows the block diagram of the output port of an upstream router, the communication link, and the input port of a downstream router along with the added circuitry. For simplicity, Fig. 4 shows only the East output of the upstream router, connected to the West input of the downstream router. Packets are being transmitted from the upstream router to the downstream router.

The parity checker in the downstream router's input checks the parity of the received data. The output of the parity checker is used to trigger the hold signal in the downstream router. When a fault is detected, the *hold_out* signal is instantly set to *"1"*. Using a flip-flop ensures that the *hold_out* signal stays at the *"1"* value, using asynchronous reset of the D flip-flop, until the fault has disappeared. Once the fault has disappeared, at the next falling edge of the system's clock, the value of *hold_out* signal is set to *"0"*.

Additional components are added to the mechanism in order to control the credit counters and the grant generation in the allocator unit. Upon receiving the hold signal, allocator unit suppresses the grants issued to the FIFO unit which is



(a) Fault duration: 10% of clock cycle (b) Fault duration: 100% of clock cycle (c) Fault duration: 200% of clock cycle

Fig. 5: Average packet latency of 4×4 2D-mesh network under different PIR and FIR on the links

accessing the output channel. This means that FIFO will not perform any read operations and the latest data will be kept on its output. Allocator unit will also keep the select signal of the Xbar intact, ensuring that the data is kept unchanged on the link and it will not update the value of the credit counter as long as the hold signal is present. The additional hardware for supporting these mechanisms is minimal and does not exceed a few gates. Next section will provide experimental results for illustrating the AWAIT mechanism's efficacy under different fault injection campaigns.

VI. Experimental Results

This section details the experiments for validating the effectiveness of the proposed AWAIT mechanism and providing support for its efficacy in mitigating SETs. Later in this section the overheads of the AWAIT mechanism are investigated in terms of area, power and critical path delay.

The experiments were carried out on a 4×4, 2D-mesh NoC with wormhole switching using 4 flit deep FIFO buffers. Each router in the network is a Bonfire credit-based router equipped with the AWAIT mechanism for handling transient faults. For measuring the latency of the AWAIT mechanism, 1500 experiments were performed using random uniform traffic pattern with different Packet Injection Rates (PIRs) and Fault Injection Rates (FIRs) – up to 80 Million random faults per second over the network's physical links. The experiments shows no failing scenario and validates the effectiveness of the AWAIT approach. In other words, despite of the existence of transient faults on the router's link at run-time, the number of received packets over the network match the number of sent packets. Fig. 5 depicts the behavior of the fault-tolerant NoC equipped with the AWAIT mechanism under different fault campaigns on network links. Similar to experiment on the baseline router (see Fig. 2), three different fault duration scenarios were used. Each data point in this plot is the average latency of 5 experiments, all of which use the same FIR and PIR, but have different seeds for the random packet generators. The experiments illustrate that the network is still operational under extreme fault conditions (fault injection rate from 5-Million to 80-Million faults per second) with acceptable latency overhead. The experiments show that while the network is not saturated, the length of the faults does not have much effect on the additional network latency from fault handling. However, once the network exceeds saturation point, the additional latency becomes more visible. The orange line in Fig. 5 marks the reference, fault-free experiment.

TABLE II: Area overhead of the AWAIT mechanism

	Area (μm^2)	Area Overhead
Baseline router	8289.38	–
AWAIT router	8719.56	5.1 %

Table II shows the area overhead imposed to the baseline router by the AWAIT mechanism. Both designs are implemented in Register Transfer Level (RTL) in VHDL and are synthesized using TSMC 40 nm CMOS technology standard cell library in Synopsys Design Compiler. It is important to note that the fault detection mechanism (parity checker) imposes the majority of the reported combinational area overhead (3.82%) due to the additional XOR gates required to perform the fault detection. It should be noted, however, that this area overhead is still very small when compared to the overhead implied by an error correction code such as Hamming [2]. The rest of the mechanism, including the additional gates imposed by the proposed mechanism, incurs only 1.3% overhead to the router's area, which makes the solution scalable.

Fig. 6 shows the comparison of the dynamic and static power consumption of a 4×4 network augmented with the AWAIT mechanism against the baseline router under different PIRs. In order to obtain the power results for each PIR, the design architecture was simulated in ModelSim. Then, the simulation traces were collected and the switching activity of the components and signals were stored. Further, the annotated switching activities are fed into the synthesis tool to calculate the power results. The AWAIT mechanism has only a negligible total power consumption overhead (around 2.89% on average), compared to the baseline router. This is mostly caused by increase in the dynamic power consumption.

Table III shows the critical path delay overhead of the AWAIT mechanism compared to the baseline router. The proposed mechanism does not impose overhead to the router's critical path delay. Due to the critical path delay of the AWAIT mechanism the faults that occur very close to the clock edge

TABLE III: Critical path delay overhead of the AWAIT mechanism

	Critical-Path Delay (ns)	Overhead
Baseline router	1.96	–
AWAIT router	1.94	$\approx 0\%$

will be latched into the FIFO buffers. However, in systems with frequency of 250 MHz and below these effects are negligible. Additionally, it should be noted that since the area and timing overhead of the AWAIT mechanism is very small, it can be coupled with other fault mitigation mechanisms which can handle faults already latched into the buffer (such as a packet dropping mechanism [2]) to solve the issue. The source code for both the baseline and the AWAIT router design along with all the experimental result scripts are maintained as an open-source project [17].

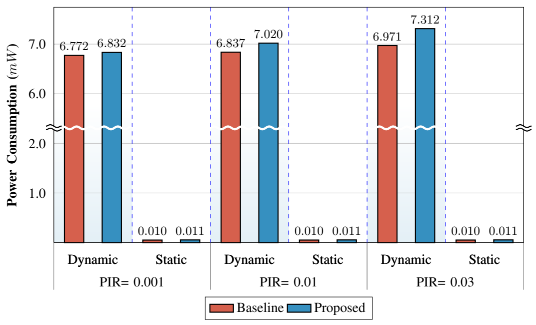


Fig. 6: Comparison of dynamic and static power consumption (in mW) of the proposed and baseline 4×4 network under different packet injection rates

VII. Conclusion

Network-on-Chip has become a widely accepted communication paradigm, replacing the bus-based communication mediums in modern multi/many-core System-on-Chips. However, the shrinking transistor's feature size and increase in the number of components integrated on a single chip, makes the systems growingly more susceptible to transient faults. On-chip networks are particularly sensitive to transient faults on the physical links between the nodes. This paper presented the ultra-lightweight AWAIT mechanism which can tolerate all single event transients on network links. The proposed AWAIT mechanism adds only 5.1% overhead to the area and only a negligible overhead to the critical path delay of the routers, which makes it a scalable solution, as it adds transient fault tolerating capability to the design by augmenting it with only a few gates. Experimental results show the effectiveness of the AWAIT mechanism even in extreme environments. As future work, the authors plan to extend this work to cover permanent faults on the router links as well. Moreover, the mechanism would be extended to cover faults occurring in the control part of the router.

Acknowledgment

The work has been supported by EU's Twinning Action TUTORIAL, Estonian Institutional research grant IUT 19-1, Estonian center of excellence in IT EXCITE, and Estonian IT Academy programme.

References

[1] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging analysis of circuit timing considering NBTI and HCI," in *2009 15th IEEE International On-Line Testing Symposium*, June 2009, pp. 3–8.

[2] S. P. Azad, B. Niazmand, K. Janson, N. George, A. S. Oyeniran, T. Putkaradze, A. Kaur, J. Raik, G. Jervan, R. Ubar, and T. Hollstein, "From online fault detection to fault management in Network-on-Chips: A ground-up approach," in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2017, pp. 48–53.

[3] G. Schley, N. Batzolis, and M. Radetzki, "Fault localizing End-to-End flow control protocol for Networks-on-Chip," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb 2013, pp. 454–461.

[4] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst, "Designing Networks-on-Chip for high assurance real-time systems," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan 2017, pp. 185–194.

[5] E. Wachter, V. Fochi, F. Barreto, A. Amory, and F. Moraes, "A hierarchical and distributed fault tolerant proposal for NoC-based MPSoCs," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2017.

[6] M. Ali, M. Welzl, S. Hessler, S. Hellebrand, and S. And, "An efficient fault tolerant mechanism to deal with permanent and transient failures in a network on chip," *International Journal of High Performance Systems Architecture*, vol. 1, 01 2007.

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

[8] S. Ogg, B. Al-Hashimi, and A. Yakovlev, "Asynchronous transient resilient links for NoC," in *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA: ACM, 2008, pp. 209–214.

[9] A. P. Frantz, F. L. Kastensmidt, L. Carro, and E. Cota, "Dependable Network-on-Chip router able to simultaneously tolerate soft errors and crosstalk," in *2006 IEEE International Test Conference*, 2006, pp. 1–9.

[10] S. R. Naqvi, V. S. Veeravalli, and A. Steininger, "Protecting an asynchronous NoC against transient channel faults," in *2012 15th Euromicro Conference on Digital System Design*, Sept 2012, pp. 264–271.

[11] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant Network-on-Chip architectures," in *International Conference on Dependable Systems and Networks (DSN'06)*, June 2006, pp. 93–104.

[12] D. DiTomaso, T. Boraten, A. Kodi, and A. Louri, "Dynamic error mitigation in NoCs using intelligent prediction techniques," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[13] A. Dutta and N. A. Touba, "Reliable network-on-chip using a low cost unequal error protection code," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sept 2007, pp. 3–11.

[14] X. Chen, Z. Lu, Y. Lei, Y. Wang, and S. Chen, "Multi-bit transient fault control for NoC links using 2D fault coding method," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, Aug 2016, pp. 1–8.

[15] A. P. Frantz, M. Cassel, F. L. Kastensmidt, É. Cota, and L. Carro, "Crosstalk- and SEU-aware Networks on Chips," *IEEE Design Test of Computers*, vol. 24, no. 4, pp. 340–350, July 2007.

[16] J. Flich and J. Duato, "Logic-based distributed routing for NoCs," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 13–16, Jan 2008.

[17] "AWAIT," https://github.com/Project-Bonfire/AWAIT, 2018.

Appendix 2 – Published AWAIT2

K. Janson, R. Pihlak, S. P. Azad, B. Niazmand, G. Jervan and J. Raik, "*Handling of SETs on NoC Links by Exploitation of Inherent Redundancy in Circular Input Buffers*," 2018 16th Biennial Baltic Electronics Conference (BEC), Tallinn, 2018, pp. 1–4. doi: 10.1109/BEC.2018.8600989

Handling of SETs on NoC Links by Exploitation of Inherent Redundancy in Circular Input Buffers

Karl Janson, René Pihlak, Siavoosh Payandeh Azad, Behrad Niazmand, Gert Jervan, Jaan Raik
Department of Computer Systems, Tallinn University of Technology,
email: {karl.janson, rene.pihlak, siavoosh.azad, behrad.niazmand, gert.jervan, jaan.raik}@ttu.ee

***Abstract*—The miniaturization of nanometer technologies beyond the sub-micron domain has jeopardized the reliability of on-chip network links, making them more susceptible to Single Event Transients (SETs) during system's run-time. Using retransmission approaches has been proposed in the literature for handling SETs on Network-on-Chip (NoC) links. However, those approaches either suffer from significant latency overhead or impose additional retransmission buffers, which consume more area. This paper proposes the ReUSE mechanism as a transient fault mitigation mechanism for Network-on-Chip links. The mechanism takes advantage of the inherent redundancy in the input buffers of NoC routers and reuses these for SET mitigation on the NoC links. By using a parity checker for fault detection, the approach can cover all SETs during run-time and return to normal operation in maximum one clock cycle after the disappearance of the fault. The experimental results show that the proposed mechanism prevents network-wide failure even in harsh environments with up to 80 million random faults on links per second. The ReUSE mechanism imposes 18% area overhead and 7.8% critical path delay overhead to the baseline NoC router.**

***Index Terms*—Network-on-Chip, Fault tolerance, Transient fault mitigation**

I. Introduction

The ongoing trend of miniaturization of the semiconductor technology beyond the sub-micron domain has made digital circuits more susceptible to faults, including Single Event Transients (SETs) [1]. As SETs are also caused by evironental factors, such as radiation, systems operating in harsh environments with high radiation levels, such as space applications, are especially at risk of SETs. Such faults occurring during run-time can affect the performance of embedded systems, including Network-on-Chip (NoC)-based System-on-Chips (SoCs). A faulty link in a NoC can corrupt the data transmitted between processing elements over the network. Thus, it is important to handle such SETs on inter-router links at run-time.

This paper focuses on management of SETs on NoC links. This is done by implementing a very low latency hop-by-hop (HBH) retransmission mechanism. The proposed ReUSE mechanism does not use additional registers and can cover 100% of SETs on inter-router links by using the inherent redundancy provided by the unusable memory slot found in the circular buffers [2] used in the NoC router input ports. The detection of SETs at each router input is performed online via a single bit parity checker. Experimental results show the effectiveness of the proposed mechanism in terms of area overhead, critical path delay overhead and power consumption.

978-1-5386-7312-6/18/$31.00 ©2018 IEEE

The rest of this paper is organized as follows: Section II provides information about related works, Section III provides details on the proposed ReUSE mechanism and Section IV will provide results of the experiments which help to illustrate the proposed mechanism's efficiency. Finally, section V will conclude the paper.

II. Related Work

The issue of Single Event Transients (SETs) on the NoC links has been addressed in several papers. Depending where and when SET detection and/or correction takes place, the approaches can be divided into End-to-End (E2E) and Hop-by-Hop (HBH) mechanisms.

In case of E2E approaches, such as [3], [4] and [5], fault detection and/or correction takes place at the final destination, while HBH approaches such as [6], [7] and [8] tend to mitigate faults in each router by avoiding their propagation. The E2E approaches, however, are unable to handle errors in the packet header before the packet has reached the destination, meaning, they cannot guarantee the arrival of the packet to the final destination where fault detection/correction takes place and, moreover, can cause network congestions due to mis-routing. Therefore, the proposed approach deploys an HBH approach.

In addition to that, the SET fault tolerance mechanisms can be further divided into three groups based on the type of mechanism used: *1)* Error correction codes (ECC); *2)* Relaxed transmission (RT); *3)* Packet/flit retransmission (R/F-RET).

Error correction code based approaches such as [9], [6] and [10] usually imply a large area and data overhead.

Relaxed transmission based approaches such as AWAIT proposed in [11] detect a fault on the link by using a fault detection mechanism, such as a parity checker. When a fault is detected the system is paused until the SET disappears. However this approach cannot handle faults which happen near the raising clock edge, since fault detectors (such as parity) will not provide the result instantaneously due to signal propagation delays in the gates. If the time between the SET and the next rising clock edge is shorter than the fault detection latency, the faulty value will be latched into the router's input buffer. With the use of high clock rates in modern systems the probability of a fault being latched into the buffer raises noticeably, thus limiting the maximum safe speed of systems using the RT approach.

Retransmission-based approaches use fault detection mechanisms and trigger retransmission of the data in case a fault is detected.

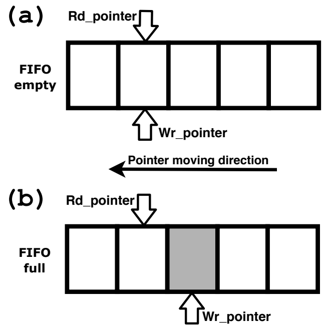


Fig. 1: A FIFO which is implemented as a circular buffer (a) empty, (b) full

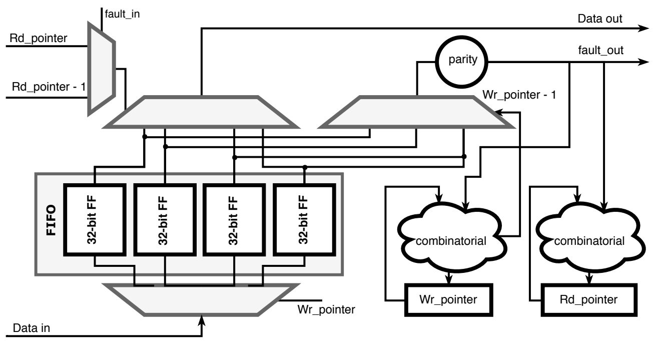


Fig. 2: Schematic of FIFO implementation in ReUSE

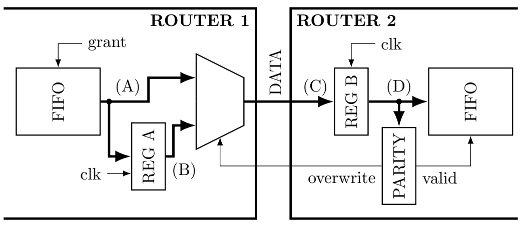


Fig. 3: Schematic of the dual register version of the ReUSE mechanism

Retransmission approaches can be categorized as packet retransmission (P-RET) or flit retransmission (F-RET). HBH P-RET approaches, such as [12] suffer from large area and slow speed, since routers need large buffers to store the entire packet. Additionally, in case of a fault, the entire packet needs to be retransmitted. In case of F-RET approaches, like [13] and [14], only one flit is retransmitted in case of the fault, thus requiring much smaller buffer and completing retransmission much faster. However, usually those approaches still have a relatively large overhead, since they require additional registers for retransmission.

In addition, the approach proposed in [15] has some similarities with the proposed mechanism. It uses Razor flip-flops [16], both, to perform online error detection and to restore the correct value of a signal after detection. The goal of the approach is to tolerate SETs on NoC inter-router links. However, a delayed clock is used in order to obtain the delayed sample of the signal.

The ReUSE approach proposed in this paper is able to detect 100% of SETs by utilizing a parity checker. However, as it is based on HBH flit retransmission, it can, unlike [11], handle all detected faults while guaranteeing returning of normal, fault free, system operation in at most one clock cycle after disappearance of the fault. Moreover, the ReUSE mechanism does not use any additional registers, but relies on inherent redundancy already found in FIFO input buffers of NoC routers, thus not requiring an increase in the area.

III. ReUSE Mechanism

A. General Concept

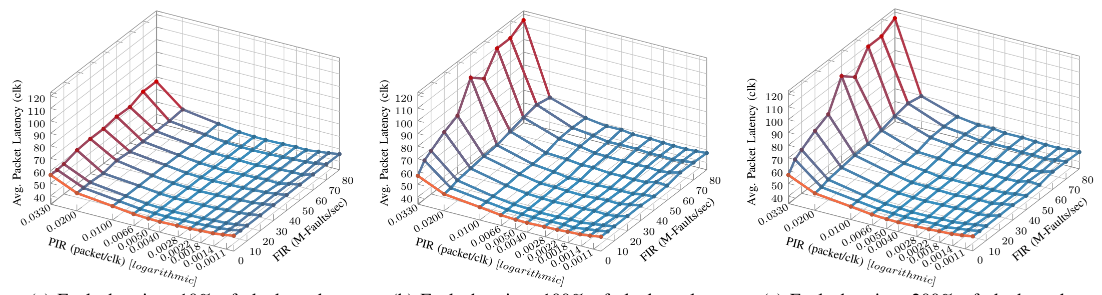
ReUSE is a HBH flit retransmission mechanism for handling SETs on the NoC links. A simplified concept of the ReUSE mechanism can be seen in Fig. 3 (data is transmitted from *Router 1* to *Router 2*). For simplification, signals used for flow control, and also router components such as the routing and arbitration modules are not shown in the figure.

On the transmitter's (*Router 1*) side, at each clock cycle, a flit is transmitted to *Router 2* (receiver) (A) and also stored into register *REG A*. Thus, the value on the *DATA* line depends on the multiplexor's (MUX) select line. If (A) input is selected, *DATA* will have the current flit read from the FIFO on it, however, if the (B) input is selected, *DATA* will be set equal to the flit which was read from the FIFO during the previous clock cycle and stored into *REG A*. By combining this with a mechanism for pausing the operation of the transmitting router, it is possible to keep the already sent flit on the DATA line until a fault disappears. While the pausing mechanism is not specified in this paper, an example of such a mechanism can be seen in [11].

On the receiver side (*Router 2*) every time a flit is received over link (C), it is stored into a register *REG B*. The value in the register is checked by a parity checker. If the parity checker detects a fault, it will cause the sending router to be paused and the MUX to be switched to input (B) using the "overwrite" line, thus holding the current flit on the *DATA* line. The *DATA* line is sampled to *REG B* every clock cycle and re-checked by the parity checker. Once the fault has disappeared, the operation of *Router 1* will be resumed and the MUX will be switched back to input (A), and writing to receiving router's FIFO is enabled by a signal from the parity checker. Since the data in register *REG B* is updated each clock cycle, the mechanism is guaranteed to return to normal operation on the next rising edge of the clock after the fault has disappeared.

B. ReUSE Concept

Since *REG B* in Fig. 3 can be thought of as an extension to *Router 2*'s FIFO, the design can be optimized by using



(a) Fault duration: 10% of clock cycle (b) Fault duration: 100% of clock cycle (c) Fault duration: 200% of clock cycle

Fig. 4: Average packet latency of a 4×4 2D-mesh network under different PIR and FIR on the links

data written to the FIFO one clock cycle earlier as input to the parity checker, thus removing the need for ***REG B*** as a separate register. This can be easily implemented, assuming a circular buffer implementation of the FIFO, as shown in Fig. 1. The buffer is accessed in such a FIFO implementation by using two pointers; one for reading ($Rd\_pointer$), other for writing ($Wr\_pointer$). If the buffer is empty, the read and write pointers will both point at the same memory slot, as seen in Fig. 1 (a). After each writing operation, the write pointer is shifted by one slot to the left. It can be seen that ***REG B*** can be implemented in such a way that parity checker is connected to the FIFO slot referred to by $Wr\_pointer - 1$.

The data is read from the memory slot referred by the read pointer. However, as it can be seen in Fig. 1 (b), the condition for the FIFO being full is when the read buffer is ahead of the write buffer by one memory slot ($Wr\_pointer = Rd\_pointer - 1$). Since writing would cause the write pointer to move to the same memory slot where the read pointer is (creating the "empty" configuration), the memory slot $Rd\_pointer - 1$ can never be overwritten in such an implementation. This definition helps to further optimize the mechanism, since ***REG A***, as mentioned in Subsection III-A, stores the flit read from the FIFO during the previous clock cycle. This means that the flit in ***REG A*** is the same as the one stored in FIFO memory slot $Rd\_pointer - 1$.

The ReUSE mechanism takes advantage of the aforementioned optimizations in order to provide fault tolerance for NoC links. The implementation of the FIFO in ReUSE can be seen in Fig. 2, data is normally read from the FIFO slot referred by $Rd\_pointer$ and written into the slot referred by $Wr\_pointer$. However, ***REG B*** has been implemented by connecting the parity checker to an additional MUX, which uses the previous write pointer as select line. If a fault is detected by the parity checker, the fault output is activated. This will cause the sending router's operation to be paused until the fault has disappeared, as discussed earlier, and reading of the memory slot referred by $Rd\_pointer - 1$, thus also removing the need for ***REG A***.

IV. Experimental Results

This sections presents results of the experiments used to validate the proposed ReUSE mechanism. The experimental

TABLE I: Area overhead of the ReUSE mechanism

	Area (μm^2)	Overhead
Baseline router	8276.45	–
ReUSE router	9777.26	18%

TABLE II: Critical path delay overhead of the ReUSE mechanism

	Critical path Delay (ns)	Overhead
Baseline router	2.28	–
ReUSE router	2.46	7.8%

results will also demonstrate the SET mitigation efficiency of the proposed mechanism. Additionally, in this section the overhead of the proposed ReUSE mechanism will be investigated in terms chip area, power and critical path delay.

For carrying out the experiments, the ReUSE mechanism was implemented for the links of a 4×4, 2D-mesh NoC utilizing the open source Bonfire NoC router [3]. The Bonfire router uses wormhole switching with 32-bit flit width. In order to reduce the area overhead, it uses Logic Based Distributed Routing (LBDR) [17] and credit-based flow control, which has the capability of transferring up to one flit per clock cycle, assuming there are free slots in the receiver's input buffer. The version of the Bonfire router used in this work does not use virtual channels. For calculating the overhead of the proposed mechanism, all experimental results were compared to a baseline Bonfire router which does not include any fault tolerance mechanisms. The source code for both the baseline router and for the proposed ReUSE router design along with all the scripts used for calculating the experimental results are maintained as an open-source project available at [18].

In order to measure the applicability of the ReUSE mechanism in harsh environments (like space), a set of random uniform fault injection experiments were run for SETs lasting 10%, 100% and 200% of the clock period. The experiments were run with different network loads and for fault rates up to 80 million faults per second. The faults were injected by forcing the signals in Modelsim simulation to take faulty values. The results of the experiments in Fig. 4 show that when compared to the fault-free run (shown with an orange line in the figure), the latency did not change much when faults were

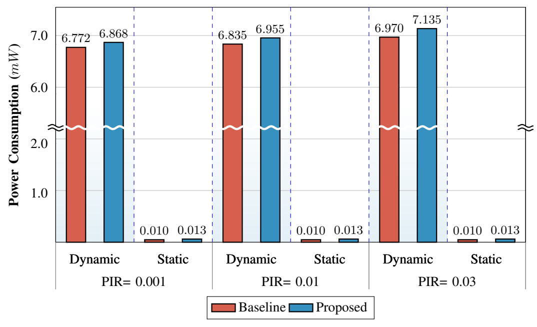


Fig. 5: Comparison of dynamic and static power consumption (in mW) of the proposed and baseline 4×4 network under different packet injection rates

injected, even under extreme fault injection rates.

The area overhead the ReUSE mechanism imposes on the baseline Bonfire router can be seen in Table I. Table II shows the critical path delay overhead. In order to obtain those results, both designs were implemented in Register Transfer Level (RTL) in VHDL and synthesized using TSMC 40 nm CMOS technology standard cell library in Synopsys Design Compiler, for 400 MHz clock frequency.

Finally, Fig. 5 shows the power usage of dynamic and static power of the ReUSE mechanism, when compared to the baseline. As it can be seen in the figure, the increase in power usage implied by the ReUSE mechanism is minimal.

V. Conclusion

Due to the trend of shrinking transistors' feature size, Network-on-Chip links have become more susceptible to run-time Single Event Transients (SETs), thus, affecting the operation of the entire system. This paper presented the ReUSE mechanism for handling SETs on inter-router links. To this end, the inherent redundancy in the input buffers is utilized, avoiding the usage of any additional re-transmission buffers when mitigating SETs at run-time. The proposed mechanism only adds 18% area overhead and 7.8% critical path delay overhead to the baseline router (router without any fault tolerance mechanisms). Experimental results show the effectiveness of the ReUSE mechanism even in extreme fault environments. As future work, the authors plan to extend this work to cover permanent faults on the router links as well. Moreover, the mechanism would be extended to cover faults occurring in the control part of NoC routers.

Acknowledgment

The work has been supported by EU's Twinning Action TUTORIAL, Estonian Institutional research grant IUT 19-1, Estonian center of excellence in IT EXCITE, and Estonian IT Academy programme.

References

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov 2005.

[2] N. Ni, M. Pirvu, and L. Bhuyan, "Circular buffered switch design with wormhole routing and virtual channels," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, Oct 1998, pp. 466–473.

[3] S. P. Azad, B. Niazmand, K. Janson, N. George, A. S. Oyeniran, T. Putkaradze, A. Kaur, J. Raik, G. Jervan, R. Ubar, and T. Hollstein, "From online fault detection to fault management in Network-on-Chips: A ground-up approach," in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2017, pp. 48–53.

[4] E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst, "Designing Networks-on-Chip for high assurance real-time systems," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan 2017, pp. 185–194.

[5] E. Wachter, V. Fochi, F. Barreto, A. Amory, and F. Moraes, "A hierarchical and distributed fault tolerant proposal for NoC-based MPSoCs," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2017.

[6] X. Chen, Z. Lu, Y. Lei, Y. Wang, and S. Chen, "Multi-bit transient fault control for NoC links using 2D fault coding method," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, Aug 2016, pp. 1–8.

[7] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 6, pp. 1053–1066, June 2013.

[8] Q. Yu and P. Ampadu, "A dual-layer method for transient and permanent error co-management in NoC links," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 1, pp. 36–40, Jan 2011.

[9] S. Ogg, B. Al-Hashimi, and A. Yakovlev, "Asynchronous transient resilient links for NoC," in *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA: ACM, 2008, pp. 209–214.

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

[11] K. Janson, R. Pihlak, S. P. Azad, B. Niazmand, G. Jervan, and J. Raik, "Await: An ultra-lightweight soft-error mitigation mechanism for network-on-chip links," in *2018 13th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC) (Accepted for publication)*, July 2018.

[12] A. Dutta and N. A. Touba, "Reliable network-on-chip using a low cost unequal error protection code," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sept 2007, pp. 3–11.

[13] S. R. Naqvi, V. S. Veeravalli, and A. Steininger, "Protecting an asynchronous NoC against transient channel faults," in *2012 15th Euromicro Conference on Digital System Design*, Sept 2012, pp. 264–271.

[14] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant Network-on-Chip architectures," in *International Conference on Dependable Systems and Networks (DSN'06)*, June 2006, pp. 93–104.

[15] R. R. Tamhankar, S. Murali, and G. De Micheli, "Performance driven reliable link design for networks on chips," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '05. New York, NY, USA: ACM, 2005, pp. 749–754. [Online]. Available: http://doi.acm.org/10.1145/1120725.1121009

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

[17] J. Flich and J. Duato, "Logic-based distributed routing for NoCs," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 13–16, Jan 2008.

[18] "ReUSE reposity," https://github.com/Project-Bonfire/ReUSE, 2018.