FAULT TOLERANT TECHNIQUES FOR ASYNCHRONOUS NETWORKS ON CHIP

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2016

By
Guangda Zhang
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

Advancing semiconductor technology is boosting the core count on a single chip to achieve continuously increasing performance, posing a growing demand for scalable, efficient and reliable on-chip interconnection. However this advance also makes the electronics increasingly vulnerable to faults. Inter-core connection is increasingly provided by Networks-on-Chip (NoCs), typically using conventional synchronous designs. Scaling makes it increasingly hard to avoid problems with clock distribution and in many chips a single, synchronous domain is inappropriate, anyway.

In place of the well-studied synchronous NoCs, event-driven asynchronous NoCs have emerged as a promising replacement. Asynchronous NoCs have many promising advantages over synchronous ones; however, their fault-tolerance has rarely been studied. Implemented in a Quasi-Delay-Insensitive (QDI) fashion, asynchronous NoCs can achieve high timing-robustness but show complicated failure scenarios in the presence of faults and behave differently from synchronous ones, posing a challenge to asynchronous circuit advocates.

This research studies the impact of different faults on QDI NoC fabrics and presents thorough and systematic fault-tolerant solutions at the circuit level, providing a holistic, efficient and resilient interconnection solution for QDI NoCs.

The contributions of this research include: 1) a thorough analysis of fault impact on QDI NoCs; 2) a Delay-Insensitive Redundant Check (DIRC) coding scheme protecting QDI links from transient faults; 3) a novel time-out technique detecting the fault-caused physical-layer deadlock in a QDI NoC (the adaptability of a QDI circuit to timing variation makes it vulnerable to this kind of deadlock); 4) a fine-grained recovery technique utilising a Spatial Division Multiplexing (SDM) implementation to recover the deadlocked network from a link fault. Both unprotected and protected QDI NoCs are implemented, along with a fault simulation environment, to provide a detailed performance and fault-tolerance evaluation of these techniques. The improvements to the NoC operation, together with the costs in circuit overhead and throughput are enumerated using a typical example of QDI interconnection.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

The past three and a half years' time in Manchester is one of the most valuable, unforgettable and successful periods in my life. I gain a lot of novel knowledge and skills from my research, and harvest precious friendships that I will cherish forever. I must express my appreciation to all people who helped me and guided me in the past.

I would like to thank my supervisor, Dr. Jim Garside for his excellent supervision. I am deeply impressed by his solid knowledge, insights to novel research problems, and enthusiasm for work and life. His guidance and constant encouragement continuously motivate me to make progress. Thanks also to my co-supervisor, Dr. Javier Navaridas. His knowledge in Networks-on-Chip inspires me lot. Without his support, I cannot finish my PhD. I would like to say special thanks to Dr. Wei Song, who acts as a strict teacher at work and a dear good friend in life. He helped check my research ideas and review my writings over and over, making a good start for my PhD.

I must express my thanks to my supervisor before PhD, Prof. Zhiying Wang, for helping me build the basis as a researcher. Many thanks to Dr. Hongyi Lu, Dr. Li Shen, Dr. Jiangchun Ren and Dr. Lei Wang for their guidance in my study and research. Thanks also to Dr. Peter Beerel, Prof. Alex Yakovlev, Prof. Jen Sparsø, Prof. Ney Calazans, Prof. Marly Roncken, Dr. Ivan Sutherlan, Dr. Fu-Chiung Cheng and Dr. Jakob Lechner for their reviews on my ASYNC papers and constructive discussions in the ASYNC conferences. Thanks Dr. Danil Sokolov for his help on Workcraft.

It is impossible to complete this PhD without the help from members in the APT group. Many thanks to Prof. Steve Furber, Dr. Doug Edwards, Dr. Dirk Koch, Dr. Mikel Luján, Dr. Vasilis Pavlidis, Dr. Luis Plana, Dr. Will Toms and Dr. John V. Woods for their direct or indirect help to my research progress. Many friends have helped me pass the time joyfully. Great thanks to Chengkun Wu, Wei Yi, Chao Li and other friends.

Finally, I must express my great thanks to my mentors for life, my parents Chuanliang Zhang and Huiyuan Li. I am proud of being your son. Thank you!

Chapter 1

Introduction

1.1 Motivation

In 1965, Gordon Moore observed that the number of components per chip doubled roughly every two years [Moo65], which was named ***Moore's law*** and approximately pertained to the scaling trend of the semiconductor industry in the past decades. The current semiconductor manufacturing technology used in commercial microprocessors has arrived at 22 *nm* and the transistor count on a single chip has reached billions [WCB+15]. Table 1.1 lists several state-of-the-art processor designs with billions of transistors. Targeting on the 10 *nm* technology node, TSMC has made an ambitious plan to enter the 7 *nm* full development in 2018 [TSM14]. This advancing semiconductor technology continues boosting the performance of processors. However, purely relying on technology progress to improve processor performance cannot always satisfy the rapidly increasing computing requirements [Gee05]. As a solution, manufacturers started building multiple processing cores on a chip to improve the overall performance, leading to a multi-core or many-core era.

Multi-core and many-core systems

Since the first general purpose dual-core processor, POWER4, was released by IBM in 2001 [BBF+01], multi-core techniques have become a trend to improve the performance of general-purpose processors and application-specific System-on-Chip (SoC) designs [Vaj11], resulting in Chip Multi-Processors (CMP) [BDM09] or Multi-Processor System-on-Chip (MPSoC) [WJM08]. The integrated processing cores do not necessarily run as fast as the highest performing single-core model, but fully utilise the parallel features of applications and achieve high performance at low energy [Vaj11].

Table 1.1: Several state-of-the-art multi-core processors

Processor	Semiconductor technology (*nm*)	Core count	Die area (mm^2)	Transistor count (*billion*)
SPARC T5 [HBC+13]	28	16	–	1.5
Xeon Ivytown [RMA+14]	22	15	–	4.31
Power 8 [FFD+14]	22	12	649	4.2
IBM System x [WCB+15]	22	8	678	4.0
Xeon E5-2600 [BSN+15]	22	18	663.52	5.56

Current mainstream commercial processors or platforms, such as Intel Core i3, i5 and i7 [Int15], AMD FX, Athlon and Opteron [AMDI15], and processors in Table 1.1 are all multi-core. As the core count on a chip increases, "multi-core" is progressing to "many-core" which usually refers to multi-core architectures with ten or more cores. Representative many-core systems include SpiNNaker MPSoC (18 cores) [FLP+13], Intel Xeon Phi Coprocessor 5110P (60 cores) [Int13] and TILE-Gx100 processor (100 cores) [Ram11]. Figure 1.1 shows the die of the SpiNNaker MPSoC built from 18 ARM cores [FLP+13]. This continuously increasing core counts lead to a growing demand for scalable and efficient on-chip communication fabrics.

Networks-on-Chip (NoCs)

Conventional communication fabrics include bus and Point-to-Point (P2P) interconnects [KZT05, BDM09, Int09]. Figure 1.2a shows the bus structure where Processing Elements (PEs) share the same interconnects so that its performance reduces linearly with the PE count. As the PE count increases, the bus could suffer from constrained bandwidth, long transmission latency and intolerable heat dissipation [DMB06]. P2P

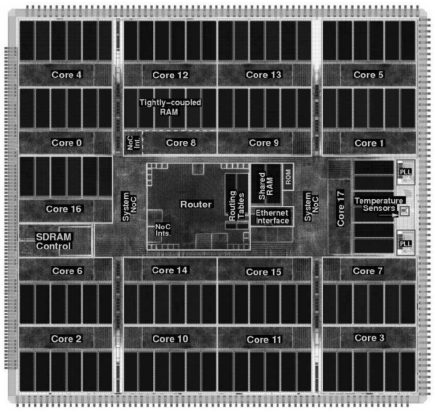
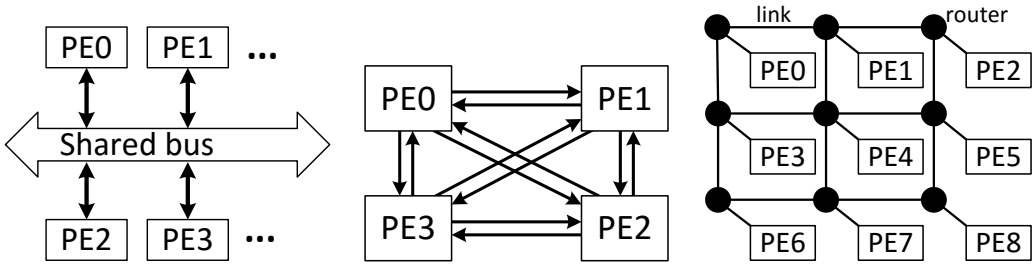


Figure 1.1: The SpiNNaker MPSoC [FLP+13]



(a) Bus (b) Point-to-Point (c) Mesh NoC

Figure 1.2: On-chip interconnects

interconnects can deliver data efficiently. As Figure 1.2b shows, each two PEs have specialized communication links so that P2P interconnection achieves a high bandwidth, but with a price of area and power due to the large number of required pins and wire resources, which limits its scalability.

As a promising candidate, Network-on-Chip (NoC) was proposed as a trade-off to support massively parallel communication [DMB06,EJP09]. It tends to structure wires (Figure 1.2c) so that the average distance between any two PEs – which are usually Intellectual Property (IP) cores – is shortened [DT01], reducing the average packet propagation delay and power consumption [BS06]. It facilitates a natural modularity and supports concurrent communication. Different nodes can communicate through different paths, increasing the network average bandwidth. Figure 1.2c illustrates a 3×3 mesh NoC, whose area increases linearly with the node count. Compared with classical interconnection structures, NoC is more efficient in large-scale multi-core or many-core systems [DT01, BS06, EJP09].

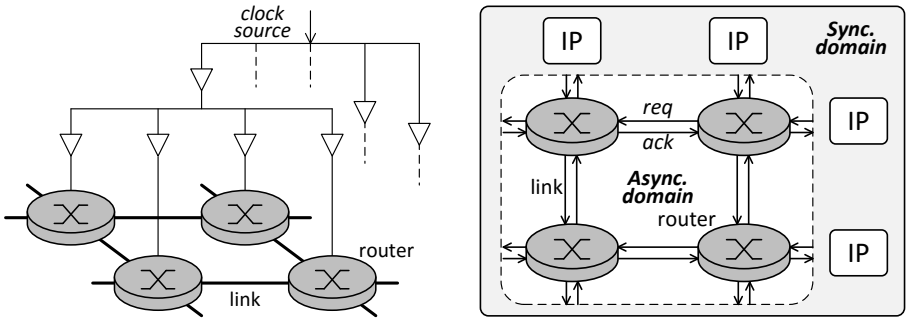
Currently, most NoC designs are built synchronously with one or multiple global clocks [BM06, AFEH12, CC15], which is well supported by current commercial Electronic Design Automation (EDA) tools. The shrinking semiconductor geometry makes circuits more sensitive to process and environmental variations, resulting in delay variations which may corrupt basic timing requirements and are harmful to synchronous circuits. In a synchronous system, all signals should comply with the timing constraints (setup/hold constraints) mandated by the clock signal. As the technology scales, however, the timing analysis becomes increasingly difficult. The gate delays are known relatively early in the implementation process (after the synthesis) while the interconnect delays, which are dominating the total circuit delay, can only be modelled when the chip layout has been performed [Lec14]. Any unsolved timing violations may

result in the redesign of the circuit. The other challenge residing at the timing is the increasing uncertainty as the process variations become commonplace as the technology progresses. These uncertainties were expected to add 30% of the total timing budget to ensure that the manufactured chips works within specification [Nas00, TCL+07]. In terms of synchronous NoCs, the global clock needs to be distributed over long distances with little clock skew and jitter, which can be a challenge. It is also a notable power drain [TSR+98] (it consumes 20% – 50% of total power in synchronous circuits [PS02, LCYH14] while synchronous NoCs could consume 33% – 36% of total power [TKM+02, HVS+07, VHR+08]). Thus, reducing or removing the clock network could save lots of chip dynamic power. It is commonplace that multiple IP cores integrated on a chip run at their own clock frequencies, partitioning the chip into multiple separate timing domains [BSN+15]. Delivering the global clock across so many timing domains in a large-scale NoC is increasingly difficult, requiring lots of extra design effort to implement the synchronization [HBC+13, BSN+15]. Too much synchronization could limit the network performance and harm the system robustness [BM06]. The modularity and scalability at the physical level is also increasingly difficult to achieve [KGGV07]. These factors urge researchers to look at other possible alternatives to replace traditional synchronous circuits.

Asynchronous NoCs

As an alternative, NoCs can be implemented using asynchronous circuits [SF01]. Having a history of over 50 years [MB59], asynchronous circuits use handshake protocols rather than global clocks to control the communication between modules, providing a fundamental solution to problems caused by the clock. Figure 1.3 compares SoCs constructed using synchronous and asynchronous NoCs. Different from the synchronous system with a clock tree, asynchronous NoC partitions a chip into multiple synchronous islands, leading to a Globally Asynchronous, Locally Synchronous (GALS) system [MVK+99]. The NoC constructs the asynchronous domain, whose communication is controlled by (*req*, *ack*) handshakes, while the attached synchronous IPs construct the synchronous domains and run at their own clock frequencies.

Compared with synchronous NoCs, getting rid of the global clock, issues such as clock skew and high clocking power inherent to the distribution of the clock tree are alleviated or removed in asynchronous NoCs [DGK09, CBL+10]. Signals switch only when a request or operation is processed, while unused circuits are not triggered due to their event-driven nature. As a result, asynchronous NoCs can achieve low dynamic



(a) Synchronous (b) GALS

Figure 1.3: SoCs constructed using synchronous and asynchronous NoCs

power consumption and near zero-standby power [SF01]. The event-driven, asynchronous NoC could convey data in a speedy and energy-efficient fashion [CBL+10]. The division of the whole chip into independent synchronous domains naturally enables individual frequency/voltage control and simplifies the chip-level timing closure [KGGV07, BCMV08], providing an efficient support for heterogeneous networks. The isolated timing domains and simple handshake interfaces provide good support for design reuse and modularity. Some classes of asynchronous circuits, such as the Quasi-Delay-Insensitive (QDI) circuits [SF01], which relax timing constraints on gate and wire delays are inherently timing-robust. Though variations of supply voltage, temperature and fabrication process parameters have increasing impact on sub-micron designs and often cause delay variations [LPP08], a QDI asynchronous NoC can tolerate these gate or wire delays, which is attractive for large-scale multi-core systems.

Fault-tolerant asynchronous NoCs

The advancing semiconductor technology on the one hand boosts the chip performance improvement and permits more processing cores to be integrated. On the other hand, accompanied with the shrinking device dimensions, factors like the lowering power voltage, the increasing clock frequency and the growing density of chip all have a negative impact on the chip reliability [Bor05].

In the deep sub-micron era, variations in manufacturing and operating conditions have a proportionately greater effect than before. Shrinking transistor dimensions means that variations in manufacturing such as dopant levels and crystal boundaries influence transistor and wire properties with time [Bor05]; growing chip density results in a high heat flux across the die, creating hot spots with a high temperature, which affects the circuit performance and accelerates the device ageing process; reducing

supplying voltage gives greater susceptibility to various noise sources [Con03, Bor05]; increasing clock frequency increases the chance that noise creates faults on the circuit. As a result, the sensitivity of electronic devices to environmental variations is significantly increased and the device ageing process is accelerated. It has been reported that the mean values of Soft Error Rate (SER) of three circuits under a 40 *nm* process are 2.2E-4 FIT, 4.7E-4 FIT and 1.2E-4 FIT respectively (1 FIT = 1 Fail per 1E9 Hours) [YYL+12]. The 24 *MByte* of Level 3 Cache in an Intel Processor encountered 0.2∼2 errors per year under the SER of 1E-4∼1E-3 FIT/bit [Sla11]. A SER in the order of 1E-3 FIT/bit has also been observed on the Altitude SEE Test European Platform [AMM+15]. It was predicted that the SER per logic state bit could increase 8% in each technology generation [HKM+03]. The SER in SRAMs would increase 6∼7× from 130 *nm* to 22 *nm* process [ITY+10]. In 65 *nm* technology, the radiation can cause a 6.45× increase in SER when the supply voltage decreases from 1.0 *V* to 0.33 *V* [PCC+14]. It is believed that both the SER and the ageing speed would increase as the technology continues scaling [Con03, Bor05, GGC+13, NCL+15]. Authors disagree on the absolute number of faults in particular circuits on particular processes, but all agree that the trend is for them to increase as processes shrink. Electronic systems are more susceptible to *faults* [Bau01], including *transient*, *intermittent* and *permanent* faults depending their properties [Muk08]. The 2015 ITRS takes *reliability* as one main challenge faced by the next generation electronics and stresses the importance of a runtime protection [Bot15]. Thus, *fault-tolerance* has become an essential design objective for critical digital systems, especially in highly specialized fields such as aerospace, military and medical equipment.

The fault-tolerance of synchronous NoCs has been extensively studied. Faults typically cause data errors (or packet loss) which can be detected and corrected during several clock cycles. A clock signal provides a timing reference for error detection and correction. Detecting the error or packet loss, a retransmission can be requested to obtain the right packet [Sor09]. However, in an asynchronous NoC, there is no such timing reference. The QDI implementations are robust to timing variations, but not to faults. A fault may pollute a transmitting packet, corrupt the handshake protocol and disrupt the normal data flow, which is a new challenge faced by asynchronous circuit designers. A single fault could even break the handshake protocol and results in a *fault-caused physical-layer deadlock*. This deadlock is different from the well-known network-layer one due to the cyclic dependence of multiple packet transmissions [DS87, DT03]. Most conventional fault-tolerant or deadlock management

techniques for synchronous NoCs cannot work in a *deadlocked* state.

The fault-tolerance of asynchronous NoCs has not been thoroughly studied. Various styles of asynchronous NoCs have been proposed but rarely do they have fault-tolerance capabilities [BF02, FF04, SG08, BS06, DGK09, PMMC11]. Therefore, providing fault-tolerant asynchronous NoCs is an imminent and game-changing technology, which will enable the usefulness of asynchronous NoCs in the many-core era.

1.2 Problem description

Networks-on-Chip (NoC) is a promising infrastructure to support on-chip communication of large-scale multi-core systems due to its efficient and scalable feature [DT01]. Most existing NoCs are built synchronously with a global clock. As the network scales, they may be restricted by problems within the distributed clock tree. As an alternative, NoCs can be implemented using asynchronous circuits controlled by handshake protocols [SF01]. The resulting asynchronous NoC achieves many attractive features and is free from the heavy burden of a global clock.

In the deep sub-micron era, operating conditions of digital systems can vary, posing a reliability challenge to critical electronics. Electronic devices become susceptible to timing errors due to delay variations and logic faults where the logic levels of signals are corrupted [Con03]. Quasi-Delay-Insensitive (QDI) NoCs are naturally timing-robust so that delay variations can be tolerated. However, the fault-tolerance of asynchronous NoCs, especially the QDI ones, has not been fully studied compared with synchronous ones. Most existing asynchronous NoCs lack fault-tolerance capability, making them susceptible to *faults*.

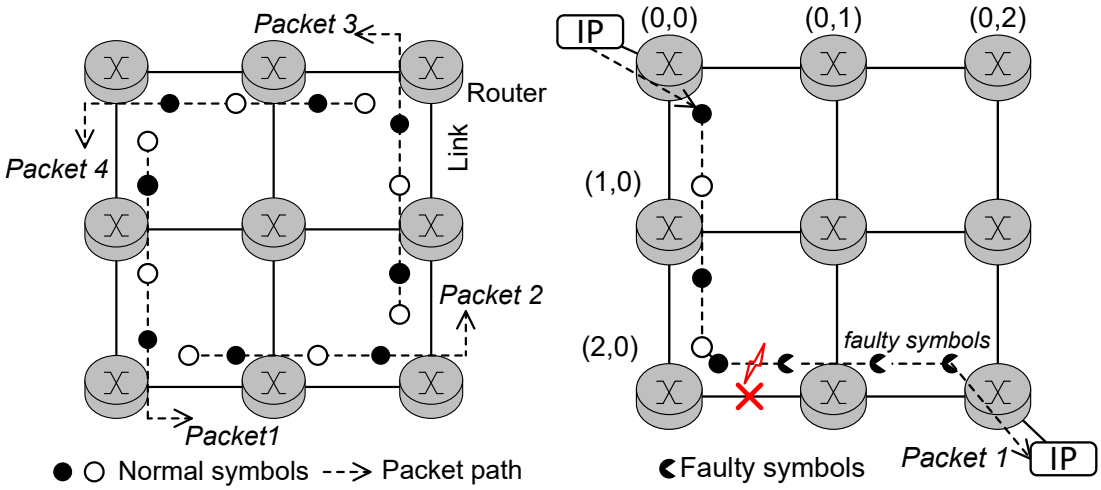
Faults can be classified into *transient*, *intermittent* and *permanent* depending on their duration [Con03]. Transient faults usually last for a short time and behave as positive or negative glitches (“0→1→0” or “1→0→1”) [Bau01, KH04]. Permanent faults will influence the victim gates or wires forever. Most permanent faults can be modelled as “stuck-at” faults [AAA87, MMC15], where the logic level of a net is always “0” or “1”. Intermittent faults usually happen as an early manifestation of permanent ones with the ageing process [Con03]. They can appear as either transient or permanent during error detection or correction.

In the presence of faults, QDI NoCs behave differently from synchronous ones. A fundamental difference between synchronous and QDI circuits is the timing reference of transmitting data symbols.

- In synchronous circuits, a data symbol typically has a constant time per bit which can be agreed – and maintained for a known time – between the transmitter and the receiver. Corruption of the transmission will therefore affect a known number of bits. Thus faults on a synchronous NoC may corrupt transmitting data packets, leading the packet to a wrong destination, resulting in packet loss or causing data errors, but the erroneous data symbol or faulty behaviour can be easily detected and further corrected or recovered.

- In QDI asynchronous circuits, there is no such timing reference, so that faults can insert, or possibly delete symbols as well as corrupt them. Managing these faulty cases represents a new challenge faced by QDI NoCs. Besides, it is obvious that a permanent fault will stall the handshake and cause a physical-level deadlock. Its detection and recovery has not been thoroughly studied in a NoC backbone. More seriously, a transient fault can not only cause data errors, but also deadlock a QDI NoC, which has been neglected by the asynchronous community. These all increase the challenge of fault detection and recovery in QDI NoCs.

Deadlock is fatal to a NoC without any management mechanisms [DS87]. It can reduce the network performance, paralyse its function and eventually cause the chip to be discarded. The well-known *network-layer* deadlock due to the cyclic dependence of packets or restricted routings [DT03] can happen in all NoCs. Figure 1.4a shows an example where four packets hold and request network resources in a cyclic fashion, which is a network-layer deadlock. It can be resolved by using specific turn models



(a) Network-layer deadlock

(b) Fault-caused physical-layer deadlock

Figure 1.4: Network-layer and physical-layer deadlocks in a QDI NoC

or providing extra escape channels [DT03]. This network-layer deadlock is common in all NoCs and not the point of this research. In QDI NoCs, a fault may break the handshake protocol, resulting in a physical-layer deadlock, which is particular to QDI NoCs. Taking a simple (*req*, *ack*) handshake process for example, if the sender sends out a request to the receiver but without getting acknowledged, in a QDI circuit which is immune to delay variations, the sender does not know whether this is caused by a fault or a delay, so it keeps waiting for the lost *ack*, resulting a physical-layer deadlock. Figure 1.4b illustrates a faulty case that a fault on a transmitting packet deadlocks the reserved data path in the network. Note that it is the adaptability of a QDI circuit to timing variations that makes it more vulnerable to this kind of deadlock-type faults. This physical-level deadlock cannot be easily resolved by higher level techniques for network-level ones.

The management of a fault-caused physical-level deadlock in QDI NoCs can be divided into deadlock *detection* and *recovery*. Since there is no data transmission and no further error history can be collected, it is difficult to detect and locate the fault in the deadlocked state. If both transient and permanent faults are considered, the fault type should be diagnosed first so that different methods can be used to recover the network. The other difficulty during the *detection* phase is differentiating the fault-caused physical-layer deadlock from the network-level one and the network congestion (packet transmission is stalled in relevant paths). In the *recovery* phase, the deadlock should be removed and the blocked, fault-free network resources should be released. The management of this physical-level deadlock represent a new challenge faced by QDI NoCs.

In the deep sub-micron era where reliability becomes one of the most critical challenges for digital systems, keeping the critical or ultra-expensive systems working is important even with some performance loss, posing a demand for fault-tolerant QDI asynchronous NoCs. Conventional fault-tolerant techniques for synchronous circuits or NoCs cannot easily be migrated to QDI NoCs. This research studies fault-tolerant techniques for QDI asynchronous NoCs and proposes mechanisms which allow recovery from various fault scenarios.

1.3 Research objectives

The overall objective of this research is to study the impact of different faults, including transient and permanent ones, on QDI NoCs and propose thorough and systematic

fault-tolerant solutions to protect QDI NoCs, providing a holistic, efficient and resilient interconnection solution. The achieved fault-tolerance capability and the incurred performance and hardware overhead are two main evaluation factors.

Protecting QDI links

A large-scale NoC may contain a large number of long link wires, which are common in large-scale Systems-on-Chip (SoCs). Exposed to the external environment, they can be easily affected by various noise or fault sources and become the victim of timing variations or transient faults [BS09]. These chip-level long interconnects can be implemented as QDI pipelines to achieve high bandwidth and timing-robustness. However, in a QDI system, a transient fault can be accepted as a valid signal, inserting, deleting or corrupting a data symbol. Fault-tolerant codes have been widely used to protect on-chip communication [Sor09]. Codes also perform an important role in QDI circuits where Delay-Insensitive (DI) codes are used to build data symbols to encode the timing information. Most existing state-of-the-art fault-tolerant codes proposed for asynchronous circuits either corrupt the timing-robustness feature of QDI circuits or result in large area and speed overhead. This research presents a novel Delay-Insensitive Redundant Coding (DIRC) scheme to protect QDI communication from transient faults, which can be easily migrated to existing DI or QDI interconnects without destroying their intrinsic timing-robustness. The protected QDI links can be constructed flexibly to satisfy various fault-tolerance requirement, with a moderate and reasonable hardware overhead.

Detecting fault-caused deadlocks

Both permanent and transient faults could break the handshake process in QDI NoCs, resulting a physical-layer deadlock, which is more serious to the system function than data errors and needs to be resolved. The management of this fault-caused physical-layer deadlock is significantly important to prolong the chip life, which has barely been studied. It includes two phases: *deadlock detection* and *recovery*.

Detection of a fault-caused physical-level deadlock is difficult in a QDI NoC since in a deadlocked state, error syndromes for fault analysis cannot be easily collected and locating a specific defective wire or gate is difficult. The ideal situation is that the faulty component can be precisely located so that a recovery method can be further used to bypass or replace it and recover the network function. Therefore, an efficient and flexible detection method is necessary. It should be able to not only precisely locate the

fault in the QDI NoC, but differentiate the fault-caused physical-level deadlock from other similar network scenarios, including the upper network-layer deadlock and the network congestion. When both transient and permanent are considered, an accurate model is needed to differentiate deadlocks caused by different faults, so as to enable the *fault diagnosis*. The proposed techniques should be able to detect, diagnose and locate the fault as long as the fault deadlocks the network.

Recovering the NoC from fault-caused deadlocks

As the fault position has been located, the next step is to recover the network function according to the deadlocked state of the handshake (4-phase, 1-of-n in this research [SF01], which was used in many state-of-the-art NoC designs [BF02, FF04, TVC10, SE11b]) and the network protocol (*wormhole switching* [DT03]). Figure 1.4b shows one possible deadlock case that a fault deadlocks a reserved long packet path, containing the faulty link, and other healthy network sources. A direct system reboot can remove this deadlock temporarily, but it is expensive and cannot deal with the deadlock caused by permanent faults. Thus, a fine-grained recovery strategy is necessary to remove the deadlock and isolate the faulty component (which is the inter-router link in this research). The recovery contains two main processes: (1) deadlock removal, which recovers the stalled packet flow in the deadlocked packet path, releasing blocked healthy network resources and eliminating the deadlock; (2) faulty link isolation: instead of using upper network-layer methods such as fault-tolerant routings to detour the faulty link, this research proposes a fine-grained recovery technique at the lower physical layer to isolate the faulty component and restore the network function. Upper layer recovery techniques can further be used to improve the network performance after the loss of the faulty component. When transient and intermittent faults deadlock the network, the isolated link should be resumed.

1.4 Research contributions

The following contributions have been made by this research:

- A fault model systematically analysing the impact of different faults on 4-phase, 1-of-n QDI pipelines, especially the fault-caused physical-layer deadlocks.
- A new *Delay-Insensitive Redundant Check (DIRC)* coding scheme protecting 4-phase, 1-of-n QDI links from transient faults.

  - It tolerates all 1-bit and some multi-bit transient faults on QDI interconnects. DIRC coding scheme can be easily migrated to the existing DI or QDI interconnects without destroying their intrinsic timing-robustness.
  - DIRC pipeline can be constructed flexibly to satisfy the various fault-tolerance requirement in practical designs, with a moderate and reasonable hardware and performance overhead. Besides data wires protected by DIRC, wires carrying acknowledgement signals for the handshake are also protected.

- A new *deadlock detection* mechanism detecting the fault-caused physical-layer deadlock and locating the fault position precisely.

  - Fault detection can be translated to the detection of physical-layer deadlocks. As long as a QDI pipeline or QDI NoC is deadlocked by a fault, it can be detected with little overhead.
  - This detection mechanism can differentiate the fault-caused physical-layer deadlock from the network-level one and network congestion.
  - The type of faults that cause physical-level deadlocks can mostly be diagnosed when different faults are considered.

- A fine-grained *network recovery* strategy recovering QDI NoCs from physical-level deadlocks, including *deadlock removal* and *faulty component isolation*.

  - *Drain&Release* technique can release fault-free network resources which were deadlocked previously due to the fault, removing the deadlock.
  - Spatial Division Multiplexing (SDM) is employed to implement the QDI NoC where each inter-router link is physically divided into multiple independent sub-links. Isolating the defective sub-link, the other healthy ones can still be used to transmit packets, providing an efficient network recovery with low hardware overhead and graceful performance degradation.
  - For deadlocks caused by transient or intermittent faults, the isolated sub-links can be resumed to use after the fault disappears.

1.5 Thesis organization

The rest of this thesis is organized as follows:

- Chapter 2 introduces fundamental concepts of asynchronous circuits and Networks-on-Chip (NoCs), and presents the state-of-the-art asynchronous NoCs
- Chapter 3 introduces different fault types and sources and briefly demonstrates existing fault-tolerant techniques. 4-phase, 1-of-n QDI pipelines are modelled to analyse the impact of different faults, especially the deadlock effect, which is the basis of all the fault-tolerant techniques proposed in this research.
- Chapter 4 presents the Delay-Insensitive Redundant Coding (DIRC) scheme to protect QDI links from transient faults. Detailed experimental results are given.
- Chapter 5 presents a new time-out deadlock detection technique which can detect the fault and locate its position precisely if the fault has caused a physical-level deadlock. A fault diagnosis method is presented to determine the fault type when both transient and permanent faults are considered.
- Chapter 6 presents fine-grained network recovery techniques, including *Drain & Release* mechanisms to remove the deadlock and release blocked fault-free network resources, and the Spatial Division Multiplexing (SDM) router design with which the faulty component – which is the inter-router sub-link in the implemented NoC – is isolated from the network. For deadlocks caused by transient faults, the isolated sub-link is healthy, which will be resumed after the recovery.
- Chapter 7 presents the design of the baseline QDI router and a fault simulation environment. Detailed experimental results of protected and unprotected routers or NoCs are given to evaluate the proposed fault-tolerant techniques.
- Chapter 8 concludes the thesis and presents the future work.
- Appendix A presents the implementation of several basic asynchronous cells.
- Appendix B presents the latency and area analytical models evaluating the proposed DIRC codes in Chapter 4.

1.6 Publications

1. **Guangda Zhang**, Wei Song, Jim Garside, Javier Navaridas and Zhiying Wang. *Transient Fault Tolerant QDI Interconnects Using Redundant Check Code*. EUROMICRO Conference on Digital System Design (DSD), Santander, Spain, 2013.

(The DIRC code in Chapter 4 was first published in this paper [ZSG+13].)

2. **Guangda Zhang**, Wei Song, Jim Garside, Javier Navaridas and Zhiying Wang. *Protecting QDI Interconnects from Transient Faults Using Delay-Insensitive Redundant Check Codes*. Microprocessors and Microsystems, Elsevier, 38(8):826-842, 2014.
   (The fault models analysing the impact of transient faults in Chapter 3 and the analytical models evaluating the performance of DIRC pipelines in Appendix B originate from this paper [ZSG+14b].)
3. **Guangda Zhang**, Wei Song, Jim Garside, Javier Navaridas and Zhiying Wang. *An Asynchronous SDM Network-on-Chip Tolerating Permanent Faults*. International Symposium on Asynchronous Circuits and Systems (ASYNC), Potsdam, Germany, May 12-14, 2014.
   (The fault model for a permanent fault in Chapter 3, and the management of a physical-layer deadlock caused by a permanent link fault in Chapters 5 and 6 were first published in this paper [ZSG+14a].)
4. Wei Song, **Guangda Zhang** and Jim Garside. *On-line Detection of the Deadlocks Caused by Permanently Faulty Links in Quasi-delay Insensitive Networks on Chip*. International Conference of the Great Lakes Symposium on VLSI (GLSVLSI), Houston, Texas, US, May 21-23, 2014.
   (The general detection technique of a fault-caused physical-layer deadlock was published in this paper [SZG14].)
5. **Guangda Zhang**, Jim Garside, Wei Song, Javier Navaridas and Zhiying Wang. *Deadlock Recovery in Asynchronous Networks on Chip in the Presence of Transient Faults*. International Symposium on Asynchronous Circuits and Systems (ASYNC), Mountain View, Silicon Valley, California, US, May 4-6, 2015.
   (The fault model for transient faults causing deadlocks in Chapter 3, and the deadlock recovery in the presence of transient faults in Chapters 5 and 6 were published in this paper [ZGS+15].)
6. Lei Wang, Yuxing Tang, Yu Deng, **Guangda Zhang** and Feipeng Zhang. *A Scalable and Fast Microprocessor Design Space Exploration Methodology*, The 9th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Turin, Italy, Sep. 23-25, 2015.

Chapter 2

Asynchronous circuits and Networks-on-Chip

Advancing semiconductor technology boosts the scale of digital systems. Traditional synchronous circuits still dominate current mainstream computing systems, but face increasing difficulties with the growing clock tree. The technology for synchronous circuits is mature at some scales, but less so over contemporary System-on-Chip (SoC). Under the control of handshake protocols rather than the global clock, an event-driven, asynchronous circuit has many potential advantages over its synchronous counterpart, such as better modularity and lower dynamic power consumption. Problems inherent to the global clock distribution are removed or alleviated. In a severe environment with Process-Voltage-Temperature (PVT) variations, synchronous circuits are susceptible to timing errors. Quasi-Delay-Insensitive (QDI) circuits, which are one class of timing-robust asynchronous circuits, are promising for designs requiring high reliability.

Meanwhile, the scaling of semiconductor technology allows more Intellectual Property (IP) cores to be integrated, leading to a multi-core era when on-chip communication becomes the main bottleneck limiting the chip's performance. Networks-on-Chip (NoCs) have been proposed to effectively support on-chip communication in large-scale multi-core systems. Asynchronous NoCs could provide scalable and efficient communication with attractive features, which are especially promising as the network scales. The timing-robust QDI NoC is the study object of this research.

This chapter reviews fundamental concepts in asynchronous circuit designs and NoCs and presents the state-of-the-art asynchronous NoC designs.

2.1 Asynchronous circuit design

2.1.1 Introduction

Asynchronous circuits have a history of over 50 years [MB59]. ITRS [ITR11] predicts that, by 2025, the percentage of asynchronous circuits on a design will rise to 52%. Different from synchronous ones governed by one or several global clocks that provide a notion of time, asynchronous circuits are event-driven and use handshake protocols rather than clocks to drive different modules. The state of an asynchronous circuit changes once a request or an operation is processed, without regard to the clock pulse in synchronous circuits. Asynchronous circuit design provides a fundamental solution to problems caused by the clock, achieving promising benefits:

- *Low dynamic power consumption*: Simplistically, a synchronous circuit switches continuously. In particular, the widespread clock network is responsible for significant dynamic power dissipation. Nowadays this is typically addressed by clock gating – at some granularity – and, sometimes, by frequency adjustment. In asynchronous circuits, signals switch only when a request or operation is processed, while unused circuits are not triggered due to their event-driven nature. As a result, asynchronous circuits can achieve low dynamic power consumption due to this "fine-grained" clock gating mechanism and near zero-standby power [SF01].

- *Support for heterogeneous networks*: A Multi-Processor System-on-Chip (MPSoC) can be heterogeneous, interconnected using a Network-on-Chip (NoC) [EJP09], where network nodes are Intellectual Property (IP) cores running at different clock frequencies and working voltages, complicating the network structure and making chip-level timing closure difficult. In asynchronous circuits, different IP cores can be easily integrated through simple handshake-based interfaces, providing good support for modularity and design-reuse. An asynchronous network partitions the chip into islands, enables individual frequency/voltage control and simplifies the chip timing closure [KGGV07, BCMV08].

- *Timing Robustness*: The variations of supply voltage, temperature and fabrication process parameters have an increasing impact on sub-micron designs and often cause delay variations [LPP08]. Some classes of asynchronous circuits relax constraints on gate and wire delays, which are timing-robust and could naturally tolerate these delay variations [SF01].

- *Reduction of Electromagnetic Interference (EMI)*: The clock is a strong noise source in synchronous circuits, which could cause EMI, possibly resulting signal delay variations and even bit-flips. EMI especially threatens the data transmission on chip-level long, parallel interconnects. Without a global clock, switching activities in asynchronous circuits are relatively randomly distributed in time, reducing the amplitude of electromagnetic interference [PDF+98, KGGV07], decreasing the possibility of incorrect bit-flips.

Considering additional, complicated circuits controlling the handshake process, asynchronous circuits usually have drawbacks in area and speed in practical designs. The possible area overhead may increase static power loss, which has become a major contributor to power consumption in nanoscale technologies due to leakage currents. This static power consumption can be reduced by adding power gating circuits [OTM10] or using leakage control techniques [HB13]. Furthermore the lack of mature Electronic Design Automation (EDA) tools and design methodology deters the wide use of asynchronous designs in commercial fields [SF01].

2.1.2 Circuit classification and QDI circuits

According to different timing assumptions at the gate level, asynchronous circuits can be designed in different styles or delay models [SF01], among which Delay-Insensitive (DI) and Quasi-Delay-Insensitive (QDI) circuits are attractive to designers due to their timing-robust nature. DI circuits assume that both the delays of gates and wires are arbitrary and positive so that the logic function of the circuit is independent of delays, making DI circuits extremely robust. According to this definition, only C-elements [SF01] and inverters can be used to make a circuit DI, making it impractical to implement complicated functions [Mar90, SF01]. Most claimed DI circuits in the literature are QDI in reality [SF01]. QDI circuits emerge as a timing relaxation of

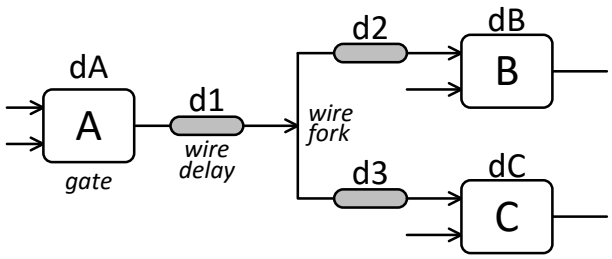
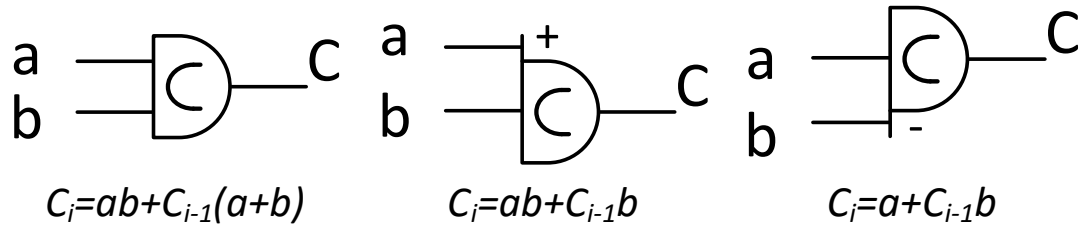


Figure 2.1: A circuit model with gate and wire forks [SF01]



(a) Systematic C-element (b) With a plus input (c) With a minus input

Figure 2.2: Muller C-elements

DI circuits. They loosen the constraint on wire delays, which allows *isochronic* wire forks [SF01]. In other words, QDI circuits work correctly if the delays of forking wires are "identical" (*d2=d3* in Figure 2.1). Isochronic forks are often found in gate-level implementations of basic blocks where designers can control the wire delays. This assumption makes QDI circuits more widely used in practical asynchronous designs than DI ones.

Besides, there are Speed-Independent (SI) circuits, which assume that gates have arbitrary, positive and bounded delays while all wire delays are zero (*d1=d2=d3*=0 in Figure 2.1). This timing assumption is not realistic in deep sub-micron process because the wire delays may be longer than the gate delay. Only if wire delays are negligible compared with delays of the connected gates, can they be lumped into gate delays to make the circuit SI, reducing their robustness to timing variations. Self-timed circuits rely on more elaborate and/or engineering timing assumptions of gates and wires, compromising their tolerance to delay variations. The worst-case timing analysis and matched delay elements are usually employed to help self-timed circuits satisfy the rigorous timing requirements [SF01].

Among these asynchronous circuits, the QDI one is extremely attractive since it is relatively easy to implement in real circuits and has a timing-robust nature [SF01]. It was preferred in many existing circuit designs, including long point-to-point on-chip interconnects and NoCs [BF02, BTEF03, LM04, JM05, CBL$^+$10, SE10a]. This research uses QDI circuits as the backbone of NoC implementations to study their fault-tolerance.

2.1.3 Basic asynchronous elements

In addition to basic logic gates like AND, OR, NOT, etc. and latches, asynchronous circuits commonly use some extra basic elements. The most common one is the Muller C-element [MB59, SF01], also known as a C-element or C-gate. It is a fundamental

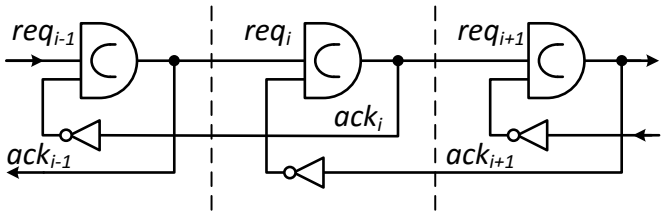
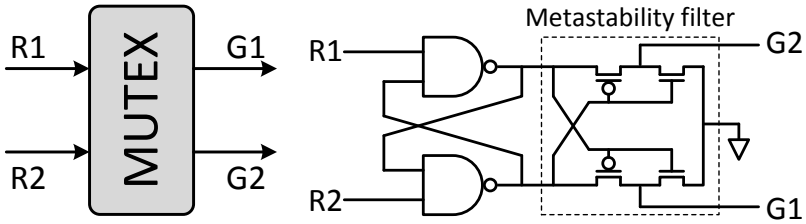


Figure 2.3: Muller pipeline



(a) Symbol (b) Implementation

Figure 2.4: Mutex Exclusion (ME) element [SF01]

control and state-holding element that has been extensively used. Figure 2.2 shows symbols of 2-input C-elements and their logic functions, where *a* and *b* are two inputs; *C* is the output. Figure 2.2a presents the symmetric C-element where the *set* or *reset* of the output is decided by both its inputs. It outputs '1' (or '0') when both of inputs are '1's (or '0's), which is a *set* (or *reset*) operation. Otherwise, if its inputs are different, the output will keep its state. Asymmetric C-elements are specialised C-elements, where some inputs affect either the set or the reset operation but not both. Figure 2.2b and 2.2c gives two examples of asymmetric C-elements with a plus and minus input respectively. A C-element can be used as an asynchronous latch to store data or an event synchronizer to synchronize two input events [Sut89]. Both the symmetric and the asymmetric C-elements are widely used in asynchronous circuits.

Built from C-elements and inverters, Figure 2.3 illustrates a simple Muller pipeline, whose communication is controlled by handshake pairs (*req*, *ack*) [SF01]. All C-elements are initialized to '0's at the beginning. The output of a C-element acts as both the request to its successor and the acknowledge to its predecessor. A C-element will propagate '1' (or '0') from its previous pipeline stage to its successor only when its successor outputs '0' (or '1'). The Muller pipeline is a backbone for most existing asynchronous pipelines. It can work correctly regardless of delays in gates and wires, showing an attractive QDI feature.

The Mutex Exclusion (ME) element [SF01] is another important asynchronous cell. Figure 2.4 presents its symbol and one possible implementation [SF01]. It reads

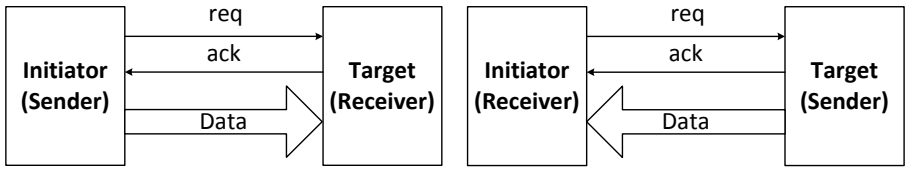
two requests and grants the one arriving first. If they arrive at the same time, the ME may enter a metastable state before making an arbitrating decision. As the timing is flexible, a filter (Figure 2.4b) can detect metastability and delay any decision until it has resolved. MEs have been widely used in arbitration models of existing asynchronous designs [BKY00, BF02, Kin07, GSX+09, SE11b].

The above basic asynchronous elements are implemented using standard cells in this research. Implementation details are introduced in Appendix A.

2.1.4 Handshake protocols

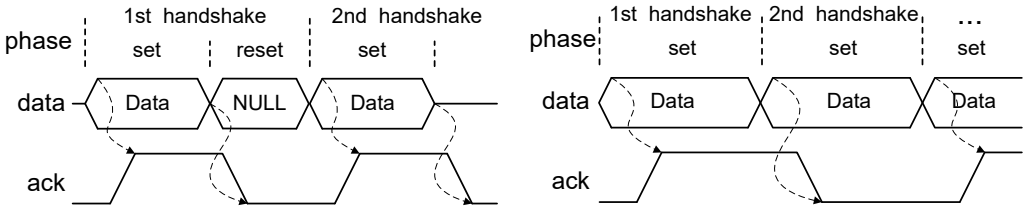
In asynchronous circuits, to start a transfer of data, the *initiator* asserts a *request* (*req*) action while the *target* responds with an *acknowledge* (*ack*) (Figure 2.5). The transferring path comprising multiple wires is termed a *channel*. The device delivering the data onto the channel is a *sender* while the other end is the *receiver*. If the initiator is putting the data onto the channel, the channel is a *push* one (Figure 2.5a); otherwise, if the initiator acts as a receiver, it is a *pull* channel (Figure 2.5b) [Bai00]. Both asynchronous channels are controlled by handshake protocols.

Asynchronous handshake protocols can be classified as 4-phase and 2-phase [SF01]. 4-phase (or *return-to-zero*) protocols are level-triggered where a reset or return-to-zero phase is always required after transmitting a valid data symbol (i.e. the set phase), as Figure 2.6a shows (the request information is implicated in the transmitting data).



(a) A push channel (b) A pull channel

Figure 2.5: Channel models of asynchronous circuits



(a) 4-phase (b) 2-phase

Figure 2.6: Handshake protocols for an asynchronous *push* channel

Figure 2.6b illustrates the edge-triggered 2-phase (or *non-return-to-zero*) protocols where each transition of the acknowledge signal starts a new data transmission. In 4-phase asynchronous pipelines, valid data symbols are separated by spacers (or *NULL*) while under 2-phase handshake protocols, data symbols are transmitted consecutively.

Compared with 4-phase protocols, 2-phase protocols avoid the reset phase and only need half the number of transitions of 4-phase signalling to transmit a single data symbol, achieving an advantage from power and speed views. However, to be aware of the current and previous circuit states, 2-phase protocols require more complex and larger circuit implementation than 4-phase ones. Consequently, 4-phase handshake protocols are more widely used for the sake of implementation simplicity [BF02, FF04, BS05, SG08, DGK09, TVC10, AN10, PMMC11, PCD$^+$11, SE11b]. This research studies fault-tolerance techniques to protect general QDI asynchronous NoCs. 4-phase handshake protocol is used to implement all QDI circuits and NoCs.

2.1.5 Data encodings

Like handshake protocols, data encoding is an important concept in asynchronous designs. It is closely relevant to delay models (DI, QDI, SI or self-timed) and has a significant impact on the circuit area, speed and energy efficiency. Besides, codes play an important role in fault-tolerance field [Sor09]. According to the timing assumptions of asynchronous circuits, data encodings can be classified into Delay-Insensitive (DI) [Ver88] and non-Delay-Insensitive (non-DI) ones.

Non-Delay-Insensitive codes

As the simplest non-DI codes, *bundled-data* codes [Bai00], also known as single-rail codes, use one wire for each bit of information; they are common in “traditional”

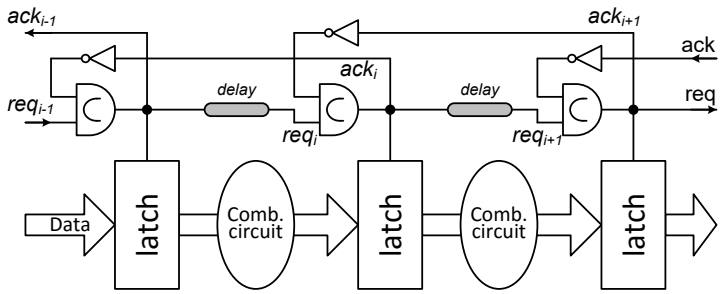


Figure 2.7: Bundled-data pipeline

synchronous circuits. Figure 2.7 illustrates a 4-phase bundled-data pipeline, which comprises a *control* path and a *data* path. The control path is a 1-bit wide Muller pipeline generating enable signals to trigger the latches of the data path, while the data path is a data pipeline where latches are separated by combinational circuits. To ensure that the triggered latches always receive stable data, delay elements are inserted between adjacent pipeline stages of the control path to match the maximum delay of the corresponding computational circuits in the data path.

Bundled-data encoding has high coding efficiency compared with the others. However, relying on timing assumptions within the delay element in the control path, bundled-data circuits are not timing-robust. It is possible to make the control delays vary according to the data being processed, with techniques such as speculation completion [Now96] or delay selection [SF01] which further complicate the timing model.

Delay-Insensitive codes

In DI or QDI circuits, data bits from a sender may arrive at the receiver at any time and in any order, reflecting the timing-robustness of these circuits. One of the crucial issues for this communication is to detect whether all data bits have arrived at the receiver or not [CH01]. Since there is no clock in DI or QDI circuits, the transmitting data should have implicit timing information. Therefore, DI codes are used in these timing-robust circuits. In DI codes, the data validity information is encoded with the data (Figure 2.6), making it possible to determine whether an entire data symbol has been received or not. Extensively used DI codes are 1-of-n and m-of-n codes.

- 1-of-n $(n \geq 2)$

  Different from an n-bit wide binary code which can represent 2^n symbols, a 1-of-n code occupies n wires and provides n valid data symbols (which have one bit '1') plus a *spacer* (or *NULL*, which are all '0's). Spacers do not represent any data information. They are used to separate two adjacent data symbols if 4-phase handshake protocol is employed. Therefore, a 1-of-n channel can deliver $\log_2 n$ bits. For each transfer of a 1-of-n code, only one of the n wires is asserted. Figure 2.8a and 2.8b illustrate 1-of-2 (or dual-rail) and 1-of-4 codes, which are two extensively used 1-of-n codes.

- m-of-n $(2 \leq m < n)$

  The m-of-n codes are a superset of 1-of-n codes but usually refer to cases of $m \geq 2$. By driving m of the total n wires high for each transfer, m-of-n codes

1-of-2 Codes	Binary
01	00
10	01
00	NULL (Spacer)

(a) 1-of-2 codes

1-of-4 Codes	Binary
0001	00
0010	01
0100	10
1000	11
0000	NULL (Spacer)

(b) 1-of-4 codes

Figure 2.8: 1-of-n codes

Table 2.1: Incomplete 2-of-7 codes

2-of-7 Codes	Binary	2-of-7 Codes	Binary
100 0001	1100	001 0001	0100
100 0010	1101	001 0010	0101
100 0100	1110	001 0100	0110
100 1000	1111	001 1000	0111
010 0001	1000	000 1001	0010
010 0010	1001	000 1010	0011
010 0100	1010	000 0101	0000
010 1000	1011	000 0110	0001
unused symbols	000 1100	000 0011	011 0000
	101 0000	110 0000	

can represent C_n^m valid symbols and show comparably higher coding efficiency than 1-of-n codes. Since the implementation overhead incurred by encoding, decoding and completion detection of m-of-n codes can be high, incomplete m-of-n codes constructed from groups of smaller m-of-n and 1-of-n codes were proposed to reduce the overhead [BTEF03]. Table 2.1 depicts the incomplete 2-of-7 codes where 5 out of the 21 codewords are unused.

- Other non-return-to-zero encodings

  Besides 1-of-n and m-of-n codes which can be used in both 2-phase and 4-phase asynchronous circuits, a Level-Encoded Dual-Rail (LEDR) signalling (Table 2.2) [DWD91] has been proposed which is a DI encoding scheme specially used in 2-phase circuits since no return-to-zero phase is required. It encodes two wires or rails – $\{data, parity\}$ – the *data* rail holding the real bit value and the *parity* rail indicates the phase by its parity relative to the data rail. The encoding alternates between two phases strictly – the even and odd phase. Therefore,

Table 2.2: LEDR codes

Phase	{Data, Parity}	Binary
Even	{00}	0
	{11}	1
Odd	{10}	1
	{01}	0

Table 2.3: Comparison between several common data encodings

Codes	DI	M	Code rate (R)	Transitions/bit /transaction
Bundled data	No	2	1	0.5
1-of-2	Yes	2	0.5	2
1-of-4	Yes	4	0.5	1
2-of-4	Yes	6	0.65	1.5
3-of-6	Yes	20	0.72	1.4
Incomplete 3-of-6	Yes	16	0.67	1.5
2-of-7	Yes	21	0.63	0.9
Incomplete 2-of-7	Yes	16	0.57	1

the encoding of bit '1' is "10" in odd phase and "11" in even phase while the encoding of bit '0' is "01" in odd phase and "00" in even phase. Another improved Level-Encoded Transition signalling (LETS) [MAMN08] was also proposed whose details are omitted here.

Comparison of common data encodings

Table 2.3 compares several data encodings which are commonly used in asynchronous communications. Since this research uses the simple 4-phase handshake protocol, 2-phase encodings [DWD91, MAMN08] are not listed. Code rate and the number of transitions per bit per transaction are two important evaluation metrics.

The code rate (R) is used to measure the coding efficiency [Ver88], as shown in (2.1), where n is the length of a codeword or the number of occupied wires; M is the number of valid code words and it equals to n for 1-of-n codes and $\binom{n}{m}$ for complete m-of-n codes.

$$R = \frac{\log_2 M}{n} \tag{2.1}$$

The average number of transitions per bit per transaction (T) is another important metric to evaluate the dynamic energy cost of the code transition. Under the 4-phase protocol, a channel initialized with a *spacer* (all '0's) needs two transitions (a rising transition at the *set* phase and a falling one at the *reset*) to transmit each '1' bit [AN12]. The calculation of the transition number T for m-of-n codes is presented in (2.2)

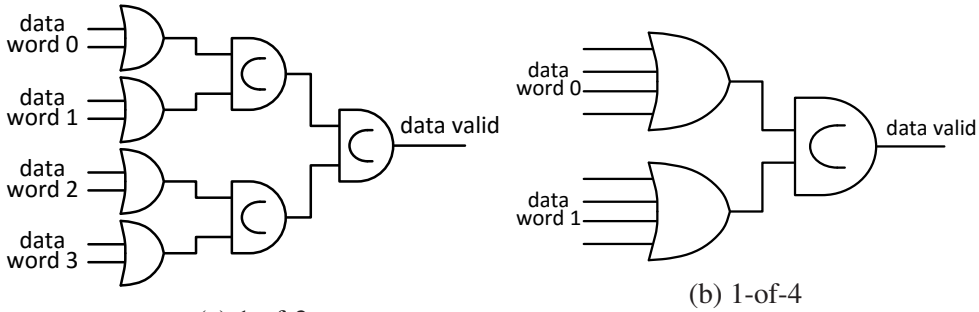
$$T = \frac{2m}{\log_2 M} \tag{2.2}$$

As an example, the 1-of-2 and 1-of-4 codes with an identical code rate (R=0.5) both need two transitions to transmit the hot bit, which, however, carries one-bit and two-bit data respectively. Therefore, the average number of transitions per bit per transaction of 1-of-4 codes is half of the number of 1-of-2 codes, indicating 1-of-4 codes are more energy efficient than 1-of-2 codes. It can be found from Table 2.3 that m-of-n codes (m$\geq$2) usually have a higher code rate than 1-of-n codes, but fewer transitions per bit per transaction than 1-of-2 codes, reflecting their coding and energy efficiency. However, m-of-n codes may bring a large encoding, decoding and completion detection (Section 2.1.6) overhead in practical implementations [BTEF03]. Considering these metrics and the design cost, 1-of-4 appears to be a promising encoding [MAMN08], which is used with the 4-phase handshake protocol in the implemented QDI NoCs (Chapter 7).

2.1.6 Completion detector and QDI pipeline model

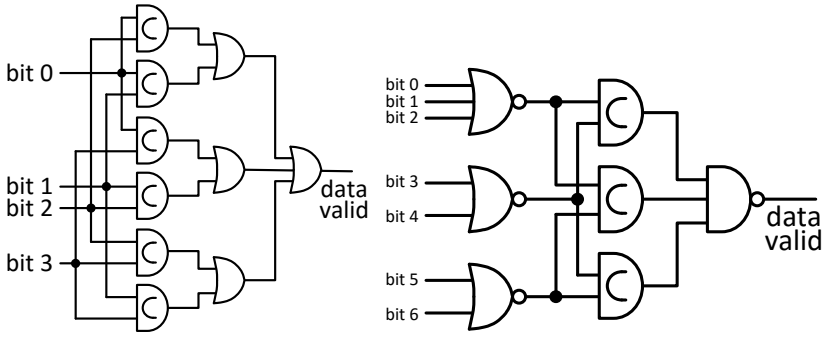
In DI or QDI circuits, without the clock signal as the timing reference, a Completion Detector (CD) is required at the receiver to determine whether a valid code word has arrived or not [SF01]. The timing information is implicit in transmitting DI codes (so that no extra *request* signals are needed as Figure 2.5 shows).

Figure 2.9 illustrates the implementation of CDs for 1-of-2 and 1-of-4 channels to detect a 4-bit binary datum. A 2-input C-element tree is used to construct a multi-input C-element. To deliver the same width binary data the completion detection tree is one level deeper for 1-of-2 codes than for 1-of-4 codes. Figure 2.10 shows CDs for complete 2-of-4 codes and incomplete 2-of-7 codes (Table 2.1), which reflects that omitting some codewords in m-of-n encodings may save a lot of logic in CDs. The encoding and decoding cost of m-of-n codes at the sender and receiver is usually much larger than for 1-of-n codes [BTEF03]. The ease of completion detection and simple encoding/decoding provide the main advantage of 1-of-n codes over the others.



(a) 1-of-2 (b) 1-of-4

Figure 2.9: Completion Detectors for 1-of-n codes



(a) 2-of-4 (b) Incomplete 2-of-7 [BTEF03]

Figure 2.10: Completion Detectors for m-of-n codes

This thesis uses 4-phase, 1-of-n as the basic asynchronous protocol. Data is encoded as DI codes which may contain one or multiple parallel 1-of-n symbols. A data word with N 1-of-n symbols can be classified into the following three sets:

- **Spacer**: all N symbols are spacers (all '0's).
- **Incomplete data**: this is the intermediate state between a spacer and a complete data word where some of the N symbols are 1-of-n while the others are spacers.
- **Complete data**: all N symbols are 1-of-n symbols.

Figure 2.11 shows the normal state transition of the transmitting data under the 4-phase handshake protocol. The data word in a pipeline stage experiences a "*spacer* $\rightarrow$ *incomplete data* $\rightarrow$ *complete data* $\rightarrow$ *incomplete data* $\rightarrow$ *spacer*" process. Corresponding to the data, a data channel may comprise one or more 1-of-n channels, each of which is responsible for transferring a 1-of-n codeword. As an example, Figure 2.12 shows a basic 4-phase, 1-of-n QDI pipeline with N parallel 1-of-n channels.

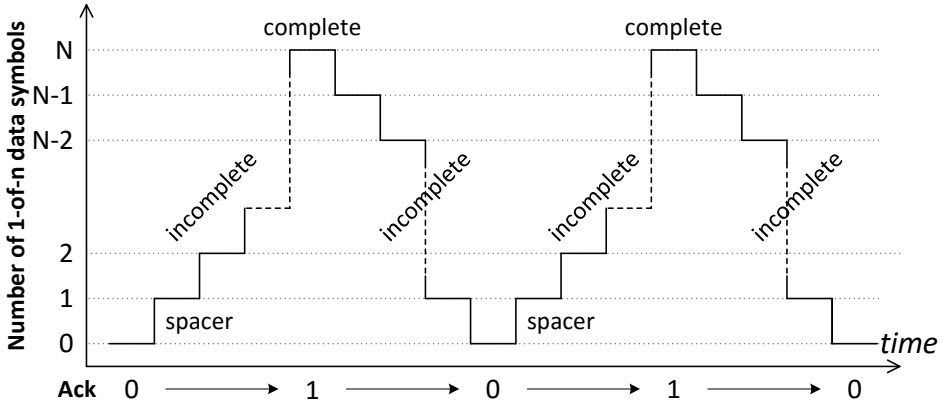


Figure 2.11: State transitions of a 4-phase QDI pipeline stage

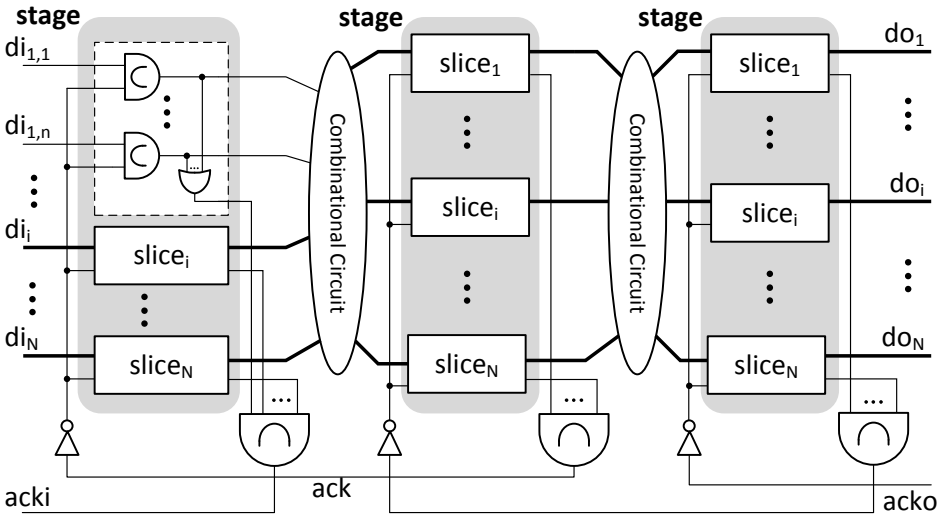


Figure 2.12: A 4-phase 1-of-n QDI pipeline

Each channel contain *n* data wires. To latch incoming data, each pipeline stage is divided into *N* 1-of-n slices, each of which is an asynchronous latch (or half buffer latch) built from *n* C-elements that can store one 1-of-n symbol [SF01]. An OR-gate acting as a Completion Detector (CD) registers the arrival of a 1-of-n symbol. To synchronize all slices of a pipeline stage, their CDs are connected to a multi-input C-element producing a common acknowledge signal (*ack*) to the preceding stage. A rising *ack* indicates a complete data symbol has been accepted by the current pipeline stage, and spacers are allowed to enter into the previous pipeline stage. A falling *ack* indicates all the symbols in a stage are just reset to spacers. This pipeline model is the basis for studying the impact and management of fault-tolerance in Chapter 3.

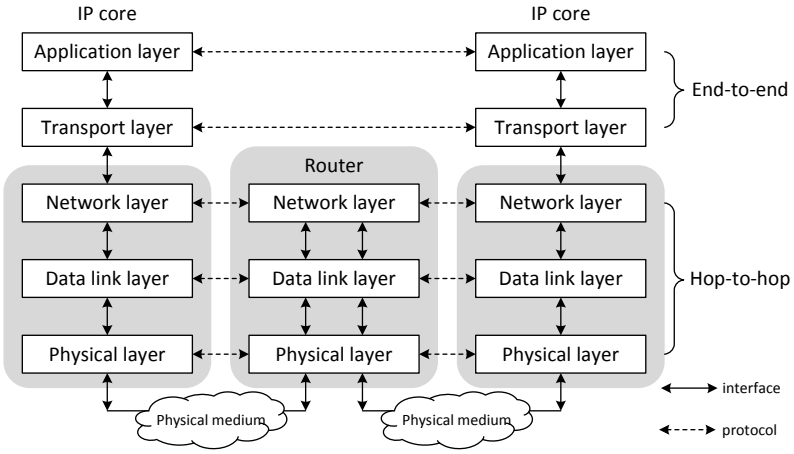


Figure 2.13: A modified OSI reference model [TW10]

2.2 Networks-on-Chip

2.2.1 Introduction

Multi-core techniques are being widely used in both general-purpose microprocessors and application-specific System-on-Chip (SoC), resulting in Chip Multi-Processors (CMP) or Multi-Processor System-on-Chip (MPSoC) [WJM08, BDM09]. Networks-on-Chip (NoCs), or on-chip networks, have been proposed as a scalable and efficient on-chip interconnection fabric to support the communication of large-scale multi-core systems [DT01, DMB06, EJP09]. This research uses NoCs as the "backbone" to discuss fault-tolerant techniques.

2.2.2 Network layer model

As Figure 2.13 shows, a NoC structure can be abstracted to five protocol layers borrowed from the modified Open Systems Interconnection (OSI) model used in traditional networks [TW10]. This research presents fault-tolerant NoC designs and demonstrates their implementation, involving the bottom three layers above the physical medium, which are the physical layer, the data link layer and the network layer:

- **Physical layer** is the layer above the physical medium, which transmits raw bit streams and specifies the electrical/physical properties of the connection.
- **Data link layer** groups bit streams into small blocks to provide reliable hop-to-hop transmission over the physical medium. Flow control methods, which decide the allocation of network resources [DT03], are defined in this layer.

- **Network layer** decides the path by which a packet reaches its destination, where the routing lies.

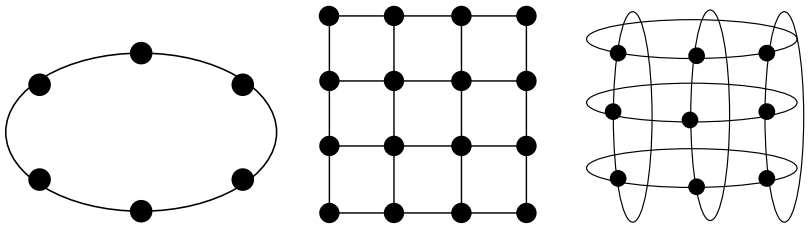
The other two upper layers, including the transport layer which deals with the packet loss or packet reordering to ensure reliable end-to-end communication, and the highest application layer which defines how application programs use the network, are out of the scope of this research.

In terms of fault-tolerance studied in this research, a fault happening on the physical medium may first manifest in the bottom "physical" layer. Without using fault-tolerant techniques, the resulting error may show up in upper layers. Adding fault-tolerance capability to different layers to avoid the faults appearing in the next higher layer is necessary to reduce the fault impact on the whole network. In the literature, it is common to co-manage multiple layers to provide systematic protection, so as to increase the reliability of the network [YA10, YA12]. In this research where NoCs are implemented using asynchronous circuits, a fault on the physical layer can corrupt the basic handshake protocol and lead to a system failure, in which case the protection from high layers (including the network layer and upper layers) cannot work effectively. Providing fault-tolerance from the bottom physical and data link layers is necessary to keep the NoC working in a faulty environment, which is the objective of this research.

2.2.3 Network topology

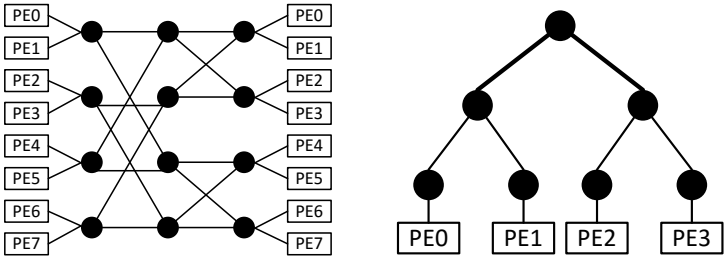
Built from routers and links, a NoC structures wires into a network, forming a certain *topology*. The topology affects the number of alternative paths between routers, the *hop* (or router) count that a message needs to traverse and the length of the wires between routers. It has a significant effect on the network traffic, latency and power consumption. Network topologies can be classified as either *direct* or *indirect* [EJP09].

- In *direct* networks, each Processing Element (PE) – which is usually an Intellectual Property (IP) core – is associated with a router which acts as both a network switch and a source or destination of the traffic. *Ring*, *mesh* and *torus* are all direct networks, as Figure 2.14 shows. A *ring* (Figure 2.14a) is the simplest topology, where the number of links at each router (i.e. the topology *degree*) is two. The distance between two nodes can be measured by the fewest hops that a packet needs to traverse. The maximum distance of a bidirectional ring network is $\lfloor N/2 \rfloor$ where N is the PE count. *Mesh* belongs to the most widely used topologies [EJP09]. Figure 2.14b shows a 4×4 2-Dimensional (2D) mesh NoC.



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

Figure 2.14: Direct network topologies



(a) Butterfly (b) Binary tree

Figure 2.15: Indirect network topologies

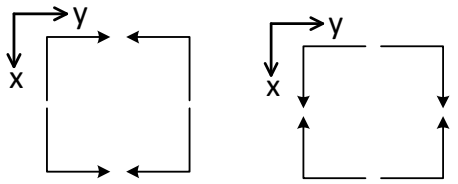
Its topology degree is not uniform. The maximum distance of an $m \times n$ 2D-mesh is $(m+n-2)$.

- In *indirect* networks, nodes are differentiated into terminal and intermediate nodes. Terminal nodes are routers connected to PEs while intermediate nodes are routers which switch the traffic between terminal nodes. Butterflies and binary trees are indirect topologies, as Figure 2.15 shows. By customizing the topology, indirect NoCs can satisfy specific communication requirements, providing an effective support for heterogeneous MPSoCs [EJP09].

The topology affects the network fault-tolerance to some degree. For different topologies, there may be a different number of paths between two network nodes. If one path between two nodes is defective, other paths can still be used to deliver the fault-free data. Thus topologies can be used along with routings to support fault-tolerance. As one of the most used topologies [EJP09], a 2D-*mesh* is used in this research to implement the asynchronous NoC.

2.2.4 Routing algorithms

Routing is a fundamental concept in the network layer, determining which path a packet will traverse from the source node to its destination [DT01, DMB06, EJP09]. A



(a) XY-DOR (b) YX-DOR

Figure 2.16: Dimension-Ordered Routings

good routing algorithm can make network resources (i.e. routers and links) efficiently utilised without causing severe traffic congestion or deadlocks. Routing algorithms can be generally classified into *oblivious* and *adaptive* depending on if the present network state is taken into consideration by the router when deciding the path to be taken.

If the routing is independent (i.e. the router is unaware) of the network state, it is an *oblivious* routing, which can be further classified into *deterministic* and non-*deterministic* routing. As the network condition is not considered, oblivious routing algorithms can be vulnerable to traffic congestion if the network traffic is not balanced.

- Deterministic routing uses fixed communication paths between source and destination nodes. One of the representatives is Dimension-Ordered Routing (DOR) where packets go through the network dimension-by-dimension. In a 2D-mesh, if packets firstly go along the X-dimension and then the Y-dimension, the routing is XY-DOR (Figure 2.16a). If packets traverse along the Y-dimension first, it is a YX-DOR (Figure 2.16b). Due to its simplicity, DOR has been widely used in existing NoC designs [DGK09, SE11b].
- Non-deterministic routings have multiple paths between the source and destination. One well-known non-deterministic routing is the minimal oblivious routing randomly choosing one shortest path from all possible paths at each hop [DT03].

Adaptive routing algorithms are aware of the present network status. They can route packets according to the network traffic or fault condition, so that there are multiple alternative paths between pairs of the source and destination routers. They can select one of these paths to avoid network contention. According to whether there are any routing restrictions, adaptive routings can be further classified into *fully* and *partially* ones. With regard to faults, under the precondition that the faulty component is detected and located so that the neighbouring routers are probably aware of this fault condition, adaptive routings (or "fault-tolerant routings" in the literature on fault tolerance [IY11]) can detour the defective or fault-prone links or routers, thus providing a better fault-tolerance capability than oblivious routings.

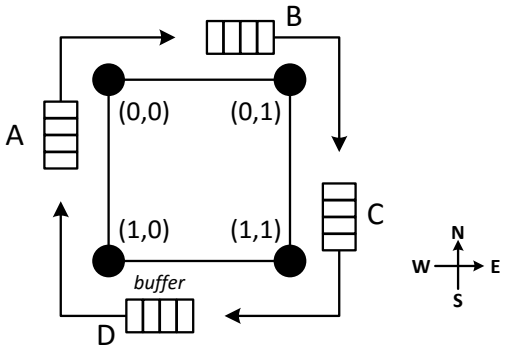


Figure 2.17: Network-layer deadlock due to cyclic dependence of packets

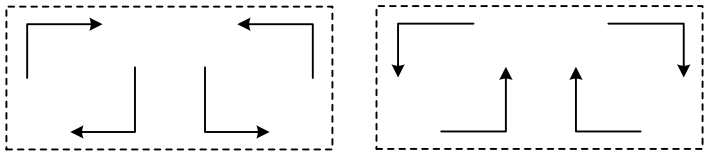
2.2.5 Deadlock avoidance and recovery

Depending on its design, NoC may suffer from network-layer deadlocks when its load is close to or beyond the saturation point [Lee07, MRLD01, ADMX+11]. This deadlock occurs in the network layer when packets cannot move forward since they all hold and request network resources in a cyclic fashion. This could be caused by unrestricted routing algorithms or limited network resources [ADMX+11]. As a result, the network has to face a significant performance degradation or even system failure.

Figure 2.17 illustrates an example of this network-layer deadlock. Packet A enters into router (0,0) from the south input port and keeps waiting for B to release router (0,1). Packet B entering (0,1) from the west input is waiting for packet C leaving router (1,1). Packet C cannot proceed to (1,0) due to the blocked D while packet D is waiting for the release of the south input of router (0,0). No packets can move forward and all packets have to wait for each other to leave.

In a fault-free environment, guaranteeing deadlock freedom of the NoC is an essential topic in the network layer. Two strategies have been proposed to deal with the deadlock: deadlock *avoidance* and deadlock *recovery* [Lee07, MRLD01, ADMX+11].

- Deadlock avoidance has been extensively used in NoC designs where packet routing or flow control methods are restricted in a way to prevent the cyclic dependence from happening, so that the whole network can be deadlock-free in a fault-free environment. The well-known turn models (Figure 2.18) prohibiting some turns belong to the simplest avoidance methods, while *virtual channels* can be used to avoid the deadlock by specifying the escape channel [DT01]. These techniques could limit the routing adaptivity, increase network latency and decrease its throughput.



(a) XY-DOR (b) YX-DOR

Figure 2.18: Dimension-Ordered Routing with some turns banned

- Deadlock recovery supports fully-adaptive routing so that the network performance is less affected. Thus network-layer deadlock may happen but the network function can be recovered after detecting the deadlock. Differentiating the real deadlock from congestion at runtime is the main challenge. The recovery cost may be large when deadlock happens. Considering the fact that deadlock could happen rarely if the network is restricted below its saturation point, deadlock recovery is often more attractive due to its lower hardware cost compared with the deadlock avoidance [ADMX+11].

In a fault environment, faults may change the address information or other control bits in a packet, in which case deadlock avoidance may be invalidated. As an example, if a packet following XY-DOR routing has reached the Y-dimension in a 2D-XY mesh NoC, a fault may mutate the packet address information, according to which the packet must turn to the X-dimension again, which is forbidden under XY-DOR routing. As a result, this packet may be blocked at one router to wait for an X-turn, which is a kind of network-layer blockage due to restricted routings. This can be resolved by careful router designs [SE11b]. When it comes to asynchronous or QDI NoCs, a fault could destroy the handshake protocol and lead to a *fault-caused physical-layer* deadlock, which is different from the network-layer one. Traditional deadlock avoidance and recovery techniques cannot manage this deadlock due to the broken handshake. This fault-caused physical-layer deadlock is the main point of this research, which will be discussed in Chapter 3.

2.2.6 Switching techniques

Switching techniques decide how network resources, including the channel bandwidth and input/output buffers in a router, are allocated to messages and how messages traverse the network. For a NoC, before a *message* is injected to the network, it is usually first disassembled into *packets*, which can be further divided into multiple, fixed-length

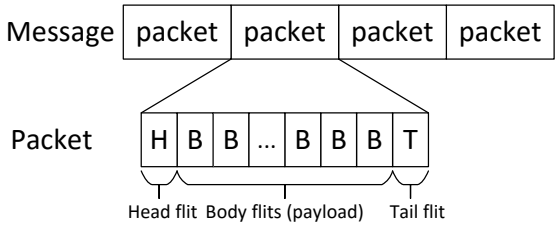


Figure 2.19: Message, packet and flit in NoCs

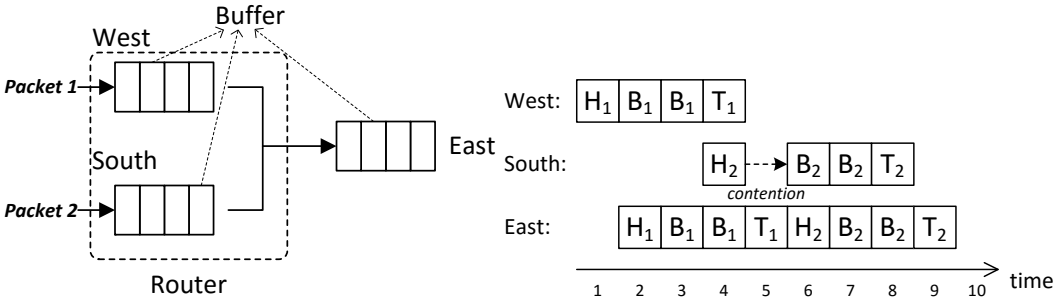
flits (as a subdivision of a packet, a flit is the basic unit of bandwidth and storage allocation used by most flow control methods [DT03]), including a *head* flit containing the address information, *body* flits containing the data information (or payload) and a *tail* flit indicating the end of a packet, as Figure 2.19 shows. Correspondingly, network resources can be organized in different sizes. There are two basic switching techniques: *circuit switching* and *packet switching* [DT03, DMB06].

Circuit switching

In *circuit switching*, a fixed physical path, which is named a *virtual circuit* [DMB06], is first set up between the source and the destination node. Then all flits of the message are transmitted along the allocated path, which will be released at the end of the message transmission. Circuit switching is connection-oriented, so that the transmission delay is generally predictable, providing Guaranteed Services (GS) [DT03]. Buffers in routers are not mandatory for circuit switched networks, saving the area and power. Due to the possible long path set-up and acknowledge time, circuit switching may suffer from poor latency. The long reserved path can also block other messages from reserving their paths, making the network congestion-prone.

Packet switching

By contrast, *packet switching* does not reserve a physical path before the message transmission. Packets can be transmitted to the destination through different routes, improving the link utilization compared to the circuit switching. Without the set-up phase, multiple packets may try to use a link at the same time, leading to *contention* that only one packet can be served at one time while the others have to wait the link to be released. This may result in variable packet transmission delays, making it difficult to guarantee the Quality-of-Service (QoS) [DMB06]. *Store-and-forward* is the simplest form of packet switching. It stores an entire packet in each router before forwarding



(a) Abstracted router (b) Packet transmission (1-flit long buffer)

Figure 2.20: Wormhole switching

any part of the packet, so that it requires a router buffer long enough to hold the longest packet. This could also cause a long hop latency. To reduce the latency, *virtual cut-through* switching permits the progress of a packet before one entire packet is received by the current router buffer, but still requires that the succeeding router buffer has sufficient storage for an entire packet.

Different from the *store-and-forward* and *virtual cut-through* switching which allocates the network resources in a packet granularity, *wormhole switching* is a widely used flit-based technique, which permits the progress of a flit as long as the subsequent router has enough storage space for it. It reduces the required buffer size from packet to flit level. Figure 2.20 illustrates the wormhole switching process which requires only 1-flit long router buffers. *Packet* 1 gets forwarded to the free east output flit-by-flit. Due to contention, only the head flit of *packet* 2 arrives at the south buffer and gets output after the east link is freed. Then the left flits start transmitting. In a *wormhole switching* network, once the head flit is blocked at one router, the subsequent flits will be stalled at a track of routers which cannot be allocated to any other packets until these occupied network resources are released. This may lead to a poor link efficiency and long transmission latency when the network load is heavy.

Accompanied with different switching techniques, buffer flow control methods are required: the upstream router should be able to know the availability of buffers at the downstream router so that it can decide when to send out the flit to avoid buffer overflow. This can be implemented using *back-pressure* mechanisms to inform the preceding router of stopping transmitting flits when the buffer of current router is full. A credit-based flow control can implement this back-pressure by sending a credit to preceding router only if there is available buffer space in current router [DT03]. A counter counts the received flits to decide if the router buffer is full. In an asynchronous NoC, the handshake protocol provides a natural back-pressure mechanism to manage buffer

resources in a network. A flit can progress only if the succeeding router acknowledges with available buffer space, so no extra buffer control is required.

Improving transmission efficiency

Multiplexing techniques, including Time Division Multiplexing (TDM) and Spatial Division Multiplexing (SDM) techniques [DT03, DMB06], have been proposed to improve the transmission efficiency of the network.

As a popular TDM method, *Virtual Channels* (VCs) [DS87, DT01] have been used to improve the network performance. Multiple independent input buffers share one physical link in a time multiplexed manner. In other word, multiple virtual channels for different packets can be time-multiplexed onto one physical link. When one packet or flit is blocked in one virtual channel (or input buffer), the others can still traverse this physical link using others, improving the link efficiency. Figure 2.21 depicts an example that two isolated input buffers share one physical link. Using a virtual channel as an exclusive *escape* channel, the link stalled due to the dependence cycle of transmitting packets formed in other virtual channels can be released, avoiding the network-layer deadlock [DS87]. VCs have been used in asynchronous NoC designs to support Quality-of-Service (QoS) [FF04, TVC10].

Differently, multiple narrow independent sub-links can be multiplexed onto a physical link (Figure 2.22), which is called Spatial Division Multiplexing (SDM) [DMB06,

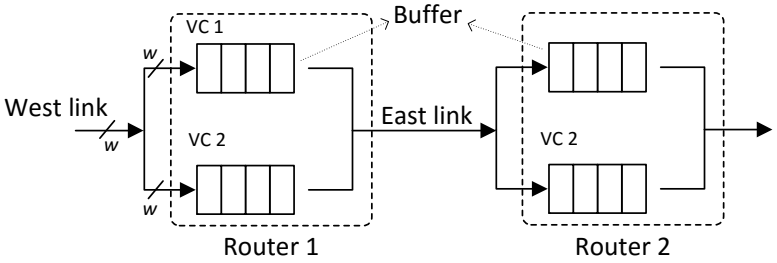


Figure 2.21: Virtual Channel router connection

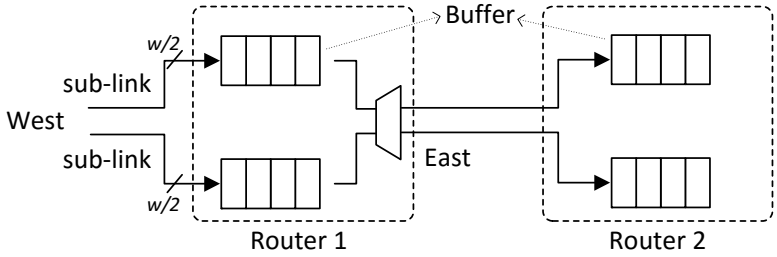


Figure 2.22: Spatial Division Multiplexing router connection

LMV+08, SE11b]. In other words, each link can be divided into several physically independent narrower sub-links, so that multiple messages could traverse the same link concurrently using different sub-links. Due to the narrowed width of each sub-link, SDM can significantly increase the transmission delay of a single packet. However, it has been demonstrated that SDM shows better parallelism with an increased overall throughput [LMV+08, SE11b]. SDM can be used with other message-based, packet-based or flit-based flow control methods to improve the network performance.

This research uses wormhole switching with SDM to design fault-tolerant QDI NoCs. Due to the simple implementation, wormhole belongs to the most popular switching techniques used in existing NoCs [DMB06]. Using SDM, a permanent fault on the link will pollute only one sub-link while the other sub-links of the same link are healthy. By isolating the defective sub-link, the other fault-free ones can still be used to transfer packets, shows a promising fault-tolerance nature. In a QDI NoC, if a fault destroys the handshake process and creates a faulty acknowledge informing the pre-fault router of unavailable buffers, the remaining flits in the packet path upstream of the fault cannot progress, leading to a *physical-layer* deadlock different from the network-layer one caused by cyclic dependence of packet transmission (Section 2.2.5). As a result, all remaining flits of the packet upstream of the fault are fault-free but cannot progress, while the fault may transmit to the destination along the built path and the downstream path will not be released without receiving a tail flit (blocked before the fault). Detection of this fault-caused physical-layer deadlock and recovery of the network from broken handshakes is an unexplored challenge faced by QDI NoCs, which is the main contribution of this research (Chapters 5 and 6).

2.2.7 Wormhole router implementation

Figure 2.23 depicts a generic wormhole router. Due to its simple implementation, *wormhole switching* is dominant in current NoC designs [EJP09]. This router is used in a 2D-mesh network. It consists of five input/output ports, four (South, West, North, East) of which are used to communicate with adjacent routers and the fifth connects to the Local PE. A central *crossbar* in the router provides multiple connection paths from input to output ports. *Buffers* at the input/output ports and the crossbar construct the data path through the router. The major components of the router's control logic comprise a *Routing Computation* (RC) unit, a *Buffer Controller* (BC) and a *Switch Allocator* (SA). The RC unit is responsible for computing a routing request according to the address information in the head flit of a packet to compete for an output port.

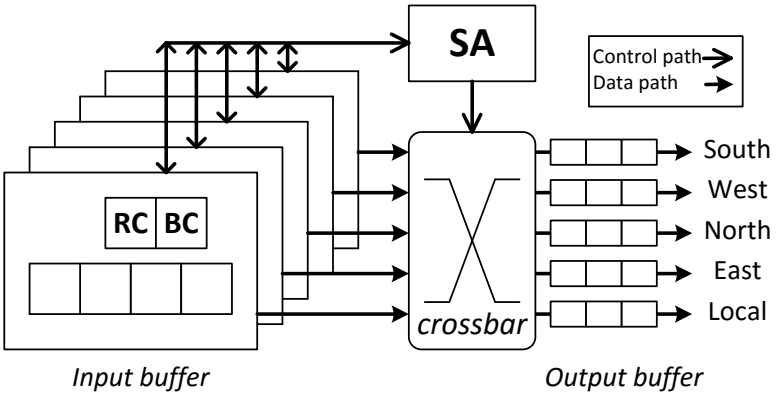


Figure 2.23: A generic router structure

The BC resides at each input buffer and regulates the incoming flit flow sequence to the router according to the used *wormhole switching* method. (This BC can be a virtual channel allocator in a VC router [Dal90, DT01]). The core control unit is a SA, which receives routing requests from different inputs and grants one (or multiple) of them to available output(s). It controls the connection between the input and output ports through the crossbar.

Therefore, when a packet arrives at the router, its head flit is first blocked in the input buffer (Packet 2 at router R2 in Figure 2.24). The routing computation unit computes the output direction and sends the request to the switch allocator. If the requesting output is busy, the head flit of the packet is blocked in the input and waits. A packet may straddle other routers, as Figure 2.24 shows. (This brings a great challenge faced by a faulty, deadlocked QDI NoC, whose recovery needs to release all healthy routers and links reserved by the faulty packet. The network recovery will be resolved in Chapter 6). Otherwise, since there may be multiple input packets requesting the same output direction, the allocator acts as an arbiter and grants one of them to proceed. The other packets keep waiting for this output to be idle and compete for it again.

2.3 Asynchronous Networks-on-Chip

2.3.1 Taxonomy of asynchronous NoCs

Applying asynchronous techniques to NoCs, asynchronous NoCs benefit from the clockless nature and are competitive candidates with their synchronous counterparts for current multi-core or many-core systems [Spa07, TVC10, CBL+10, Pon12]. According to different implementations, asynchronous NoCs can be generally classified

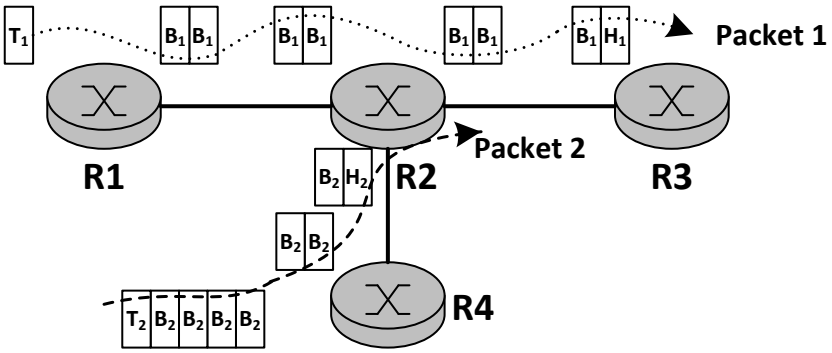
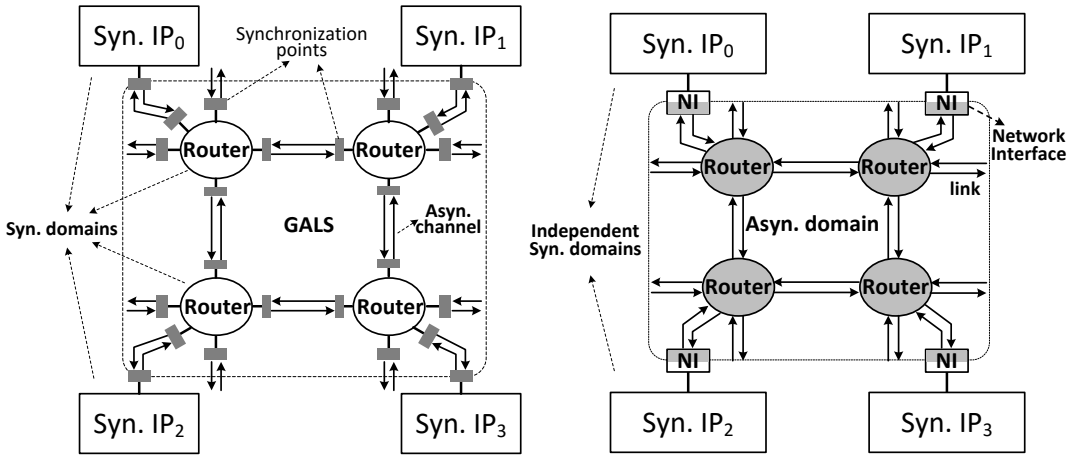


Figure 2.24: Flit flows under the wormhole switching



(a) GALS NoC (b) Fully-asynchronous NoC

Figure 2.25: Asynchronous NoCs

into Globally Asynchronous Locally Synchronous (GALS) and fully-asynchronous NoCs (Figure 2.25) [MVF00, KGGV07, Spa07, Pon12].

GALS NoCs introduce multiple different synchronous domains to the network. As an implementation example, Figure 2.25a illustrates a GALS NoC where routers have their own clock domains and inter-router channels are implemented using handshake protocols. Both the NoC and the whole system, including independent PEs, are GALS. Depending on the practical design requirements, the router and its attached PEs may or may not use the same local clock. Synchronization points, or sync./async. interfaces are mandatory between the synchronous and asynchronous domains, where the metastability problem would unavoidably reside. The sync./async. interfaces can be implemented using synchronizers [Spa07, Gin11], pausible-clock generators [MM07] or asynchronous First-In-First-Out (FIFO) buffers [KGGV07] to resolve the metastability problem.

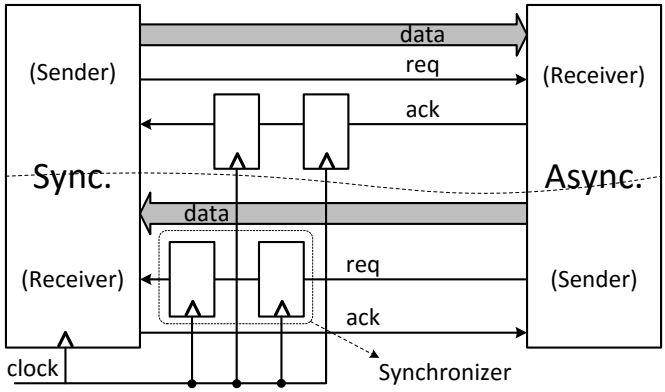


Figure 2.26: Boundary synchronizers [SE11b]

Figure 2.26 depicts one possible implementation using two-flip-flop synchronizers. For a sync./async. interface, synchronization is required only when signals enter into the synchronous domain [KGGV07]. Signals entering into the asynchronous domain can be treated as an event triggering the handshake process, which does not require synchronization [SE11b, Spa07]. Therefore, as Figure 2.26 shows, if the *data* goes to the asynchronous domain, only the backward *ack* signal needs to be synchronized; in the other way around, the request signal going to the synchronous domain should be synchronized. The synchronizer could reduce the metastability issue to an acceptable level and increase the Mean Time Between Failures (MTBF), with a price of decreased throughput caused by the induced extra clock cycle(s) [Kin07].

An entire fully-asynchronous NoC is possible where both routers and links are asynchronous circuits controlled by handshake protocols, so that it can fully utilise the asynchronous features. PEs, which are usually synchronous Intellectual Property (IP) cores, are connected to the asynchronous network through Network Interfaces (NIs). Figure 2.25b depicts the GALS system structured by a fully-asynchronous NoC. Attached synchronous IP cores may have their own clock domains isolated by the asynchronous network, while the internal communication of the network does not require synchronization, simplifying the chip-level timing closure. NIs implement the conversion between asynchronous protocols and synchronous clock domains. They are the only synchronization points. This framework allows an easy assembly of complicated GALS SoCs comprising independent synchronous IP cores. If PEs are implemented using the same asynchronous protocol, they can be connected to the asynchronous NoC easily through the handshake interface. Another advantage of the fully-asynchronous NoC is the timing-robustness if it is designed in a QDI fashion. However, without the clock signal, implementing complicated network protocols on the fully-asynchronous

NoC and testing the network can be more difficult than that on the synchronous or GALS one and the resulting area can be large [Spa07, DGK09]. Their fault-tolerance is another challenge which has not been thoroughly studied; this research will concentrate on the QDI (fully-asynchronous) NoCs and will study their fault-tolerance.

2.3.2 Previous asynchronous NoCs

Many asynchronous on-chip interconnection and NoC designs were proposed in recent years. This section reviews several representatives and describes the basic asynchronous and network protocols they used.

Chain [BF02] belongs to the first generation of asynchronous on-chip interconnection using the 4-phase handshake protocol. 1-of-4 codes are used to encode data while one End-of-Packet (EoP) signal is used to separate consecutive packets. Thus each channel contains one acknowledge wire, four data wires and one EoP wire. Input and output buffers of the router are pipelined using the C-element built half-buffer pipeline stages (Figure 2.12). Source routing (where the entire packet route is embedded in the packet header at the source terminal [DT03]) is used so that synchronous IP cores have a full knowledge of the network topology. A routing control module samples the address information in each packet header, leading the packet in the right direction. Chain provides a feasible asynchronous communication framework and can be used to create networks of different topologies with different network protocols.

A QDI NoC [FF04] was proposed as an early effort to support Quality-of-Service (QoS) in fully asynchronous NoCs. Network traffic can be generally divided into two classes: connection-oriented Guaranteed Services (GS) and connection-less Best-Effort (BE) classes [DT03]. GS requires that network resources are reserved during the packet transmission, ensuring that the network will satisfy some specific performance requirement even in the worst-case traffic scenarios. In contrast, BE does not require reserved resources and the network just makes its effort to transfer packets to the destination. The price is an unpredictable, long transmission in the worst case. Supporting different service levels could increase the link utilization efficiency and highly improve the network performance. The implemented QoS router follows the structure of a synchronous Virtual Channel (VC) router shown in Figure 2.27. Four VCs are used at each physical link to support four QoS levels. The VC allocator is a static priority arbiter which always allocates the output port to the requesting VC with the highest service level. The 4-phase, 1-of-4 QoS NoC is implemented as a 2D-mesh using DOR and wormhole switching as the network protocol.

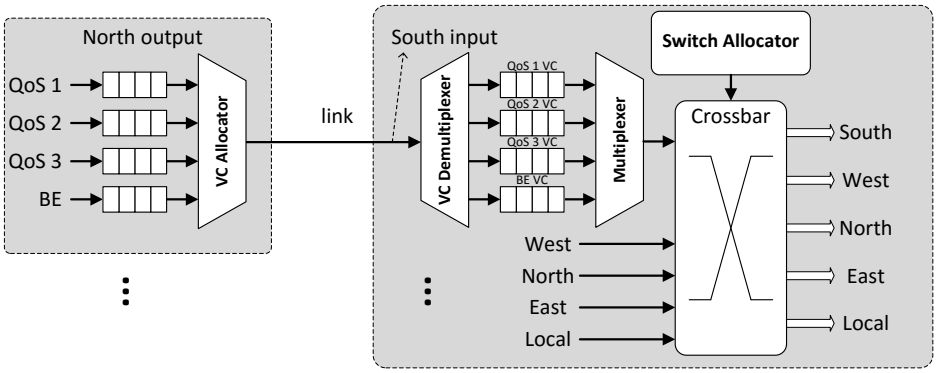


Figure 2.27: QoS-supporting QDI router based on Virtual Channels

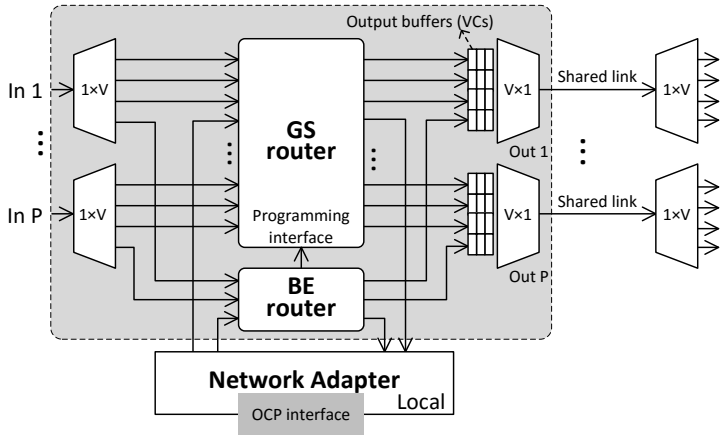


Figure 2.28: MANGO NoC router

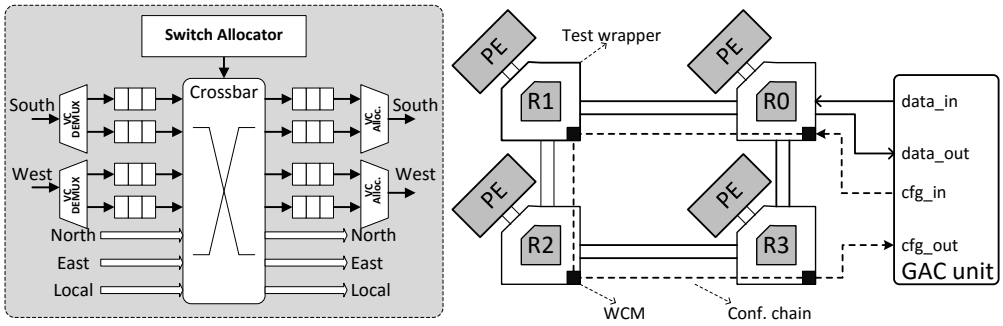
Asynchronous Scalable Packet-switching Integrated Network (ASPIN) [SG08] is a fully-asynchronous NoC based on a scalable programmable interconnection network (SPIN) [ACG+03]. It uses a 2D-mesh topology and XY-DOR routing. The 4-phase, bundled-data protocol is used inside routers while links use 1-of-2 encodings to guarantee delay insensitivity. Pipeline stages are inserted on links to alleviate the long wire delay. Asynchronous FIFOs are used to alleviate the metastability between the asynchronous network and the synchronous IP domains.

A Message-passing Asynchronous NoC providing Guaranteed services over Open core protocol interfaces (MANGO) [BS05, BS06] was proposed, supporting circuit-switched GS and packet-switched BE routing. Figure 2.28 depicts the structure of a MANGO router, whose central switch consists of a GS router and a BE router, responsible for corresponding services. In the implemented router with five ports, each local port supports four GS service levels and one BE service. The other ports use eight VC buffers to implement seven GS service levels and one BE service. VC buffers are

put at the router output rather than the input to support different service levels. Using standard Open Core Protocol (OCP) sockets at network interfaces (or network adapters in Figure 2.28), MANGO permits any IP cores compliant with the OCP to be plugged into the system. 4-phase, bundled-data protocol is used to implement the router to minimise the area, but losing robustness to delay variations without applying the QDI manner. Inter-router links use 2-phase, 1-of-2 protocol to achieve timing-robustness.

An asynchronous Quality-of-Service NoC (QNoC) [DGK09] was proposed to support multiple communication requirements. It is implemented as a 2D-mesh using XY-DOR source routing and the wormhole switching. Each input/output port has multiple VCs to support different Service Levels (SLs). Firstly a head flit is routed to an output VC (VC-OP) according to its address information, SL and VC bits. Two-level arbitration is required at the output side to decide which flit will be granted the output link: the first level is a VC arbiter, implemented as an M-way MUTEX-NET arbiter [DGK09], which selects one flit from all VC-OPs of the same SL; the second level is an SL arbiter, implemented as a static priority arbiter, determining which flit will be sent to the output link according to its priority. 4-phase, bundled-data protocol is used inside the QNoC router so it is not a QDI design. Due to the complicated VC designs, the area overhead can be very large. No fault-tolerance is supported.

ANoC [TVC10] is a fully-asynchronous NoC framework implemented using 4-phase, 1-of-4 asynchronous protocol. It employs source routing and wormhole switching. In the given 2D-mesh instance, each router has five input/output ports and each port has two VCs to provide two service levels. Similar to the QoS router in Figure 2.27 [FF04], VC multiplexer at each input port is used to route flits from the physical link to corresponding VC queues. The switch allocator performs the arbitration between different directions within each VC first, then the VC allocator at the output side determines which VC uses the link (Figure 2.29a). The difference is the ANoC router removes the multiplexer arbitrating the input VCs, improving the network throughput. In addition, this ANoC employs a Design-for-Test (DfT) architecture [TTD$^+$09] detecting stuck-at faults on the network. As Figure 2.29b shows, each router is surrounded by a Test Wrapper, containing a Wrapper Control Module (WCM) controlling the test process. Test flows of the whole network are defined by an external Generator-Analyser-Controller (GAC) unit, which generates test vectors and configurations, and analyses the test results. The transmission of the test vectors reuses the network link while the test configurations are transmitted through a 2-bit configuration chain. It uses the property that stuck-at faults will stall the QDI circuit, and achieves



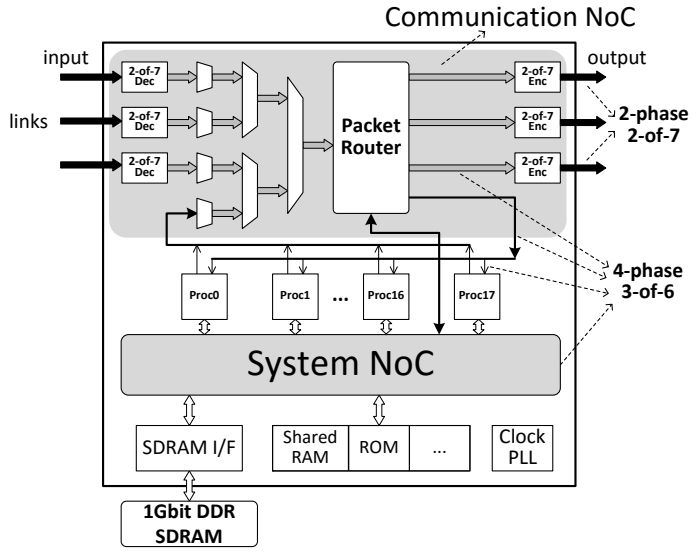
(a) Router structure (b) Design-for-Test architecture

Figure 2.29: ANoC architecture

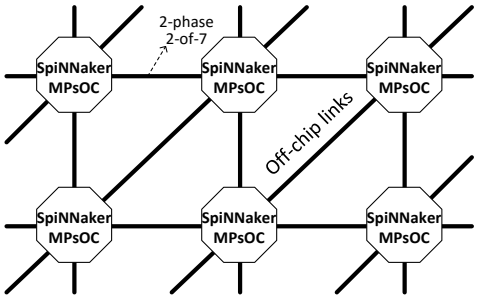
a test coverage of 99.86% per router in the case of a single stuck-at fault. However, it is not a runtime fault-tolerant design and cannot locate the defective component. No recovery mechanisms are provided.

Hermes-A [PMMC11] is an asynchronous NoC router where most of components use the 4-phase 1-of-2 asynchronous protocol except for the arbiter at each output port which uses bundled-data encoding. Thus an encoding controller is required to implement the conversion between bundled-data and 1-of-2 encodings. A 2D-mesh instance was implemented using the wormhole switching. A further version is Hermes-AA using west-first routing rather than the previous XY-routing to improve routing adaptivity [PMMC10].

SpiNNaker [PFT+07, PCD+11, FLP+13] is a massively-parallel computer system designed to model up to a billion spiking neurons in biological real time. It is built from around 60,000 nodes and each node is a MPSoC, as Figure 2.30a shows. The SpiNNaker system contains two kinds of NoCs: an off-chip Communication NoC used for on and off-chip interprocessor communication and a System NoC to handle the on-chip processor-to-memory/peripheral communication. Both of them are implemented as DI interconnection. As Figure 2.30b shows, a 2D toroidal triangular mesh is used to build the off-chip network, i.e. the Communication NoC. It uses 2-of-7 codes to encode transmitting data. 2-phase handshake protocol is employed rather than the 4-phase one to improve the communication speed. The asynchronous System NoC employs a crossbar topology, connecting eighteen synchronous ARM processing cores, one multicast router and many slaves on a SpiNNaker MPSoC. 4-phase, 3-of-6 encodings are used inside the chips. Therefore, the off-chip 2-of-7 codes under the 2-phase handshake should be transformed to the 3-of-6 codes using 4-phase logic when they enter into a SpiNNaker node through the interface. Under a faulty environment,



(a) SpiNNaker MPSoC



(b) Interconnected SpiNNaker Nodes

Figure 2.30: SpiNNaker MPSoC system

this protocol transformation at the inter-chip link interfaces may fail, which was studied [SFGP09, Shi10]. Transient faults were deemed more likely in the inter-chip links, so the 2-of-7 coding receivers were designed to tolerate most transient fault scenarios and possible deadlocks residing at the phase converter [SFGP09, Shi10]. This is different from this research, which studies the deadlock scenarios in general 4-phase QDI pipelines, as introduced in Chapter 3. Here, fault tolerance is extended to the on-chip network. In addition, it was observed from several faulty scenarios that a single-bit 4-phase dual-rail pipeline is free of deadlock but a multiple-bit one is not, without further mechanisms. No formal verification was given. Alternatively, this research systematically studies different fault effects in Chapter 3 and summarises deadlock patterns using formal proofs, which are further used to manage the fault-caused deadlock in Chapters 5 and 6. This work is new and belongs to the main contributions of this research.

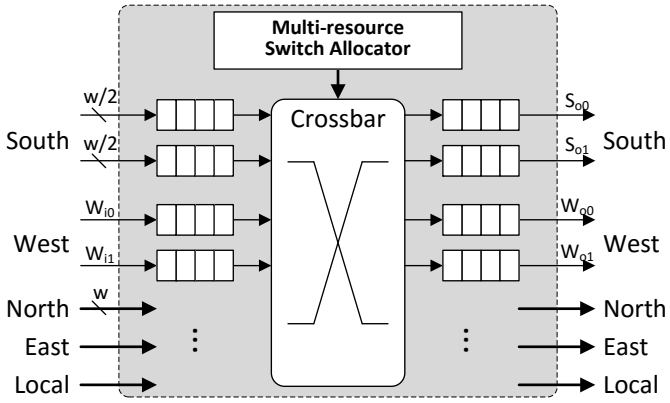


Figure 2.31: SDM NoC router

A fully-asynchronous QDI Spatial Division Multiplexing (SDM) NoC (Figure 2.31) was proposed to improve network throughput [SE11b, SE11a]. The link between routers is physically divided into several sub-links, which can be used to transmit different packets in parallel. It has been shown that the asynchronous SDM router shows better parallelism than VC routers. By using channel slicing techniques [SE11b, SE10b], the network throughput can be further improved. These techniques have been used to design 2D-mesh and Clos networks [SE11b, SE10a]. The 2D-mesh SDM NoC uses XY-DOR and wormhole switching as the basic network protocol. The 4-phase 1-of-4 protocol is employed to provide a QDI feature.

Summary

Table 2.4 summarises the above asynchronous NoC representatives. Existing asynchronous NoCs focus on either providing QoS support or improving the performance while most of them have no or very limited fault-tolerant capabilities.

- ANoC [TVC10] provides a DfT design which can detect off-line stuck-at faults. It is a defect-tolerant design [TM04, MG03, KK98] (Section 3.1) which is unsuitable for tolerating on-line faults. No recovery mechanism is proposed.
- Based on the SpiNNaker system, the fault-tolerance of inter-chip communication is studied [SFGP09, Shi10]. With the view that the deadlock scenario is much more severe than illegal data symbols, this work focuses on minimizing transient-fault-caused deadlocks rather than correcting data errors. A transient-fault-tolerant phase converter (between 2-phase and 4-phase) at the chip interface was studied, which is not general and cannot be migrated to the large number of existing 4-phase, 1-of-n QDI NoCs.

Table 2.4: Comparison of previous asynchronous NoCs

NoC	Asynchronous protocols	Topology	Routing	Flow control	QoS	QDI
Chain [BF02]	4-phase 1-of-4	Reg/Irreg	Source	P2P framework	No	Yes
QoS NoC [FF04]	4-phase 1-of-4	2D-mesh	DOR	WH, VCs	Yes	Yes
ASPIN [SG08]	4-phase; Router: bundled-data; Link: 1-of-2	2D-mesh (Reg)	XY-DOR	WH	No	No
MANGO [BS06]	Router: 4-phase bundled-data; Link: 2-phase 1-of-2	2D-mesh (Reg/Irreg)	Source	WH, VCs	Yes	No
QNoC [DGK09]	4-phase; Router: bundled-data; Link: 1-of-4	2D-mesh (Reg/Irreg)	XY-DOR Source	WH, VCs	Yes	No
ANoC [TVC10]	4-phase 1-of-4	2D-Mesh (Reg/Irreg)	Source	WH, VCs	Yes	Yes
Hermes [PMMC10, PMMC11]	4-phase, 1-of-2 and bundled-data	2D-mesh (Reg)	XY (west-first)	WH	No	No
SpiNNaker System NoC [PCD+11, FLP+13]	4-phase, 3-of-6	Crossbar-like	Routing table	Packet switching	No	No
SDM NoC [SE11b]	4-phase,1-of-4	2D-mesh (Clos, Reg/Irreg)	XY-DOR, Comb. circuits	WH, SDM, channel slicing	No	Yes

∗: Reg/Irreg: Regular/Irregular; WH: wormhole

- The SDM router [SE11b] has a mechanism to drop packets at the router input when the decoded 1-of-4 routing request signal is invalid due to the wrong address information in head flits. This invalid address information violates the used XY-DOR routing, which can be detected by the routing computation unit. Without dealing with data errors, the provided fault-tolerance of the SDM NoC is very limited.

It can also be found from Table 2.4 that asynchronous NoCs can be implemented using different asynchronous protocol combinations, such as 4-phase, 1-of-n routers and links [BF02,FF04,TVC10,SE11b], 4-phase, bundled-data routers with 1-of-n links [SG08, DGK09], 4-phase, bundled-data routers with 2-phase, 1-of-n links [BS06], and 4-phase, bundled-data/1-of-n hybrid routers with 1-of-n links [PMMC10, PMMC11]. Among these options, only the NoCs using 4-phase, 1-of-n are QDI, which can tolerate delay variations on both routers and links. Since the 4-phase, 1-of-n protocol has been widely adopted in previous QDI NoC designs and it can be implemented using simple circuits; this research uses 4-phase, 1-of-n as the basic asynchronous protocol in the implemented QDI NoCs and study their fault-tolerance. The baseline NoC employs the 2D-mesh topology, the XY-DOR routing and the wormhole switching. It should be noted that the proposed fault models and fault-tolerant techniques are general for 4-phase, 1-of-n QDI NoCs. They are independent of the upper-layer network protocols.

Therefore, the primary purpose of this research is to study the fault impact and propose general fault-tolerant techniques for asynchronous NoCs to tolerate different kind of faults, providing a holistic, efficient and resilient interconnection solution for future large-scale multi-core systems.

2.4 Summary

This chapter presented basic background knowledge of asynchronous circuits and designs and Networks-on-Chip (NoCs). State-of-the-art asynchronous NoC designs were introduced, from which it can be concluded that most existing asynchronous NoC designs are implemented using relatively simple asynchronous and network protocols to simplify the design complexity. Compared with synchronous NoCs, these asynchronous NoCs have no or very limited fault-tolerant capabilities. The following chapters will introduce the fault context and analyse the fault impact on asynchronous NoCs and then several fault-tolerant techniques are proposed to protect asynchronous NoCs from different network levels and different faults. The widely used 4-phase, 1-of-n asynchronous protocol is used in the studied asynchronous NoC. To make the analysis simple, the baseline NoC is structured as a 2D-mesh network and uses XY-DOR and wormhole switching as the basic network protocols.

Chapter 3

Faults and fault impact on QDI pipelines

Advancing semiconductor manufacturing technology permits the design of more sophisticated electronic systems, continuously increasing the number of transistors and wires integrated onto a single chip. Scaling effects, such as the reduced transistor dimensions, the decrease of critical charge[1], the increase of clock frequency and the increase of the power density, intensify the frailty of electronic devices to environmental variations, which also has a negative impact on long-term chip lifetime and consequently accelerates the occurrence of faults of the circuit [Bau01, Con03].

Fault-tolerance becomes an essential design objective for critical digital systems, especially those in highly specialized fields. It has been extensively studied in synchronous NoCs but rarely in asynchronous ones, including Quasi-Delay-Insensitive (QDI) NoCs. Without a timing reference, faults on QDI circuits can not only cause data errors, but also stall the handshake process, causing a physical-layer deadlock. Even a transient fault can disrupt the handshake process and deadlock the network. This fault-caused physical-layer deadlock proposes a new challenge for fault-tolerant asynchronous NoCs.

This chapter introduces different fault classifications and existing fault-tolerant techniques. 4-phase, 1-of-n QDI pipelines are modelled to study the impact of different faults, from which the fault characteristics can be abstracted. Besides data errors, the production and behaviour of fault-caused physical-layer deadlocks are thoroughly studied. By analysing the patterns shared by the physical-layer deadlocks caused by

[1] Critical charge, Q_{crit}, refers to the minimum charge that differentiates the state of a node, i.e. '1' and '0' [MW79]

different faults, a general deadlock management strategy is proposed to handle various fault scenarios.

3.1 Faults classification

In a computer system, which can be abstracted into multiple layers, a fault denotes the malfunction of a component in one specific layer. It can be a software fault happening in applications or operating systems, or a hardware fault due to radiation or wear-out faults so that the silicon chip malfunctions [Con03, Bor05]. In this thesis, faults refer to the malfunction of integrated circuits. Different faults may occur on gates or wires during the bit transmission and cause bit-flips of some signals.

According to their duration time, *faults* in integrated circuits can be classified into two general categories: *transient* and *permanent* faults [Con03]; *intermittent* faults are sometimes included as a third category. They can be taken as transient but occur repeatedly, or permanent if they last for a long enough period.

Figure 3.1 illustrates the relationship amongst three commonplace terms used in fault-tolerance literature: ***fault***, ***error*** and ***failure*** [Muk08]. *Errors* are manifestations of *faults*. In integrated circuits, *errors* are defined as captured *faults* by memory components (such as C-elements in asynchronous circuits [SF01]), so that they can also be classified into transient (or *soft*) and permanent (or *hard*). *Faults* are necessary to *errors* but not all *faults* cause *errors* since many of them will be masked during their propagation. If captured errors are not corrected, they may cause malfunction of the circuit or chip outputs, or even lead to a circuit *failure*, which may cause faults in upper layers of a computer system (such as the operating system level or the application software level) whose detection and correction may be more complex and expensive.

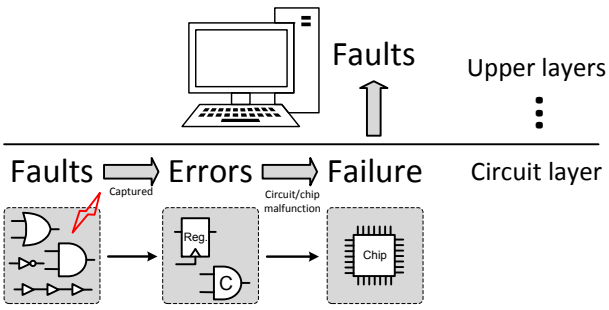


Figure 3.1: Relationship between *fault*, *error* and *failure*

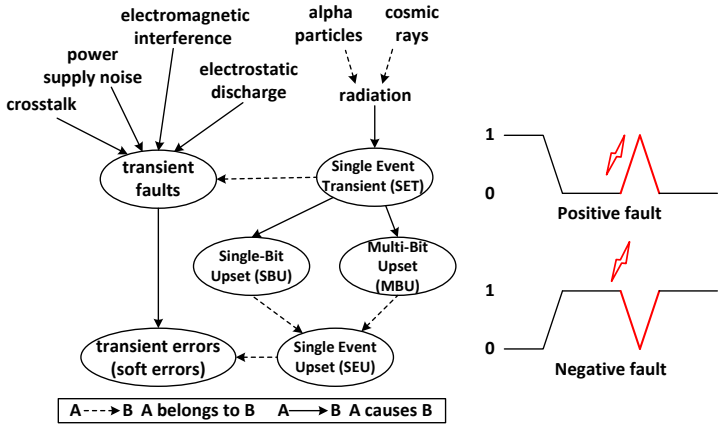


Figure 3.2: Sources of transient faults [SN96, Con03, KH04, MT03, Bau05]

Therefore, a multi-layer, fault-tolerant architecture protecting a computer system from the bottom circuit to the high software level is required to deal with different fault scenarios [Yu11]. This thesis concentrates on fault tolerance in the circuit level to reduce circuit errors and prevent a circuit *failure* from occurring.

3.1.1 Transient faults

Shrinking transistor dimensions, increasing clock frequency, increasing density of integrated circuits and decreasing critical charge, increase the sensitivity of electronic devices to environmental variations, largely increasing the occurrence of transient faults [Bau01, Con03]. As Figure 3.2 shows, transient faults can be provoked by many sources, including noise (such as crosstalk and power supply noise [SN96]), Electro-Magnetic Interference (EMI), electrostatic discharge [Con03] and radiation (including alpha particles and high energy cosmic rays like neutrons [KH04]). The typical phenomenon caused by a transient fault is a bit-flip, also known as a glitch, which can be positive or negative. They usually last for a short period and cause *soft* errors when captured by memory components. *Soft* errors are non-permanent and non-recurring.

Transient faults caused by radiation have been widely studied in the literature due to their unpredictable nature and increasing occurrence [MW79, KH04, MHH+05]. Two leading radiation sources are alpha particles and high energy cosmic rays (mainly neutrons), which are from the radioactive decay of isotopes in device materials and space, respectively [MW79, KH04]. The first report of electronic device malfunctions caused by cosmic rays was in 1975 when anomalies emerged in communication satellites [BSH75, KH04]. In 1978, alpha particle induced soft errors were first

observed at ground level [MW78]. After that, widespread of evidence of soft errors was found [OBF+93, TN93, BHS+95, BN98, KH04, MHH+05]. As the semiconductor geometry shrinks, the possibility that neutron and alpha particle strikes cause bit-flips in a cell significantly increases, leading to an increasing *Soft Error Rate* (SER) [KH04, Bor05], which is a threat to current digital systems. It has been reported that the SER per logic state bit increases 8% in each technology generation [HKM+03].

The effect of a radiation event is commonly termed *Single Event Effect* (SEE) [Bau05, HS12]. As Figure 3.3 shows, a strike of particles on silicon devices can generate a cloud of electron-hole pairs that allow a current flow. This may be evident as a transient voltage change, which can flip bit(s) in memory. This transient voltage fluctuation at a logic node is termed a *Single Event Transient* (SET, in Figure 3.2) [MT03, KM04, HS12]. A SET propagates in combinational logic and becomes a *Single Event Upset* (SEU) [HS12] when it is captured by a memory component. SEUs are *soft* errors which are random, transient and non-recurring [Muk08]. Usually a SEU lasts about 100 ps [KH04] and affects one bit only, which is termed *Single-Bit Upset* (SBU) [Bau05]. If the radiation event is of high energy, a *Multi-Bit Upset* (MBU) may be created [Bau05] affecting multiple bits (or *Multiple Cell Upset* (MCU) in Static or Dynamic Random Access Memories (SRAMs/DRAMs) [LK08]). The relations of these terms are illustrated in Figure 3.2. Without struggling to differentiate these terms for various usage scenarios, this research uses "transient faults" and "transient errors" as the unified names to study their impact and behaviour, so as to concentrate on the fault management techniques.

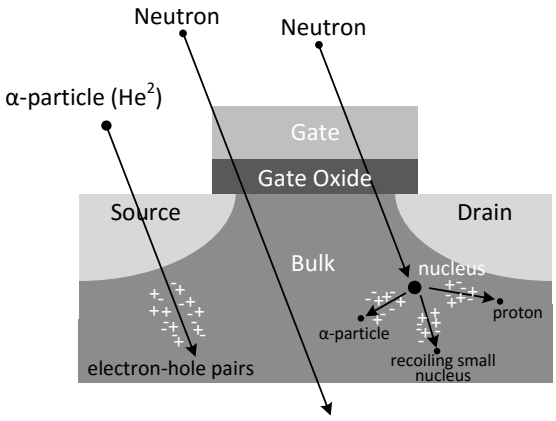


Figure 3.3: Transient faults caused by ionising radiation

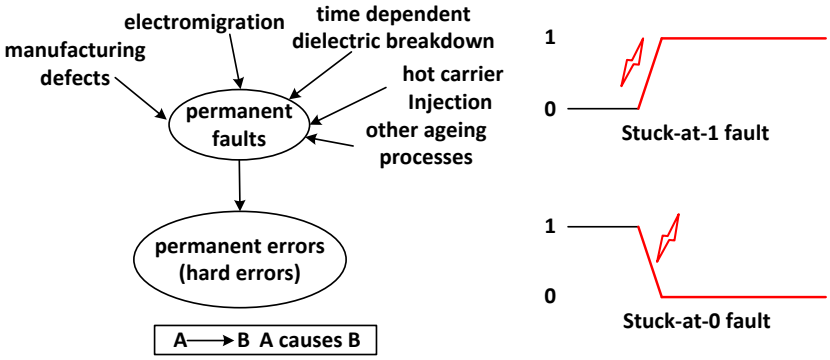


Figure 3.4: Sources of permanent faults [TS83, Bor05, McP12, PBP15]

3.1.2 Permanent faults

Permanent faults (or hard faults) can be classified into *manufacturing defects* and *operational hard faults* according to their occurrence time. Manufacturing defects are introduced during the chip manufacturing process and can lead to a chip yield loss [BSO05]. *Defect-tolerance* techniques are used to enhance the chip yield [KK98, MG03, TM04]. With the improvement of semiconductor manufacturing techniques, this kind of permanent faults decrease [Con03].

The increasing power density, the high temperature and other scaling effects coming along with the sub-micron era can accelerate the ageing process and have a sustained negative impact on long-term chip lifetime [SABR04, ZPJ+15]. Electronics become more susceptible to permanent faults. Permanent faults can happen at runtime with the ageing process, leading to *operational hard faults* [LMSP99, BSO05]. Figure 3.4 shows possible sources of permanent faults [TS83, Bor05, McP12, PBP15]. The two dominant factors causing operational hard faults are electromigration and Time Dependent Dielectric Breakdown (TDDB) [SABR04, BSO05, McP12].

- Electromigration: Metal atoms migrate over time because of current flow. This migration will cause an accumulation and depletion of the metal, which may create hillocks or voids in wires (Figure 3.5). The eventually resulting open or short circuits lead to permanent faults [Tu03, SSC08, PBP15]. A delay fault may happen before an open fault due to increased resistance [Con03].
- Time Dependent Dielectric Breakdown (TDDB) is caused by manufacturing imperfections in a gate oxide with the ageing process [BSO05]. Imperfections produce electron "traps" in the oxide which may accumulate over time due to the continuous electric field stress and electron tunnelling [LBTW97, McP12]. The

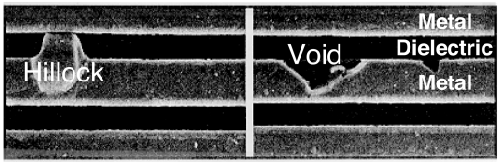


Figure 3.5: Scanning Electron Micrograph (SEM) of hillock and voids caused by electromigration in an Al (Cu,Si) wire [Sar03]

trapped electrons may build a conductive path between the gate and the bulk of the device, leading to a gate oxide breakdown, which causes bit-flips or delays.

In some cases, manufacturing imperfections are so tiny that they cannot be detected during the initial test process or by *defect-tolerance* techniques [KK98, MG03, TM04] but evidence after a long period, leading to operational hard faults [LMSP99].

"Permanent faults" in this thesis refers to the operational hard faults without specific clarification. The resulting permanent errors exist for the remaining life of the circuits, bringing lifetime reliability problems. This may lead to a system failure or cause chips being discarded. For critical devices, to keep them working is significantly important even with some performance loss, leading to a demand for runtime permanent-fault-tolerant systems [BSO05].

The gate-level stuck-at fault model, which has been widely accepted in the semiconductor industry, is employed to study the behaviour of permanent faults [AAA87, MMC15]. According to the definition of the stuck-at model, the state of a faulty wire – which can be a gate input/output or a long interconnect – is either stuck-at-0 or stuck-at-1 (Figure 3.4), masking details of fault behaviour and making it simple to analyse the impact of permanently faulty circuits.

3.1.3 Intermittent faults

The wear-out of transistors and interconnects is a gradual process which first affects timing, probably making combinational circuits violate timing requirements and leading to delay faults [SGH$^+$07]. As a result, intermittent faults usually happen at the same location, first manifesting as delay faults, then frequent transient faults and finally permanent faults [Con03]. It has been discussed that some classes of asynchronous circuits – Delay-Insensitive (DI) and Quasi-Delay-Insensitive (QDI) circuits – are timing-robust and can tolerate these delay faults [SF01]. Therefore, according to their behaviour or duration length, intermittent faults can be regarded as either transient or permanent during the fault management process.

3.1.4 Masking factors

In common combinational logic, three intrinsic masking factors could prevent transient faults from being captured as transient errors [KH04, KM04]. In any of these cases the fault goes undetected and never causes an error.

- Logical masking: A logic gate in the propagation path of a fault has multiple inputs while the output may be decided by the other inputs rather than the faulty one, so that this transient fault can be logically masked.
- Electrical masking: As a transient fault propagates in a combinational path, the pulse amplitude may reduce while the rise and fall time increases due to the electrical properties of gates, leading to an attenuated SET which will not be captured by the final memory component.
- Temporal masking: A transient fault reaches a memory component (such as a flip-flop) but not at the clock edge or the latching window where the component captures values. This fault will not cause errors as well.

3.1.5 Traditional redundancy techniques

Though many faults are masked during propagation [KH04], extra fault-tolerant measures should be used to create a more secure environment considering the increasing fault possibility, requiring redundant circuit designs. Traditional redundancy techniques include physical, temporal and information redundancy [Muk08, Sor09].

Physical (or "*spatial*") redundancy, achieves fault-tolerance by replicating hardware. A well-known physical redundancy technique is Triple Modular Redundancy (TMR) [Sor09], which adds two replicas of the original circuits. A single error can be detected and masked by a voter. TMR induces an area overhead of at least 200%, along with a multiplied power consumption.

Temporal (or "*time*") redundancy, achieves fault-tolerance by performing one operation multiple times. Errors can be found and corrected by comparing multiple results. Extra hardware is required to store the previous data. Due to the repeated operation, the consumed power will be multiplied as well as the latency, which constrains the circuit performance. A permanent fault will be repeated and influence all operations so that this method is unsuitable for permanent faults.

Information redundancy has been widely used to protect interconnection and memory. Extra check bits are added to the original data word, constructing an Error Detecting Code (EDC) or Error Correcting Code (ECC), where faulty bits can be detected

or corrected [Ham50, Muk08]. The error detection and correction capability is determined by the *Hamming Distance* (*HD*) of the possible codes, which is the number of bit positions in which they differ [Sor09, Muk08]. Taking two codes *A*=0010 and *B*=0011 for example, their *HD* is one since they only differ in one bit. If *B* is 0100, their *HD* becomes two. Given a coding space, its *HD* is defined as the minimum HD of any two valid code words. The *HD* of all 1-of-n codes is two. An EDC/ECC code can detect up to $(HD-1)$ bit errors and correct up to $\lfloor (HD-1)/2 \rfloor$ bit errors [Sor09].

The above techniques belong to Forward Error Recovery (FER) processes [Sor09], where faults or errors are detected/corrected as the data word progresses. The flow of data is not interrupted if a fault occurs. Redundant circuits usually lie in critical paths, inducing a slight performance penalty even when no errors happen. An additional slight performance penalty may be caused when correcting errors. Alternatively, the error correction process can be a Backward Error Recovery (BER) [Sor09] process which only uses redundancy to *detect* faults and store backups. Fault-free data is retained as a recovery point before it is sent out. When faults are detected at the receiver, a retransmission operation is requested. In situations with few errors, BER shows better performance than FER because of the omitted error correction circuits and lower overhead on transmission – detection is cheaper than correction. However, in a fault-prone environment, BER may suffer from the costly data storing and retransmission operations. Extra storage is needed for the BER to record recovery points which may introduce a large area overhead. In addition, retransmissions cannot resolve permanent faults. If a fault destroys the handshake process in asynchronous circuits, the local faulty handshake signals cannot be simply recovered by using retransmission. Therefore, the proposed fault-tolerant techniques in this research use the FER scheme.

3.2 Impact of transient faults on QDI pipelines

A generic NoC router acts as a switch which directs traffic from inputs to different outputs. Buffers are usually inserted to or between input and output ports of a router to support specific flow control and improve the network performance. As a result, the data path through a router can be modelled as a pipeline where different buffers construct pipeline stages, as Figure 3.6 depicts. As stated in Section 2.1, the simple 4-phase, 1-of-n asynchronous protocol is chosen to implement the QDI NoC in this research so that the data path built by a packet is a 4-phase, 1-of-n QDI pipeline. The pipeline may comprise one or more parallel 1-of-n channels, as shown in Figure 2.12.

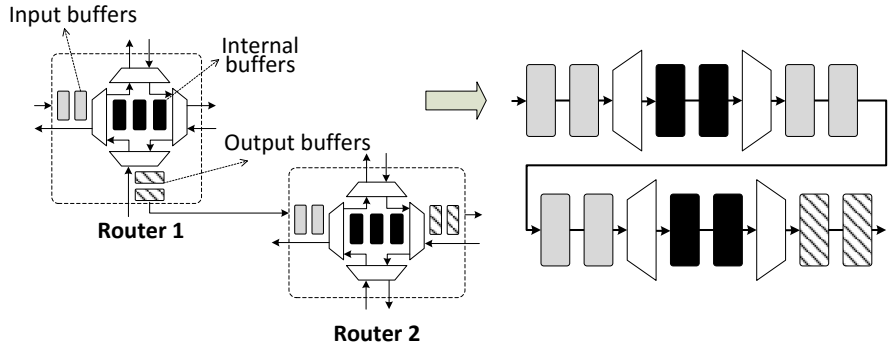


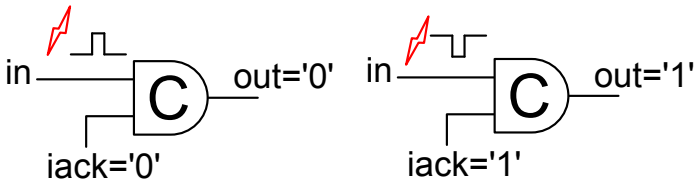
Figure 3.6: Pipelined NoC data path

3.2.1 Transient faults on synchronous and QDI pipelines

Faults may happen on any gates and wires of a circuit at any time. In the presence of transient faults, QDI pipelines behave differently from synchronous ones due to their clockless nature. A fundamental difference between QDI and synchronous circuits is the timing reference of data words.

In synchronous circuits, the clock signal acts as a timing reference controlling the data transmission. Each bit of a data word typically has a constant duration which is agreed between the transmitter and the receiver. Faults are therefore only able to corrupt the *value* of symbols, which can be used to analyse the fault type [Sor09], locate the fault position and further correct errors.

QDI circuits have no such timing reference; data-validity is implicitly encoded within the data word, which controls its own transmission. As a result, the corrupted data words caused by faults can disorder the data transmission. In other words, faults on QDI pipelines may produce fake data-validity or remove the correct data-validity, resulting in erroneous data insertions or data removals. A physical-layer deadlock could happen if the handshake process stalls due to a fault. Even a transient fault can deadlock a QDI pipeline, which is a new challenge that has been ignored by the asynchronous community (Section 3.3.2). This deadlock is harmful to the circuit function. As an example, an asynchronous NoC can be used to construct a Globally Asynchronous, Locally Synchronous (GALS) system [KGGV07]; redundant information can be added in the synchronous domain and transmitted through the asynchronous network. Errors are corrected when the data leaves the network and arrives at the destination [CH01,Pon12]. This can save lots of overhead but the risk is the destination may never receive the packet due to a fault-caused physical-layer deadlock. Conventional fault-tolerant techniques used in synchronous circuits cannot work in QDI pipelines.



(a) Masked positive fault (b) Masked negative fault

Figure 3.7: Transient faults masked by C-elements

Therefore, it is necessary to provide fault-tolerance for asynchronous interconnects.

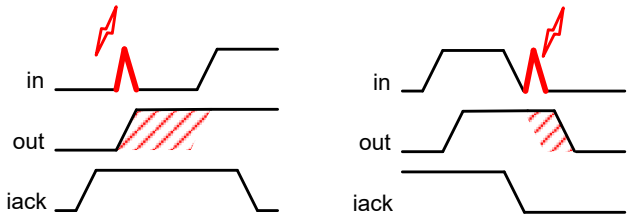
Not every transient fault causes an error. Some transient faults are masked automatically during their propagation by intrinsic masking factors of the circuit [KH04]. The three aforementioned conventional masking factors determining whether a transient fault will be captured as an error (Section 3.1.4) are not fully suitable for QDI circuits. Temporal masking which relies on the clock latching-window has to be excluded since there is no clock in QDI circuits. Electrical masking can occur in long interconnects, which is common in both synchronous and asynchronous circuits. Logical masking plays an important role in asynchronous circuits. The use of C-elements, which are the most fundamental and significant cells in asynchronous circuits, enriches the concept of logical masking. A C-element outputs '1' or '0' if both of its inputs are '1's or '0's; otherwise, it keeps the last output. As Figure 3.7 shows, when *iack* is '0', a positive glitch at the other input of the 2-input C-element is masked; When *iack* is '1', a negative glitch at the other input is masked. If a glitch at the input *in* is to the same level as *iack*, the fault will be latched, but can be tolerated depending on the normal input:

- If this C-element output will flip to the same level in the later normal process after the fault disappears, this fault causes a *premature* firing as Figure 3.8a shows, which can be tolerated naturally by QDI circuits.
- Similarly, a fault can delay a firing if it has the same level with the concurrent firing, as Figure 3.8b shows, which can be tolerated as well.

Therefore, a key issue faced be fault-tolerant QDI pipelines is to prevent transient faults from being incorrectly latched and interrupting the normal handshake process.

3.2.2 Modelling impact of transient faults

A fault model was built to analyse the impact of a 1-bit transient fault on QDI pipelines. Figure 3.9 presents a single-symbol 4-phase 1-of-n pipeline model with two successive stages. The handshake between two adjacent pipelines stages forms a loop, including



(a) Premature firing (b) Delayed firing

Figure 3.8: Premature and delayed firing of a C-element due to a fault

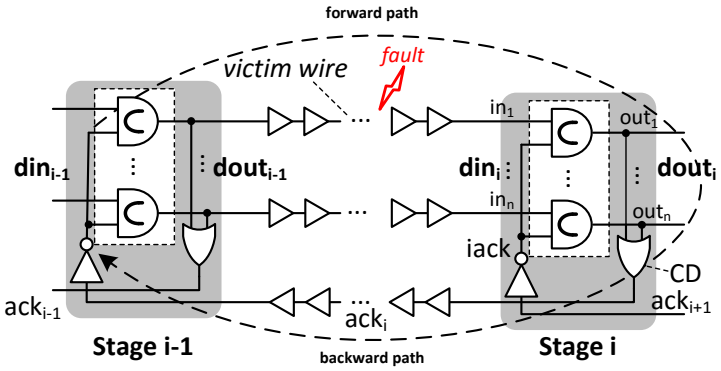


Figure 3.9: Two-stage pipeline model with one 1-of-n channel

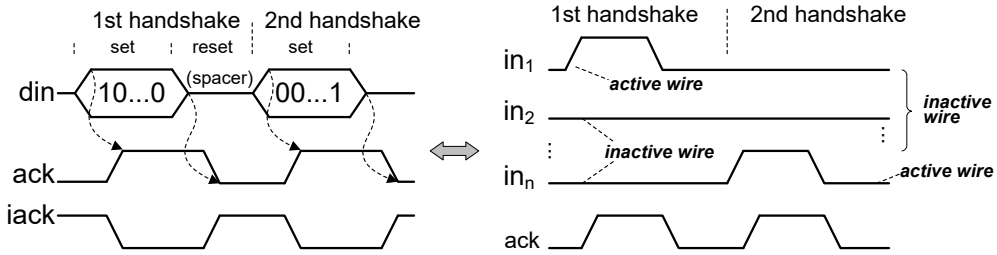
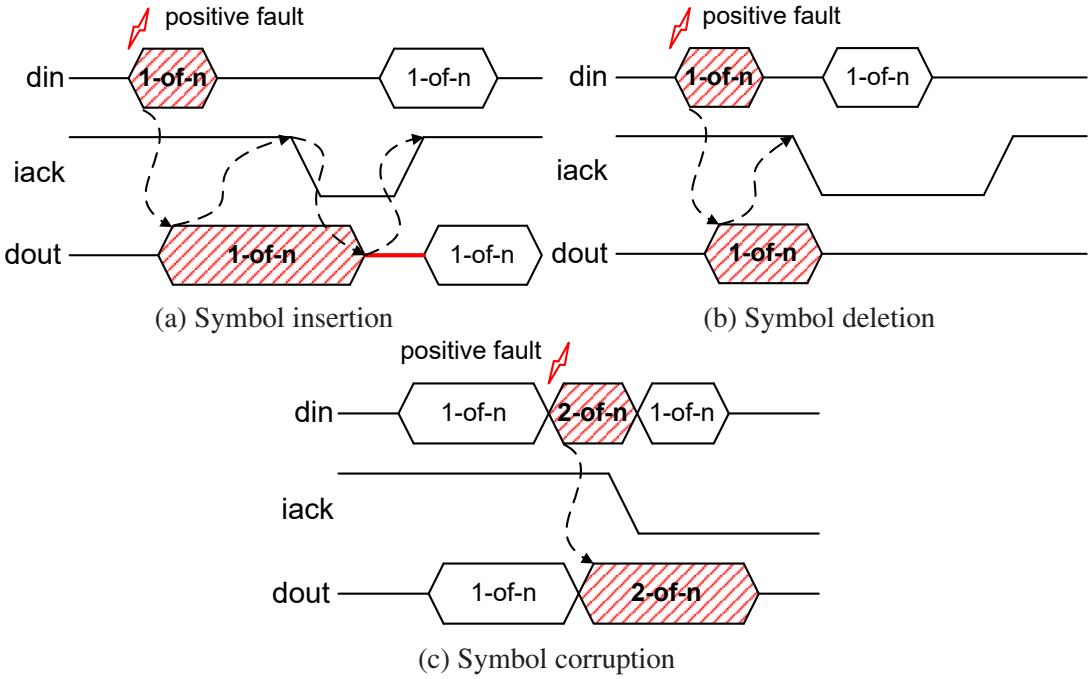


Figure 3.10: Fault-free 4-phase handshake process

a forward data path and a backward acknowledge path. A transient fault can happen on any gate or wire of a QDI pipeline, including pipeline stages (the C-element built asynchronous latch and the Completion Detector (CD)), the *n data* wires and the *ack* wire. This results in a "victim" wire if it is not masked (Figure 3.9). Figure 3.10 models the 4-phase handshake process corresponding to the pipeline segment in Figure 3.9. In one 1-of-n channel, the data wire that should carry the correct '1' in the normal case is defined as the "active" wire while the other data wires are "inactive" in the current handshake period. The *iack* is the inverted *ack* signal used to drive the preceding pipeline stage. To analyse all faulty occasions, the proposed fault model divides one handshake period into two intervals according to logical level of *iack*.



(a) Symbol insertion

(b) Symbol deletion

(c) Symbol corruption

Figure 3.11: Transient faults happen when *iack* is high

Transient faults on data wires with a positive *iack*

When *iack* is high, the QDI pipeline (and the C-element latch) is susceptible to positive data faults. Both active and inactive wires can be the victim, leading to an invalid 1-of-n code insertion or a 2-of-n code insertion.

1. *Invalid 1-of-n code insertion*: A positive fault on an inactive wire may be latched and output before the arrival of the next valid code, producing an invalid 1-of-n code insertion and resulting an early drop of *iack*. As shown in Figure 3.11a, if the valid code comes so late that a full reset phase of the invalid code has completed, the invalid 1-of-n code is *inserted* into the original code sequence. Otherwise, if the following valid code overlaps with the reset phase (*iack*−) from the inserted invalid code, as Figure 3.11b depicts, this original valid code may be *deleted* from the data sequence as the negative *iack* caused by the invalid code will block it from being latched.

   A positive fault on the "active" wire of the 1-of-n channel may lead to the cases shown in Figure 3.11a and Figure 3.11b when the fault makes *iack* go low earlier than the arrival of the valid data, but the difference is the inserted 1-of-n code by the fault is *valid* now. Therefore, the situation of Figure 3.11b will not make any

errors while the situation of Figure 3.11a will output two valid codes, which can be taken as an invalid spacer insertion.

2. *Invalid 2-of-n code insertion*: As shown in Figure 3.11c, a positive fault on one of the inactive wires may coincide with the transition of the active wire, resulting in an invalid 2-of-n code at the output.

Transient faults on data wires with a negative *iack*

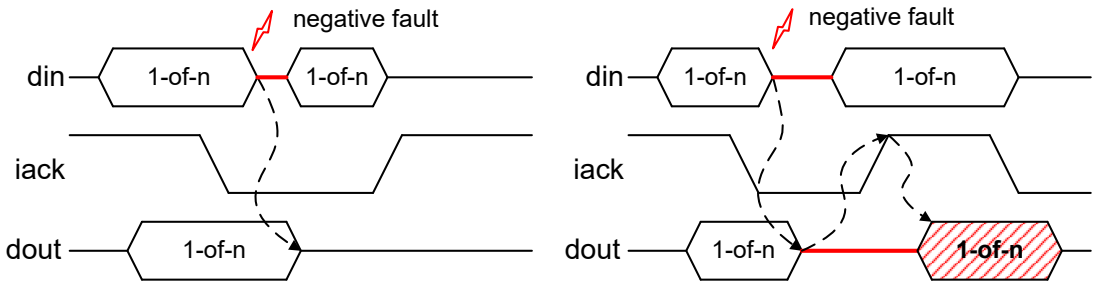
When *iack* is low, all positive data faults are masked and the pipeline is only vulnerable to negative faults happening on the active wire.

1. *Early spacer* (not errors): A negative fault on the active wire may cause an early spacer to be latched (Figure 3.12a), which is tolerated by QDI pipelines.

2. *Invalid 1-of-n code insertion*: A negative fault may lead to a premature reset phase. If *iack* has time to return high as a reaction to the data being withdrawn, as shown in Figure 3.12b, the original valid code will be output twice, leading to an invalid code insertion to the original data sequence.

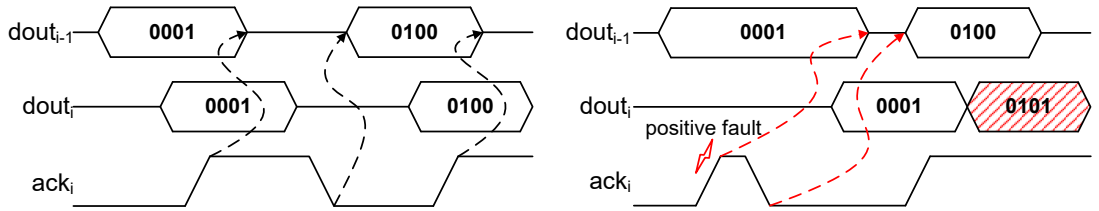
Transient faults on CD and *ack* wires

Acknowledge (*ack*) signals are equally critical to the handshake process in comparison to the transmitted data. Though the number of *ack* wires can be far less than the number of data wires, faults on *ack* wires could cause complex faulty behaviour as well. Protection of acknowledge wires has been ignored by many existing fault-tolerant designs [PCV12]. An example is shown in Figure 3.13 where $dout_{i-1}$ and $dout_i$ are outputs of *Stage* i-1 and *Stage* i respectively; ack_i is the acknowledge signal



(a) Early spacer (b) Invalid 1-of-n code insertion

Figure 3.12: Transient faults happen when *iack* is low



(a) Correct situation (b) Incorrect situation

Figure 3.13: Transient faults on acknowledge wires

from *Stage* i to *Stage* i-1. Figure 3.13a shows the correct situation of these signals. Figure 3.13b depicts the faulty situation where a positive fault happens on ack_i indicating that a valid code has been latched by *Stage* i, causing *Stage* i-1 to reset. When the fault disappears, *Stage* i-1 starts a new transmission of "0100", which may augment the first data word, leading to an invalid 2-of-n code ("0101") insertion.

Physical-layer deadlock caused by transient faults

From the above analysis, it can be found that a transient fault could cause symbol *insertion*, *deletion* and *corruption* on one 1-of-n channel, which could disorder the original symbol sequence. In a wide QDI pipeline comprising many parallel 1-of-n channels, multiple symbol sequences are transmitted separately through the multiple parallel channels, which need to be synchronized at various places (the CD of each pipeline stage in Figure 2.12) to achieve the complete data. If there is an extra (or missing) symbol on one channel, this synchronization may fail and will wait indefinitely. The data transmission in the pipeline is stalled and cannot recover by itself. This indefinite stall is termed a "**physical-layer**" deadlock since it destroys the bottom handshake protocol. The behaviour of this deadlock caused by transient faults has long been neglected by the asynchronous community besides some early exploration [Shi10]. Along with the physical-layer deadlock caused by permanent faults (which stall the handshake permanently apparently), it will be thoroughly discussed in the next Section 3.3.

In conclusion, transient faults on QDI pipelines may cause symbol insertion (a 1-of-n code is inserted to the original data sequence), symbol deletion (the original code is removed from the data sequence), symbol corruption (a 1-of-n code is corrupted into an invalid m-of-n code), or a physical-layer deadlock where the handshake process is permanently stalled.

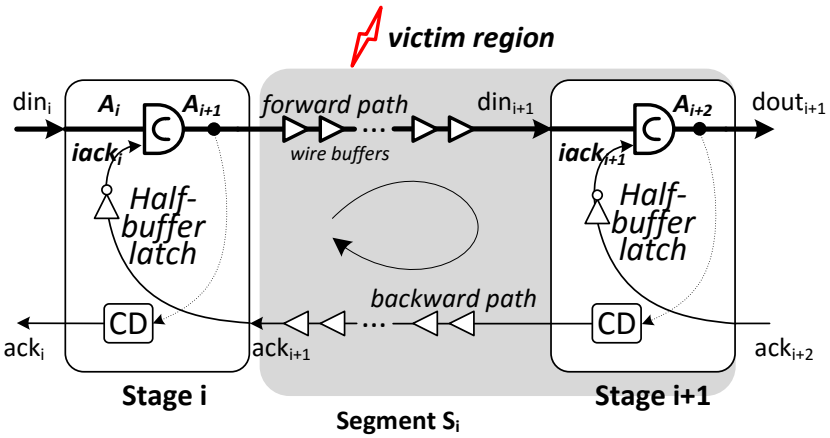


Figure 3.14: A QDI pipeline model divided into *segments*

3.3 Modelling the deadlock caused by different faults

Both transient and permanent faults can stall the handshake process in the bottom, physical layer permanently, leading to a *fault-caused physical-layer deadlock*, which is a new challenge faced by the asynchronous community. This deadlock is more serious to QDI NoCs than packet errors, since it happens in the physical layer and cannot be solved by network-layer deadlock management techniques (Section 2.2.5). In a deadlocked state, most existing fault-tolerant techniques for synchronous circuits cannot work [Ham50, Sor09]. Most existing fault-tolerant QDI designs are also deadlocked by faults and fail to work [CH01, JM05, PCV12, ZSG+14b]. This section models QDI pipelines deadlocked by different faults to study their behaviour, with which the fault can be detected and recovered from.

A linear pipeline with depth d can be divided into $(d-1)$ *segments* ($d \geq 2$). A segment begins and ends with contiguous half-buffer latches, as Figure 3.14 shows. *Stage* i+1 ($1 \leq i < d$) is included in the segment S_i. The *direct* result of a fault on the forward data path of a segment (S_i) may be a faulty input (din_{i+1}) to the half-buffer latch of the second stage (*Stage* i+1), which can be latched and further affect the produced *ack* signal to the preceding stage (*Stage* i). This fault on the forwarding *data* wires and asynchronous latch is termed a "*Data*" fault. On the backward *ack* path, including the Completion Detector (CD) and the *ack* wire, the *direct* fault impact may be a faulty *ack* signal (ack_{i+1}) to the preceding stage; this is an "*Ack*" fault.

Segment is used as the basic unit to study the fault impact (the grey "*victim*" region in Figure 3.14). *Stage* i is termed the "pre-fault" stage while *Stage* i+1 is the "post-fault" one. The proposed fault model is based on the following assumptions or rules:

- Though multi-bit faults could happen in practice, only a 1-bit fault is considered since it is enough to deadlock a QDI pipeline.
- If a fault happens at the *latch input* or in the *latch*, whose value is stored by the latch, this fault is a "Data" fault to the current pipeline stage.
- If a fault is declared to happen at the *latch output*, it is an "Ack" fault. The fault is not stored by the current latch but affects the CD.
- If a fault happens at the *stage output* of the pipeline stage, it indicates that this fault neither gets stored by the current latch nor goes to the CD. This is a "Data" fault to the next pipeline stage.

3.3.1 Deadlock caused by permanent faults

Since QDI asynchronous circuits are event-driven and controlled by handshake protocols, besides corrupting the transmitting data, it is obvious that a permanent stuck-at fault will halt the handshake process, resulting a "*physical-layer*" deadlock.

In synchronous circuits, permanent faults typically cause consecutive data errors. The circuit continues under the control of the clock, making it possible to detect the fault using accumulated history statistics of the circuit, i.e. *error syndromes* which are usually obtained by using transient-fault-tolerant techniques [LLP07]. If the collected error syndromes satisfy specific patterns or some time-out conditions, the fault is taken as permanent and the recovery process is invoked; otherwise, it is transient or intermittent [LLP07, LWL+10, FLJ+13].

However, in QDI circuits, most existing transient-fault-tolerant QDI designs [CH01, JM05, PCV12, ZSG+13] are also deadlocked when a permanent fault happens. In this case, error syndromes cannot be easily obtained, making traditional fault-detection techniques [LLP07, LWL+10, FLJ+13] fail. Furthermore, when it comes to a QDI NoC, this physical-layer deadlock may reserve a sequence of network resources, which will cause more upper-layer deadlocks of the network, dramatically reducing the network performance and finally paralysing the whole network. This section models the deadlocked QDI pipeline to find common deadlock patterns with which the permanent fault can be detected and located.

In a 4-phase, 1-of-n QDI pipeline, data is encoded as Delay-Insensitive (DI) symbols which are either *complete* or *incomplete*. Incomplete data exist between a complete data word and a spacer. According to the state transition in a pipeline stage (Figure 2.11), Figure 3.15 illustrates a possible transient state of a 6-stage pipeline without faults. The consequence of a permanent fault on segment S_i of a 4-phase, 1-of-n QDI

pipeline can be classified into following four classes (Figure 3.16, 3.17 and 3.18):

1. **Data stuck-at-0** (Figure 3.16a): Caused by a stuck-at-0 fault on the forward data wire or in the asynchronous latch, an input (din_{i+1} in Figure 3.14) to the asynchronous latch can get stuck at '0', preventing a valid '1' from arriving at the post-fault stage (*Stage* i+1). As a result, all pipeline stages downstream of the fault are stuck at the *set* phase with an incomplete data word and keep waiting for the lost '1'. Their *ack* signals are all '0's awaiting a complete data word.

2. **Data stuck-at-1** (Figure 3.16b): Due to a stuck-at-1 fault on the forward data wire or the latch, the input to the asynchronous latch of the post-fault *Stage* i+1 could get stuck at '1', preventing all pipeline stages downstream of the fault from being reset as they keep holding the incomplete data word with the invalid '1'. As a result, all their *ack* signals are '1's.

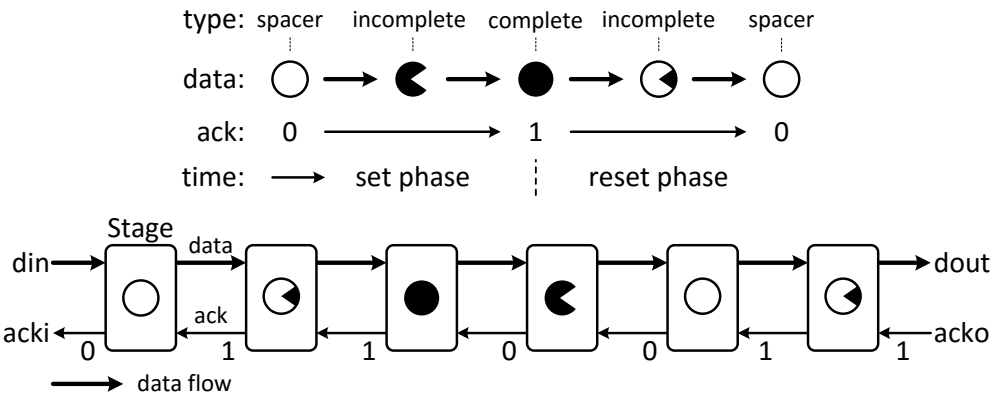
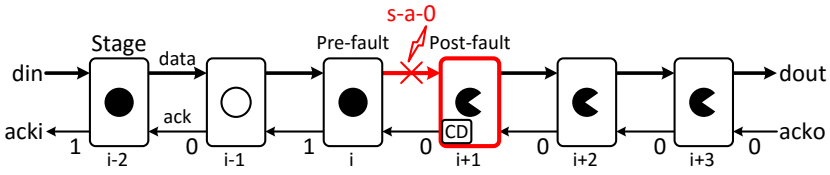
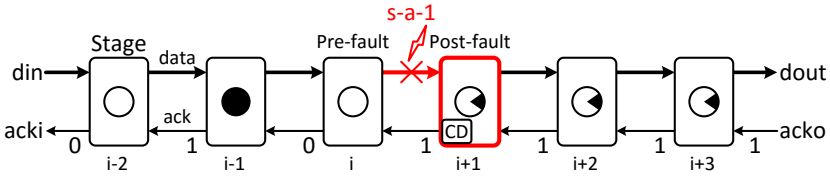


Figure 3.15: One possible transient state of the pipeline without faults

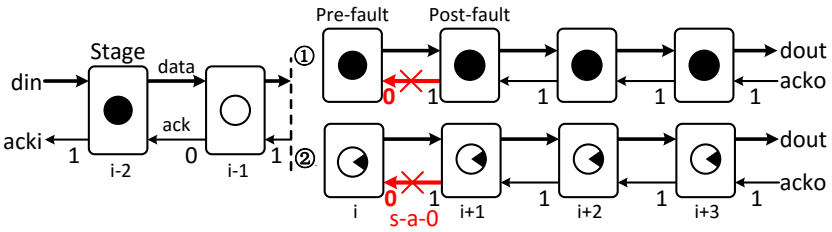


(a) Data stuck-at-0 fault

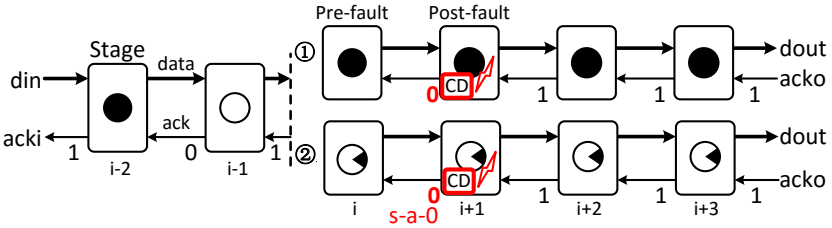


(b) Data stuck-at-1 fault

Figure 3.16: Deadlocked pipelines due to stuck-at faults on the forward data path



(a) Ack stuck-at-0 fault on the *ack* wire



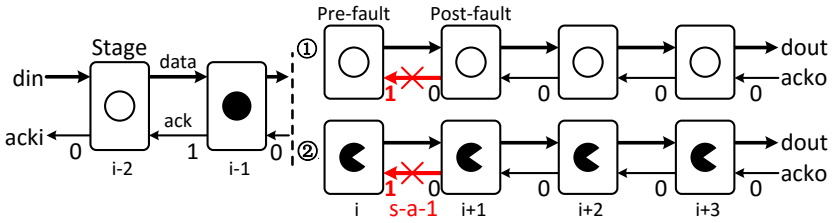
(b) Ack stuck-at-0 fault on the CD

Figure 3.17: Deadlocked pipelines due to stuck-at-0 faults on CD or *ack* wires

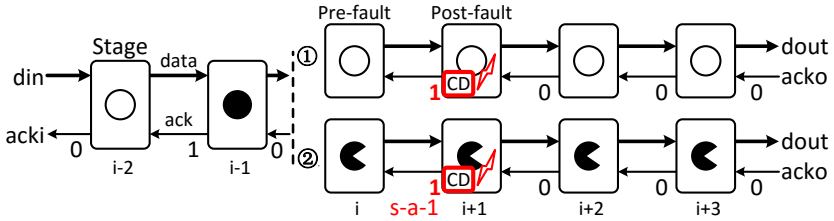
3. **Ack stuck-at-0** (Figure 3.17): The *ack* signal (ack_{i+1}) to the preceding pipeline stage (*Stage* i) could get stuck at '0' due to a permanent fault on the backward *ack* wire (Figure 3.17a) or the CD of the current stage (*Stage* i+1, Figure 3.17b). As a result, *Stage* i may hold either a complete data word if the faulty *ack* arrives at the preceding stage when it was in the *set* phase, or an incomplete data word if the preceding stage is resetting (the faulty *ack* stalls the *reset* operation). Thus, the pre-fault stage and all pipeline stages downstream of the fault cannot be fully reset. Their *ack* signals are all '1's. Considering the *ack* signal back to the pre-fault *Stage* i, it may have two possibilities depending on the fault position as Figure 3.17a and 3.17b show.

4. **Ack stuck-at-1** (Figure 3.18): The *ack* wire (ack_{i+1}) to the preceding stage may get stuck at '1' as well due to a stuck-at-1 fault on the backward acknowledge path, so that incoming '1's or 1-of-n symbols to the pre-fault stage cannot be latched. The pre-fault stage and all pipeline stages downstream of the fault may hold a spacer or an incomplete data word, getting stuck at the *set* phase, with their *ack*s being '0's.

For all the above cases, since pipeline stages before the pre-fault stage are fault-free, they would hold fault-free complete data words and spacers alternately according to the 4-phase handshake protocol, leading to alternate "*ack* signal states" [SF01] as Figure 3.16, 3.17 and 3.18 show. It can be concluded that, all the above cases share



(a) Ack stuck-at-1 fault on the *ack* wire



(b) Ack stuck-at-1 fault on the CD

Figure 3.18: Deadlocked pipelines due to stuck-at-1 faults on CD or *ack* wires

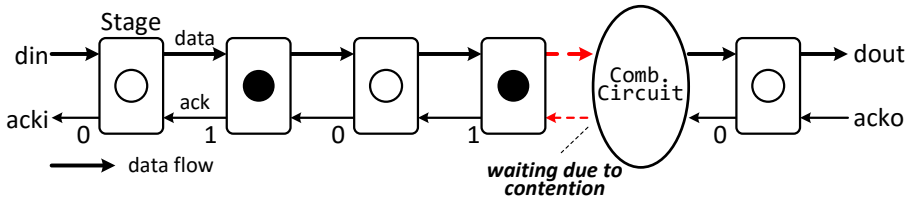


Figure 3.19: Blocked data flow without faults

the same deadlock pattern defined in Theorem 3.3.1, which is the key to detecting permanent faults in QDI pipelines.

Theorem 3.3.1. ***Pattern of the fault-caused physical-layer deadlock****:*

The physical-layer deadlock caused by a permanent fault on a 4-phase 1-of-n QDI pipeline leads to a steady state where all pipeline stages downstream of the fault have the same ack while the ack signals between upstream stages are alternately valued.

It should be noticed that the data transmission through some pipeline stages may take a long time, which can be caused by traffic contention happening in QDI NoCs due to a high workload. As a result, the packet transmission is stalled for some time. All stalled pipeline stages store fault-free data. According to 4-phase handshake protocols, complete data words and spacers are stored alternately in this stalled packet path (Figure 3.19), which is different from the deadlock pattern caused by faults (Figure 3.16 $\sim$ 3.18). This feature can be used in QDI NoCs to differentiate the fault-caused physical-layer deadlock from the network-layer one and contention (Chapter 5).

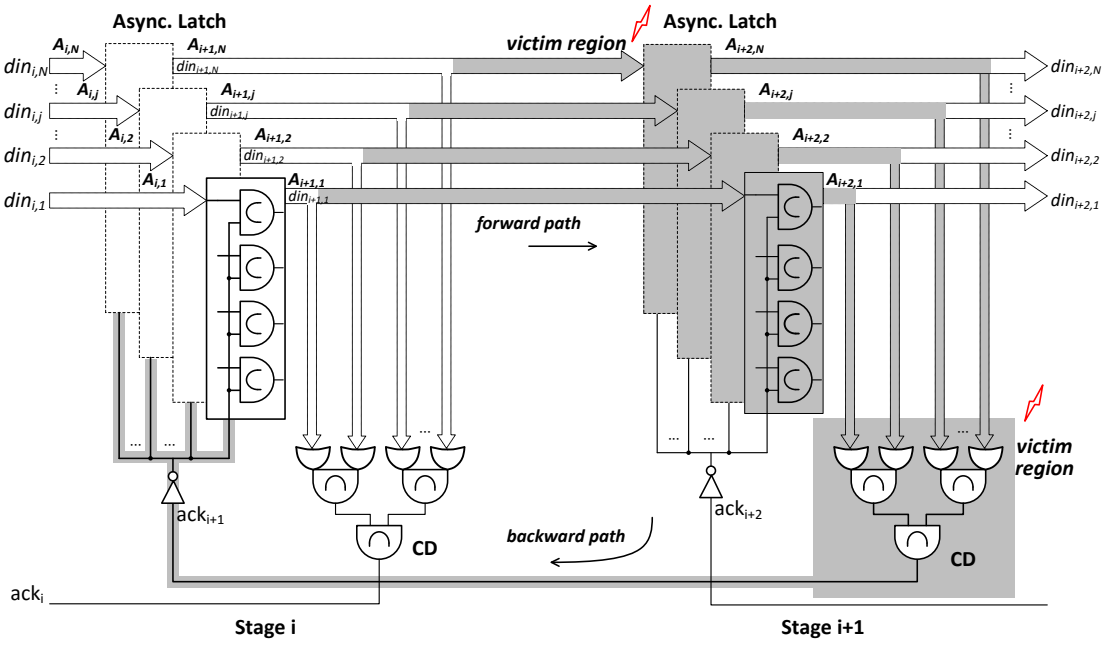


Figure 3.20: Pipeline model for transient faults

3.3.2 Deadlock caused by transient faults

As mentioned in Section 3.2, a transient fault may insert or delete a symbol from a 1-of-n channel, stall the handshake process and deadlock the N-symbol wide QDI pipeline ($N>1$), which has barely been studied. The occurrence of this deadlock is more complicated than the one due to a permanent fault. This section will study the behaviour of a transient fault stalling the 4-phase handshake process and the resulting deadlock patterns, which can be used to detect the fault and diagnose the fault type.

For an N-symbol 4-phase 1-of-n QDI pipeline, the input data to a pipeline stage can be expressed by using "active" signals (Figure 3.20) transmitting on the "active" wires belonging to the N 1-of-n channels (Section 3.2.2), so that din_i to *Stage* i can be expressed as $\{A_{i,1}, A_{i,2}...A_{i,N}\}$ $(1 \leq i \leq d)$. $A_{i,j}$ represents the jth output active wire from the (i-1)th pipeline stage, as Figure 3.20 shows. Active signals ($A_{i,j}$) to, and *ack* signals (ack_i) from, pipeline *Stage* i and *Stage* i+1 are used to denote the state of a pipeline *segment*, S_i=($\{A_{i,1}...A_{i,N}\}$, ack_i, $\{A_{i+1,1}...A_{i+1,N}\}$, ack_{i+1}), whose initial state after reset is S_i = ($\{0...0\}$, 0, $\{0...0\}$, 0).

The Production Rule Set (PRS) for basic logic gates [Mar91] is borrowed to describe the behaviour of a *segment*. A production rule with a form of $A \wedge B \longrightarrow C\uparrow, D\downarrow$ is equivalent to $A \wedge B \longrightarrow C\uparrow$ and $A \wedge B \longrightarrow D\downarrow$, meaning that if both A and B are *true*, the assignments $C\uparrow$ and $D\downarrow$ are fired. The firing of $C\uparrow$ and $D\downarrow$ is independent

and has no sequence. According to the 4-phase handshake protocol, the PRS for an N-word pipeline *segment* S_i is ($1 \leq j \leq N$) shown below.

- For *segment* S_i

$$A_{i,j} \wedge \neg ack_{i+1} \longrightarrow A_{i+1,j} \uparrow \tag{3.1}$$

$$A_{i+1,1} \wedge \ldots \wedge A_{i+1,j} \ldots \wedge A_{i+1,N} \longrightarrow ack_i \uparrow \tag{3.2}$$

$$\neg A_{i,j} \wedge ack_{i+1} \longrightarrow A_{i+1,j} \downarrow \tag{3.3}$$

$$\neg A_{i+1,1} \wedge \ldots \wedge \neg A_{i,+1j} \ldots \wedge \neg A_{i,N} \longrightarrow ack_i \downarrow \tag{3.4}$$

The PRS of the environment of *segment* S_i is defined as follows:

- Left of *Stage* i

$$ack_i \longrightarrow A_{i,j} \downarrow \tag{3.5}$$

$$\neg ack_i \longrightarrow A_{i,j} \uparrow \tag{3.6}$$

- Right of *Stage* i+1

$$A_{i+2,1} \wedge \ldots \wedge A_{i+2,j} \ldots \wedge A_{i+2,N} \longrightarrow ack_{i+2} \uparrow \tag{3.7}$$

$$\neg A_{i+2,1} \wedge \ldots \wedge \neg A_{i+2,j} \ldots \wedge \neg A_{i+2,N} \longrightarrow ack_{i+2} \downarrow \tag{3.8}$$

Therefore, if a fault-caused physical-layer deadlock happens where the 4-phase handshake process is broken, none of the above rules should be fired any more. The pipeline gets stuck at a "stable" state without any signal transitions. Considering the deadlock state of a pipeline *segment* S_i containing *Stage* i and *Stage* i+1 (Figure 3.20), its victim region comprises a *forward* data path – including the input data wires to and the asynchronous latch of the post-fault *Stage* i+1, and a *backward ack* path – including the CD of the post-fault *Stage* i+1 and the *ack* wire (ack_{i+1}) back to the pre-fault *Stage* i. The input active wire ($A_{i,j}$) to and *ack* wire (ack_i) from the pre-fault stage are out of the victim region and thus they are fault-free. Theorem 3.3.2 can be inferred in a deadlocked state:

Theorem 3.3.2. *In a deadlocked 4-phase QDI pipeline, the two contiguous ack signals from and to the pre-fault pipeline stage are complementary.*

Proof. It is assumed that the two contiguous *ack* signals from and to the pre-fault *Stage* i, ack_i and ack_{i+1}, are equal in the deadlock state. Thus (ack_i, ack_{i+1}) gets stuck at (11) or (00).

a) (ack_i, ack_{i+1}) get stuck at (11).

If ack_i is stuck at '1', $\{A_{i,1}...A_{i,N}\}$ is stuck at $\{0...0\}$ according to (3.5). The low $A_{i,j}$ and the high ack_{i+1} causes a low $A_{i+1,j}$ using (3.3), further leading to a low ack_i according to (3.4), which is in contradiction to the assumption.

b) (ack_i, ack_{i+1}) get stuck at (00).

If ack_i is stuck at '0', $\{A_{i,1}...A_{i,N}\}$ is stuck at $\{1...1\}$ according to (3.6). The high $A_{i,j}$ and the low ack_{i+1} causes a high $A_{i+1,j}$ using (3.1), further leading to a high ack_i according to (3.2), which is in contradiction to the assumption.

Therefore, ack_i and ack_{i+1}, which are the two contiguous *ack* signals from and to the pre-fault pipeline *Stage* i, are complementary in a deadlocked state. In other words, the *segment* state ($\{A_{i,1}...A_{i,N}\}$, ack_i, ack_{i+1}) should get stuck at either ($\{1...1\}$,0,1) or ($\{0...0\}$,1,0) when the physical-layer deadlock happens. □

Theorem 3.3.3. *A transient fault happening on a single-symbol 4-phase 1-of-n QDI pipeline can cause symbol insertion, deletion and corruption, but not a physical-layer deadlock in the current segment.*

Proof. It has been shown that a transient fault could cause symbol insertion, deletion and corruption in a single-symbol 4-phase 1-of-n QDI pipeline [JM05, ZSG+14b] (Section 3.2.2). Here we need to prove only that a transient fault cannot deadlock a single-symbol 1-of-n pipeline. This behaviour was observed by Shi [Shi10] from several faulty scenarios without validation. This research uses mathematical deduction to prove the theorem in a formal way. The *segment* state of a single-symbol pipeline *segment* is indicated by S_i=(A_i, ack_i, A_{i+1}, ack_{i+1}). According to Theorem 3.3.2, (ack_i, ack_{i+1}) should get stuck at either (01) or (10) if deadlock happens.

a) (ack_i, ack_{i+1}) get stuck at (01).

The statement that ack_i being stuck at '0' is true iff A_{i+1} is stuck '0' according to (3.4). The ack_{i+1} gets stuck at '1' iff A_{i+2} is '1' according to (3.2), so that ack_{i+2} is high according to (3.7).

Using (3.3) and (3.4), the low A_{i+1} and the high ack_{i+2} result in a low A_{i+2}, leading to a low ack_{i+1}, which contradicts to the assumption that ack_{i+1} gets stuck at '1'.

b) (ack_i, ack_{i+1}) get stuck at (10).

Similarly, the assumption that ack_i is stuck at '1' iff A_{i+1} keeps high according to (3.2). ack_{i+1} is '0' iff A_{i+2} is low according to (3.4), so ack_{i+2} is low according to

(3.8). Using (3.1) and (3.2), the high A_{i+1} and low ack_{i+2} result a high ack_{i+1}, which contradicts to the assumption that ack_{i+1} gets stuck at '0'.

Therefore, (ack_i,ack_{i+1}) can get stuck at neither (10) nor (01). In other words, a transient fault cannot deadlock a single-symbol 4-phase 1-of-n QDI pipeline. □

It should be noted that the symbol insertion and deletion may cause synchronization failure at a higher level (the data-link layer [TW10]) where the data sequence in this 1-of-n channel needs to be synchronized with the data in other channels. Due to the unmatched symbol count, these independent and parallel channels cannot be correctly synchronized at some points, which is another kind of (data-link-layer) deadlock [LM04]. It is not the point of this research. Thus, Theorem 3.3.3 points out that the deadlock is a physical-layer one which generates from the current *segment*.

In a multi-symbol pipeline, multiple 1-of-n channels are synchronized at each pipeline stage using a multi-bit wide C-element connected to their CDs (Figure 3.20), which is different from a single-word pipeline where the CD is an OR-gate. In other words, there are two kinds of synchronization points in a multi-symbol pipeline. One is the asynchronous latch which synchronizes the forward and backward paths (denoted by (3.1) and (3.3)) and outputs the latched data (A_i). The other is the CD synchronizing multiple parallel forwarding 1-of-n channels (denoted by (3.2) and (3.4)) and meanwhile generating the backward event (ack_i). The introduction of the synchronization point at the CD makes the multi-symbol pipeline different from the single-symbol one when considering the impact of a transient fault.

Theorem 3.3.4. *A transient fault can cause a physical-layer deadlock in a multi-symbol 4-phase 1-of-n QDI pipeline.*

Proof. This theorem has been mentioned by Shi [Shi10] as well. An instance was used to explain the deadlock process without any validation while the management techniques were not studied. It is difficult to make this statement just from one deadlock scenario without any further exploration. This research derives this theorem in a formal way step by step, from which deadlock patterns can be abstracted to support the deadlock management methods proposed in Chapters 5 and 6. This proof is new and belongs to the main contributions of this research. It provides a new way of thinking to handle this deadlock issue in asynchronous circuits or NoCs.

According to Theorem 3.3.2, when a physical-layer deadlock happens, the *segment* state ($\{A_{i,1}...A_{i,N}\}$, ack_i, ack_{i+1}) should get stuck at either ($\{1...1\}$,0,1) or ($\{0...0\}$,1,0).

First, the statement of $(\{A_{i,1}...A_{i,N}\}, ack_i, ack_{i+1})$ being stuck at ($\{1...1\}$,0,1) in the deadlock state is proved. In this case, whether the N active signals, $(A_{i,1}...A_{i,N})$, to the pre-fault stage are latched by *Stage* i depends on the time when ack_{i+1} goes high. The output active signals $(A_{i+1,1}...A_{i+1,N})$ of the pre-fault *Stage* i may have different combinations of logic values:

a) $\{A_{i+1,1}...A_{i+1,N}\}$ are stuck at $\{1...1\}$.

In this case, all active signals to the pre-fault *Stage* i are latched and output, so that ack_i is high using (3.2), which contradicts to the precondition that ack_i is stuck at '0'.

b) $\{A_{i+1,1}...A_{i+1,N}\}$ are stuck at $\{0...0\}$.

In this case, none of the input active signals gets latched and output by *Stage* i, denoting that (3.1) is not fired. It means when the new data word arrives at the *Stage* i, ack_{i+1} to *Stage* i is high. This violates the precondition in a 1-bit transient fault environment where ack_{i+1} cannot remain high.

Therefore, the only possible case for a deadlocked pipeline is that $\{A_{i+1,1}...A_{i+1,N}\}$ are neither all '0's or all '1's.

c) $\{A_{i+1,1}...A_{i+1,N}\}$ are neither all '0's nor all '1's in the final deadlock state.

As Figure 3.20 shows, a transient fault in the victim region of *segment* S_i may happen on either the forward data path (including input data wires to the post-fault *Stage* i+1 and the asynchronous latch of *Stage* i+1), resulting in a faulty $A_{i+2,f}$ ($1 \leq f \leq N$), or the backward *ack* path (including the CD of the post-fault *Stage* i+1 and the backward *ack* wire to the pre-fault *Stage* i), resulting in a faulty ack_{i+1}. The following proofs are divided into two cases for discussion:

1. The 1-bit transient fault is on the forward data path

   Under the precondition that $\{A_{i+1,1}...A_{i+1,N}\}$ are neither all '0's nor all '1's in the final deadlock state, to ensure ack_{i+1} to be stuck at '1' which means (3.1) and (3.2) have been fired, but (3.3) and (3.4) have not, the only possibility is that a 1-bit positive transient fault leads to a faulty active signal, $A'_{i+2,f}$, at the output of the post-fault *Stage* i+1. It is latched by the CD along with other fault-free active signals, resulting a high ack_{i+1} according to (3.2). This faulty active signal $(A'_{i+2,f})$ could traverse through the fth 1-of-n channel to the destination with other fault-free symbols, resulting high *ack* signals at all succeeding pipeline stages. Meanwhile, the fault-free $A_{i,f}$ arrives at the pre-fault *Stage* i after ack_{i+1} goes high, preventing the complete data word from being latched. As a result, ack_i keeps low, which further ensures this complete data

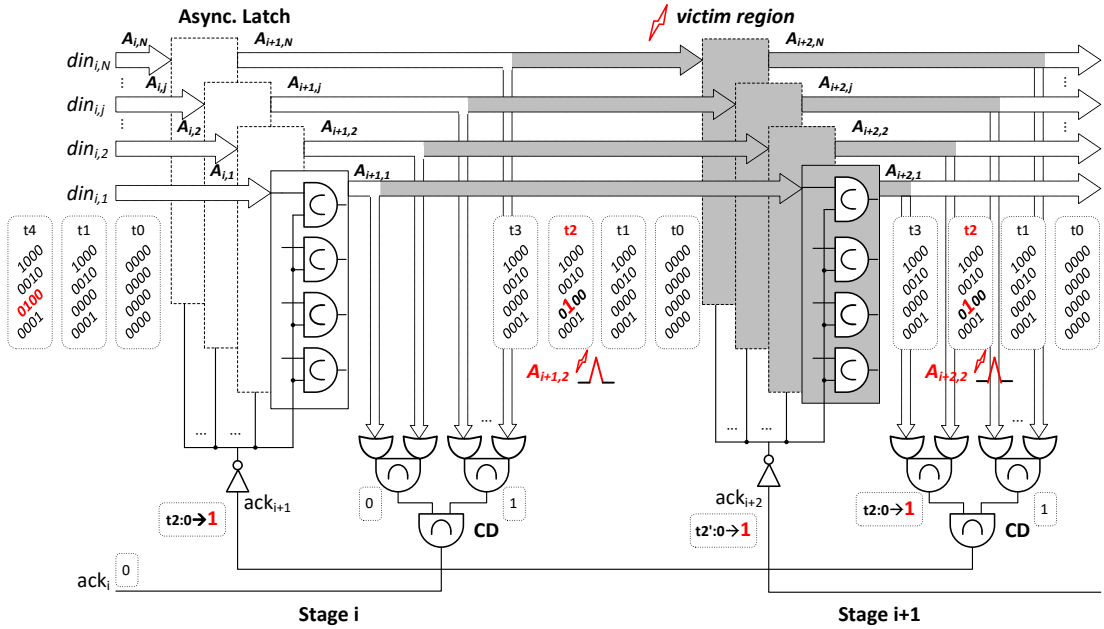


Figure 3.21: A positive transient fault on the forward path of the pipeline

($\{A_{i,1}...A_{i,N}\}$=$\{1...1\}$) to the pre-fault stage cannot be withdrawn according to (3.6), resulting a physical-layer deadlock.

Even after a short time when this positive transient fault disappears, the faulty active signal $A_{i+2,f}$ resets to '0', which traverses through all pipeline stages downstream of the fault. Since the other fault-free active signals to the CD are high, the disappearance of this fault will not trigger (3.3) and (3.4), this ack_{i+1} will be steady at '1', preventing the complete data word from being latched by the pre-fault stage.

Figure 3.21 illustrates an example that a positive transient fault on the forward data path (the shaded victim region) deadlocks the N-symbol 1-of-4 pipeline. All the 1-of-4 symbols except for the delayed one on the second channel have been latched by *Stage* i at *t1*, and output to the succeeding pipeline stages. At *t2*, a positive transient fault happens on the second channel of the forward path creates a faulty $A_{i+1,2}$ latched by *Stage* i+1 or a faulty $A_{i+2,2}$. The resulting faulty $A_{i+2,2}$ will trigger the CD of the post-fault *Stage* i+1, leading to a high ack_{i+1}, which further prevents the later coming $A_{i,2}$ from being latched by the pre-fault *Stage* i. After the fault disappears at *t3*, all stages downstream of the fault and the pre-fault stage keep holding the incomplete data word with a *spacer* on the second channel. Their *ack* signals are all '1's, while the pipeline stages before

the pre-fault stage hold fault-free data following a "complete data – spacer – complete word –spacer" pattern and their *ack* signals are alternately valued.

In this deadlock state, the input to the pre-fault *Stage* i is a complete data word, while all pipeline stages downstream of the fault (including the post-fault *Stage* i+1) hold an *incomplete* data word. Under the precondition that only a 1-bit transient fault happens, this incomplete word needs one more 1-of-n symbol to become *complete*, which is named as an "almost-full" data word. The deadlock state of the *segment* S_i can be expressed as (3.9).

$$\begin{aligned}(\{A_{i,1}...A_{i,N}\}, ack_i, \{A_{i+1,1}...A_{i+1,f-1}A_{i+1,f}A_{i+1,f+1}...A_{i+1,N}\}, ack_{i+1}) \\ = (\{1...1\}, 0, \{1...101...1\}, 1)\end{aligned} \tag{3.9}$$

2. The 1-bit transient fault is on the backward *ack* path

   A positive transient fault on the backward *ack* path could cause a persistent high ack_{i+1} directly. If ack_{i+1} goes high before data symbols arrive at the pre-fault *Stage* i, the later coming symbols cannot be latched. (3.1) and (3.2) cannot be fired, resulting a permanently low ack_i. At the post-fault *Stage* i+1, this fault should happen before the *last synchronization point* at the backward path (which is the C-element at the root of the CD tree in Figure 3.22) so that the backward ack_{i+1} is permanently high even after the disappearance of the fault. The 1-bit positive transient fault causes a fake event triggering the CD high along with other normal events, which is equivalent to the firing of (3.2). After the disappearance of the fault, since the other inputs to the synchronization point of the CD are still high, ack_{i+1} stays high permanently, preventing the later coming fault-free data symbols from being latched by the pre-fault stage. The low ack_i prevents the incoming data word to the pre-fault stage from being reset. The handshake protocol is stalled, resulting in a physical-layer deadlock.

   Figure 3.22 illustrates an example that a 1-bit positive transient fault happens on the CD of the post-fault *Stage* i+1 deadlocks the *N*-symbol 1-of-4 pipeline. At *t1*, all 1-of-4 symbols except for the symbols on the first and second 1-of-4 channels have been latched by *Stage* i, and output to all following stages. At *t2*, a positive fault happens on the CD of *Stage* i+1, leading to a high ack_{i+1} to the pre-fault *Stage* i, which prevents the later coming symbols on the first and second channels from being latched (ack_i remains low). After the fault disappears at *t3*, ack_{i+1} remains high since the other input(s) to the last C-element is (are) high,

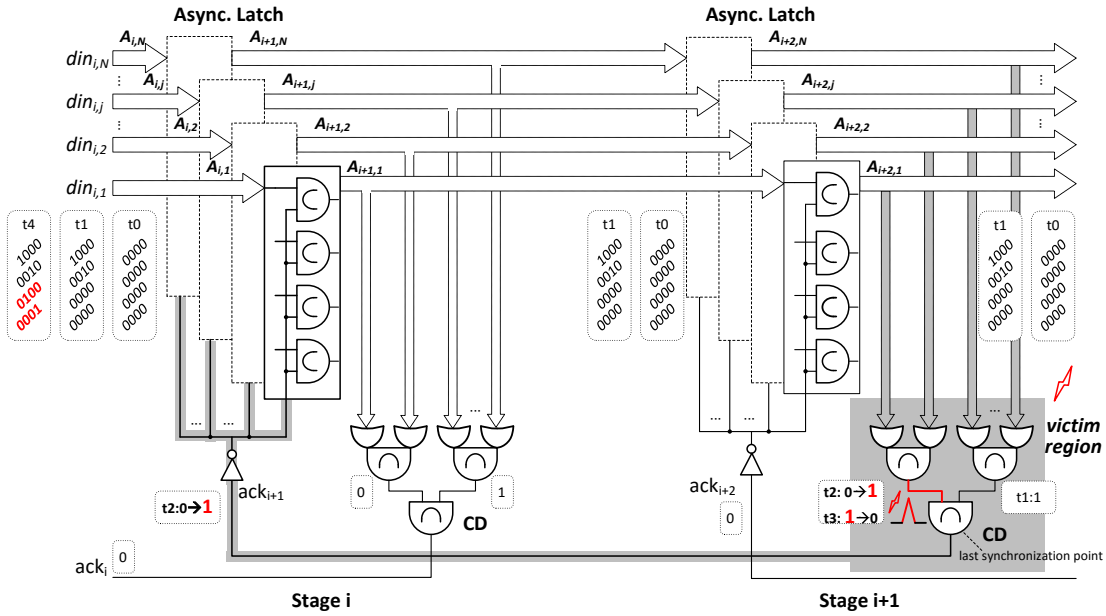


Figure 3.22: A positive transient fault on the backward path of the pipeline

resulting a physical-layer deadlock that the handshake is permanently stalled.

In this deadlock state, the input to the pre-fault *Stage* i is a fault-free complete data word. All pipeline stages before the pre-fault *Stage* i hold fault-free data following a "complete data – spacer – complete data – spacer" pattern, with alternately valued *ack* signals. Pipeline stages after the fault and the pre-fault stage hold the same incomplete data. The different points from the deadlock state caused by a transient fault on the forward path are:

(a) It is not necessary that the incomplete data held by the pre-fault stage and stages downstream of the fault are "almost full" word.

(b) Pipeline stages downstream of the post-fault stage have the same *ack* signals while the post-fault stage has a complementary *ack* signal.

The above proof also fits the case that *segment* state ($\{A_{i,1}, ..., A_{i,N}\}$, ack_i, ack_{i+1}) gets stuck at ($\{0...0\}$,1,0). A negative transient fault on the forward data path will cause an "almost empty" data, which contains only one 1-of-n symbol while the others are spacers. This "almost empty" data word is latched by the pre-fault stage and transmitted to all pipeline stages downstream of the fault. They all have the same *ack* signals. Pipeline stages before the pre-fault stage hold a fault-free data stream with alternately valued *ack* signals. The deadlock state of the victim *segment* can be expressed by (3.10). If the negative transient fault happens on the backward *ack* path (or

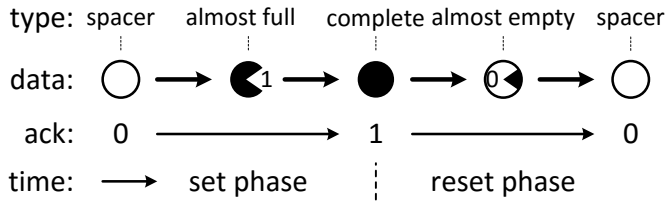


Figure 3.23: State transitions of a 4-phase pipeline stage with "almost" words

the CD), it could deadlock the pipeline as well while the incomplete word latched by pipeline stages downstream of the fault are not necessary to be "almost empty".

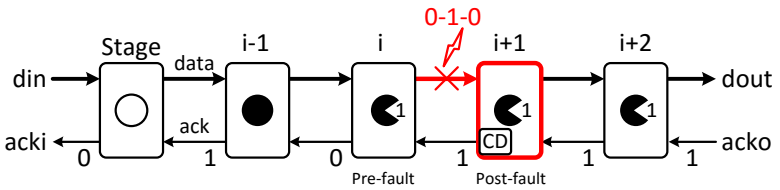
$$\begin{aligned}(\{A_{i,1}...A_{i,N}\}, ack_i, \{A_{i+1,1}...A_{i+1,f-1}A_{i+1,f}A_{i+1,f+1}...A_{i+1,N}\}, ack_{i+1}) \\ = (\{0...0\}, 1, \{0...010...0\}, 0)\end{aligned} \tag{3.10}$$

Therefore, it can be concluded that a transient fault could deadlock a multi-symbol 4-phase 1-of-n QDI pipeline. The above proofs can be extended to m-of-n ($2 \leq m < n$) pipelines, which can be deadlocked by a transient fault as well. □

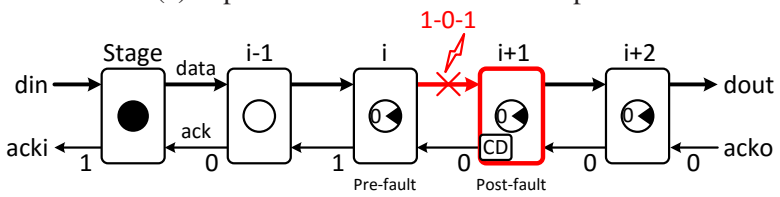
3.3.3 Deadlock analysis

For an N-symbol wide 4-phase 1-of-n QDI pipeline, five kinds of binary sets are defined as follows ($\mathbb{A}_i = \{A_{i,j} | 1 \leq j \leq N\}$), including the three defined in Section 2.1.6. The **almost_full** and **almost_empty** data words are **incomplete** which have only one spacer and one 1-of-n symbol respectively. Figure 3.23 illustrates the updated state transition graph based on Figures 2.11 and 3.15, with "almost full/empty" data words as the intermediate states between the spacer and a complete data word.

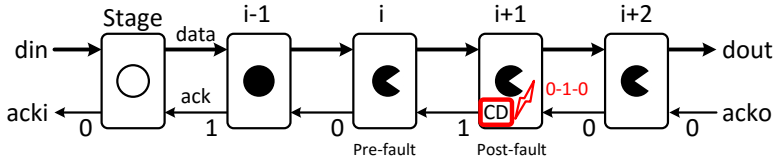
- **complete_data** ($\mathbb{A}_{i,complete}$): all elements are '1's.
- **incomplete_data** ($\mathbb{A}_{i,complete}$): this is the intermediate state between a spacer and a complete data word where some of the N elements are 1-of-n while the others are spacers.
  - **almost_full** ($\mathbb{A}_{i,almost\_full}$): all elements are '1's except for the fault-related one which is '0'.
  - **almost_empty** ($\mathbb{A}_{i,almost\_empty}$): all elements are '0's except for the fault-related one which is '1'.
- **spacer** ($\mathbb{A}_{i,spacer}$): all elements are '0's.



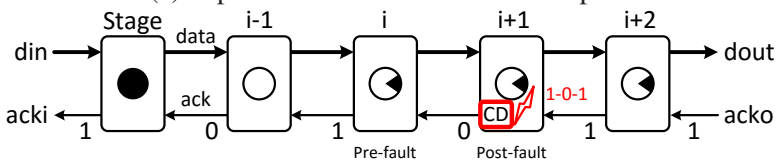
(a) A positive fault on forward data path



(b) A negative fault on forward data path



(c) A positive fault on backward ack path



(d) A negative fault on backward ack path

Figure 3.24: Deadlocked pipelines caused by transient faults

It can be concluded from Theorem 3.3.4 that possible deadlock states of an N-symbol 4-phase QDI pipeline caused by a transient fault can be classified into two kinds according to the fault position:

- If a transient fault on the forward data path deadlocks the pipeline, the pre-fault stage and all pipeline stages downstream of the fault keep the same "almost full" or "almost empty" incomplete data word, while pipeline stages before the pre-fault stage hold a fault-free data stream. As a result, any two contiguous *ack* signals after the fault are the same while the *ack* signals before the fault are alternately valued. (3.11) and (3.12) presents the concluded expression of the deadlocked pipeline *segment* while their pipeline models are shown in Figure 3.24a and 3.24b.

$$(\mathbb{A}_i, ack_i, \mathbb{A}_{i+1}, ack_{i+1}) = (\mathbb{A}_{i,complete}, 0, \mathbb{A}_{i+1,almost\_full}, 1) \tag{3.11}$$

$$(\mathbb{A}_i, ack_i, \mathbb{A}_{i+1}, ack_{i+1}) = (\mathbb{A}_{i,spacer}, 1, \mathbb{A}_{i+1,almost\_empty}, 0) \tag{3.12}$$

- If a transient fault on the backward *ack* path (before the last *synchronization point*, which is the root of the CD tree at the post-fault stage in Figure 3.22) deadlocks the pipeline, the pre-fault stage and all pipeline stages downstream of the fault keep holding the same incomplete data word, while the preceding pipeline stages hold the fault-free data stream. As a result, any two contiguous *ack* signals after the post-fault stage are the same while the *ack* signals before the faulty CD are alternately valued. (3.13) and (3.14) presents the expression of the deadlocked pipeline *segment* while their pipeline models are shown in Figure 3.24c and 3.24d.

$$(\mathbb{A}_i, ack_i, \mathbb{A}_{i+1}, ack_{i+1}) = (\mathbb{A}_{i,complete}, 0, \mathbb{A}_{i+1,incomplete}, 1) \tag{3.13}$$

$$(\mathbb{A}_i, ack_i, \mathbb{A}_{i+1}, ack_{i+1}) = (\mathbb{A}_{i,spacer}, 1, \mathbb{A}_{i+1,incomplete}, 0) \tag{3.14}$$

It can also be concluded from Theorem 3.3.4 that a "long enough" time difference between the two slowest 1-of-n channels is necessary to the production of this physical-layer deadlock. A similar observation has been made by Shi [Shi10]. T_{skew} is used to denote the data skew between the two slowest parallel 1-of-n channels. The loop latency T_{loop} between two adjacent pipeline stages can be expressed as (3.15) where $T_{forward}$ and $T_{backward}$ represent the propagation delay of the forward *data* and the backward *ack* respectively. $T_{forward}$ and $T_{backward}$ can be expressed as (3.16) and (3.17) where t_{latch}, t_{CD}, t_{data} and t_{ack} are propagation delays of the asynchronous latch, the CD, *data* wires and the *ack* wire, respectively. Thus, to ensure that a 1-bit transient fault could deadlock the pipeline, (3.18) should be satisfied as the precondition and the fault happens on the slowest channel.

$$T_{loop} = T_{forward} + T_{backward} \tag{3.15}$$

$$T_{forward} = 2t_{latch} + t_{data} \tag{3.16}$$

$$T_{backward} = t_{CD} + t_{ack} \tag{3.17}$$

$$T_{skew} > T_{loop} \tag{3.18}$$

This data skew or the time difference of data transmission between 1-of-n channels of a pipeline can be long because of many factors [BY92, RJDC98, BDF+14]. Due to the increasing chip size and more function on a chip, on-chip wiring delay is having an increasing impact on the chip performance [RJDC98]. The geometry, design constraints and manufacturing variations can cause significant data skew between different

link wires [BY92, BDF+14], which becomes common as the semiconductor technology scales. The resulting data skew becomes more easily to reach the time to transmit multiple bits, and may get augmented after a long distance. Besides these internal factors to the circuit itself, external parameter variations such as crosstalk [NKM10], worn-out wires or transistors with the ageing process [Con03], Electromagnetic interference (EMI) [KSDH00] are able to introduce data skew between the large amount of on-chip interconnects as well. Therefore, if the data skew is common in a multi-symbol channel and the time difference between the two slowest 1-of-n channels is longer than the loop latency, satisfying (3.18), the possibility that transient faults on the asynchronous pipeline cause a deadlock significantly increases. As a result, the incoming large-scale QDI NoC becomes more sensitive to a physical-layer deadlock caused by a transient fault. Data errors can be corrected in different network layers, even at the interface to synchronous circuits so that the error management process can be executed in the synchronous domain of a Globally Asynchronous, Locally Synchronous (GALS) system, which can utilise the well-studied conventional fault-tolerant techniques. However, without protection inside the asynchronous network, a fault-caused physical-layer deadlock can fail the whole system easily. Therefore, it is essential to use extra fault-tolerant techniques in the asynchronous network to reduce the fault impact (Chapter 4), and also manage the physical-layer deadlock caused by different faults (Chapters 5 and 6).

Comparing patterns of the physical-layer deadlock caused by a permanent and a transient fault (Figure 3.16, 3.17, 3.18 and Figure 3.24), it can be found that these faults cause the same deadlock patterns, which are:

1. No transitions are detected on the pipeline;
2. All pipeline stages downstream of the fault have the same *ack* while the *ack* signals in pipeline stages upstream of fault are alternately valued.

Therefore, Theorem 3.3.1 can be extended to transient faults, which is the key to detecting the fault that results in this physical-layer deadlock.

3.4 Related work

Fault-tolerance is a well-studied topic in synchronous circuits and NoCs [RFZJ13]. As discussed before, event-driven asynchronous circuits manifest much more complicated faulty scenarios than synchronous ones. Several techniques have been proposed to

protect asynchronous circuits or interconnection from transient faults, but the tolerance of permanent faults and the management of the fault-caused physical-layer deadlocks in asynchronous circuits have rarely been studied. This section introduces several fault-tolerant techniques as representatives to demonstrate the state-of-the-art work related to this research.

3.4.1 Tolerating transient faults

Section 3.1.5 discussed that (transient) faults can be dynamically detected using different redundancy techniques [Ham50, LLP07, FLJ+13]. Information redundancy techniques, or fault-tolerant codes [Sor09], are important concepts in the field of fault-tolerance. In QDI asynchronous circuits, Delay-Insensitive (DI) codes, which implicitly include timing information, are used to build data channels. Thus, information redundancy techniques can be used in QDI circuits to obtain both timing-robustness and fault-tolerance.

Information redundancy techniques

Systematic codes [AN10] (Section 4.1.1), such as parity check codes and Hamming codes [Ham50] have been widely used for fault-tolerance purposes [LLP07, LWL+10, Yu11, Pon12, FLJ+13]. Encoding them into DI codes for QDI interconnection may result in low coding efficiency [Pon12, ZSG+14b]. Among existing asynchronous designs, information redundancy techniques are often used in bundled-data rather than QDI designs to provide fault-tolerance [LLP07, LWL+10, LLP12].

The first paper that systematically addressed transient faults in 4-phase DI codes during transmission was presented by Cheng and Ho [CH01]. An unordered systematic code using a parity check code was proposed to correct 1-bit errors and detect completion simultaneously on 4-phase asynchronous links. Two error models were proposed, the first of which assumes only unidirectional errors (either $0 \rightarrow 1$ or $1 \rightarrow 0$) could happen. For the $1 \rightarrow 0$ error model, t '1's will be missing at the receiver if at most t errors can happen during the transmission, requiring a t-error correcting DI code to correct these errors. For the $0 \rightarrow 1$ model, where each error could contribute two errors at the receiver side, a $2t$-error correcting code is required under the assumption of up to t transmission errors. In the reset phase of the 4-phase handshake process, where the transmission wires may not be reset to '0's due to either an unexpected and arbitrarily long delay or permanent faults, a bounded delay model is used in these error

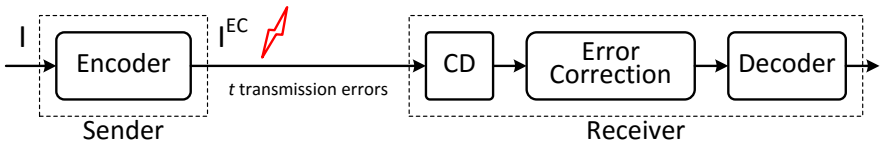


Figure 3.25: Error-correcting model proposed by Cheng and Ho [CH01]

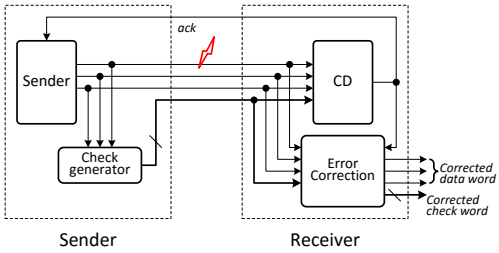
models to constrain the reset time so it is not a QDI design. Taking the $1 \rightarrow 0$ case for example (Figure 3.25), the information bits, I, are first encoded into a DI codeword I^{EC} at the sender. The DI code has a weight of n (n '1's) and supports a t-error correction capability. t errors may occur and remove t '1's during the transmission. Detecting (n-t) '1's at the receiver side, all errors can be corrected and then the original information bits decoded. Since the data are not protected during their transmission in the asynchronous interconnects, symbol insertion, deletion and corruption, even a fault-caused physical-layer deadlock can happen, which can paralyse the whole communication. No hardware implementation was disclosed.

Similarly, a parity check was used to implement the error detection and correction in QDI communication [Pon12]. Taking a 8-bit binary vector "01011100" for example, it is equivalent to a data word with four 1-of-4 codes ("0010", "0010", "1000" and "0001"). The produced parity bits at the sender are $0010 \oplus 0010 \oplus 1000 \oplus 0001 = 1001$, which are further transformed to two 1-of-4 codes "0100" and "0010" to transmit. At the receiver, parity bits are regenerated and compared with the transmitted ones through the XOR operation. The bit position of the faulty code can be decided and the faulty bit is further corrected. However, this method can only resolve symbol corruption on data wires. If the transmitted parity bits are faulty, error correction is disabled. Any faults in the data field will be transmitted to the destination. Since the NoC itself was not protected, the redundancy added to network interfaces incurs 11.6% area overhead and 8.1% speed overhead. The achieved fault-tolerance is limited.

Another unordered systematic code named Zero-Sum [AN10, AN12] was proposed to protect 4-phase asynchronous links. As Figure 3.26 shows, each bit position of a Zero-Sum code is assigned an index – check bits are indexed by powers of two while indices of data bits occupy the remaining consecutive position. The check word is the binary representation of the sum of indices of data bits whose values are '0's. Figure 3.26b presents the Zero-Sum communication system. The check word is generated at the sender and then transmitted to the receiver with the data word. Once a valid codeword is detected by the Completion Detector (CD) at the receiver, the error correction

indices	data word		check bits			
	5	3	8	4	2	1
	0	0	1	0	0	0
	0	1	0	1	0	1
	1	0	0	0	1	1
	1	1	0	0	0	0

(a) Zero-Sum code with 2 data bits



(b) Protected communication

Figure 3.26: Zero-Sum codes [AN12]

unit recalculates the check field C' and compares it with the incoming check word C. If their absolute difference $|C-C'|$ is zero, it is assured no error has occurred. Otherwise, the value of $|C-C'|$ represents the index of the erroneous bit. A time-out mechanism is used to force the erroneous data which cannot be detected by CD to be corrected and latched by the receiver to ensure forward progress, so that it is not a QDI design. Carefully checking the codes, it can be found that a 1-bit fault could mislead the error corrector to decode a wrong code word: assuming that a transient fault mutates the data word "10" into "11" during its transmission while the transmission of its check word "0011" is delayed (Figure 3.26a). The erroneous "11 0000" will be incorrectly regarded as a valid code without error correction, even if the check word arrives at the receiver later. Extensions, Zero-Sum$^+$ and Zero-Sum* [AN12], have been proposed to improve the error detection and correction capability, but with the same problems mentioned above and a larger hardware overhead.

The work of Lechner et al. [LSH15] studied the general fault-tolerance properties of DI codes and proposed methods for improving the fault resilience of 4-phase DI asynchronous communication links. In DI or QDI communication which is free from timing assumptions of signal delays, signal transitions on long wires may appear at any time and in any order at the receiver. In a faulty environment, rising transitions caused by faults can make an intermediate transmission state of a DI code into a complete data word latched by the receiver. Taking a 2-of-4 code "0011" for example, one possible intermediate transmission state could be "0001", which can be mutated into "1001" by a positive fault and further be mistaken into a valid 2-of-4 code at the receiver. Resilient sub-codes can be built from the original DI codes to avoid this issue. It is declared that two codewords $x = (x_{n-1}, x_{n-2}, ..., x_0)$ and $y = (y_{n-1}, y_{n-2}, ..., y_0)$ *overlap* at bit position i if both x_i and y_i equal '1'. If the number of '1's in two codes – which are not the overlapping bits – is larger than f, it can be inferred that f faults cannot change an incomplete code word into a complete one. Resilient sub-codes are a set

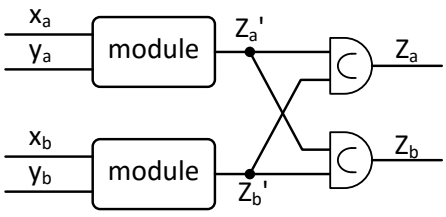


Figure 3.27: Double-checking protection [JM05]

of DI code words where any two of the code words satisfy this condition. {0011, 1100} are resilient sub-codes of 2-of-4 codes tolerating one bit transient fault. {10010, 01100} are sub-codes of 2-of-5 codes tolerating two bits transient faults.

The above resilient sub-codes are immune to a specific number of transient faults and no detection/correction is required. The price is the extremely poor coding efficiency due to the loss of code space. To improve the coding efficiency, a two-step data encoding method was proposed [LSH15]: at the sender, an error detection coding method first adds enough bit redundancy to the original data, which is then encoded into DI codes transmitting to the receiver. To achieve an f-bit fault-tolerance capability, the mapping between the error detection codes and the DI codes must ensure that f transient faults on the transmitted DI codes will cause at most t-bit errors on the decoded error detection codes, which can be detected. Producing this mapping requires complex processes in searching a graph. No hardware details were disclosed.

In addition, using multiple fault-tolerant methods, a self-timed bundled-data link is able to tolerate transient and permanent faults [LLP07, LWL+10]. Hamming coding and interleaving is used to detect transient faults while a retransmission is used to correct errors. The extra de-interleaving and Hamming decoding process introduces a large area overhead and it is not a QDI design. Lechner et al. [LLP12] proposed a robust asynchronous interfacing scheme for Globally Asynchronous, Locally Synchronous (GALS) systems. The interface performs a transformation between bundled-data and dual-rail encodings. Parity bits are employed to do error-correction. Delay elements are used to guarantee the correct operation of the code generator, error detection and correction. This is not a QDI design either.

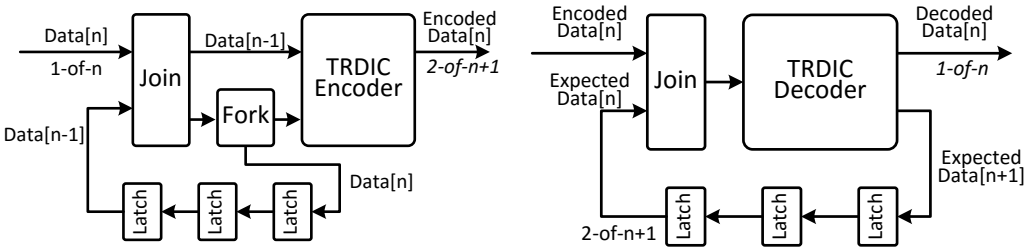
Physical and other redundancy techniques

Many physical and temporal techniques have been proposed to provide fault-tolerance for asynchronous circuits or on-chip communication. As for QDI designs, Jang et

al. [JM05,Jan08] proposed a physical-redundant double-checking technique which tolerates 1-bit faults and some multi-bit faults. As Figure 3.27 shows, one module can be a basic logic gate or a combinational circuit, which is duplicated. The variables $\{x_a, x_b\}$, $\{y_a, y_b\}$ and $\{Z_a, Z_b\}$ are copies of the variable x, y and Z respectively. A transient fault on inputs may result in a glitch on one of the *check-in* variables $\{Z_a^{'}, Z_b^{'}\}$, which can be filtered by the double-checking C-elements. Two extra weak C-elements can be added to the two double-checking C-elements respectively to tolerate accumulated soft errors (or Multiple-Event Upset). The main difference from other communication-centric fault-tolerant techniques is that it can be used to build computational circuits. Compared with an unprotected QDI circuit, the area of the double-checking protected one is enlarged by a factor of between two and three, and the throughput is reduced between 40% and 50%. This double-checking technique has also been used in several other fault-tolerant asynchronous designs [ASLM09, IY11, LHHA14].

Similar to the double-checking technique [JM05], Almukhaizim et al. [ASLM09] proposed replication-based soft-error-tolerant solutions for Asynchronous Burst-Mode Machines (ABMM). Both the TMR-based [Sor09] and duplication-based [JM05] techniques utilising the inherent functionality of C-elements were explored to improve the circuit robustness. Considering the large area overhead due to a pure duplication of the original circuit ($\geq$100% of the cost), three partial duplication options were explored to protect sensitive gates or blocks, requiring 87%, 60%, and 50% of its cost while providing 68%, 47%, and 24% of its transient-error tolerance respectively, providing a trade-off between the area overhead and the error reduction in ABMMs.

Monnet et al. studied the sensitivity of QDI circuits to transient faults [MRL04, MRL05a, MRL06], and proposed a rail synchronization technique between multiple parallel channels to filter glitches [MRL05b, MRL06]. Compared to the duplication-based techniques proposed by Jang et al. [JM05] and Almukhaizim et al. [ASLM09], duplication is avoided so that the area overhead is reduced. The synchronization between two channels, for example, ensures that in one channel data symbols cannot be latched without the presence of valid data symbols in the other channel, thus most transient faults can be filtered at the synchronization point. When there is only one circuit without a redundant channel to be synchronized with, a redundant control circuit is required to synchronize the original channel. A delay line compromising the DI feature is required. Some minor timing assumptions are also required between synchronized channels. Some faulty scenarios (such as a 1-of-2 code “01” changed to a faulty “10”) and the possible deadlock cannot be resolved. The incurred area overhead



(a) Encoder (b) Decoder

Figure 3.28: TRDIC communication system

depends highly on the circuit architecture to harden. A hardened asynchronous Data Encryption Standard (DES) crypto-processor is 7.7% larger and 18% slower than the unprotected one. A robust C-element latch scheme using a similar synchronization technique was proposed [GYB07] but also requires a number of timing assumptions.

A duplication and recalculation technique was proposed [FS09,FPD09] for 2-phase 1-of-2 QDI circuits. The original channel is duplicated and a checker samples data from the original pipeline stage and the replica to detects errors. Data transmission will be stalled until the preceding combinational circuit recalculates a fault-free result. A cross-coupling technique was proposed to ensure that the circuit will not proceed until the error is resolved. The Register Transfer Level (RTL) simulation results were presented, which demonstrated that the proposed techniques can tolerate data errors but will fail in the presence of physical-layer deadlock.

Julai et al. [JYB12] proposed a fault-tolerant latch architecture which duplicates the original data rail so that XOR gates can compare the two "identical" rails and detect faults. Similar to the Razor flip-flop technique [EKD$^+$03] for synchronous pipelines, a shadow latch controlled by a delayed clock is added to support the error correction. A multiplexer at the output will choose either the main latch as the output if no faults are detected or the shadow latch if faults are detected. Many timing assumptions are required in the latch so it is not a QDI design.

Temporal redundancy has been used to tolerate transient faults on 1-of-n QDI pipelines. As Figure 3.28a shows, the Temporally Redundant Delay Insensitive Code (TRDIC) system [PCV12] converts 1-of-n DI codes to 2-of-n+1 codes at the sender, which are transmitted to the QDI data links. The conversion from 1-of-n to 2-of-n+1 code can be easily implemented using a bitwise OR-operation between the previous and the current 1-of-n codes while the most significant bit indicates if the two consecutive codes are the same. In this way, two code words are transmitted inside

the 2-of-n+1 code every time and every code word is transmitted twice. At the receiver (Figure 3.28b), using a feedback loop of asynchronous latches (which exist at the sender as well) to store the code word based on the first transmission (which is the next expected data), faults occurring during the second transmission of the same code word can be filtered by comparing the previously stored one and the new incoming one. Double checkers built from C-elements [JM05] are used at the receiver which could filter most transient faults. Without including the cost of TRDIC encoder and decoder, the area of a protected 2-of-5 pipeline is around three times larger than unprotected 1-of-4 pipeline. The throughput decreases 20%. If the decoding/encoding component is considered, the hardware cost must be large; it was not revealed. As mentioned by the authors, some transient faults cannot be tolerated with the simple double checker, such as an insertion of a valid code word. A three-stage trellis structure was used in an extended version to improve fault-tolerance [Pon12]. No implementation details were given but the area/speed overhead is expected to increase further.

Ogg et al. [OAHY08] proposed a technique utilizing phase relationships between multiple 1-of-2 data symbols and a redundant reference symbol to achieve the transient resilience of asynchronous links. The generation of the reference symbol follows the sequence of "00" $\rightarrow$ "01" $\rightarrow$ "11" $\rightarrow$ "10". If a data symbol is the same as the reference, they are "in-phase" and a '0' bit is decoded. If a data symbol is complementary to the reference, they are "180° out of phase" and a '1' bit is decoded. A data symbol satisfying other phase relationships ($\pm 90^\circ$) will not be latched. However, this technique is time-dependent since wire delays can violate the normal phase relationships between the data and reference symbol. A faulty data symbol satisfying the phase relationship can be decoded into an incorrect value. For example, to transmit a bit '0' when the reference symbol is "01", the received data symbol should be "01" in the normal case. Any faults which change the data to "11" or "00" will not be latched or decoded, but a temporarily faulty "10" occurring before the right data symbol will be incorrectly latched and decoded into '1'. It is not a QDI design.

Considering Null Convention Logic (NCL) designs [SF01], Kuang et al. [KXIZ07] studied the fault-tolerance of NCL QDI circuits and proposed that reducing the sensitivity duration of the NCL logic to glitches could significantly suppress the soft error rate. In its extension [KZYD10], a Schmitt trigger [SNI06] implemented at transistor level was used to prevent most glitches from being latched into transient errors. Besides, to protect combinational blocks, extra latches, detection and reset circuits were

added. When erroneous transitions are detected, the reset circuits clear the combinational circuits. However, these redundant circuits were assumed to be fault-free without any protection. Delay elements were used to mitigate some fault scenarios, compromising the timing-robustness feature of the circuits. Mosaffa et al. [MJM11] proposed techniques to improve the robustness of NCL gates to transient faults by building robust threshold gates at transistor level, without being sensitive to transistor sizing and load capacitance. These robust threshold gates were later used in a Macro Synchronous Micro Asynchronous (MSMA) pipeline proposed by Lodhi et al. [LHHA14] to achieve fault-tolerance.

In addition, Bainbridge et al. [BS09] proposed a series of techniques to protect QDI pipelines from transient faults, which can be classified into: 1) preventing a glitch from causing an effect, 2) mapping the consequences of a glitch onto a different consequence that is easier to detect/recover from and 3) reducing the window of sensitivity to a glitch. However, none of them is robust enough to qualify as fault-immune. The transient-fault effects on different kinds of implementations of C-elements have been evaluated by means of fault-injection simulations at transistor level [BSK+10].

Although the above techniques seem promising, they cannot be easily migrated to protect QDI links or the large number of existing QDI NoCs without modifying most of the original circuit or disturbing the timing-robustness nature. It can be summarized from the above techniques that the fault-tolerance and timing-robustness are difficult to achieve at the same time. Many existing techniques improve the circuit robustness but do not qualify as fault-immune. A trade-off should usually be made between the incurred hardware/performance overhead and the achieved fault-tolerance capability, so that a flexible protection strategy depending on the practical design requirement is valuable. Chapter 4 will propose a general fault-tolerant coding scheme to protect QDI interconnects from transient faults according to the fault-tolerance requirement while keeping the timing-robust nature, providing a flexible way to enhance the robustness of QDI links or NoCs.

3.4.2 Tolerating permanent faults and managing deadlocks

Conventional fault detection and recovery techniques

Transient faults may corrupt the transmitting data, which can be either detected or corrected on-the-fly so that no further recovery operation is required. If the fault is detected but not corrected, the end-to-end retransmission is a popular way to avoid

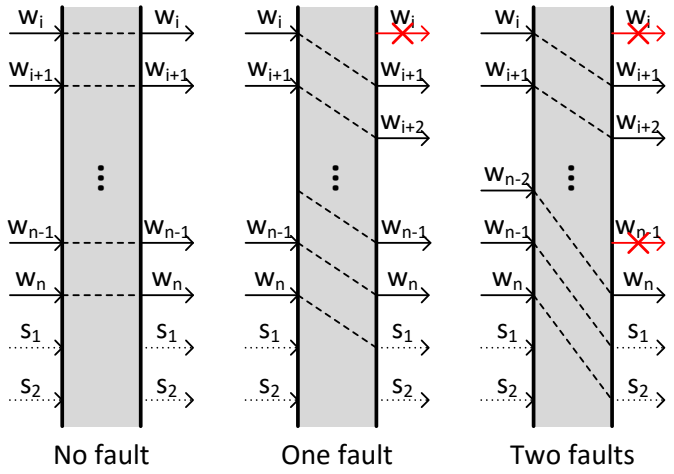


Figure 3.29: Spare wire replacement example

packet loss in NoCs, which, however, works for transient faults only. Permanent faults can be checked off-line using self-test mechanisms, such as scan chains [TTD+09], or detected on-line by searching for persistent errors using fault-tolerant coding schemes [LLP07, LWL+10, FLJ+13]. As discussed in Section 3.3.1, if the collected error syndromes satisfy specific patterns or some time-out conditions, the fault is taken as permanent and the recovery process starts. Although permanent faults cannot be eliminated, a network may survive with redundant or reduced resources, which is important to critical digital equipment. Based on the assumption that the permanent fault has been detected and located, many recovery techniques have been proposed [RFZJ13], including spare wire replacement [LLP07, YA10, LWL+10], splitting transmission [LLP07, ZYA12, YA10] and fault-tolerant routings [RFR+10, IY11, FLJ+13].

Spare wire replacement [LLP07, YA10, LWL+10] is a physical-redundant technique substituting extra wires for faulty ones, whereas in the normal case these spare wires are unused. Figure 3.29 illustrates the function of the reconfiguration unit implementing the switching between faulty and spare wires. If the output wire w_i is defective in the one-fault case, all input wires numbered no less than i will be redirected to new outputs numbered with an increment of one. The two-fault case is also depicted in Figure 3.29. Complicated control circuits are usually required to implement the wire switching. Splitting transmission [LLP07, ZYA12, YA10] divides a physical link into several groups and one flit into the same number of parts. When a link wire of a group is diagnosed as defective, the corresponding group is discarded and the other groups still work. As a result, transmission of one flit is split into two transmissions;

the discarded group will be compensated by a group of wires in the second transmission. Data packets are disassembled at the transmitter and reassembled at the receiver side to obtain the original data in a faulty environment. A small group size provides high bandwidth and throughput in the presence of faults, but incurs more complicated control logic, consuming significant area and power. As the group size increases, the network throughput decreases as more wires are abandoned in case of a faulty wire; so does the complexity of the control logic. Note that splitting the transmission, which dynamically configures the link for one flit when a link wire is found defective, is different from the Spatial Division Multiplexing (SDM, Section 2.2.6) which initially divides one link physically into multiple independent sub-links for different packets. Splitting transmission is a network-layer or data-link-layer technique. After a link wire is found defective the following flit should be disassembled to fit the remaining link width and reassembled at the link end. Differently, the SDM applied in this research is a physical-layer technique which provides multiple narrow physical channels on each link. The flit width is adjusted to the width of each sub-link before a packet enters the network so that no disassembly or assembly operation is required even if a permanent fault removes a sub-link. Fault-tolerant routings have also been widely used to recover the network function in the network layer [RFR+10, DT03, IY11, ZYA12, FLJ+13]. It will detour the located faulty link and choose an alternative, healthy one to carry data. A defective router can also be bypassed if its location is known. However, using only the network-layer fault-tolerant routing cannot manage a fault-caused deadlock in the lower physical layer.

As an instance, Lehtonen et al. [LLP07] used Hamming codes [Ham50] to detect transient faults and retransmissions to correct errors in an asynchronous bundled-data NoC. Similar to the detection of permanent faults in synchronous NoCs, error syndromes are recorded when detecting transient faults. A number of equivalent syndromes indicate a permanent fault. By decoding syndromes, the position of the permanent fault is located. Spare wires and half-splitting transmission were explored to bypass permanent faults respectively. However, this bundled-data design is similar to a synchronous network rather than QDI.

The later work of Lehtonen et al. [LWL+10] proposed an in-line test method to detect permanent faults in synchronous on-chip interconnects. A test pattern generator at the sender injects test vectors to each adjacent pair of link wires periodically. An error detection unit at the receiver can tell if a permanent fault happens by checking the received test outputs against specific patterns. This in-line test method was later used

in a transient and permanent faults co-management synchronous NoC [YA10] to detect permanent faults. Another detection method using Hamming codes was also proposed [LWL+10]. When one or multiple faults are detected by the fault-tolerant coding scheme, retransmissions are requested to transmit data through suspicious wires. If collected error syndromes satisfy specific patterns, faults are taken as permanent and the recovery process is executed; otherwise, the faults are transient. Spare wires are used to recover from permanent faults.

Feng et al. [FLJ+13] proposed an on-line fault diagnosis mechanism for synchronous NoCs. Transient faults are detected and corrected using Hamming codes [Ham50]. At the receiver, a 1-bit error will be directly corrected no matter what type of fault causes it. If a 2-bit error is detected, a retransmission is requested. If the same 2-bit error happens again in the retransmitted packet, the router enters a test mode and a group of test vectors are transmitted through the suspicious link multiple times. By checking the correctness of the received test vectors, permanent faults can be diagnosed. A fault-tolerant routing algorithm was proposed to bypass defective links.

It should be noted that, the above conventional detection & recovery techniques for synchronous circuits or NoCs cannot work for deadlocked asynchronous NoCs easily. In a deadlocked state, conventional techniques using fault-tolerant codes cannot obtain error syndromes any more, making it difficult to detect and locate the faulty component. In terms of the network recovery, the aforementioned recovery techniques (spare wire replacement, splitting transmission and fault-tolerant routings) try to restore the network function in the data link layer or network layer (Chapter 2.2.2); they cannot resolve the fault-caused deadlock occurring in the lower physical layer. Only after the physical-layer deadlock (or the broken handshake protocol) is resolved, can they be employed to further recover the network function. Therefore, considering the case that both transient and permanent faults can deadlock the asynchronous NoC, a new, effective detection and recovery technique targeting on the fault-caused physical-layer deadlock is necessary.

Existing research on fault-caused physical-layer deadlocks

Existing research on permanent faults is mostly on synchronous circuits or NoCs [RFZJ13]. There is rarely research on the management of permanent faults in QDI NoCs. This chapter demonstrated that both transient and permanent faults can break the handshake protocol and cause physical-layer deadlocks, which is fatal to the network. Thus, the topic of tolerating a permanent or transient fault is transferred to the

detection and recovery of the physical-layer deadlock. It is straightforward that a permanent fault could deadlock a QDI circuit, which has been studied [MH91, DGY95, HBB95, JM05] but not under the context of NoCs. The statement that a transient fault could stall the handshake process was mentioned in [LM04, Jan08, Shi10] without further exploration. In a deadlocked state, traditional information redundancy [CH01, AN12, ZSG+14b] or duplication-based [JM05, MRL06] fault-tolerant techniques are stalled as well and cannot work.

To detect the fault in asynchronous circuits, Concurrent Error Detection (CED) or self-checking techniques have been proposed to provide a circuit with the capability to monitor its own function and report potential deviations from correct functionality [VM02]. Rennels and Kim [RK94] studied the CED in asynchronous Differential Cascode Voltage Switch Logic (DCVSL). Fault effects on different DCVSL elements were investigated. A checker using a time-out counter was added to the outputs of the original circuits to signal errors. The fault-caused deadlock caused by stuck-at faults can be detected using the time-out mechanism. Traditionally, duplication-based CED techniques have been proposed to detect faults in synchronous circuits. Verdel and Makris [VM02] pointed out that the difficulty in duplication-based CED for asynchronous circuits is the lack of a global synchronization mechanism (i.e. the clock) so that it is unclear when the outputs of the original and the replica circuit are expected to match. A comparator circuit attached to two identical D-elements was proposed with a comparison synchronizer. A custom delay line defines a time window. If the outputs of the original and the replica circuits are the same in the window, the circuits are operating correctly. Otherwise, either of the circuits is operating too fast (a premature firing) or too slowly (or a halt in the presence of stuck-at faults). Extra circuits are required to differentiate different fault effects. Applying a fault-tolerant coding scheme [Don84] and duplication, Hyde and Russell [HR04] proposed a CED architecture for bundled-data asynchronous Reduced Instruction Set Computer (RISC) processors. 12% area overhead was caused by the CED technique while a fault coverage of 98.5% of all unidirectional errors was achieved. Significant delay assumptions are required in the design of the asynchronous CED processors. David et al. [DGY95] studied the detection of a stuck-at permanent fault on asynchronous circuits and proposed self-checking designs which either stop operating or produce illegal outputs when faults happen. The halt condition can be detected using a simple watchdog timer. Mathematical proofs were given in details while the fault detection circuits were not provided. The self-checking property of asynchronous circuits with respect to permanent faults was also

studied in several publications [PN95, YCCP03, PFS10].

With respect to asynchronous NoCs, Tran et al. [TTD+09, TVC10] proposed a Design-for-Test architecture to improve the testability of an asynchronous NoC, which has been illustrated in Figure 2.29. Each asynchronous router is surrounded by a test wrapper controlled by a local control module. A 2-bit configuration chain is built to connect serially the wrapper control modules of all routers so that test vectors can be inserted to the network and test results can be achieved. The whole test process is controlled by a Generator-Analyser-Controller (GAC) unit. Using the single stuck-at fault model, the proposed test approach can achieve 99.86% test coverage. The area of a test wrapper is 32.7% of the router area. Considering the whole chip, the incurred area overhead is about 3 – 5% of the chip area depending on the chip size. Since the handshaking loop between the input and output of the router is the critical path, where redundant circuits do not reside, the network throughput does not degrade.

Shi [SFGP09, Shi10] investigated the fault sensitivity of C-elements, 4-phase and 2-phase pipelines. It has been mentioned in Section 3.3 that the physical-layer deadlock due to transient faults (which is studied in this research) was also observed by Shi [Shi10] with several deadlock scenarios enumerated. The occurrence condition of this deadlock was also explored. However, no formal verification was conducted. Without resolving this deadlock issue, the fault-tolerance of an asynchronous interface circuit between the 4-phase, 3-of-6 on-chip link and the 2-phase, 2-of-7 off-chip link was researched as the main contribution [SFGP09, Shi10]. The interface implements the protocol transformation between the two asynchronous domains, where transient faults can also cause deadlocks (which is different from the one in general QDI pipelines studied in this research). Several methods were proposed to reduce or eliminate the deadlock, including a novel phase-insensitive 2-phase to 4-phase converter, a priority arbiter for reliable code conversion and a scheme that allows independent resetting of the transmitter and receiver to clear deadlocks. In some cases an over-length packet due to faults may arrive and occupy all available buffer space, which can be detected using flit counters. The results show a reduction in deadlocks with 4% larger area compared with the baseline.

Peng and Manohar [PM05, PM06, Pen06] proposed a failure-detection technique for QDI circuits which can achieve “fail-stop” with respect to both transient and permanent faults. It utilizes the deadlock behaviour in the faulty circuit to realize the fault detection and online recovery. Instead of comparing the results from redundant computations or modules to detect the fault, by adding appropriate redundant logic at

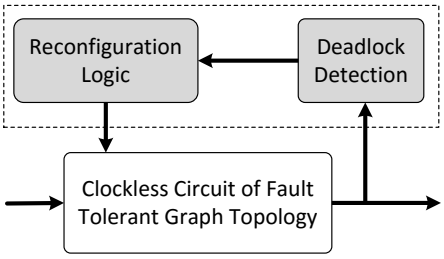


Figure 3.30: A reconfigurable self-healing asynchronous circuit [Pen06]

transistor level, the QDI circuit will stall permanently in the presence of any faults, resulting in a "deadlock". This deadlock can be discovered using a detector, as Figure 3.30 shows. Controlled by a timer implemented as delay lines, the deadlock detector keeps monitoring the handshake activities of the data channel. A transition occurring on the data channel can be detected, which starts the timer. If the next transition or protocol state does not occur for a long time, the detector assumes that the circuit has deadlocked. To recover the circuit function, self-reconfiguration logic is implemented as synchronous circuits. It uses the time-out signal from the deadlock detector as the clock signal and try all possible valid configurations to find a workable one. To support multiple configurations, each module of the system should be built on a fault-tolerant graph (Figure 3.30) with spare resources corresponding to different configurations. Thus the whole system achieves self-checking and self-healing. The "fail-stop" augmentation logic increases the hardware cost of asynchronous adders by 92%, which is close to the price of duplication. The circuit speed is reduced by around 30%. For the weak-condition half buffer, the "fail-stop" increases the area by 200%, while the speed decreases by 36%. The overhead of the deadlock detector and the reconfiguration logic was not given. In addition, many custom delay lines are required inside the deadlock detector to detect different signal transitions. The proposed technique is not easy to use in standard cell designs.

Imai and Yoneda [IY11, YIO+12] proposed a time-out detection and a recovery mechanism in a self-timed NoC using QDI inter-router links. A delay line is used to detect the abnormal data skew among the data wires of the same pipeline stage, so that a lost data bit due to a permanent fault can be detected. However, this method does not work in a pure QDI NoC. Since self-timed bundled-data pipelines are used inside the router and they do not latch incomplete data words, the fault-caused partial or incomplete data (which leads to the large skew) is isolated in one inter-router link only, which can be detected by the time-out mechanism and reported. However, in a pure QDI NoC, this faulty incomplete data word will propagate to all pipeline stages

downstream of the fault. As a result, all the downstream stages are timed out and multiple faults are reported. The fault position cannot be located. This issue also lies in most of the above aforementioned fault detection techniques. These methods cannot be used easily in QDI NoCs and they cannot locate the faulty component [PM05]. As a result, an expensive system reboot may be used to eliminate the deadlock and recover the network after the system deadlock is detected. For a permanent fault, even the system reboot cannot restore the system. A fine-grained recovery mechanism was proposed [IY11, YIO+12] though it was not implemented in the final chip: incoming flits to the faulty link are immediately accepted and acknowledged while a pseudo tail, generated at the end of the faulty link, releases all occupied channels along the deadlocked path. It was designed for an asynchronous NoC with self-timed bundled-data asynchronous routers (which is much like a synchronous circuit) and Level Encoded Dual Rail (LEDR) encoded inter-router links, which cannot work in a QDI NoC. Fault-tolerant routing was further used to recover the network function.

It can be summarised that, the self-checking or fail-stop feature of asynchronous circuits in the presence of faults has been studied to detect faults or the "deadlock". Time-out mechanisms have been proposed to provide a time reference for the fault detection though most early studies did not present any implementation details. An asynchronous circuit can even be forced to "deadlock" in the presence of faults to enable the fault detection. However, most of these techniques target on specific circuits and cannot work in QDI NoCs. In a QDI NoC, a fault can break the handshake protocol, resulting in a long, deadlocked packet path (Figure 1.4b). In this case, most above techniques [TTD+09, TVC10, IY11, YIO+12], may report multiple deadlocks along the whole packet path so that it is difficult to locate the fault position. As all network resources on this packet path cannot be allocated to other fault-free packets, the fault-caused physical-layer deadlock may cause more packet stalls that spread over the whole network, in which case the physical-layer and network-layer deadlocks may coexist, making the fault or deadlock detection more difficult. The recovery of this deadlocked network is difficult since the handshake protocol is broken in the physical layer and the faulty packet path is stalled. In terms of network recovery, a system reboot can release all deadlocked, fault-free network nodes which can be too expensive for a critical system. In some cases this reboot is not allowed [AIKR13]. The aforementioned traditional fault recovery techniques targets recovering the network function in the data-link or network layer; they cannot resolve this physical-layer deadlocks. Therefore, it is necessary to propose an on-line deadlock detection technique to isolate

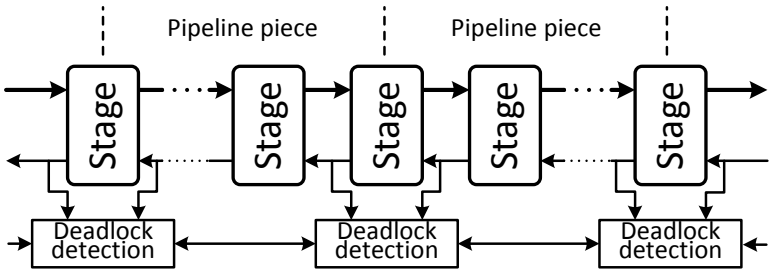


Figure 3.31: Protected pipeline pieces with deadlock detection circuits

the faulty component in QDI NoCs, so that further custom fine-grained operations can be used to recover the deadlocked NoC at runtime according to the employed asynchronous and network protocols, without disturbing other fault-free components too much (Chapter 6).

3.5 General deadlock management strategy

It has been discussed that avoidance and recovery techniques can be used in NoCs to deal with network-layer deadlock (Section 2.2.5). Since the physical-layer deadlock invalidates the avoidance techniques implemented in the network layer, recovery techniques are employed in this research to manage fault-caused physical-layer deadlocks. The management of the physical-layer deadlock can be divided into two phases: *detection* and *recovery*. When both permanent and transient faults are considered, *fault diagnosis* is required after the deadlock detection to diagnose the fault type so that different recovery methods can be used to recover the network.

- *Deadlock detection* phase, during which the fault-caused physical-layer deadlock should be precisely located and differentiated from a network-layer deadlock or a temporary stall due to congestion;
- *Fault diagnosis* phase, during which the fault type is diagnosed when both transient and permanent faults are considered.
- *Deadlock recovery* phase, during which the deadlock should be removed first. If this deadlock is caused by a permanent fault, defective components should be isolated and the network continues working; if it is caused by a transient (or intermittent) fault, the isolated component should be recovered to use.

Figure 3.31 presents a long QDI pipeline model cut into multiple small sections, each of which contains multiple pipeline stages. Adding detection circuits to the two

terminal stages of a pipeline piece to monitor their activities and check their *ack* signals, the fault-caused physical-layer deadlock can be detected and differentiated from the others (including the network-layer deadlock or temporary stall due to congestion). Under the assumption of a 1-bit fault each time, only one pipeline piece could satisfy the deadlock pattern mentioned in Theorem 3.3.1 so that the fault position can be precisely located. A time-out mechanism can be applied in the detection circuits to monitor the switching activity of each pipeline piece. If no switching activities are detected for a "long" period and the *ack* sequence satisfies the deadlock pattern in Theorem 3.3.1, the faulty pipeline piece is found. The detected fault can be transient or permanent. If both transient and permanent faults are considered, the fault type should be diagnosed first and then different recovery methods are used. Details will be discussed in Chapters 5 and 6.

3.6 Summary

This chapter introduced different kinds of faults and fault sources. Some traditional fault-tolerant techniques were also introduced. Since faults on QDI asynchronous pipelines can cause much more complicated effect than faults on synchronous circuits, and the fault-tolerance of QDI circuits has not been fully researched, this chapter systematically modelled and studied the faulty scenarios of QDI pipelines. It demonstrated that faults on QDI pipelines could not only cause data errors, including symbol insertion, deletion and corruption, but also disrupt the handshake process, leading to a physical-layer deadlock. A permanent fault can deadlock a QDI pipeline. Transient faults can also deadlock QDI pipelines, which is a serious issue and new challenge that has been ignored by the asynchronous community. This chapter proved that a transient fault can cause a physical-layer deadlock as well if several conditions are satisfied. Finally, this chapter analysed the resulting deadlock patterns caused by both permanent and transient faults, and proposed a general deadlock management strategy.

Since a random transient fault can cause a physical-layer deadlock, which can easily paralyse the QDI NoC, it is necessary to add fault-tolerance to the asynchronous network to reduce the fault effect. The next Chapter 4 will discuss the protection of long QDI interconnects from transient faults to filter most of the fault effect.

Chapter 4

Protecting QDI links with fault tolerant codes

As an important issue in the design of asynchronous NoCs, the large number of long inter-router link wires, which are exposed to various environmental noises and fault sources, are susceptible to delay variations and transient faults [BS09]. Quasi-Delay-Insensitive (QDI) circuits [SF01] are a family of asynchronous circuits that tolerate all delay variations but they are vulnerable to faults. Occasionally, an erroneous signal transition may be accepted as a valid signal, producing a fault in QDI circuits. Fault-tolerant codes [Sor09], which are important concepts in the field of fault-tolerance, have been widely used to protect interconnection communications. Coding performs an important role in QDI asynchronous circuits as well where Delay-Insensitive (DI) codes are used to build data channels, achieving timing-robustness.

This chapter presents a novel Delay-Insensitive Redundant Check (DIRC) coding scheme to protect QDI links from transient faults. The DIRC coding scheme tolerates all 1-bit transient faults and some multi-bit transient faults. It can be easily adopted in all existing 1-of-n QDI pipelines to provide fault-tolerance. The resulting DIRC pipeline, with different construction patterns, can provide flexible fault-tolerance for a QDI communication infrastructure with a moderate and reasonable hardware overhead. Another fault-tolerant technique, Redundant Protection of Acknowledge wires (RPA), is proposed to protect the acknowledge wires from transient faults, which has extremely low area overhead and can be used independently of DIRC. Detailed experimental results are provided to demonstrate the implementation overhead.

4.1 Introduction

In a Globally Asynchronous Locally Synchronous (GALS) system [KGGV07] which is constructed by asynchronous or QDI NoCs, its fault-tolerance can be achieved through different ways of protection. The end-to-end protection, which adds redundant information when data or packets enter into the network and detects/corrects errors when data leave the network, can provide fault-tolerance capability for the asynchronous network with little disturbance. In other words, fault-tolerant encodings [Sor09] for example can be used at the network interface to encode the original data into error detecting or correcting codes. These codes are then translated to DI codes required by the asynchronous NoC and sent across the network. During the data transmission, no error detection and correction is required. When data leave the network and pass the network interface again, these DI symbols are translated back to the usual binary data so that the erroneous data can be detected or corrected, depending on the fault-tolerance capability of the applied error detecting/correcting coding schemes.

In this way, the error detecting and correcting operations are executed in the synchronous domain, rather than the asynchronous domain of the network. The asynchronous network is rarely disturbed and it is only responsible for delivering packets. This method has been proposed and utilised in several existing asynchronous interconnections [CH01, Pon12], which demonstrated that managing errors in the synchronous domain can be cheaper and more effective than that in the asynchronous domain. However, this end-to-end protection has a limitation which cannot be ignored. Without protection inside the network or the asynchronous domain, a transient fault can insert, delete or corrupt a DI symbol, bringing difficulty to the synchronization at some places of the network [LM04]. More seriously, even a transient fault can break the handshake and cause a physical-layer deadlock. As a result, the destination node or the receiver may never receive a data packet. This fault-caused physical-layer deadlock can cause large amount of packet loss, in which case the end-to-end error detection or correction cannot work. Therefore, it is necessary to add fault-tolerance to protect the large number of long asynchronous interconnects inside the asynchronous domain, so as to significantly decrease the impact of faults on the network.

Many fault-tolerant techniques [OAHY08, LLP07, LLP12] have been proposed to protect asynchronous circuits but they cannot easily be used to protect QDI communications. Some duplication-based circuit redundancy techniques [JM05, MRL06, ASLM09] are efficient to protect computation-centric QDI circuits, but they may not be suitable for communication-centric systems which may have a large number of

long wires, making the duplication expensive. Codes play an important role in both asynchronous and fault-tolerance fields. Information redundancy techniques, or fault-tolerant codes, are attractive to fault-tolerant asynchronous designs. They are promising candidates to protect on-chip communication. Several fault-tolerant codes were proposed but none of them can be easily used in QDI interconnects [AN12, PCV12, LSH15]. Their QDI implementation may be difficult, require much design effort, or lead to a large hardware overhead. This chapter focuses on providing fault-tolerance for QDI interconnects with code redundancy while keeping the timing-robust nature.

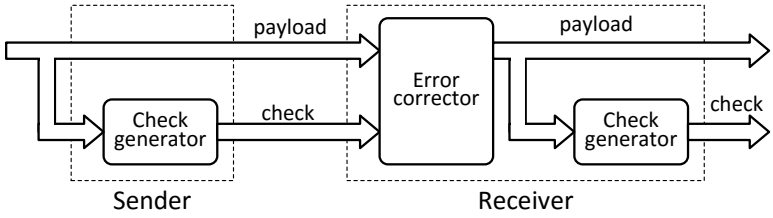
4.1.1 Unordered and systematic codes

Two important code categories in the fault-tolerance field are *unordered* and *systematic* codes. The mathematical definition of *unordered* codes is given below:

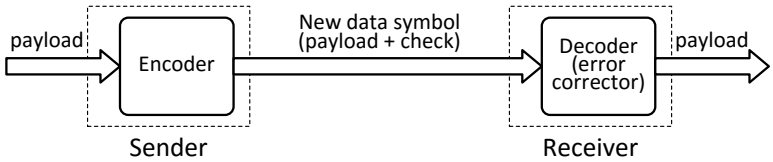
Definition 4.1.1 (Unordered codes [Bos91, BB92])**.** *: Given two code words* $X = (x_0, x_1, ..., x_{n-1})$ *and* $Y = (y_0, y_1, ..., y_{n-1})$ *with the same length n. If X is contained in Y, it means when* $x_i = 1, y_i = 1$ *(*$i \in [0, n-1]$*). If neither of them is contained in the other, the two code words are unordered. In a coding scheme if each pair of the code words is unordered, i.e. no code word is contained in the others, the coding scheme is unordered. This unordered coding scheme encodes data as "unordered codes".*

For example, given three code words $a = 011, b = 001, c = 100$, a and b are not unordered (because b is contained in a) while the other two pairs (a and c, b and c) are unordered. This unordered feature ensures that valid data words in an unordered coding scheme are separated, which is the need of the Delay-Insensitive (DI) communication [CH01]. Thus DI codes are also known as unordered codes [Ver88]. Extensively used unordered codes or DI codes are 1-of-n and m-of-n codes (Section 2.1.5). The unordered coding supports the detection of all unidirectional errors ($0 \rightarrow 1$ or $1 \rightarrow 0$) in a data word [Bos91]. Take a 1-of-2 code "01" as an example which is mutated into "11" during its transmission. "11" is an invalid symbol and can be easily detected with a full symbol test [BS09].

Fault-tolerant codes can be classified into *systematic* and *non-systematic* codes according to their construction manners. A systematic code [AN10] comprises two fields: an information field and a check field. The information field contains the original data payload while the check field is generated from the data and can be used to recover the original data when errors occur. Figure 4.1a presents an example of transmitting a systematic code. A systematic code is separable so that the encoding and decoding



(a) Systematic codes



(b) Non-systematic codes

Figure 4.1: Implementation of systematic and non-systematic codes

process has little overhead [Jha89, AN10]. The main overhead comes from the check generator at the sender and the error corrector at the receiver.

Comparatively, non-systematic codes are inseparable codes because there is only one field for the encoded data, as Figure 4.1b shows. Since the payload and the check bits are inseparable, an area-consuming decoder with error correction functionality at the receiver is usually required.

Therefore, applying DI encoding to systematic codes, both the information and check fields are DI codes, transmitted in parallel, omitting the encoding and decoding process and simplifying the completion detection designs. If non-systematic codes are used, the resulting non-systematic DI codes could use more compact completion detectors [AN10], but with a high overhead from the special encoder and decoder. All m-of-n ($m > 1$) codes with $n > 2$ are non-systematic codes [BTEF03].

4.1.2 Arithmetic rules of 1-of-n codes

This chapter proposes a new fault-tolerant coding scheme to provide a flexible and efficient protection for QDI links. Belonging to the simplest asynchronous protocols, the 4-phase, 1-of-n handshake protocol has been widely used in existing asynchronous (or QDI) NoCs [BF02, FF04, TVC10, SE11b]. Thus the proposed fault-tolerant codes are based on 4-phase, 1-of-n protocols so that they can be easily used in numerous existing asynchronous designs to protect QDI links. The proposed fault-tolerant coding scheme uses the arithmetic rules of 1-of-n codes to correct transmitting errors, which are first introduced.

- 1-of-n codes

  Let i be an integer less than n $(0 \leq i < n)$; its 1-of-n code representation is an n-bit vector $D^n(i)$ where the $(i+1)$ th bit is high (e.g. $D^4(2)$ = "0100" and $D^4(0)$ = "0001'). The basic arithmetic rule of 1-of-n codes is defined as (4.1) ~ (4.3).

  For two integers a and b,

$$D^n(a) = D^n(a \bmod n) \tag{4.1}$$

$$-D^n(a) = D^n(-a) = D^n(-a \bmod n) = D^n(n-a) \tag{4.2}$$

$$D^n(a) + D^n(b) = D^n((a+b) \bmod n) \tag{4.3}$$

- m-of-n codes

  Extending the above arithmetic rules of 1-of-n codes to general m-of-n codes which have m '1's, a position set $A^m = (a_0, a_1, ..., a_{m-1})$ $(1 \leq m \leq n)$ is defined to denote positions of the m '1's in an m-of-n code. Let $D^n(A^m) = D^n(a_0, a_1, ..., a_{m-1})$ be an m-of-n code. A union operation (denoted by the symbol "$\bigcup$") is used in (4.4) to construct an m-of-n code with m 1-of-n codes.

$$D^n(A^m) = \bigcup_{i=0}^{m-1} D^n(a_i) = D^n(a_0) \cup D^n(a_1) \cup ... \cup D^n(a_{m-1}) \tag{4.4}$$

  Taking a 2-of-4 code "1100" for example, it can be considered as a union of two 1-of-4 codes $D^4(2)$ and $D^4(3)$. Thus both $D^4(3,2)$ and $D^4(2,3)$ denote "1100". The extended arithmetic rules of m-of-n codes are shown in (4.5) and (4.6), which are unions of multiple 1-of-n operations.

$$-D^n(A^m) = -\bigcup_{i=0}^{m-1} D^n(a_i) = \bigcup_{i=0}^{m-1} [-D^n(a_i)] = \bigcup_{i=0}^{m-1} D^n(-a_i) \tag{4.5}$$

$$D^n(A^m) + D^n(B^{m'}) = \bigcup_{i=0}^{m-1} D^n(a_i) + \bigcup_{j=0}^{m'-1} D^n(b_j) = \bigcup_{j=0}^{m'-1} \bigcup_{i=0}^{m-1} D^n(a_i + b_j) \tag{4.6}$$

  Taking a 1-of-4 code $D^4(3)$ and a 2-of-4 code $D^4(3,2)$ for example, we have:

$$-D^4(3,2) = -[D^4(3) \cup D^4(2)] = D^4(-3) \cup D^4(-2) = 0010 \cup 0100 = 0110,$$

$$D^4(3) + D^4(3,2) = [D^4(3) + D^4(3)] \cup [D^4(3) + D^4(2)] = D^4(1,2) = 0110.$$

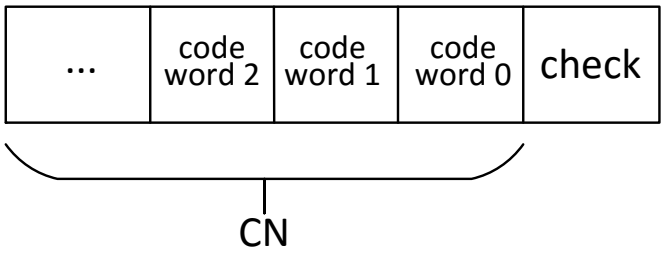


Figure 4.2: A systematic DIRC code

4.2 Delay-Insensitive Redundant Check (DIRC) codes

The basic idea of the proposed fault-tolerant codes is simple: given two code words – (x_0, x_1) – comprising the information field of a systematic code, the check field can be easily achieved by "adding" them together, so that the check word $c = x_0 + x_1$. If either one of the code words (x_0, x_1) is incorrect during the transmission, the faulty one can be corrected by "subtracting" the other fault-free one from the check word. A faulty check word can be thrown away if it is not needed at the receiver or regenerated by adding x_0 and x_1. This process can be extended to the case of protecting multiple code words in the information field. It is simple and useful. Parity checking [Sor09] belongs to this kind of method, whose process of generating the *parity* bit from the other data bits is an "addition" operation. At the receiver, a fault can be detected by comparing the regenerated and the old parity bit, which is a kind of "subtraction" operation.

This research applies the above process to the widely used DI 1-of-n codes, achieving the Delay-Insensitive Redundant Check (DIRC) coding scheme which gains both timing-robustness and fault-tolerance. DIRC uses arithmetic rules of 1-of-n codes discussed in Section 4.1.2. The definition of DIRC codes is given below:

Definition 4.2.1 (DIRC Codes)**.** *Let* $X = (x_0, x_1, ..., x_{CN-1})$ $(CN \geq 2)$ *be a data vector containing CN 1-of-n codes as the payload. A check word c can be generated from X using a check generation process, which adds all the CN codes together. Each of the* x_i*, as well as the check word c, is 1-of-n. Together they comprise a DIRC code word* $(x_0, x_1, ..., x_{CN-1}, c)$ *containing* $(CN+1)$ *1-of-n codes.*

It can be found that the DIRC code is systematic (Figure 4.2): the information field contains the original 1-of-n data payload while the check field is a single 1-of-n check word. This systematic feature makes the completion detection of a data word easy to conduct and simplifies its implementation. Since the transmitted DI codes are not

changed, this coding scheme can be easily migrated to the existing large number of 1-of-n asynchronous designs to provide fault-tolerance.

4.2.1 Check generation and error correction

The check word generation process of DIRC, $f(X)$, is defined as (4.7):

$$c = f(X) = \sum X = x_0 + x_1 + ... + x_{CN-1} \tag{4.7}$$

Therefore, if any one of the code words, x_f, in a DIRC code $(x_0, x_1, ..., x_{CN-1}, c)$, is faulty during the transmission, the correct one can be regenerated at the receiver using the error correction process $g(X_{\neq f})$ defined in (4.8), where $X_{\neq f} = (x_0, ..., x_{f-1}, x_{f+1}, ..., x_{CN-1}, c)$ and x'_f is the regenerated data word of x_f.

$$x'_f = g(X_{\neq f}) = c - \sum_{i=0, i\neq f}^{CN-1} x_i = -\left[(-c + \sum_{i=0, i\neq f}^{CN-1} x_i)\right] \tag{4.8}$$

In (4.8), subtraction has been transformed to addition so that both the check generation and error correction processes can be implemented using the same addition unit (Section 4.3). Figure 4.3 gives examples of the DIRC coding scheme for several popular 1-of-n codes when $CN = 2$. The $CN > 2$ cases can also be inferred using (4.7) and (4.8). Figure 4.4 demonstrates one implementation of a DIRC channel where the check word is generated at the sender while the error correction operation is conducted at the receiver.

4.2.2 Error filtering

For any code word x_i $(0 \leq i < CN)$ in a DIRC code $(x_0, x_1, ..., x_{CN-1}, c)$, the regenerated data word x'_i at the receiver is the same as x_i if no faults occur. Under the assumption of a 1-bit transient fault, either x_i or x'_i may be altered by a fault but not both. To obtain the error-free data word, an error filter $h(x_i, x'_i)$ is defined in (4.9), where "&" denotes the logical operation of a C-element. A 2-input C-element will output '1' (or '0') if both of its inputs are '1's (or '0's); Otherwise, it keeps the last output (Section 2.1.3).

$$x''_i = h(x_i, x'_i) = x_i \& x'_i = x_i \& g(X_{\neq i}) \tag{4.9}$$

Thus, the final error-free data is $X'' = (x''_0, x''_1, \ \ldots, x''_{CN-1})$.

x_0	x_1	c
01	01	01
	10	10
10	01	10
	10	01

(a) 1-of-2 codes

x_0	x_1	c
001	001	001
	010	010
	100	100
010	001	010
	010	100
	100	001
100	001	100
	010	001
	100	010

(b) 1-of-3 codes

x_0	x_1	c
0001	0001	0001
	0010	0010
	0100	0100
	1000	1000
0010	0001	0010
	0010	0100
	0100	1000
	1000	0001
0100	0001	0100
	0010	1000
	0100	0001
	1000	0010
1000	0001	1000
	0010	0001
	0100	0010
	1000	0100

(c) 1-of-4 codes

Figure 4.3: Examples of DIRC coding scheme applied to 1-of-n codes (*CN*=2)

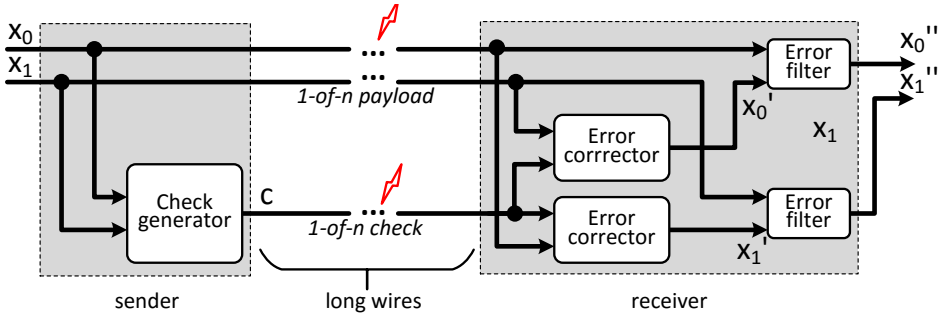


Figure 4.4: Example of a DIRC channel (*CN*=2)

If a transient fault happens on one code word x_i during the transmission, it may erase the valid '1' (negative fault) or create a faulty '1' (positive fault) in this 1-of-n code for a while, but eventually the transient fault will disappear. This fault can be filtered using (4.9) because x_i' is a 1-of-n code calculated from the other fault-free codes $X_{\neq i}$. The fault will only create a temporary "01" or "10" to the two-input C-element filter, which will not change the output. Otherwise, if the transient fault happens on the other code words, leading to a temporarily faulty $X_{\neq i}$, the fault-free 1-of-n code x_i is able to filter the faulty bit in the regenerated x_i' as well to ensure the eventual output x_i'' equals x_i.

Taking two 1-of-4 code words x_0 ("0010") and x_1 ("1000") for example, the check word c is "0001". Assuming a positive transient fault converts x_0 to the faulty "1010", the regenerated x'_0 and x'_1 are "0010" and "1010" respectively. For x'_1, the first '1' from the right is generated from the invalid '1' in the faulty x_0 which can be filtered by using C-elements. Using (4.9), the correct x''_0 and x''_1 are obtained:

$$x''_0 = x_0 \& x'_0 = 0010,\ x''_1 = x_1 \& x'_1 = 1000$$

If the check word is faulty while the incoming data words are error-free, both regenerated data words will be erroneous. These errors can also be filtered by the C-elements.

Considering the impact of a transient fault on the long data wires between the sender and the receiver (Figure 4.4), a transient fault may corrupt a 1-of-n to a 2-of-n symbol (symbol corruption), making one of the operands of (4.9) 2-of-n. The faulty '1' will be filtered out by the C-element. The other fault scenarios, including the symbol insertion, deletion and the fault-caused physical-layer deadlock, have a requirement on the data skew (the time difference) between parallel 1-of-n channels. Their occurrence can be significantly reduced since the error filter of each DIRC pipeline stage creates one synchronization point on the forward data path, which synchronizes these parallel 1-of-n data channels (including the CN+1 channels for transmitting a DIRC code) at each DIRC pipeline stage, reducing the possibility of the long data skew happening inside a DIRC channel.

It can be concluded that all 1-bit transient faults on 4-phase QDI interconnects can be tolerated using DIRC to 1-of-n codes. The proposed DIRC coding scheme can also tolerate some multi-bit transient faults. If multi-bit, unidirectional transient faults happen in a single word while the other words of this DIRC code are fault-free, the faulty word can be corrected. Applying DIRC, the original data sequence cannot be easily disordered, and the harmful physical-layer deadlock becomes difficult to happen.

4.2.3 Code evaluation

Table 4.1 summarises the characteristics of the DIRC and several existing fault-tolerant codes. Since a large number of existing asynchronous designs [TVC10, FF04, BS06, PMMC11] use 1-of-n codes to encode data, the hardware support for 1-of-n implementation is also important. Among these coding schemes, the DIRC code is the only one which is unordered, systematic and can be implemented using QDI circuits.

It has been mentioned in Section 2.1.5 that coding efficiency can be measured by the code rate [BTEF03], which was depicted in (2.1). Both code rates of the 1-of-2

Table 4.1: Comparison of different fault-tolerant codes

Codes	Unordered	Systematic	1-of-n	QDI
Hamming [Ham50]	No	Yes	No	No
Blaum [BB92]	Yes	Yes	No	Unprovided
Cheng and Ho [CH01]	Yes	Yes	No	No
Zero-Sum [AN12]	Yes	Yes	No	No
TRDIC [PCV12, Pon12]	Yes	No	Yes	Yes
DIRC	Yes	Yes	Yes	Yes

and 1-of-4 codes are 0.50. Applying DIRC, one extra check word is added for every CN 1-of-n data words. The code rate of the DIRC (R_{DIRC}) can be expressed as (4.10). It decreases $1/(CN+1)$ compared with the code rate of the original 1-of-n code.

$$R_{DIRC} = \frac{CNlog_2 n}{n(CN+1)} \tag{4.10}$$

It can be inferred from (4.10) that the code rate of the DIRC code increases with CN. In the worst-case of $CN = 2$, the code rates of 1-of-2 and 1-of-4 codes are both 0.33, which is 33.3% less than their original code rates. When $CN = 5$, the code rate decreases only 16.7%. The code rates of most existing code redundancy techniques decrease further when applied to 1-of-n codes. For example, the code rate decreases to 40.0% when applying the parity check to 1-of-4 codes [Pon12]. Applying the TRDIC code [PCV12] which changes 1-of-n codes to 2-of-(n+1) codes to obtain fault-tolerance, the code rate decreases by 33.3% for 1-of-2 codes (from 0.50 to 0.33) and 20.0% for 1-of-4 codes (from 0.50 to 0.40) respectively.

4.3 Implementation of DIRC pipelines

Figure 4.5 presents an implementation of a DIRC pipeline stage transmitting one DIRC code (x_0, x_1, c) (CN=2). It includes 1-of-n adders, error filters, CDs and an Acknowledge Generator (AckGen).

4.3.1 1-of-n adders and error filters

The check word generation and error correction processes can be implemented using QDI circuits. The key components are 1-of-n adders implementing the addition operation of 1-of-n (or m-of-n) codes. In Figure 4.5, the upper two adders are used to regenerate data words while the bottom one is used to generate the check word. If no

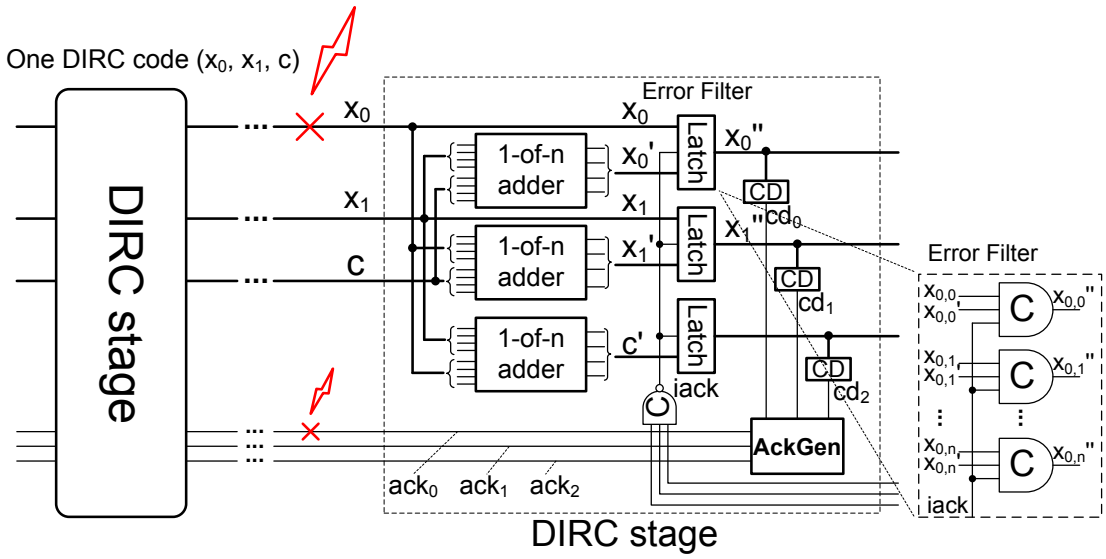


Figure 4.5: Implementation of a DIRC pipeline stage for one DIRC code (CN=2)

fault happens, the operands of the adder are 1-of-n codes. When faults convert 1-of-n codes to erroneous m-of-n codes, the adder produces m-of-n codes as well. These faulty bits will be filtered by the error filters.

The mathematical representation of a 1-of-n adder unit is depicted in (4.11):

$$s_i = \cup\{(a_j \& b_k) | i = (j+k) \bmod n, i, j, k \in [0, n)\} \tag{4.11}$$

where A ($a_{n-1}a_{n-2}...a_0$), B ($b_{n-1}b_{n-2}...b_0$) and S ($s_{n-1}s_{n-2}...s_0$) are all m-of-n codes, and the subscript denotes the bit index. S is the sum of A and B.

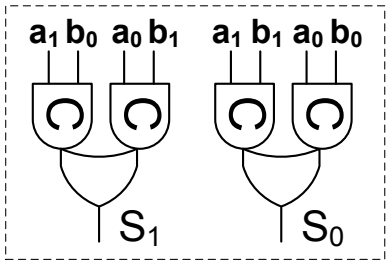
Proof. Because $S = A + B$, it can be inferred from (4.6) that:

$s_i = 1$ $(0 \leq i < n) \Leftrightarrow \exists j, k \in [0, n)$ s.t. $(j+k) \bmod n = i$, $a_j = 1$ and $b_k = 1$.

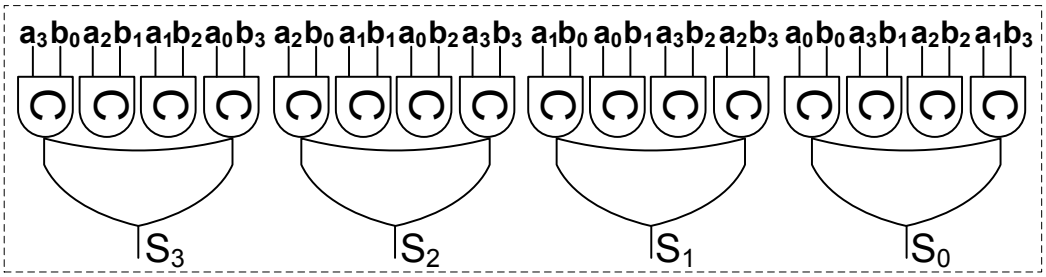
Therefore, we have (4.11). □

Figure 4.6 shows a hardware implementation of the 1-of-2 and the 1-of-4 adder. Only C-elements and OR-gates are employed to ensure the adder is QDI. Figure 4.7 shows structures of 1-of-n adders when the adder has multiple operands (corresponding to different CN). It can be inferred that the 1-of-n adder is a modulo one which involves rotating one code by a distance specified by the other. The adder area approximately scales with n^2.

During the error correction process, the negating operation of an m-of-n code is required in (4.8). According to (4.2) and (4.5), the negation of a 1-of-n code is merely

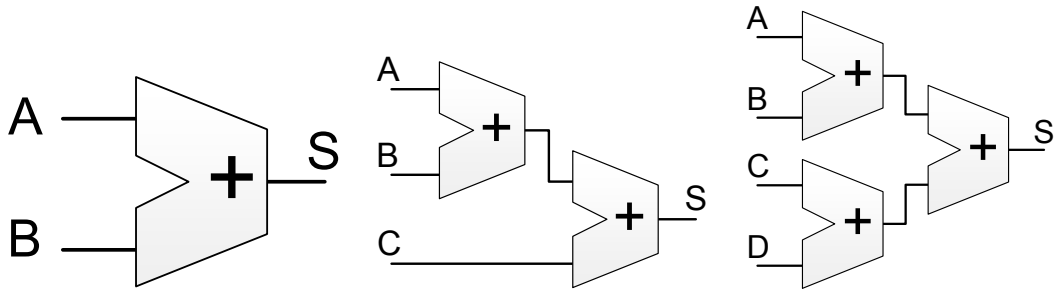


(a) 1-of-2 adder



(b) 1-of-4 adder

Figure 4.6: Implementation of 1-of-n adders



(a) 2 operands (*CN*=2) (b) 3 operands(*CN*=3) (c) 4 operands (*CN*=4)

Figure 4.7: Construction of 1-of-n adder trees with multiple operands

a bit-reshuffle which is depicted in (4.12) where *inv_A* is the negated *A* (both *A* and *inv_A* are m-of-n codes).

$$inv\_A_i = A_{(n-i) \bmod n}, \quad \forall i \in [0,n) \tag{4.12}$$

The error filters described by (4.9) are combined with pipeline latches to improve area and speed performance. The resulting 3-input C-element latch structure for data words is inset in Figure 4.5. The asynchronous latch for check words is built from 2-input C-elements.

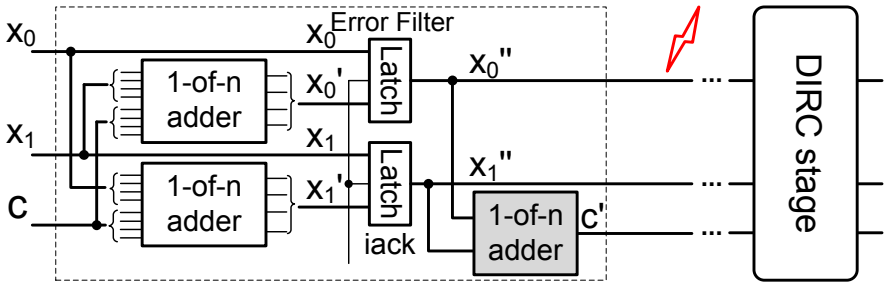


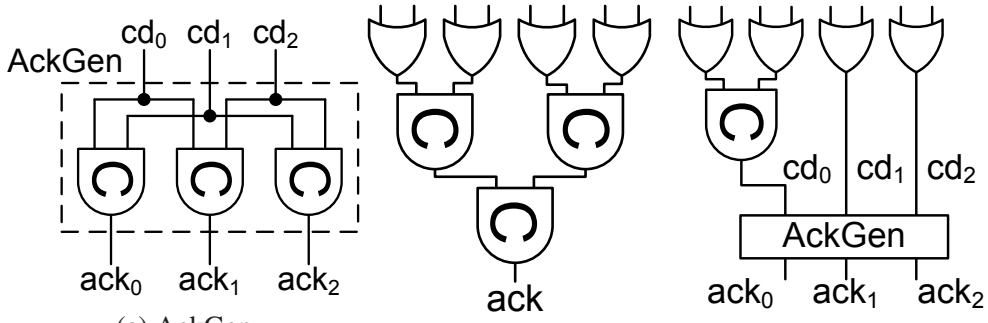
Figure 4.8: Generating the check word using recovered data words

4.3.2 Generation of check words

In the hardware implementation of a DIRC pipeline stage shown in Figure 4.5, the check word is generated from the incoming data words rather than the recovered data words (the implementation using recovered data to generate the check word is shown in Figure 4.8). Consequently, the check word generation process and the error correction process run in parallel which reduces the forward delay. However, the newly generated check word may be erroneous if the incoming data word is wrong. Since the wrong data words will be corrected and not propagated to the next stage, the check word would not be an issue as long as no fault occurs on the corrected data to the next pipeline stage. Under the assumption of a 1-bit transient fault, executing the error correction and check word generation processes in parallel is acceptable since the possibility of faults occurring on wires of adjacent stages is extremely low in practice. In an environment requiring especially high fault-tolerance instead of permanence, the implementation shown in Figure 4.8 can be employed.

4.3.3 Redundant Protection of Acknowledge wires (RPA)

It has been stated that *ack* signals are important for a reliable QDI pipeline (Section 3.2). A Redundant Protection of Acknowledge wires (RPA) technique is proposed to protect acknowledge wires from transient faults. As shown in Figure 4.9a, three C-elements are used to build an Acknowledge Generator (AckGen) which outputs three acknowledge signals (ack_0, ack_1, ack_2). The three inputs of AckGen are cd_0, cd_1 and cd_2, coming directly from the completion detection circuit. The original CD of a pipeline stage (Figure 4.9b) can be easily divided into three sub-CDs by cutting off the bottom one or two C-elements when the number of 1-of-n slices of a pipeline stage is at least three (Figure 4.9c). The generation of acknowledge signals is presented in



(a) AckGen (b) Original CD (c) Cut-off CD

Figure 4.9: Implementation of the acknowledge generator (AckGen)

(4.13) where "&" denotes the logical operation of a C-element. Finally, at the input side of the previous stage, an inverted 3-input C-element is used to generate the *iack* to asynchronous latches (Figure 4.5).

$$\begin{cases} ack_0 = cd_0 \& cd_1 \\ ack_1 = cd_0 \& cd_2 \\ ack_2 = cd_1 \& cd_2 \\ iack = \neg(ack_0 \& ack_1 \& ack_2) \end{cases} \tag{4.13}$$

It can be found that the *iack* flips only when cd_0, cd_1 and cd_2 are all set high or all reset low. Any one of the three acknowledge signals relies on two sub-CDs and any one sub-CD decides two acknowledge signals, which ensures that a 1-bit transient fault on any one of the acknowledge wires will be masked. Only one extra C-element is added to the original CD (Figure4.9). The area overhead brought by RPA is negligible compared with the large number of data latches. This technique can be implemented independently of the DIRC coding scheme to protect acknowledge wires.

4.3.4 Constructing DIRC pipelines with different patterns

DIRC and RPA can be easily used in existing 4-phase 1-of-n QDI pipelines to provide fault-tolerance with little modification. Figure 4.10 presents two contiguous stages of a DIRC pipeline. Assuming the unprotected basic QDI pipeline comprises N 1-of-n channel(s) ($N \geq 1$, Figure 2.12), all the N 1-of-n slices in a pipeline stage can be divided into GN groups ($GN \geq 1$). For different pipeline width, the pipeline can be protected differently using the following rules:

- If N=1, it means the original pipeline contains only one 1-of-n channel. Instead

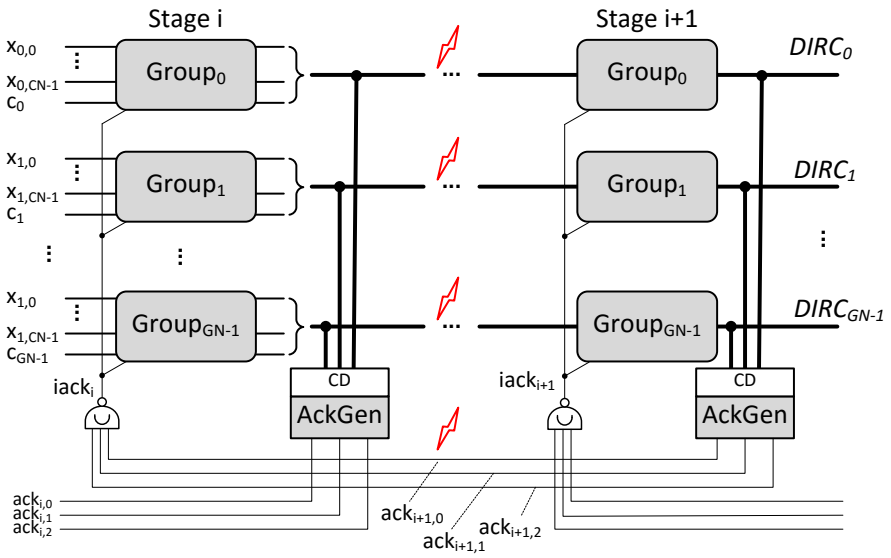
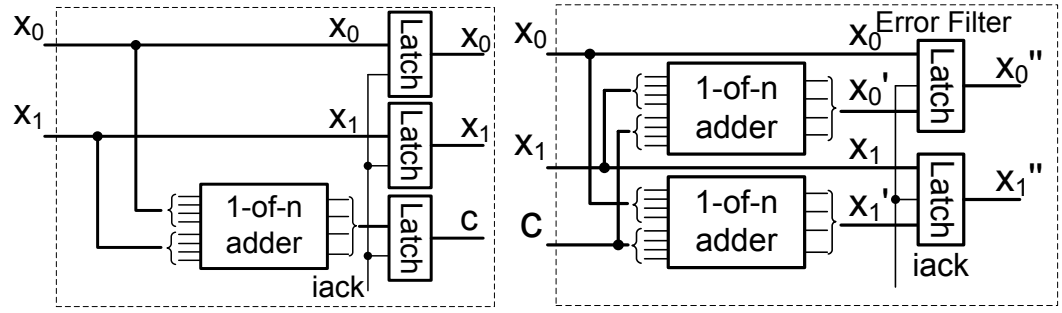


Figure 4.10: A DIRC QDI pipeline

of using DIRC which requires at least two data words to generate a check word, the double-check technique [JM05] can be used to protect the pipeline from transient faults by duplicating the only 1-of-n channel.

- If N=2 or 3, DIRC can be used directly by adding a check word, resulting in a DIRC channel containing two or three data channels and one check channel (GN=1).
- If N=4, the four 1-of-n data channels can be either taken as one group which needs one extra check channel to build a DIRC pipeline (GN=1), or divided into two groups, to each of which is added one check channel (GN=2).
- If N>4, all N 1-of-n data channels can be divided into GN groups to build a DIRC pipeline. Different groups can have different numbers of data channels. For simplicity, in the following it is assumed that N 1-of-n channels are averagely divided into GN groups, each of which contains CN 1-of-n data channels so that CN=N/GN. Applying DIRC, one extra 1-of-n check word is added to each group to achieve fault-tolerance, resulting in a DIRC pipeline stage with $GN(CN+1)$ slices. Each DIRC code contains CN code words and one check word.

In a DIRC pipeline, *complete* DIRC stages (Figure 4.5) are placed between a pair of *incomplete* DIRC stages, which are the sender and receiver denoted by sDIRC and rDIRC respectively (Figure 4.11). The incomplete sDIRC stage only generates check words, requiring one 1-of-n adder as the check generator for each DIRC channel so



(a) sDIRC stage (sender) (b) rDIRC stage (receiver)

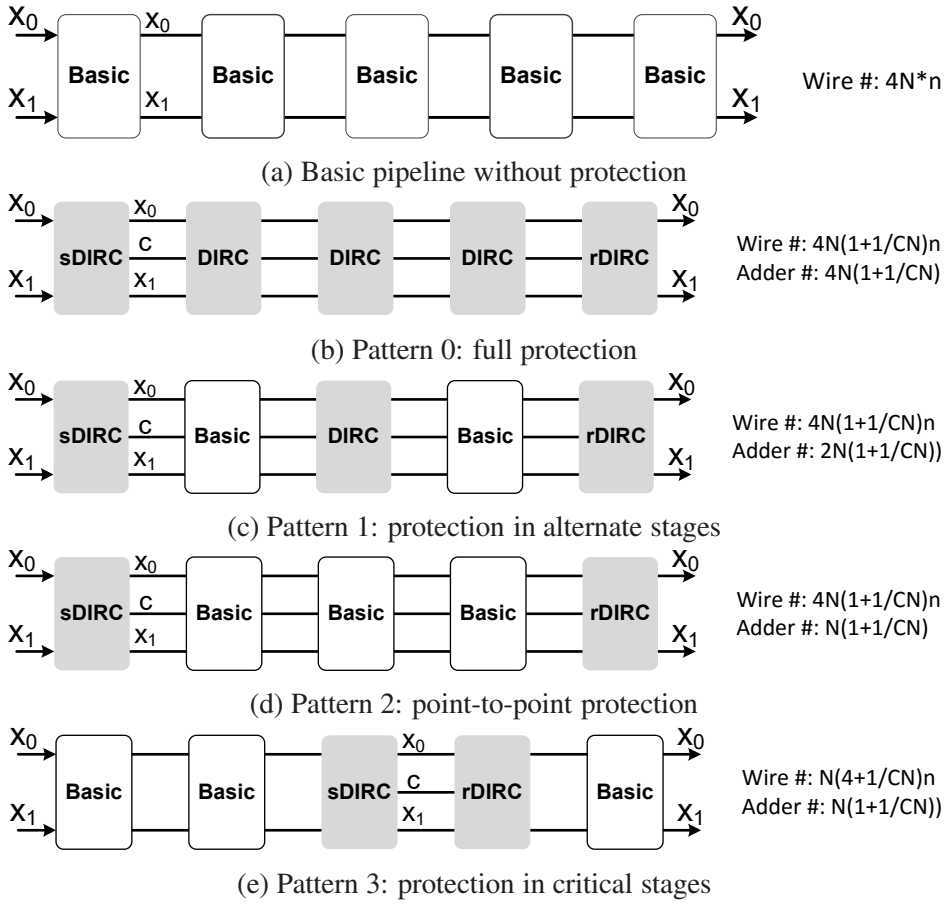
Figure 4.11: Incomplete DIRC stages

that there are *GN* adders totally. The rDIRC stage only corrects errors, requiring *N* 1-of-n adders as error correctors for each channel. They are smaller than intermediate complete DIRC pipeline stages. The complete DIRC stage can correct errors and generate check words simultaneously, requiring (*N*+*GN*) 1-of-n adders to generate check and correct errors.

Latency and area analysis

A forward data path and a backward *ack* path between two contiguous pipeline stages build a loop (Figure 3.9), whose latency decides the saturation throughput of the pipeline. The loop latency is the sum of the forward and backward path latency. The 1-of-n adders are the main latency overhead contributor on the forward path, whose depth increases with $\log_2 CN$ (Figure 4.7). Completion Detector (CD) is the main latency contributor on the backward *ack* path. Built as a C-element tree (Figure 4.9b), the latency of a CD at the receiver stage scales approximately with $\log_2 N(1+1/CN)$ for a complete DIRC stage and $\log_2 N$ for an unprotected basic stage. The latency overhead mainly comes from the adder and the CD tree. A latency analytical model presented in Appendix B demonstrates that in many cases the DIRC pipeline is generally more than half the speed of the unprotected basic pipeline.

Considering the area overhead, 1-of-n adders are the main contributor: the area of a single adder unit increases with the code width, *n*. It can be noticed from Figure 4.6 that, the number of 2-input C-elements in the adder unit increases with n^2. The narrow 1-of-2 adder is much smaller than the 1-of-4 adder. The number of 1-of-n adders in a DIRC stage increases with *CN* (Figure 4.7). The RPA technique (Section 4.3.3) rarely brings any overhead. Therefore, it can be inferred that, to control the area overhead under a reasonable level brought by the DIRC code, a relatively narrow 1-of-n code



(a) Basic pipeline without protection

(b) Pattern 0: full protection

(c) Pattern 1: protection in alternate stages

(d) Pattern 2: point-to-point protection

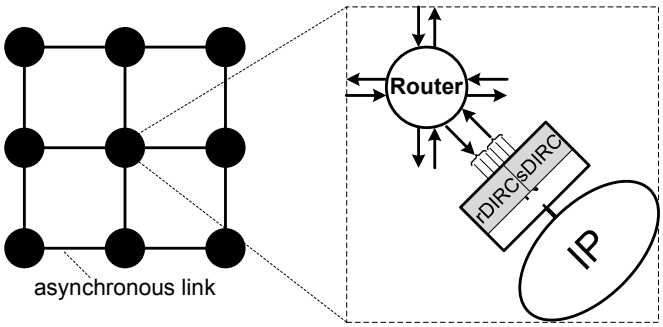
(e) Pattern 3: protection in critical stages

Figure 4.12: Different construction of DIRC pipelines

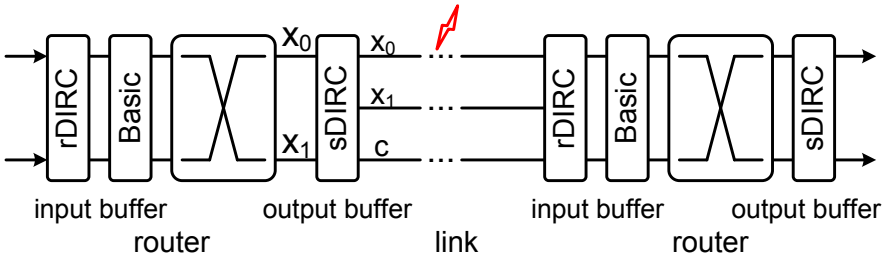
should be used and the width of the DIRC code should be small as well, in which case the interconnects can get more fault-tolerance capability. Latency and area analytical models are built in Appendix B to evaluate of the DIRC implementation. Detailed experimental results are revealed in Section 4.4.

DIRC pipelines with different construction patterns

Since the DIRC code is systematic in that the original data words are transmitted transparently, it allows DIRC pipeline stages to be placed arbitrarily in an unprotected basic QDI pipeline, thus some chosen pipeline segments can be protected. According to the practical fault-tolerance requirement, the DIRC pipeline using different construction patterns can provide enough fault-tolerance for the communication infrastructure with a moderate and reasonable hardware overhead, making DIRC especially attractive to large-scale communication-centred fabrics such as NoCs and buses.



(a) Using DIRC stages at network interfaces



(b) Using DIRC to protect a specific link

Figure 4.13: DIRC applied to asynchronous NoCs

As Figure 4.12 shows, DIRC and basic pipeline stages are mutually exchangeable. Arbitrary basic stages can be replaced by DIRC ones to strengthen fault-tolerance. Figure 4.12a presents an unprotected basic pipeline with five stages. There are $4N \cdot n$ data wires between the first and the last pipeline stage. In Figure 4.12b, all pipeline stages are complete DIRC ones, providing a *full-protection* (pattern 0). This pattern makes the pipeline robust enough to tolerate all 1-bit transient faults between any contiguous pipeline stages but incurs a large area overhead. Adding one check channel to each DIRC group, the pipeline contains $N(1+1/CN)$ parallel 1-of-n channels, so that there are $4N(1+1/CN)n$ data wires and $4N(1+1/CN)$ adders are added to the pipeline.

The systematic feature of DIRC codes allows DIRC pipeline stages to be placed discontinuously to reduce the area overhead. In a design, some parts may be critical or more susceptible to faults than other parts. In this case, DIRC stages can be used only in these parts to protect the communication. As Figure 4.12 shows, DIRC stages can be placed using an arbitrary or specific pattern to provide protection, including full protection, protection in alternate stages, point-to-point protection and protection in critical stages. The protected pipeline segment starts from the sDIRC stage and ends with the rDIRC stage. Between the sDIRC and rDIRC stages, extra long wires are introduced for the redundant check words.

Using DIRC in asynchronous NoCs

DIRC code is especially suitable for large-scale QDI NoCs due to its systematic and DI feature. A 3×3 mesh asynchronous NoC is illustrated in Figure 4.13a. Network interfaces (NIs) are used to connect synchronous IP cores to the asynchronous network, implementing the transformation between the synchronous and asynchronous domains. The large number of long asynchronous links connecting routers are susceptible to transient faults. DIRC stages can be flexibly distributed in the network to protect on-chip communication. As an example, DIRC stages can be placed at the NIs (Figure 4.13a), so that the check word generation and error correction operations are only executed when data passes NIs, providing end-to-end protection. DIRC can also protect specific links or routers. Figure 4.13b illustrates an example where a specific inter-router link is protected.

4.4 Evaluation and experiments

To evaluate the hardware overhead of the proposed fault-tolerant techniques, DIRC pipeline stages (Figure 4.10) were implemented using the UMC 130 *nm* standard cell library and synthesized by the Synopsys Design Compiler [Syn13] with default wire load models. Results were collected from post-synthesis gate-level netlists (i.e. no place and route was performed). Because this is a study of logic circuit design, the implementation technology is not of primary importance. This older, but freely available, library still provides data for relative comparisons of size and speed which should be good as guides across a range of fabrication processes. As a comparison, unprotected basic pipeline stages (Figure 2.12) were also implemented. The experiments target evaluating the general performance and overhead of the proposed techniques, which can be achieved by comparing the protected and unprotected asynchronous pipelines.

Table 4.2 shows detailed experimental results for the $CN = 2$ case (CN is the number of 1-of-n codes in a DIRC word), including the area of a pipeline stage, forward delay, equivalent period and power consumption under different pipeline configurations. The area information was obtained from the synthesis report directly. The QDI link, or the interconnection "wiring" – which includes periodic asynchronous buffers – was modelled using SystemC [IEE12] to insert an (arbitrary) extra ten pipeline stages (which are post-synthesis netlists annotated with gate latency). This can represent a

Table 4.2: Experimental results of the basic and DIRC pipelines (*CN*=2)

Code	Data width	GN	Unprotected Basic Pipelines				DIRC Pipelines				DIRC/Basic			
			Area (μm^2)	*Delay (ns)*	*Period (ns)*	*Power (mW)*	*Area (μm^2)*	*Delay (ns)*	*Period (ns)*	*Power (mW)*	*Area*	*Period*	*Delay*	*Power*
1-of-2	4	2	193	0.071	1.34	0.41	960	0.260	2.10	0.76	4.97	1.57	3.67	1.86
	8	4	400	0.071	1.80	0.60	1791	0.266	2.65	1.06	4.48	1.47	3.75	1.76
	16	8	822	0.075	2.18	1.03	3321	0.279	3.04	1.79	4.04	1.39	3.73	1.74
	32	16	1727	0.075	2.76	1.74	6482	0.278	3.54	2.99	3.75	1.28	3.70	1.71
	64	32	3227	0.075	2.97	3.04	12673	0.290	3.92	5.34	3.93	1.32	3.86	1.76
	128	64	6481	0.078	3.32	5.55	24204	0.294	4.25	9.28	3.73	1.28	3.80	1.67
1-of-4	4	1	176	0.075	1.19	0.30	1424	0.257	1.93	0.56	8.09	1.63	3.44	1.91
	8	2	365	0.075	1.64	0.45	2842	0.266	2.28	0.93	7.79	1.39	3.56	2.07
	16	4	738	0.074	1.90	0.76	5535	0.277	2.80	1.39	7.50	1.48	3.73	1.84
	32	8	1357	0.075	2.29	1.17	10958	0.278	3.23	2.33	8.08	1.41	3.70	2.00
	64	16	2745	0.075	2.72	1.99	21711	0.301	3.64	4.28	7.91	1.34	4.01	2.15
	128	32	5664	0.079	3.09	3.80	43478	0.317	4.37	7.76	7.68	1.41	4.00	2.04

connection across a significant part of a chip where a fault may be injected. In practice the number of stages will depend on the particular application, but with an asynchronous pipeline they can be inserted arbitrarily. The logically more complex stages will tend to be slower than these simple repeaters and, for modelling purposes, it was assumed that the delay due to long wires could be factored into that at layout time. Thus, to achieve the time and power information, millions of data were transmitted through the ten-stage pipelines. The forward delay was achieved by averaging the time that a data word transmit through an idle pipeline. Injecting data words to the pipeline as long as the first stage can receive data (so that the asynchronous pipeline is saturated), the equivalent period can be obtained by counting the received data words at the output of the pipeline during a specific length of time. The power consumption was calculated from the recorded signal transition activities using Synopsys PrimeTime PX [Syn14].

4.4.1 Performance evaluation

Since DIRC pipelines can be constructed using various patterns, the practical pipelines may have quite different areas. To provide a general evaluation, the area results of *complete* DIRC stages are presented to demonstrate the area overhead brought by the DIRC and RPA. Figure 4.14a compares the area of different pipeline stages, which increases with the pipeline data width. DIRC and RPA introduce some area overhead

due to the check generation and error correction mechanisms. On average, this ratio is around 4.15 for 1-of-2 pipeline stages and 7.84 for 1-of-4 ones (Figure 4.14b), which is generally consistent with the area model proposed in Appendix B. When the area of the large number of long wires is considered, this ratio will decrease. In practical designs where the pipeline may be constructed using different patterns including incomplete , the area overhead can be further reduced.

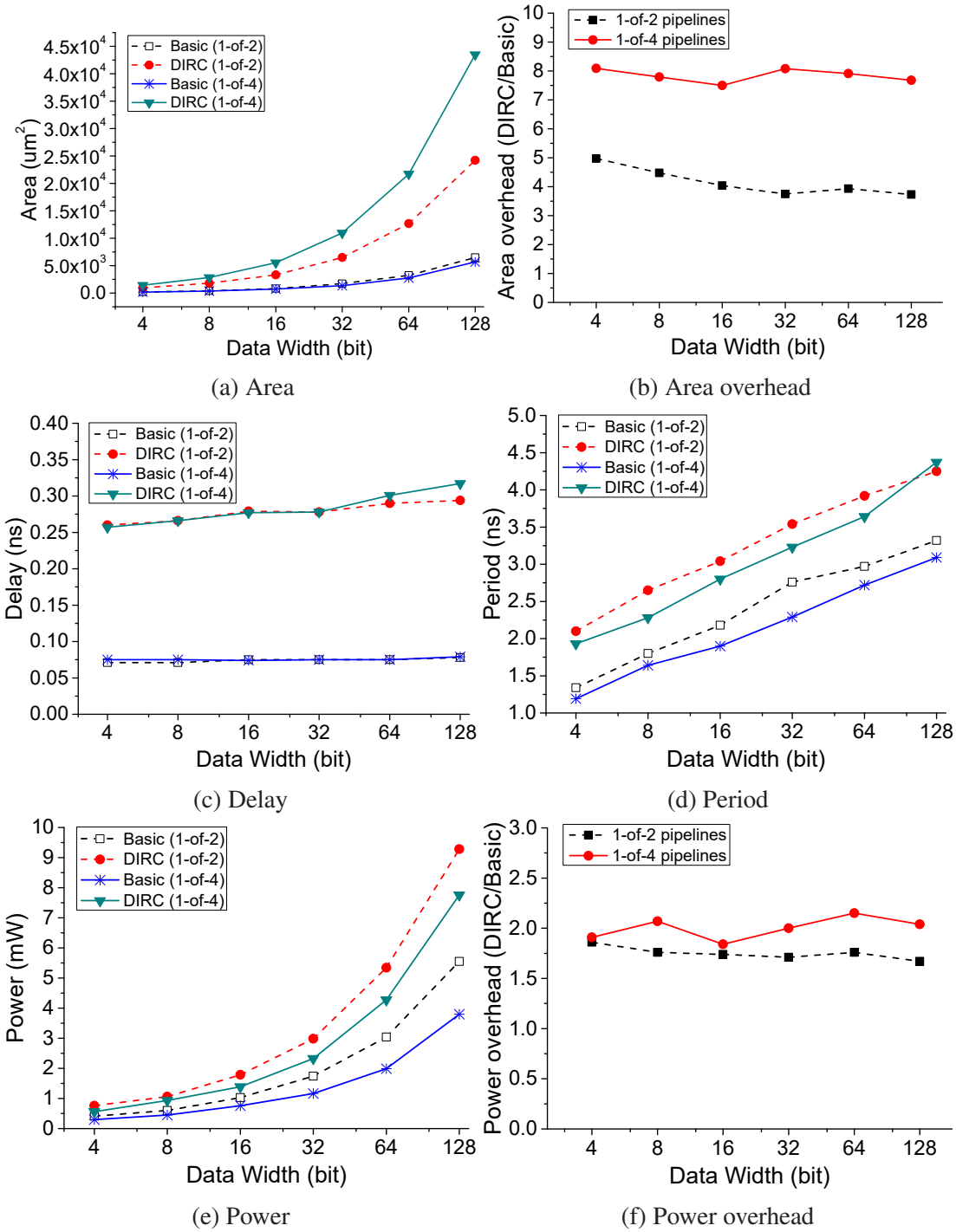
The forward delay of an asynchronous pipeline stage is the time needed by the data to traverse the asynchronous latch. Figure 4.14c shows that the average forward delay increases slightly with data width. On average, the forward delay of DIRC pipelines is 3.75 times the delay of basic pipelines. The fault-tolerant mechanism of the DIRC pipelines causes an extra delay (DIRC-Basic) which is only 0.21 *ns* on average for both 1-of-2 and 1-of-4 pipelines.

The equivalent period is an important factor that affects the maximum pipeline throughput (Section 4.3.4). Figure 4.14d shows that the period increases with the data width of a pipeline. Since the CD tree is one level shallower in 1-of-4 pipelines than that of 1-of-2 pipelines, 1-of-4 pipelines have relatively shorter periods with the same data width. In most cases, the equivalent period of the DIRC pipeline is less than 1.5 times of the basic pipeline (Figure 4.2). The period of the 128-bit wide 1-of-2 DIRC pipeline is only 1.28 times as long as that of the basic pipeline. Compared with the fault-tolerant design proposed by Jang et al. [JM05] whose average speed is only half of the basic one without fault-tolerance, the speed overhead of DIRC pipelines is moderate and competitive.

Figure 4.14e shows the power consumption of different pipelines. The redundant circuit brought by DIRC and RPA causes more transition activities. On average, the power of 1-of-2 DIRC pipelines are 1.75 times greater than basic pipelines, while for 1-of-4 pipelines the ratio is about 2 (Figure 4.14f).

To evaluate the effect of CN (the number of data words included in a DIRC code) on the hardware overhead of DIRC pipeline stages, 60-bit wide 1-of-2 pipeline stages are implemented and synthesized. Figure 4.15 compares the basic and the DIRC pipeline stages with different CN. It can be found that the area of a pipeline stage increases approximately linearly with CN. The forward delay increases with CN since a larger CN means a deeper adder tree inserted into the data path. For the $CN = 3$ and $CN = 4$ cases, the adder trees have the same depth (both are 2-level) but the $CN = 3$ case requires more asynchronous latch for the check words than the $CN = 4$ case, leading to a deeper Completion Detector (CD) tree. As a result, the speed of the DIRC stage with

$CN = 3$ is a bit slower than the stage with $CN = 4$ (Figure 4.15b and Figure 4.15c). The power consumption also increases with CN due to the increased transition activities.



(a) Area (b) Area overhead

(c) Delay (d) Period

(e) Power (f) Power overhead

Figure 4.14: Comparison between the basic and DIRC pipelines ($CN = 2$)

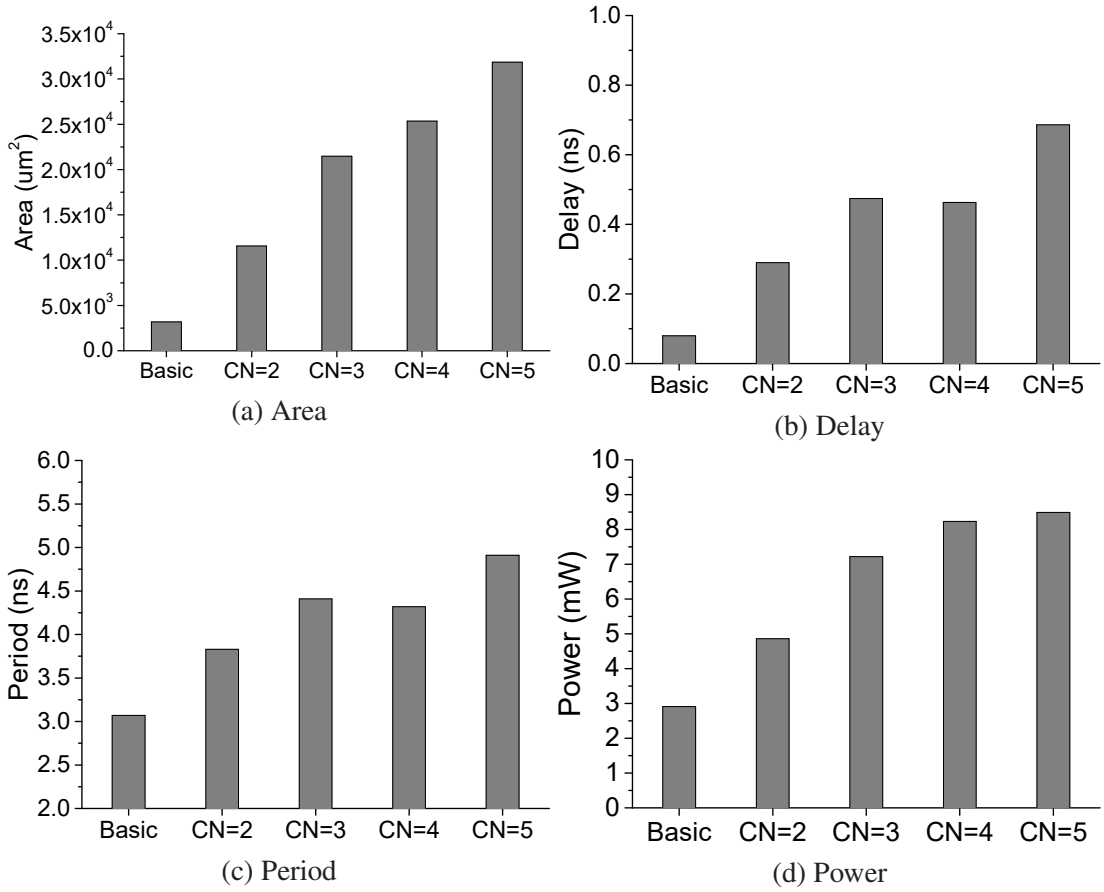


Figure 4.15: Comparison between the basic and DIRC pipelines with different *CN*

4.4.2 Comparison with related work

Compared to the state-of-the-art work presented in Section 3.4, this implementation of the proposed DIRC coding scheme is QDI, which means the fault-tolerance is achieved without compromising the timing-robustness nature of QDI circuits. This is one of the major advantages of the DIRC coding scheme compared with most existing fault-tolerant asynchronous designs. It can be found that, besides the existing fault-tolerant coding schemes for asynchronous communications summarised in Table 4.1, most fault-tolerant designs either target protecting non-QDI designs [LLP07, OAHY08, LWL$^+$10, LLP12], or obtain the fault-tolerance by sacrificing the timing-robustness nature (the fault-tolerant code in [CH01], Zero-Sum and its extensions in [AN10, AN12], the rail synchronization technique in [MRL05b, MRL06], the fault-tolerant NCL designs in [KZYD10], the fault-tolerant latch architecture in [JYB12]). Some techniques can improve the robustness of QDI circuits but do not qualify as fault-immune [BS09, MJM11, LHHA14]. Similar to the proposed DIRC techniques,

the double-checking technique [JM05, Jan08], the duplication-based Asynchronous Burst-Mode Machines (ABMM) [ASLM09] and the Temporally Redundant Delay Insensitive Code (TRDIC) system [PCV12] can protect 4-phase 1-of-n QDI links from transient faults without losing the timing-robustness advantage. Besides the timing-robustness and fault-tolerance nature of the DIRC implementation, the advantage of DIRC coding scheme includes:

- DIRC coding scheme tolerates all 1-bit transient faults and some multi-bit transient faults. It can be easily implemented using standard cell libraries and adopted in existing 1-of-n QDI link designs to provide fault-tolerance.
- DIRC pipelines can be implemented in a flexible way using different construction patterns to achieve a flexible fault-tolerance capability with a moderate hardware overhead.

The area and performance overhead incurred by DIRC and RPA techniques has been demonstrated in Table 4.2. The throughput of the DIRC-protected pipeline decreases less than 50%, which is competitive compared with previous research [JM05, Jan08, PCV12]. A latency analytical model is described in Appendix B.1 to study the speed overhead, theoretically.

The area overhead of complete 1-of-4 DIRC stages can be as high as $\sim 8\times$ compared with unprotected ones. Even the area of a complete 1-of-2 DIRC pipeline stage can be $\sim 4\times$ larger than the basic one. Considering that the double-checking technique incurs an overhead between $2\times$ and $3\times$ [JM05, Jan08], using complete DIRC pipeline stages is quite area-consuming. A trade-off was made between the hardware overhead and the fault-tolerance capability in the duplication-based ABMM designs [ASLM09] where the function of the implemented circuit was considered. The incomplete duplication resulted in an area overhead less than $2\times$. This kind of trade-off between the fault-tolerance and the incurred hardware overhead is common and necessary in practical designs where the practical fault-tolerant requirement and the expected hardware overhead should be evaluated in the beginning. Table 4.2 evaluated the implementation of *complete* DIRC pipelines, while it has been demonstrated that the protected QDI pipeline can be flexibly configured using different construction patterns (Figure 4.12). *Incomplete* DIRC pipeline stages (sDIRC as the sender stage and rDIRC as the receiver stage) are used to replace complete ones at some segments to reduce hardware overhead. As an instance, a pair of sDIRC and rDIRC pipeline stages can protect a section of QDI pipelines (Figure 4.12d). The area of a 32-bit 1-of-2 sDIRC and a rDIRC stage

(*CN*=2) is 3,643 μm^2 and 4,215 μm^2, which is $\sim$2.1$\times$ and $\sim$2.4$\times$ larger than the basic one respectively. This is competitive compared with the duplicated-based double-checking technique [JM05, Jan08] and the full-duplication ABMM design [ASLM09].

It should be noticed that these area results represent the area of pipeline stages. No place and route was done so that the link wires between the stages were not considered. The double-checking or duplication-based techniques result in a full replica of the link while the DIRC coding scheme incurs only $(1/CN)$ of the link wires ($CN > 1$), avoiding the full duplication of the link. Taking the 32-bit 1-of-2 pipeline with a 64-wire-wide channel for example, 128 wires are required when using the double-checking technique [JM05, Jan08]. For a DIRC pipeline with *CN*=2, only 96 wires are needed, so the overhead is 50%. Therefore, when the wire area is considered, the area overhead incurred by DIRC is further reduced. As the area analytical model in Appendix B.2 shows, when wires consume significantly larger area than pipeline stages, the ratio of the area of protected pipelines to unprotected ones will approach to $(1+1/CN)$, which is 1.5 in the case that *CN*=2. The fault-tolerant TRDIC coding system [PCV12] avoids the full duplication of a link as well. However, without including the cost of TRDIC encoder and decoder, the area of a protected 2-of-5 pipeline is $\sim$3$\times$ larger than the unprotected 1-of-4 one. The cost of the TRDIC encoder and decoder was not given, but will incur extra cost.

The 1-of-n adders in DIRC pipeline stages are currently implemented using the Delay-Insensitive Minterm Synthesis (DIMS) approach [SS93] with standard cells, which is relatively expensive. Custom cells for asynchronous circuit [SF01] can be used to further reduce the hardware overhead. In addition, 1-of-4 DIRC pipelines are less area efficient that 1-of-2 ones due to the area-consuming 1-of-4 adders. The area of a 32-bit 1-of-4 sDIRC and rDIRC pipeline stage can still reach to 4,457 μm^2 and 6,837 μm^2, which is $\sim$3.3$\times$ and $\sim$5$\times$ larger than the basic one respectively. More details are discussed in the area model in Appendix B.2.

4.4.3 Fault-tolerance evaluation

To evaluate the fault-tolerance capability of the whole scheme (DIRC+RPA), a SystemC [IEE12] test environment was built (Figure 4.16). It included a sender, a receiver, a DIRC pipeline stage, Fault Generators (FGs) and a stage wrapper. The DIRC pipeline stage was a synthesized gate-level netlist while other parts are SystemC models. The sender works as the incomplete sDIRC stage (DIRC stage at the sender with check generation functionality), producing random data and corresponding check words to

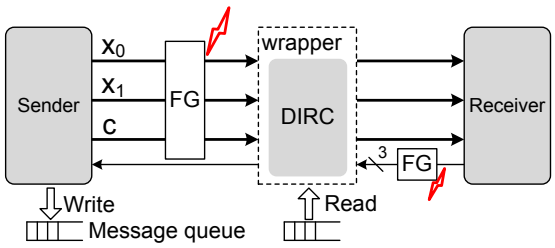
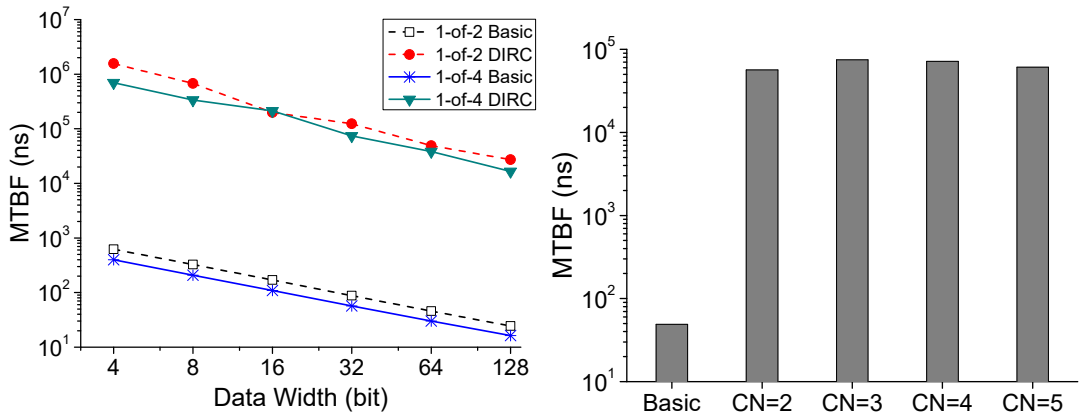


Figure 4.16: Test environment for DIRC pipelines

the DIRC stage while the receiver works as rDIRC stage (DIRC stage at the receiver with error correction functionality). Fault Generators (FGs) generate random faults on all wires including data wires (between the sender and the DIRC stage) and acknowledge wires (between the DIRC stage and the receiver). The stage wrapper checks the correctness of the output data and produces statistics. A shared message queue is used to store the error-free data being transmitted.

In this test environment, faults can be inserted on any wires at any time, which mimics a real environment where not only 1-bit faults but also multi-bit faults may happen. Assuming that the occurrence of faults on a single wire is a Poisson process [Saa61], the intervals between adjacent faults are randomized using an exponential distribution. Faults are inserted on different wires independently. The mean interval between faults is set to 1 μs while the duration of faults is randomized using a uniform distribution between 10 *ps* and 2 *ns*. These create a more comprehensive and severe fault environment than most in existing literature [JM05, OAHY08, PCV12]. Data are transmitted continuously using the maximum injection rate. In total one million data packets were transmitted during the simulation.

The transient-fault-tolerance capability is evaluated by the Mean Time Between Failures (MTBF). Figure 4.17a illustrates the MTBFs of different pipelines with $CN = 2$. Since the increasing number of wires leads to a higher occurrence of faults, the MTBFs of all pipelines decrease with data width. When the data width is 4-bit, the MTBFs of the DIRC 1-of-2 and 1-of-4 pipelines are 2,520 and 1,748 times longer than the basic 1-of-2 and 1-of-4 pipelines respectively. When the data width rises to 128-bit, the ratio becomes 1,117 for 1-of-2 pipelines and 1,012 for 1-of-4 pipelines. For the basic 1-of-4 pipeline with a data width of 128 bits, 174,647 out of 730,498 transient faults result in errors during simulation period, while only 222 out of 1,420,558 transient faults lead to errors when using DIRC and RPA. (Note that this test uses a



(a) $CN = 2$ (b) Different CN

Figure 4.17: Comparison of MTBF between the basic and DIRC pipelines

multi-bit fault environment so that DIRC pipelines can make errors, while 1-bit transient faults will be fully tolerated.) The resulting MTBF for the basic pipeline is 16 *ns* while it is prolonged to 16,561 *ns* for the DIRC pipeline. It indicates the basic pipeline without protection can easily make errors while the fault-tolerance capability of DIRC pipeline increases thousands-fold.

For DIRC pipelines using different *CN*s, Figure 4.17b compares their MTBFs with the MTBF of a 60-bit wide 1-of-2 basic pipeline. It can be found that, under such a severe environment with multi-bit faults, the MTBFs of all DIRC pipelines are more than 1,000 times longer than the basic pipeline. For the basic pipeline, 57,067 out of 334,679 faults result in errors while for the DIRC pipeline with $CN = 5$, only 62 out of 553,143 faults result in errors. This demonstrates that a more severe multi-bit fault environment may largely reduce the MTBF of pipelines (although this rarely happen). Most of the resulting errors can be filtered by using DIRC coding scheme. DIRC pipelines achieve thousands-fold increment of fault-tolerance capability compared with basic pipelines.

4.5 Summary

This chapter proposed a new coding scheme that significantly improves the tolerance of 4-phase 1-of-n QDI links to transient faults, namely the Delay-Insensitive Redundant Check (DIRC) code [ZSG+13]. The DIRC coding scheme tolerates all 1-bit transient faults and some multi-bit transient faults. Furthermore, it can easily be adopted in all existing 1-of-n QDI pipelines to provide fault-tolerance. Since the DIRC code is

systematic, it allows DIRC pipeline stages to be placed arbitrarily in an unprotected QDI pipeline, thus some certain pipeline segments can be protected. According to the practical fault-tolerance requirement, the DIRC pipeline using different construction patterns can provide enough fault-tolerance for the communication infrastructure with a moderate and reasonable hardware overhead, making DIRC especially attractive to large-scale communication-centred fabrics such as NoCs and buses. DIRC is especially suitable for large-scale communication-centric designs. A new technique named Redundant Protection of Acknowledge wires (RPA) was also proposed to protect acknowledge wires. It can be used independently but causes little hardware overhead.

Detailed experimental results showed that the DIRC pipelines achieve thousands-fold improvement on the fault-tolerance capability even in a severe simulation environment. The hardware overhead of DIRC pipelines (using DIRC plus RPA) is moderate and can be further reduced using different construction patterns. In most cases, the DIRC pipeline is less than 1.5 times slower than the basic one, while the fault-tolerance is measured to be more than 1,000 times stronger. However, in the presence of permanent faults, DIRC pipelines can still deadlock, which is harmful to a QDI NoC. Without a full protection, data words transmitted in some pipeline stages are not protected, so that a transient fault can still deadlock the QDI pipeline though the possibility is lowered. The following chapters will discuss the management of the fault-caused physical-layer deadlocks in QDI NoCs.

Chapter 5

Detecting fault-caused deadlock in QDI NoCs

It was observed that faults could cause errors in Quasi-Delay-Insensitive (QDI) channels, which may destroy the handshake protocol and deadlock the communication (Chapter 3). When these faults occur in Networks-on-Chips (NoCs), the situation could be complicated and severe. Data errors, or packet loss, can be managed through different network abstraction layers, but a deadlock can paralyse the whole network. This fault-caused deadlock happening in the physical layer is different from the usual network-layer one due to cyclic dependence of transmitting packets (Section 2.2.5), which cannot be resolved using conventional deadlock management techniques. As a solution, a general framework designed for QDI NoCs is proposed in this chapter to detect this fault-caused physical-layer deadlock caused by different faults and locate the faulty component, so that further operations can be taken to recover the network function (Chapter 6).

This chapter is organized as follows: Section 5.1 introduces a baseline QDI NoC. Section 5.2 studies the general impact of faults on NoCs. Using the baseline NoC, Section 5.3 studies different deadlock scenarios caused by permanent faults and proposes a detection mechanism which can detect the fault-caused physical-layer deadlock and locate the faulty link and router precisely. As long as a fault (permanent, intermittent or transient) causes a deadlock, the fault can be detected and located. Section 5.4 presents an improved deadlock detection process with a fault diagnosis strategy to determine the fault type when both transient and permanent faults are considered. Section 3.4.2 introduces representative related work. Finally, this chapter is concluded.

5.1 Baseline QDI NoC

5.1.1 Network principles

A 2D-mesh QDI asynchronous NoC was shown in Figure 2.25b, constructing a Globally Asynchronous, Locally Synchronous (GALS) system with synchronous IP cores attached [KGGV07]. Though a QDI NoC has many potential advantages over its synchronous counterpart due to its clockless nature, its area can be larger with complicated control logic, affecting the communication efficiency. Therefore, many existing QDI NoCs prefer simple and cheap network protocols [BF02, FF04, TVC10, SE11b]; so does the baseline QDI NoC used in this research. Since the deadlock detection technique proposed in this chapter does not rely on the network topology, for simplicity of purpose a 2D-mesh is used as the backbone. To focus on fault-tolerance and study the effect of fault-caused deadlocks in the physical layer, the popular XY-Dimension-Ordered Routing (XY-DOR) is employed to avoid "traditional" network-layer deadlocks [DT03]. Packets traverse the X-dimension first and then the Y-dimension (Figure 2.16a).

Wormhole switching (Section 2.2.7) is employed in the baseline NoC because of its simple implementation and wide use in existing asynchronous NoC designs. According to the wormhole switching, data is transmitted in packets made of different types of flits, comprising a head flit, multiple body flits and a tail flit. The destination address is stored in the head flit, with which the routing request is generated to compete for the appropriate output port. Body flits carry the main data information. The tail flit is used to separate consecutive packets and release reserved network resources on the path built by the packet. The head flit leads the route and the remaining flits follow in a pipelined fashion, so that a packet with many flits may span multiple routers and links, reserving a long data path in the network (Figure 2.24). As the tail flit progresses, all network resources on this reserved path are released in sequence.

5.1.2 Asynchronous protocols

Routers and links of asynchronous NoCs can be implemented with different asynchronous protocols, including handshake protocols and data encoding methods. Table 2.4 lists representatives of existing asynchronous NoCs using different protocols. Among these options, only the NoCs using 4-phase, 1-of-n are QDI [BF02, FF04, TVC10, SE11b], which can tolerate delay variations on both routers and links. This

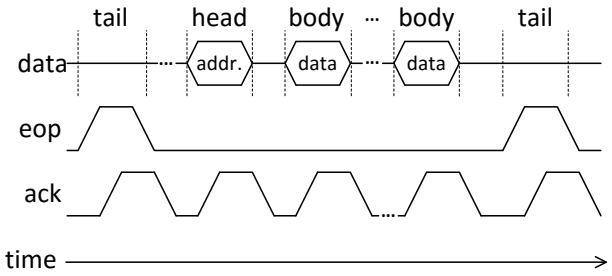


Figure 5.1: Flit sequence

research uses 4-phase, 1-of-n asynchronous protocol as well to implement the QDI NoC and study its fault-tolerance. A baseline QDI NoC is proposed as a design case to demonstrate the fault impact on QDI NoCs and how proposed fault-tolerant techniques can be used in real designs.

In the baseline NoC, data bits are grouped and encoded into multiple 1-of-n symbols, each of which can deliver $\log_2 n$ bits. The data channel of the network – for example carrying 32-bit flits – is broken down into 2-bit quantities, each represented by a parallel 1-of-4 bus or sub-channel. The whole flit is accompanied by a single End-of-Packet (*EoP*) indicator and a single handshake *ack*. The *EoP* flag indicates the tail of the packet, which separates consecutive packets so that no more flags are required to represent the head flit, simplifying the NoC design. In a fault-free environment, the head flit is defined as the first flit after system reset or the one after a tail flit. Figure 5.1 demonstrates the flit sequence under the 4-phase, 1-of-n protocol. A *Spacer* is inserted between two consecutive 1-of-n symbols as the 4-phase handshake protocol requires, providing a *return-to-zero* phase. Therefore, the head and body flits, which are a group of 1-of-n symbols, are separated by *spacers*. A high *ack* wire notifies the successful acceptance of a complete flit or data word. For the last tail flit, without any payload, a high *EoP* signal indicates the end of the packet. As the tail flit progresses, network resources on the packet path previously reserved are released in sequence.

5.2 Fault impact on the data path of QDI NoCs

5.2.1 Fault classifications

A router can be divided into the data path and control logic (Figure 2.23): pipelined buffers at the input/output ports and the crossbar construct the data path through the router, while the control logic comprises a routing computation unit, a buffer controller

and a switch allocator. Faults on NoCs can be classified into two types according to their locations, faults on the data path and in the control logic. Neither kind of fault has been thoroughly studied in QDI NoCs.

Faults on the control logic of the NoC can be harmful as long as they produce wrong control signals, which may lead to erroneous network configuration and interrupt the normal flit flow. A direct way to protect the control logic of the NoC is using replication-based techniques [JM05, MRL06, ASLM09], with which the temporary faulty behaviour can be filtered and the permanently defective component can be replaced or bypassed [FLJ+13, YA10]. Besides common masking factors, a large proportion of this kind of fault can be masked due to the function of the control logic. For example, faults on the routing computation unit can be masked if a route has been configured and flits are traversing the router.

Compared with faults on the control logic, faults on the data path are more straightforward and likely to cause errors on the high-density data flowing in the network. All kinds of flits may be polluted by a fault during their transmission. As an example, the long inter-router links, which are usually routed on high metal layers, are prone to be affected by fault sources from the environment. Faults on these large number of wires are likely to be latched, causing different errors which are transmitted with the flit flow and affect the network. Assuming that the control logic has been protected [JM05], this research studies the fault-tolerance of the data path through a QDI NoC.

5.2.2 General fault impact

Before going into the fault-caused physical-layer deadlock, the general fault impact on NoCs (synchronous or asynchronous) is briefly introduced first. Under the assumption that the control logic is fault-free, a fault may affect the traversal of a packet through a router, resulting different network behaviour depending on the specific design. Figure 5.2 describes several typical faulty scenarios. Figure 5.2a shows the fault-free case where a flit-based packet traverse a router.

- As the most likely case, a fault may pollute one (or several) of the multiple body flits (Figure 5.2b). The faulty bit reaches the right destination and cause data errors, which can be corrected by using error correction codes [Sor09].

- A fault on the head flit may change the destination address information and lead the packet in the wrong direction. Besides packet loss, this misrouting could

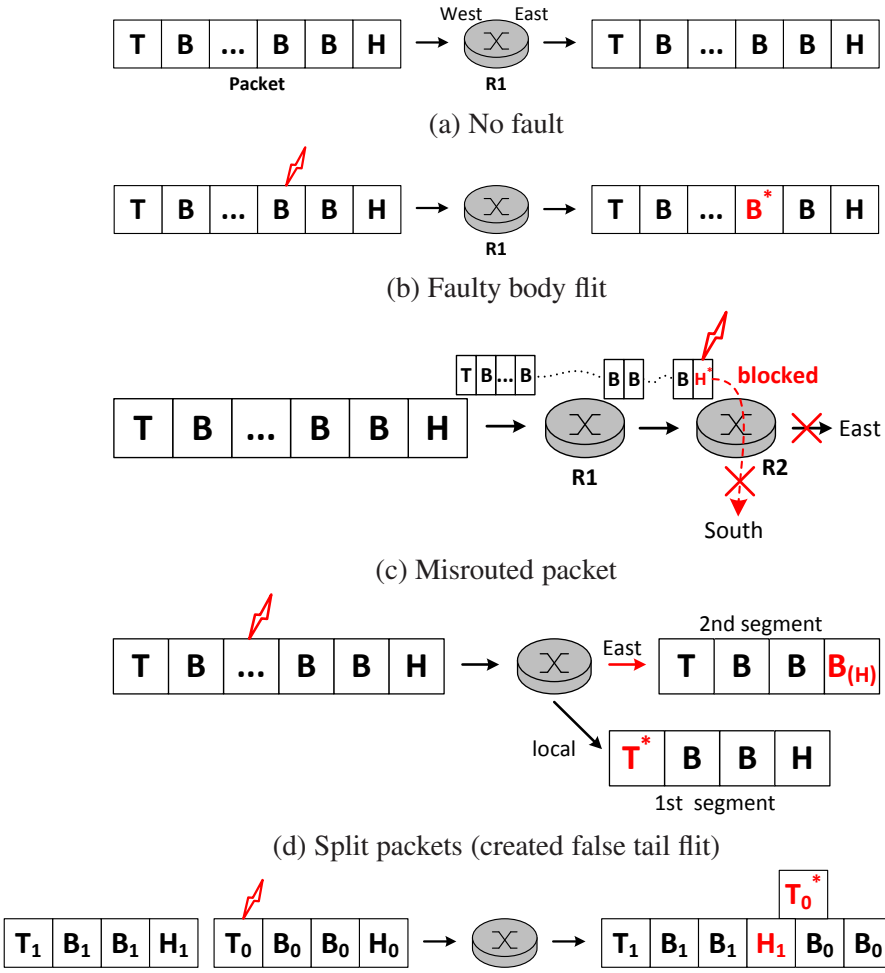


Figure 5.2: Several faulty scenarios of NoCs

cause forbidden turns in the network, or blockage due to the undefined/unrecognised route or network edge, which may further lead to a network-layer deadlock or livelock [DT03]. Figure 5.2c shows an example that a head flit which was originally directed to the East output of router R2 encounters a fault during its transmission, producing a faulty routing request to the South. The faulty head flit is blocked in the west input buffer of R2 since the routing request is invalid under the employed turn model. These scenarios can be avoided by carefully designing the routing computation unit [SE11a].

- Figure 5.2d presents a case that a fault creating a false tail flit splits a long packet into two segments. The front segment goes to the right destination (Local) eventually with severe data loss. If a head flit is identified by specific flag bits which

are fault-free, the second segment may be jammed at the router input and occupies a number of network resources, which needs extra measures to deal with. If a head flit is simply defined as "the one following a tail flit", a body flit will now be incorrectly interpreted as a header in this case, which may cause another corrupted "packet" to travel in an inappropriate direction (East, Figure 5.2d).

- A tail flit may be removed by a fault during its transmission. If the head flit is defined as the one after the tail flit without extra indication flags, the head flit of the following packet is incorrectly taken as a body flit and traverses along the reserved path. Multiple packets could be assembled together, forming a long packet as Figure 5.2e shows. Otherwise, the following packet may get blocked at the input which cannot be reallocated without being released first.

Table 5.1 summarises the general fault impact. A fault may cause packet loss, misrouting, network-layer deadlock or livelock and different kinds of blockage in the network layer. Permanent faults rarely happen compared with transient faults [Con03], but they can remove signal bits completely, causing more severe results. Many techniques have been proposed in previous literature to deal with these traditional faulty scenarios [YA10, IY11]. Different from these general fault effects listed in Table 5.1, faults on QDI NoCs could break the handshake protocol, leading to a fault-caused physical-layer deadlock, which is the main point of this research.

5.3 Detecting a permanent fault on the data path

A NoC architecture can be abstracted to the physical layer, the data link layer and the network layer from bottom to top (Section 2.2.2). A "traditional" network-layer deadlock occurs when the transmission of message results in a cyclic dependence,

Table 5.1: General fault impact on NoCs

Faults on head flit	H.1 No routing request is generated
	H.2 Misrouting (Incorrect routing request) (1) Packet loss (2) Livelock (network layer) (3) Deadlock (network layer)
Faults on body flits	B. Data errors
Faults on tail flits or false tail flit	T.1 No tail flit
	T.2 False tail flit (1) Split packet (2) Assembled packet

which can be avoided by either providing enough network resources in data link layer or using restricted routing algorithms in network layer (Section 2.2.5). Under a faulty environment, the network may go wrong as discussed above, which exists in both synchronous and QDI NoCs and is not of direct interest in this research.

Unique to QDI NoCs, a fault could break the handshake process and cause a physical-layer deadlock (Section 3.3). This fault-caused physical-layer deadlock cannot be easily managed using traditional deadlock management techniques. Keeping the network working even with some packet loss or network performance degradation, is essential for critical digital equipment. Therefore, an on-line deadlock detection and recovery mechanism is necessary to maintain the network function in the face of the fault-caused physical-layer deadlock.

5.3.1 Data path partition

It has been demonstrated that the data path in a NoC typically includes inter-router links, input/output buffers and crossbars, constructing a long QDI pipeline. This pipelined *data* path can be flexibly partitioned in different ways according to the fault-tolerance requirement (Section 3.5). Figure 5.3 presents the one partition used where the link (including its end router buffers) and router (crossbar) are separately protected within two pieces. Thus faults on the *data* path can be classified into *link* and *router* faults depending on their location.

A permanent link fault may corrupt the routing request and affect the router control logic, which can be captured in the *link* piece. A fault on the router piece of a

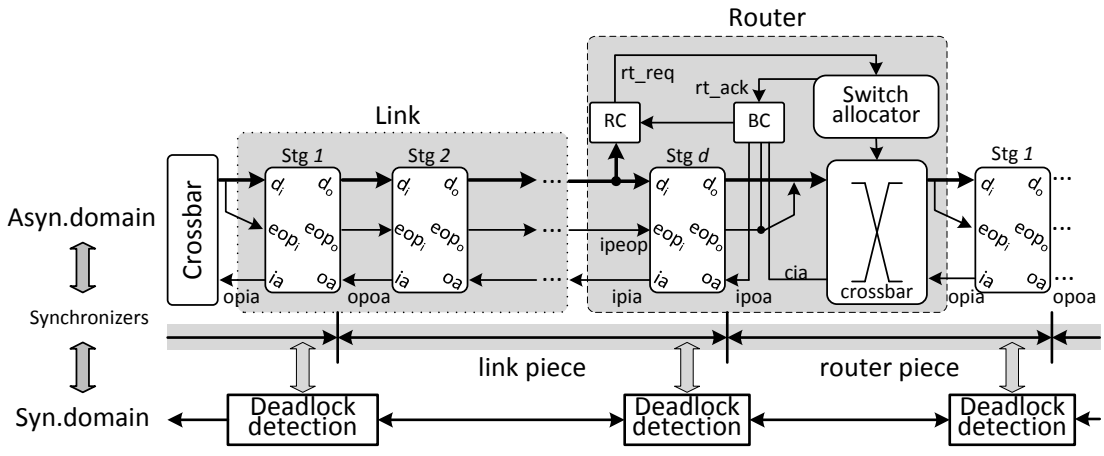


Figure 5.3: Protected pipeline pieces with deadlock detection circuits

previously reserved data path will cause data errors. According to the proposed general deadlock detection strategy (Section 3.5) and the summarised deadlock patterns (Theorem 3.3.1), adding a pair of deadlock detection circuits to the ends of each piece (input/output buffers) to monitor its activities, whether the monitored link/router piece is defective can be determined by checking if the following conditions are satisfied.

1. No transitions are detected on the pipeline piece for a "long" period, indicating it is idle, temporarily blocked due to congestion or deadlocked;

2. The link piece (including the inter-router links and pipelined input/output buffers) and the router piece (the crossbar) show different deadlock patterns in the presence of a permanent fault.

    - **Link fault** The *ack* signals at the output of the pre-fault router are alternately valued (so that the link is not idle). For the input of the post-fault router, all pipeline stages on the reserved packet path have the same *ack* (so that the blockage is not caused by either congestion or the network-layer deadlock where the *ack* should be alternately valued as well).
    - **Router fault** A *data* path has been built by a packet through the router. The *ack* signals at the router input are alternately valued and its granted output has the same *ack*.

5.3.2 Deadlock caused by a permanent link fault

Faults may happen on any gates and wires of the links at any time, affecting the transmitting *data*, the End-of-Packet (*EoP*) indicator or the *ack* signal. This means, there are six types of permanent stuck-at (s-a) faults: *data/EoP/ack s-a*-1/0, affecting all three kinds of flits (head, body and tail). Faults or faulty flits can transit to the input buffer and interrupt the normal router behaviour. The control logic inside a router, including the routing computation unit and the buffer controller, samples the head and tail flits directly from the input buffer. They may get stuck at some specific states due to the permanent fault, which can be used to detect and recover the fault. For an incoming packet, a wormhole router generally experiences three states – **Route Setup**, **Data Transmission** and **Route Release** – as Figure 5.4 shows, where the post-fault router could get deadlocked.

1. **Route Setup**. For the data path, the head flit of a packet is writing into the router input buffer and gets blocked, waiting to be granted an output. For the control

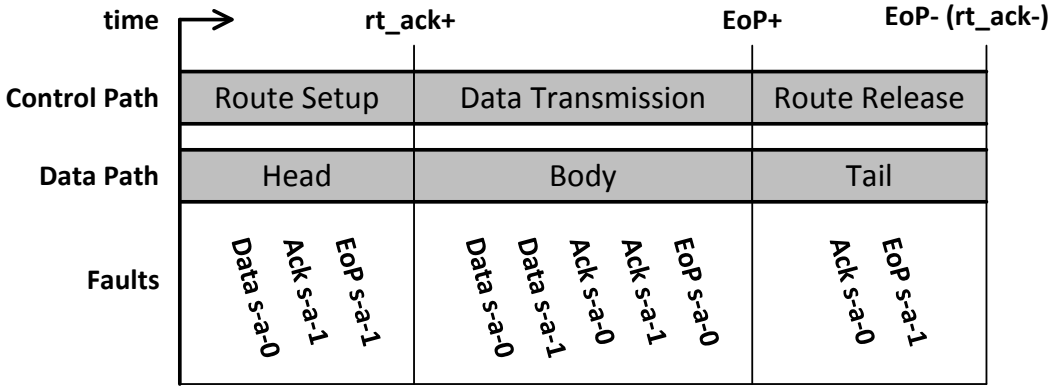


Figure 5.4: Router state transitions under permanent link faults

logic, the routing computation unit samples the address information from the head flit, generates a routing request and waits for the grant from the allocator.

- *Data s-a-*0 fault. The address information is encoded into 1-of-n codes in the head flit. A *data s-a-*0 fault may corrupt the '1' of a 1-of-n symbol into '0', preventing a complete head flit from arriving.
- *Ack s-a-*1 fault. A high *ack* signal notifying the successful acceptance of a complete word, starts the *reset* phase that 1-of-n codes reset to spacers. An *ack s-a-*1 fault may prevent the whole or part of the head flit from arriving at the front of the input buffer depending on its occurrence time.
- *Data s-a-*1 or *ack s-a-*0 faults could prevent a flit from being fully reset, stalling the 4-phase handshake at the *set* phase. The address information can get accepted by the routing computation unit successfully, generating a routing request which will be granted by the switch allocator. Note that, a *data s-a-*1 fault may create a 2-of-n code, which can be forced to 1-of-n by using Mutex Elements [SF01]. If the routing request generated from this head flit is invalid (for the currently used turn models for example), it can be recognized by the routing computation unit and get dropped [SE11a]. Otherwise, this packet will be directed to the destination. In both cases, this post-fault router will get stuck at the next **Data Transmission** state where these two faults take effect.
- An *EoP s-a-*0 fault will not manifest since *EoP* should be low in this state.
- An *EoP s-a-*1 fault will create a fake tail flit in this state: (1) it may take priority at the front of input buffer (leading to a high *ack* signal) and prevent

the following head flit from arriving. No routing request is generated; (2) if the fake tail flit "arrives late" and the incoming packet has been granted an output, the fake tail flit pushes the router to the **Route Release** state.

Therefore, in the presence of a *Data s-a*-0, *Ack s-a*-1 or *EoP s-a*-1 fault, the post-fault router can get deadlocked at the **Route Setup** or **Release** state. A complete head flit is not received by the post-fault router. No routing request is generated or granted. The incoming packet is stalled at the front of the router input buffer.

2. **Data Transmission**. The switch allocator has allocated an output to a packet request in this state and the head/body flits are traversing the router. All *data* and *ack* stuck-at faults can deadlock the reserved data path. They will cause data errors and break the handshake but not disturb the network protocol (a route has been built and the control logic does not need to sample the data).

   - *Data s-a*-0/1 fault: A *data s-a*-0 fault can remove a 1-of-n symbol in a flit. A *data s-a*-1 fault may create a 2-of-n symbol depending on the fault position and later the fault-free '1' gets reset. The faulty '1' will traverse the reserved data path.
   - *Ack s-a*-0/1 faults could prevent one flit from being reset or set, halting the handshake. An incomplete packet is received at the destination without the tail flit.
   - *EoP s-a*-0 fault may happen, preventing the tail flit indicator from arriving. As a result, routers downstream of the fault are stalled at the **Data Transmission** state.
   - *EoP s-a*-1 fault will create a tail flit, enabling the router to transit to the **Route Release** state.

3. **Route Release**. In the baseline QDI NoC, the tail flit separates two consecutive packets so that the head flit can simply be defined as the one after the tail (Figure 5.1). In the end of the **Data Transmission** state, a high *EoP* indicator enables the **Route Release** state where the buffer controller starts monitoring the *EoP* signal. After the *EoP* is reset, the buffer controller will release the built route inside the router and the router state goes to **Route Setup** again.

   - *Data s-a*-0/1 and *ack s-a*-1 faults will not take effect in this phase since they cannot prevent the resetting operation of the EoP indicator.

Table 5.2: Deadlocked states of a post-fault router

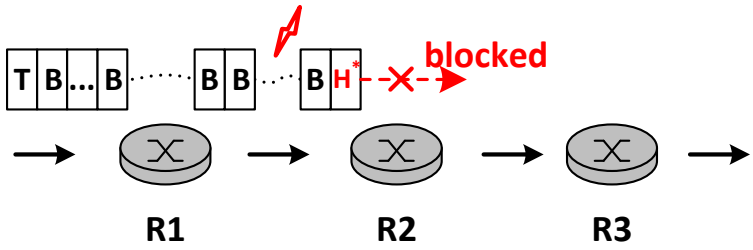
Router state	Fault Type	Impact	Pattern No.	Boolean Expression
Route Setup	Data s-a-0	No request is granted. Pipeline stages downstream of the fault latch an incomplete flit, waiting for the lost valid '1'.	①	!*rt_ack* & !*ipia* & !*ipoa*
	Ack s-a-1	No request is granted. Pipeline stages downstream of the fault latch a spacer or an incomplete flit.		
	EoP s-a-1	No request is granted. The fake tail flit gets stuck at the front of the input buffer.	②	!*rt_ack* & *ipeop* & !*ipoa*
Data Transmission	All types except for EoP s-a-1	Request is granted. All pipeline stages store the same incomplete/complete flit or spacers.	③	*rt_ack* & (*ipia* == *ipoa*)
Route Release	Ack s-a-0	Request is granted. All stages hold the tail flit, waiting for being reset.		
	EoP s-a-1			

- *Ack s-a-*0 and *EoP s-a-*1 faults may prevent the *EoP* signal from resetting, blocking the router in this state.
- *EoP s-a-*0 fault in this phase will enable the router to complete the release operation. It will not affect the current packet but stall the next packet to the **Data Transmission** state.

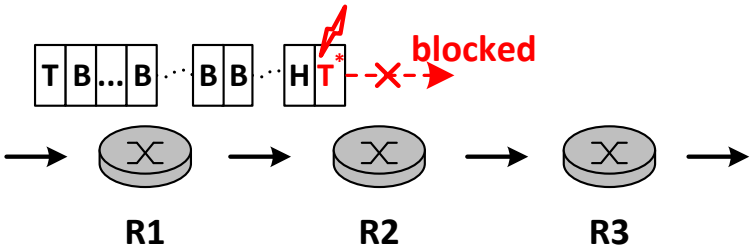
Table 5.2 summarises the impact of all kinds of permanent faults on the NoC data path. Note that the effect of stuck-at faults may not appear immediately but manifest later and causes the post-fault router to be stuck in a specific phase, they end up in the state that they exhibit these effects. In the baseline QDI NoC, *rt_ack* is the signal indicating if the switch allocator has granted an output to the input. The EoP indicator used by the buffer controller at the router input is denoted by the *ipeop* signal. Therefore, the data path may deadlock in all three states with **Route Setup** (*rt_ack*-), **Data Transmission** (*rt_ack*+ & *ipeop*-) and **Route Release** (*rt_ack*+ & *ipeop*+). In a deadlocked state, packet transmission is stalled and all control signals are steady, making it safe to sample control signals to locate the fault position.

5.3.3 Deadlock patterns due to a permanent link fault

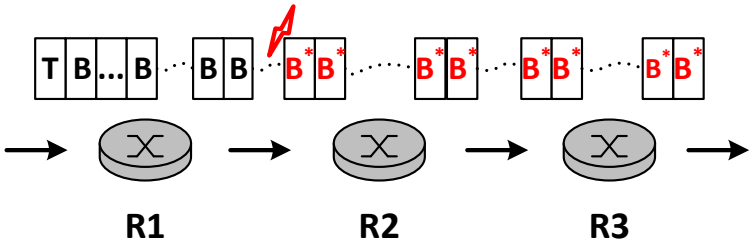
If a permanent fault on the link piece deadlocks a reserved data path, the fault may traverse to the destination and affect other healthy routers and links downstream of the faulty link. By analysing the above fault scenarios and the deadlock model proposed in Section 3.3, it can be found that the effect of stuck-at faults on the post-fault router comprises the following three deadlock cases. These cases can be checked by a pattern checker at each router input to sample values of some control signals (Figure 5.3),



(a) Case 1: stalled at **Route Setup**



(b) Case 2: stalled at **Route Setup**



(c) Case 3: stalled at **Data Transmission** and **Route Release**

Figure 5.5: Deadlocked states due to a permanent link fault

including two contiguous *ack* signals (*ipia* and *ipoa*), the EoP indicator (*ipeop*) and the grant indicator (*rt_ack*). By checking values of these signals at the stalled state, it can be determined if the router is *downstream* of the faulty link.

Case 1: A *data s-a-0* or *ack s-a-1* fault in the **Route Setup** phase will prevent the input buffer from receiving a complete head flit. Pipeline stages after the fault may hold spacers or the same incomplete flit, producing low *ack* signals (*ipia-* & *ipoa-*). As a result, no routing request is generated and granted (*rt_ack-*). This case is indicated by (!*rt_ack* & !*ipia* & !*ipoa*) (Figure 5.5a).

Case 2: An *EoP s-a-1* fault happening when the link is idle (**Route Setup**) creates a fake tail flit (*ipeop+*) which reaches the front of the input buffer and prevents it from receiving following packets. No routing request is generated or granted (*rt_ack-*), which is indicated by (!*rt_ack* & *ipeop* & !*ipoa*) (Figure 5.5b).

Case 3: As the most general scenario for all kinds of stuck-at faults in the **Data Transmission** and **Route Release** state, the routing request has been granted and body flits of a packet are traversing the router (*rt_ack*+). All pipeline stages downstream of the fault store the same incomplete/complete flit or spacers so that any two contiguous *ack* signals after the fault are equal. This case is indicated by (*rt_ack* & *ipia*==*ipoa*) (Figure 5.5c).

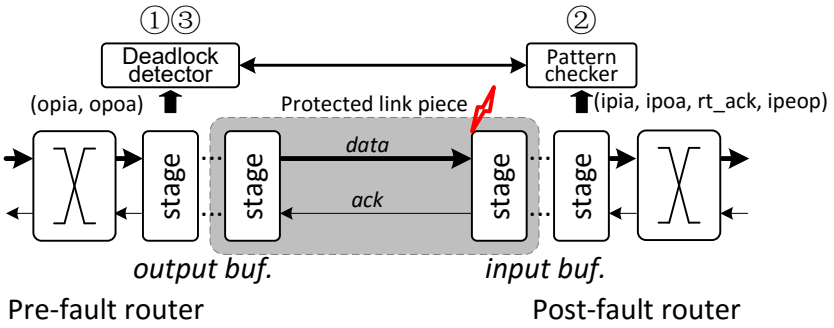
Therefore, in a deadlocked state if none of these cases is satisfied, this router is not *downstream* of the faulty link. Network congestion and the network-layer deadlocks due to the cyclic dependence of transmitting packets can stall the packet transmission, but can be distinguished from a fault-caused deadlock because the stalled packet flits are fault-free (alternately valued *ack* signals) and none of the above cases is satisfied. In the next section, a detection mechanism will be proposed to detect this deadlock using a time-out mechanism and locate the position of the faulty link piece using these deadlock patterns.

5.3.4 A time-out mechanism

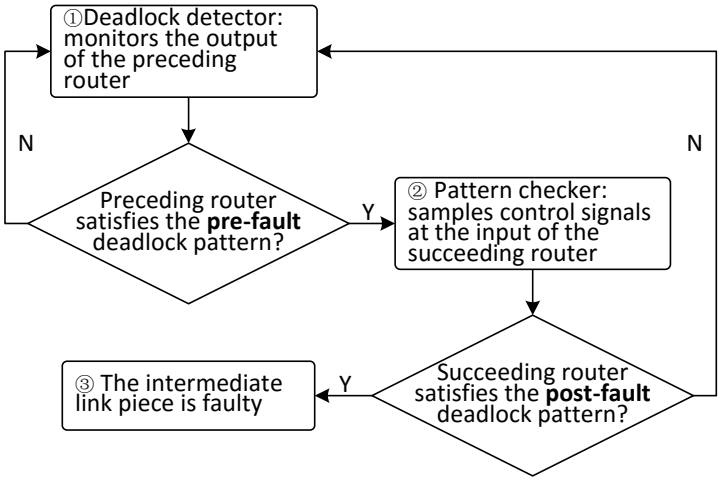
A general time-out method was proposed in Section 3.5 to detect the fault-caused physical-layer deadlock on 4-phase, 1-of-n QDI pipelines. When it comes to a QDI NoC, this deadlock shows more patterns as discussed above (Section 5.3.3). This section proposes a novel time-out strategy to detect the physical-layer deadlock due to a permanent link fault and locate the faulty link piece in QDI NoCs.

To locate the faulty component or pipeline segment, deadlock detection circuits are put at the ends of the protected region (link piece in Figure 5.3) so that the long pipelined data path in the network is cut into multiple segments, each of which is protected by a pair of detection circuits. **Only in this way which locates the fault, a fine-grained recovery can be implemented to isolate the faulty component, avoiding a system reboot or replacement of the whole or large part of the system**.

The protected link piece includes pipelined input/output buffers and the intermediate long link wires (Figure 5.6a). To detect the permanent link fault, a pair of deadlock detection circuits are added to the output and input buffer of two adjacent routers to monitor the intermediate link. Detection circuits include a deadlock detector at the input end of the pipelined link (*Stg* 1 at the output of the preceding router in Figure 5.3), controlling a pattern checker at the link output end (*Stg d* at the input of the succeeding router). Figure 5.6b presents the general flow of detecting a deadlocked link piece:



(a) Link piece



(b) Detection flow

Figure 5.6: Flow of detecting a deadlocked link piece

1. The output buffer of the preceding router (or the input end of the link) is monitored by a deadlock detector. If the pre-fault deadlock pattern is satisfied (no transitions are detected and two contiguous *ack* signals are complementary), this router is upstream of the faulty link and starts enquiring the succeeding router.

2. The input buffer of the succeeding router (or the output end of the link) is monitored by a pattern checker. If the post-fault deadlock pattern (one of the three cases in Section 5.3.3) is satisfied and no signal transitions are detected, this router is downstream of the faulty link.

3. After 1 and 2, the intermediate link piece is confirmed to be the faulty one while the two routers are the pre-fault and post-fault routers respectively.

The key component in the deadlock detector is a Transition Detector (TD) monitoring the transition of certain signals as Figure 5.7 shows [SFGP09]. The *start* is

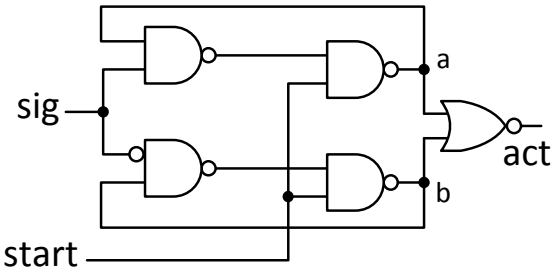
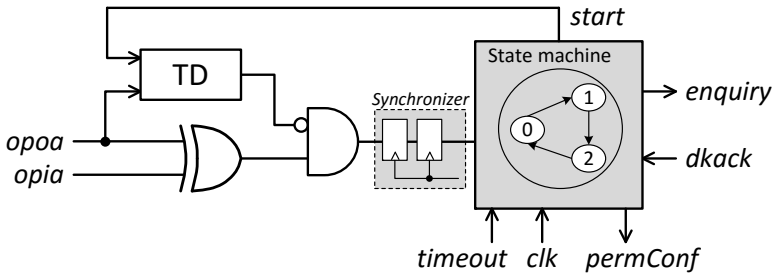
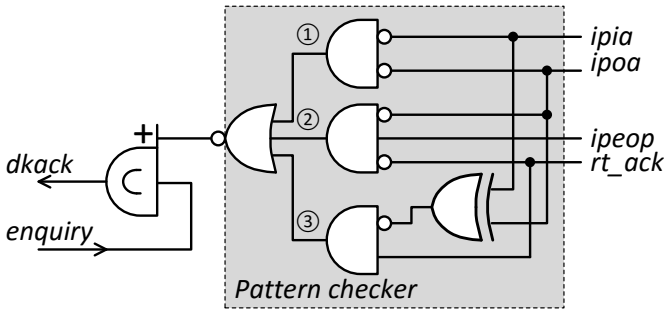


Figure 5.7: Transition detector [SFGP09]



(a) Deadlock detector at router output



(b) Pattern checker at router input

Figure 5.8: Detection circuits protecting a link piece

an active-high enable signal. During the detection process (*start*+), a positive *act* denotes transitions are detected at the monitored signal (*sig*). The reset of *start* causes *act* to reset to '0'. The transition detector is used to determine if the monitored link is transmitting packets. If no transitions are detected (*act*-) for a long time, indicating either the link is idle or packets are stalled due to the network congestion or deadlock, *ack* sequences at the two ends of the link are checked against the deadlock patterns (Theorem 3.3.1) to determine if this link is deadlocked by a fault (this fault can be permanent, intermittent or transient in fact).

Figure 5.8 illustrates the detection circuits added to protect an inter-router link. Besides the redundant detection logic, two extra wires (*enquiry* and *dkack*) are added to each link to exchange the information between the output and input of each pair of

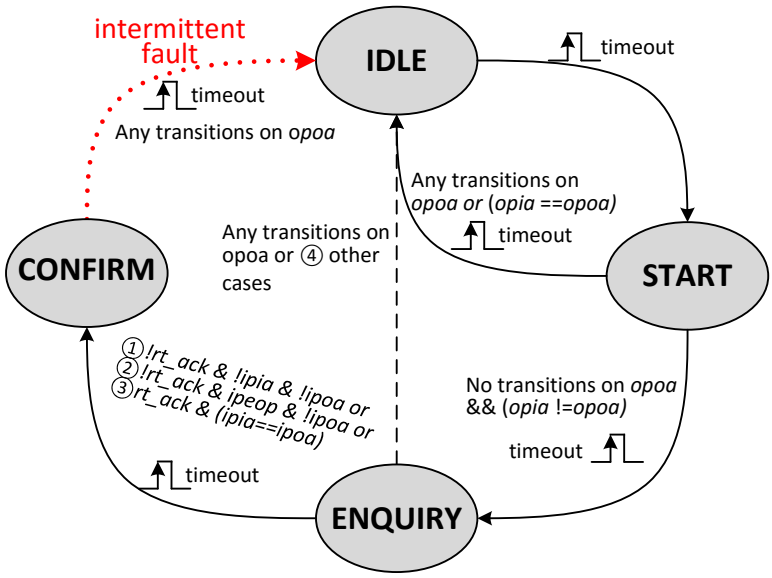


Figure 5.9: State machine used to detect defective link

routers. The whole deadlock detection process is controlled by a state machine in the deadlock detector at each output port. Each router has a counter generating a *time-out* signal to control the state machine, so that the deadlock detector is a sync./async. hybrid circuit where the metastability could happen [Kin07, Gin11]. In this design, detection circuits are assumed safe to sample asynchronous signals from the QDI pipeline if monitored links have been inactive for a "long" timeout period. Synchronizers [Spa07, Gin11] are added to the sync./async. interface to further reduce the possibility of metastability (Figure 5.8a). The state machine has four states: **IDLE**, **START**, **ENQUIRY** and **CONFIRM**, as shown in Figure 5.9. State transitions are enabled only by the *timeout* signal except when returning to **IDLE** from **ENQUIRY**.

- **IDLE**: This is the default state after reset with all flip-flops of the state machine being low. After a time-out period, the state machine transits to **START** (indicated by a high *start*).

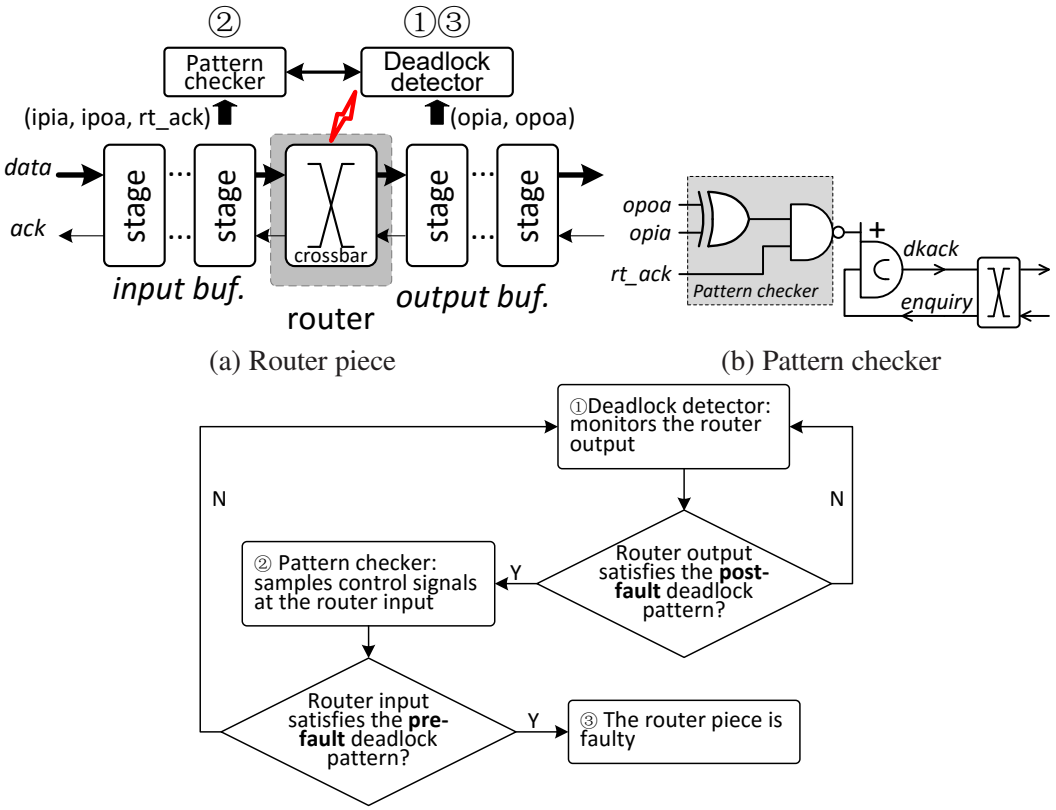
- **START**: In this state, the transition detector in the deadlock detector is enabled to monitor transition activities of the *ack* signal (*opoa*) at the output buffer of the preceding router (the input end of the link piece). At the end of the second time-out period, the deadlock detector either transits to **ENQUIRY** to check the succeeding input buffer (the output end of the link) if: (1) no transitions were detected during the second time-out period and (2) two contiguous *ack* signals at the output (*opia* and *opoa*) are complementary, satisfying the deadlock

pattern of the pre-fault stage according to Theorem 3.3.1, or gets reset to **IDLE** if transitions are detected or *opia*==*opoa*, indicating that a fault-caused physical-layer deadlock does not occur.

- **ENQUIRY**: In this state, the deadlock detector sets an *enquiry* signal high to interrogate the output end of the protected link piece about the deadlock pattern. The pattern checker samples three asynchronous signals from the headmost pipeline stage (*Stg d* in Figure 5.3) of the input buffer of the succeeding router. It should be reminded that in this state, the input end of the link has been taken as pre-fault in **START**. Therefore, if no transitions are detected and one of the three deadlock patterns concluded in Section 5.3.3 keeps being satisfied during the third time-out period, which indicates this router is post-fault (indicated by an active-low *dkack*), the faulty link piece is detected and located. The state machine will transit to **CONFIRM** in the end of the third *timeout* (indicated by a high *permConf*). If the pattern checker fails to obtain the deadlock pattern (none of the three cases in Section 5.3.3 is matched) or transitions are detected, the state machine is reset to **IDLE** immediately.

- **CONFIRM**: This is the state indicating that a fault-caused physical-layer deadlock has been detected and the faulty link piece located. The intermediate link between the pair of deadlock detection circuits is the defective one. For a permanent fault, the state machine will get stuck at this state to block the faulty link permanently. Different recovery methods can be further employed to recover the network function (Chapter 6).

Considering a long lasting intermittent fault which behaves like a permanent one and is able to stall the packet transmission for a long time, a recovery mechanism is required to resume the usage of the previously blocked link when the fault disappears. A transition from **CONFIRM** to **IDLE** is added. The disappearance of the intermittent fault would bring signal transitions on *data*, *eop* or *ack* wires, which further leads to the transition of the *ack* signal at the preceding router output buffer (*opoa*). This could be detected by the transition detector at the input end of the monitored link piece. Consequently, when the next *timeout* arrives, the state machine is reset to **IDLE** and afterwards the blocked link can be reused. This method can also tackle permanent faults with floating values [AAA87] to some extent as long as the fault deadlocks a packet path.



(a) Router piece (b) Pattern checker

(c) Detection flow

Figure 5.10: Detection flow of a permanent router fault

5.3.5 Detecting a permanent router fault

A fault may happen inside a router, deadlocking the reserved data path as well. The main component or router logic on the *data* path is a crossbar connecting input to output ports (Figure 5.10a). Similar to the detection circuits for link faults, a pair of detection circuits is added before and after the crossbar to monitor activities of its input and output. A deadlock detector is put at the output buffer of the router as well which contains a state machine. The general detection flow is illustrated in Figure 5.10c. The main difference from detecting a link fault is, to detect the router fault, the deadlock detector at the router output needs to interrogate the input of the same router in the **ENQUIRY** state about the deadlock pattern (Section 5.3.1), instead of enquiring the succeeding router in detecting a link fault. The pattern checker at the router input is simpler than the one for a link fault (Figure 5.10b). Thus, the deadlock detection process for a permanent router fault is executed locally at one router. Figure 5.11 depicts the modified state machine.

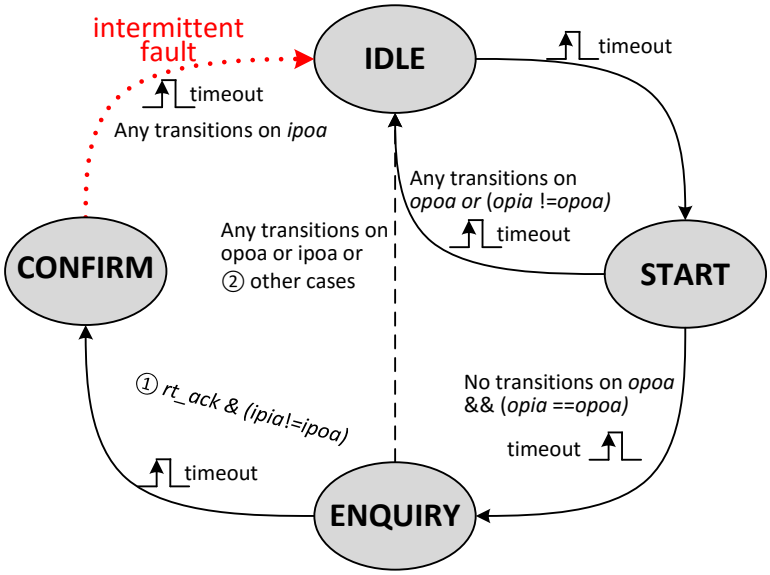


Figure 5.11: State machine used to detect defective router piece

In the state **START**, the deadlock detector at the router output port checks if two contiguous *ack* signals are the same. If no signal transitions are detected and the two *ack* signals stay the same (*opia*==*opoa*) for a whole time-out period, the output buffer is downstream of the fault. The state machine transits to **ENQUIRY** at the end of the second time-out period to interrogate the router input.

In the state **ENQUIRY**, the pattern checker at the input port of the same router checks if two contiguous *ack* signals at the input are complementary as a potential pre-fault stage. If no transitions are detected and the two *ack* signals are different (*ipia*!=*ipoa*) during the third time-out period, a permanent router fault is confirmed to be detected and the state machine transits to **CONFIRM**. It should be noted that a fault on the router logic will not exhibit if it is not on any reserved routes. Only if the fault happens on a path allocated for a packet, will the fault be apparent and affect the network (indicated by *rt_ack*+ at the router input), so that the fault can deadlock the reserved packet path.

It can be concluded that, as long as a permanent (intermittent or transient) fault on the network data path breaks the handshake protocol and causes a physical-layer deadlock in the QDI NoC, it can be detected using the proposed deadlock detection technique. Two to four time-out periods are required to locate the faulty component from the occurrence of the deadlock. There is no requirement on the skew, jitter and frequency of the clock used by the deadlock detection. This clock can be easily got from local synchronous IP cores or other clock sources, and needs to drive only the

state machine. A synchronizer [Spa07, Gin11] is required between the synchronous detection circuit and the asynchronous router to reduce metastability (Figure 5.8a). More technical issues will be discussed in Section 6.4.

5.4 Handling deadlocks caused by different link faults

It has been observed that even a transient fault could break the handshake process and deadlock the QDI communication (Section 3.3), which has rarely been studied before, not to mention this physical-layer deadlock under the context of a QDI NoC. It is common that transient faults cause data errors, most of which can be tolerated using the proposed DIRC codes (Chapter 4) or other techniques [JM05,MRL06]. The physical-layer deadlock can significantly reduce the network performance, which cannot be recovered using upper network-layer methods. The process of detecting a physical-deadlock due to a transient fault is the same as detecting a permanent fault according to the analysis in Section 3.3.3. The difference is, for a transient fault, the faulty component is healthy and can be reused rather than thrown away for a permanent fault. Therefore, when both transient and permanent faults are considered, detecting the deadlock and diagnosing the fault type is necessary to the following network recovery.

5.4.1 Fault diagnosis

Characteristics of the physical-layer deadlock caused by different faults have been thoroughly studied in Chapter 3. A transient fault happening on the forward data path could deadlock an N-symbol-wide QDI pipeline (Figure 5.12), where the data word latched by pipeline stages downstream of the fault is either an **almost full** word with a positive *ack* or an **almost empty** word with a negative *ack* (depicted in (3.11) and (3.12), Section 3.3.3). Comparably, a permanent fault on the forward data path (*data* stuck-at faults) could cause an **almost full** word with a negative *ack* (Figure 3.16a) or an **almost empty** word with a positive *ack* (Figure 3.16b). This discovery is summarised in Figure 5.13, which is the key to diagnosing the fault type. The region of the forward data path is defined in Figure 3.21. It should be noted that faults can also happen on the backward *ack* path. Pipeline stages downstream of the fault may store any kinds of data words. Diagnosis of the fault on the *backward* path is left as future work. This research presents the diagnosis of the fault on the forward data path.

Assume the link piece is protected (Figure 5.3). Located at the last stage of the

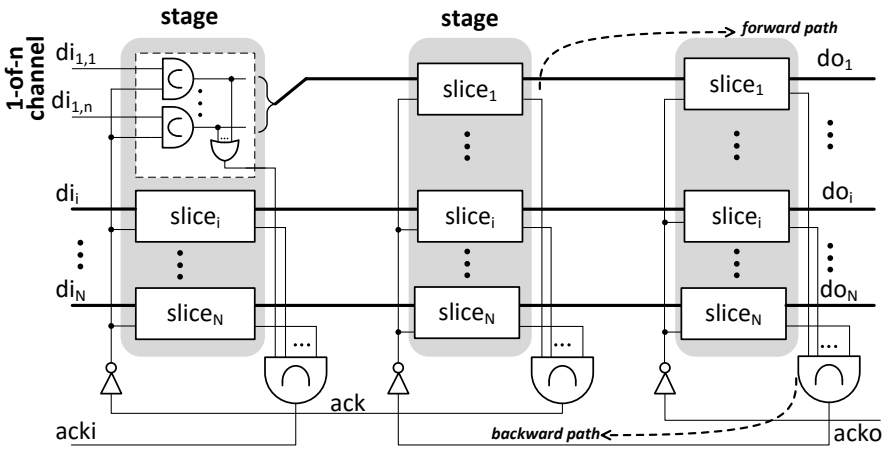


Figure 5.12: An N-symbol-wide 1-of-n pipeline

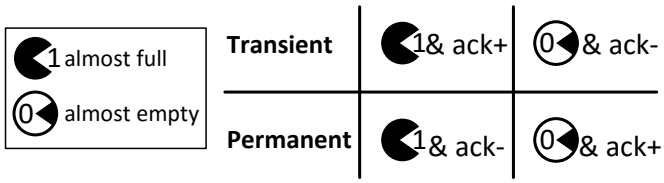


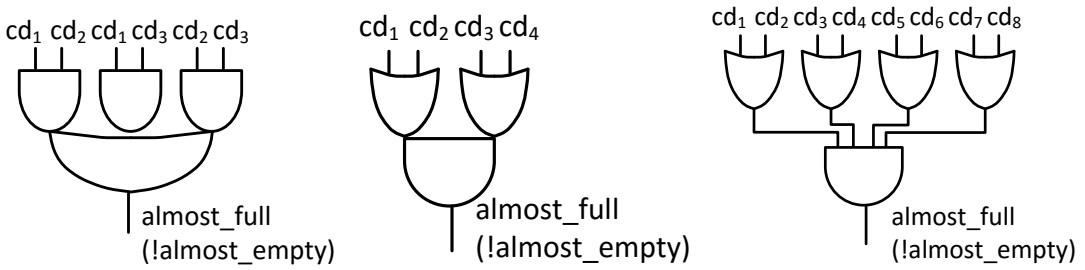
Figure 5.13: Deadlock pattern of pipeline stages downstream of the fault

protected region (*Stg d* at the router input), a fault diagnosis circuit uses the completion detection signal from the Completion Detector (CD) of each 1-of-n channel of the pipeline (Figure 5.12) as its input. Therefore, if the pipeline is N-symbols wide (N>2), the fault diagnosis circuit has N inputs, denoted by S_{CD}=$\{cd_1, ...cd_N\}$ where cd_i can be either '0' or '1'. *Num*(1) denotes the number of '1's in S_{CD} and *Num*(0) denotes the number of '0's. Under the assumption of a 1-bit fault on the forward data path, only one "*cd*" in S_{CD} at most could be different from the others in the deadlocked state. The output of the fault-diagnosis circuit is *almost_full* (whose inverse is *almost_empty*): a high *almost_full* indicates the latched data word is **almost full**; while a low *almost_full* indicates the latched data symbol is **almost empty**. Therefore, it can be inferred that:

- If *Num*(1)>*Num*(0), *almost_full*=1;
- If *Num*(1)<*Num*(0), *almost_full*=0.

The type of the latched data symbol can be determined by comparing *Num*(0) and *Num*(1) using the following rules (*Num*(1) and *Num*(0) are not allowed to be equal):

1. If N=1, only a permanent fault could deadlock the pipeline so that no fault diagnosis is required (Section 3.3).



(a) N=3 (b) N=4 (c) N=8

Figure 5.14: Fault diagnosis circuits for an N-symbol pipeline

2. If N=2, adding one more redundant check channel to make the pipeline 3-symbols wide, the fault type can be diagnosed using the circuit for N=3. In addition, this enables the usage of the proposed DIRC codes (Section 4.3.4) with which symbol corruption caused by transient faults can be tolerated.

3. If N=3, $almost\_full = cd_1cd_2 + cd_1cd_3 + cd_2cd_3$ (Figure 5.14a shows the diagnosis circuit. For an **almost full** symbol with two '1's in S_{CD}, *almost_full* is high. For an **almost empty** symbol which has two '0's in S_{CD}, *almost_full* is low.

4. If $N \geq 4$, the diagnosis circuit can be implemented as a symmetric tree structure as shown in Figure 5.14b and Figure 5.14c. It requires the N to be an even number, which can be easily implemented by adding one extra 1-of-n channel (which usually could provide extra fault-tolerance [JM05, ZSG+14b]) if N is odd. Thus, $almost\_full$=cd_1cd_2 +cd_3cd_4+...+$cd_{N-1}cd_N$.

Figure 5.14b illustrates the diagnosis circuit for a 4-symbol-wide pipeline. Under the assumption of a 1-bit fault, at most one "cd" in S_{CD} could be different from the others. Since the data word in the deadlock state is either **almost full** or **almost empty**, if *almost_full* is low, the data word must be an **almost empty** word that there are (N-1) '0's in S_{CD} ($Num(0)>Num(1)$). If the latched data word in the deadlock state is an **almost full** word, there are (N-1) '1's in S_{CD}, *almost_full* is high. The data type is diagnosed. Figure 5.14c presents the diagnosis circuit for an 8-symbol pipeline.

Fault type is denoted by a 1-of-2 signal *ft_type*[1:0] whose default value is 2'b00. When a deadlock is detected, it should be either 2'b01 (a transient fault) or 2'b10 (a permanent fault), while 2'b11 is invalid (it is assumed that the transient and permanent fault cannot happen and deadlock the same pipeline segment at the same time). According to Figure 5.13, the fault type can be decided using (5.1):

$$ft\_type = \{(almost\_full \ \& \ !ack) \mid (almost\_empty \ \& \ ack), \\ ((almost\_full \ \& \ ack) \mid almost\_empty \ \& \ !ack)\} \tag{5.1}$$

where *almost_empty is the inverse of almost_full*. Equation (5.1) corresponds to Figure 5.13. For example, if S_{CD}={00010000} where cd_4 from the fourth channel is affected by the fault, using the diagnosis circuit we have *almost_full*=0 (*almost_empty*=1). The fault type is decided using (5.1). This proposed diagnosis circuit can be extended to the case of multi-bit faults which is the future work.

5.4.2 Modified time-out mechanism

The time-out mechanism proposed to detect the physical-layer deadlock caused by permanent faults can be modified to manage both transient and permanent faults. This section uses the protection of the link piece to demonstrate the fault detection and diagnosis process.

Figure 5.15 illustrates the modified state machine controlled by a time-out mechanism, which adds one more state **TF_CONFIRM** compared with Figure 5.9. For link faults, the fault diagnosis circuit is at the headmost pipeline stage of the protected link. In the end of state **ENQUIRY** where the input of the succeeding router is interrogated

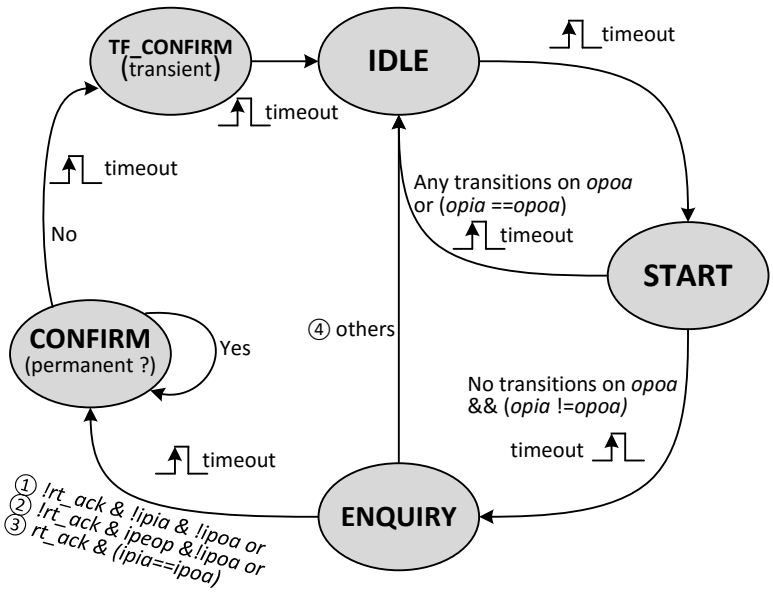


Figure 5.15: Modified state transition graph for transient and permanent faults

about the deadlock pattern, the fault type (*ft_type*) information is reported to the deadlock detector at the other end of the link piece. Therefore, when the physical deadlock is detected or confirmed in **CONFIRM**, the deadlock detector knows the fault type (*ft_type* is either "01" or "10", indicating a transient or permanent fault). The fault is taken as a permanent one in default. The faulty link piece is blocked first and some other recovery measures start (Chapter 6).

If it is a permanent fault that deadlocks the link, the state machine will stay at **CONFIRM**, where the defective link is permanently blocked and bypassed to avoid affecting the other healthy network components. If the fault is diagnosed as transient, after one *time-out* period during which default fault recovery processes finish, the state machine transits to **TF_CONFIRM** to restore the previously blocked but healthy link to use. One more *time-out* period is added between **TF_CONFIRM** and **IDLE**, allowing all necessary recovery processes to finish.

It can be inferred that it needs two to four time-out periods to detect the deadlock from its occurrence, and four to six time-out periods in total to recover the network from the physical-layer deadlock caused by a transient fault. Compared with detection circuits for a permanent link fault (Figure 5.8), one more wire is added between the two routers to carry the fault type information so that three redundant wires are added to each link in total to support the deadlock detection. Adding fault diagnosis circuits to the output buffer of a router, faults on the crossbar can also be diagnosed when they deadlock the built packet path. Thus, most of the data path in a NoC is protected.

5.5 Summary

A permanent fault coming along with the ageing process can break the asynchronous handshake protocol and lead to a physical-layer deadlock. This deadlock is different from the traditional network-layer one due to the cyclic dependence of packet transmissions [DT03]. It cannot be detected and recovered using traditional deadlock management techniques (Section 2.2.5). This chapter thoroughly studied the fault impact on QDI NoC data paths, and proposed a new time-out mechanism, which can detect the fault and locate its position precisely as long as it causes a physical-layer deadlock. As design cases, this chapter demonstrated the detection processes of a permanent link fault and a router fault, with which the data path of a QDI NoC is protected.

A transient fault could also stall the handshake protocol, leading to a physical-layer deadlock, which is a new challenging research topic. The faulty link can be detected

using the proposed time-out mechanism as well. When both transient and permanent faults are considered, fault diagnosis is required to enable different recovery methods. This chapter presented a simple fault diagnosis strategy using the different deadlock patterns caused by transient and permanent faults on the forward data path, with which the fault type can be diagnosed. Correspondingly, an updated detection process was presented. The next step is to recover the network function, which is discussed in Chapter 6.

Chapter 6

Recovering QDI NoCs deadlocked by link faults

Using the detection techniques proposed in Chapter 5, the physical-layer deadlock caused by a fault can be detected and the faulty component can be located precisely. To recover the network, two processes are usually required: (1) deadlock removal – eliminating the deadlock and releasing fault-free network resources; (2) faulty component isolation – reconfiguring the network to ensure all its communication function is unhindered. Recovery methods change with different faults. For transient faults, the temporarily faulty component should be reused after the deadlock is removed. For permanent faults, the defective component should be isolated to prevent it from affecting the following traffic and deadlocking the network again.

This chapter proposes a fine-grained strategy to recover the QDI NoC from physical-layer deadlocks caused by link faults. The proposed *Drain&Release* techniques can recover the broken handshake process according to the specific state where the network is deadlocked, so as to release the deadlocked fault-free network resources. To bypass the faulty component, Spatial Division Multiplexing (SDM) [LMV$^+$08,SE11b] is used to implement the NoC. Along with the router input/output buffers, every inter-router link is physically divided into multiple sub-links; a single fault affects only one of them, so that the faulty sub-link can be blocked and bypassed, preventing succeeding traffic from being allocated to it. Other healthy sub-links of the same link can still be used to transfer packets to the same direction and the network function is recovered.

Router logic is complicated and vulnerable to faults as well. It is difficult to isolate one specific defective logic element without a full duplication. Based on the fault

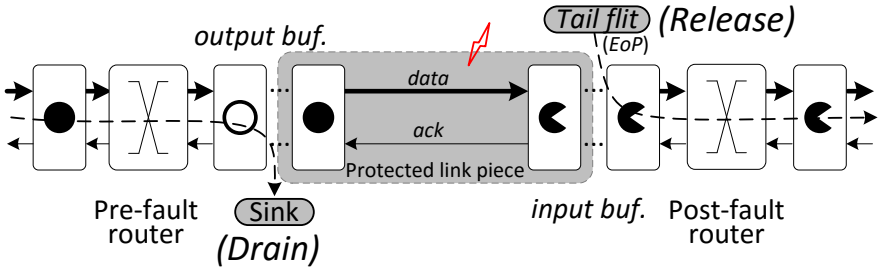


Figure 6.1: Deadlock removal using *Drain&Release*

location information obtained from the proposed detection method (Chapter 5), conventional network recovery techniques, such as fault-tolerant routings in the network layer, can be used to bypass the defective router and recover the network communication, which is left as future work; this chapter focuses on the recovery of the deadlocked NoC caused by link faults.

To achieve the goal of fine-grained network recovery, the current state of the deadlocked handshake should be realized first. Through recovering the destroyed handshake protocol, the physical-layer deadlock can be removed along with the (faulty or fault-free) flits remaining in the reserved packet path; this is a unique challenge faced by QDI NoCs. Having identified a fault position the defective link is cut (Chapter 5), so that a short length of the network (i.e. the faulty inter-router link), containing the fault, is isolated. Any traffic trying to enter the faulty section – which will have been stalled by the fault – is cleared by "sinking" it and discarding the flits to clear the incoming path. Similarly, if there is stalled outgoing traffic from the faulty section, a tail flit is synthesized to terminate the broken packet and clear the outgoing reserved path in the normal way. The broken packet can be trapped by a higher level of fault-tolerant protocol at its destination.

6.1 Deadlock removal using *Drain&Release* technique

A faulty link divides the packet path into an upstream and downstream pipeline segment. *Drain&Release* techniques are proposed to free deadlocked pipeline segments *upstream* and *downstream* of the faulty link respectively, as Figure 6.1 shows. A similar method was proposed for a self-timed bundled-data asynchronous router (which is much like a synchronous circuit) [YIO+12], which cannot work in a QDI NoC.

- **Drain**: By draining the fault-free flits from the *upstream* pipeline segment at the output of the pre-fault router, deadlocked network resources *upstream* of the

fault are released;

- **Release**: By producing a fake tail flit at the input of the post-fault router and letting it go through the network along the existing packet path, deadlocked routers and links *downstream* of the fault will be released.

Figure 6.2 shows modified input/output buffers of two adjacent routers where the intermediate link (the shaded area, including link wires and pipelined router buffers) is the protected region. Extra circuits are added before and after the protected link respectively to support deadlock recovery. Since the critical path limiting the throughput of the network is at the router crossbar, two more pipeline stages (*Stg* 0 and *Stg d*+1) are added to the link ends to avoid the added redundant circuits being in the critical path, reducing the performance decrease accompanied with the fault tolerance.

6.1.1 *Drain* operation at the router output

At the output of the pre-fault router, the *Drain* operation releases network resources *upstream* of the faulty link which hold fault-free flits. According to the 4-phase handshake protocol, complete data symbols and spacers are distributed alternately along the upstream pipeline segment (Section 3.3). Therefore, using a *sink* at the start of the protected link piece which sends back an *ack* signal when detecting a complete data word, the data stream remaining in routers upstream of the faulty link will be drained at the output buffer of the pre-fault router. With the passage of the tail flit of the blocked packet, the allocated routers and links are released one by one. They can be reused by other packets after the recovery process.

As Figure 6.2 shows, the *sink* at the beginning of the link (router output buffer) mainly comprises a Completion Detector (CD) and a multiplexer replacing the *ack*

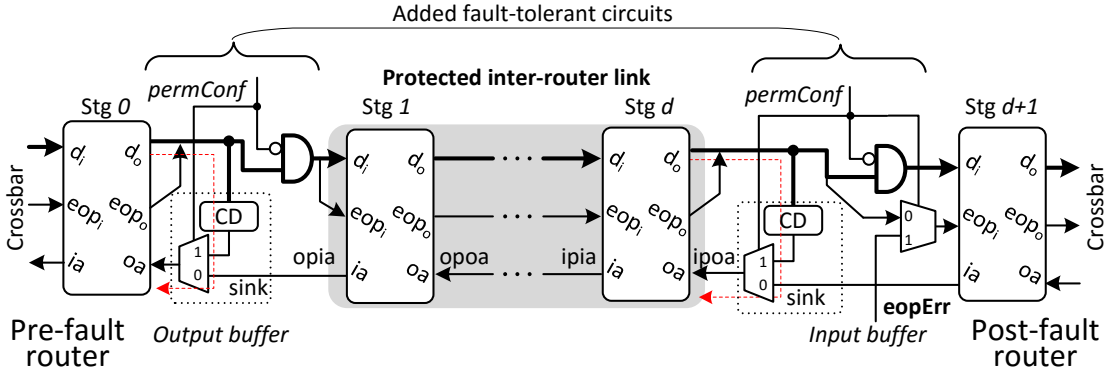
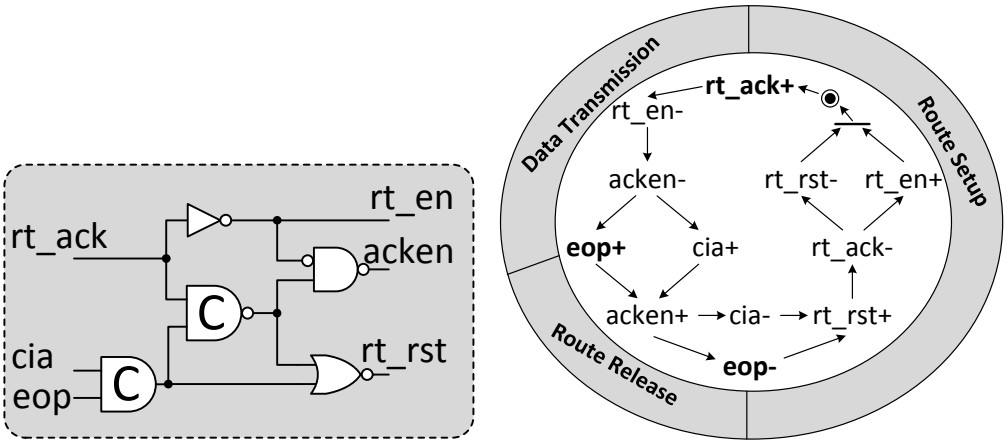


Figure 6.2: Modified input/output buffers for network recovery



(a) Implementation (b) Signal Transition Graph

Figure 6.3: Buffer controller at the router input

signal to the preceding stage (*Stg* 0) when the recovery process starts (*permConf*+). This sink disconnects the data just after the CD and causes incoming *ack*s to be ignored. The CD can reuse the one belonging to the preceding pipeline stage (*Stg* 0). An AND gate is added to prevent the protected link from being polluted by the flits from the *upstream* routers during the *Drain* operation.

6.1.2 Buffer controller at the router input

Different from the *upstream* pipeline segment of the fault which holds fault-free data and can use the original tail flit to release the reserved packet path, the pipeline segment *downstream* of the faulty link may be polluted by the fault and it may never receive the tail flit. Therefore, clearing the faulty data remaining in the pipeline stages and producing a fake tail flit are two necessary operations to release the *downstream* network resources. These operations are executed at the input buffer of the post-fault router (output of the link). It was shown that the Buffer Controller (BC) at the input of the post-fault router, which controls the flit flow through the router, may get stalled at one specific deadlocked state due to the fault (Section 5.3.3). This is critical to the recovery of network resources *downstream* of the router. Extra circuits are added to each router input to determine the deadlocked state of the BC and implement the recovery. The BC is introduced first before going deep into details of the recovery operation.

Figure 6.3a presents the structure of the BC, controlling the packet transmitting process following the wormhole switching [DT03] (Section 2.2.7). Its working manner can be depicted using a Signal Transition Graph (STG) [SF01] in Figure 6.3b

surrounded by the corresponding data path state transition graph (Chapter 5).

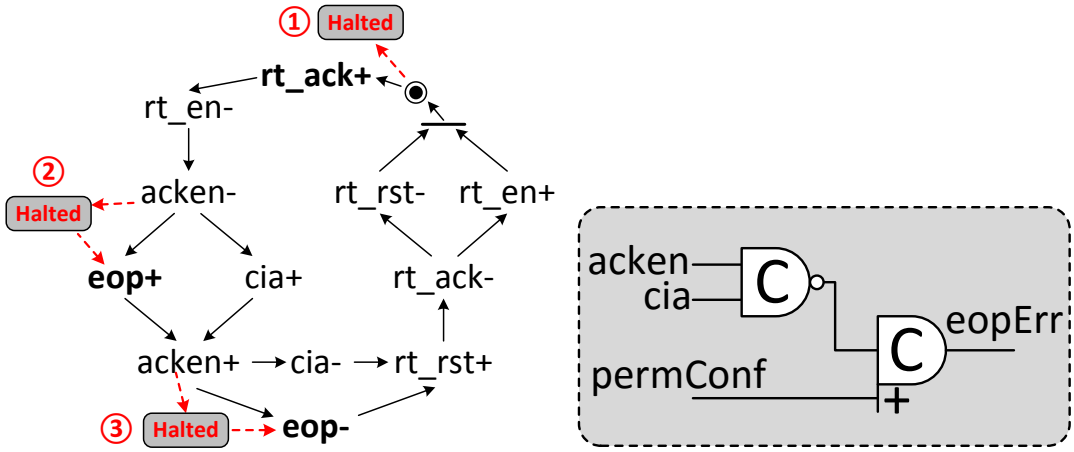
Initially the head flit of the arriving packet is blocked before the front pipeline stage of the input buffer (*Stg d*+1 in Figure 6.2). The Routing Computation (RC) unit samples the destination address in the head flit and generates a routing request (*rt_req*+). If the requested output is busy, the requesting packet has to stay in the input buffer and waits to be granted by the Switch Allocator (SA). Multiple routing requests will be arbitrated by the allocator. The winner will be allocated an idle output port, indicated by *rt_ack*+, with a packet path built across the crossbar. Then the RC unit is disabled (*rt_en*-) to stop sampling the data path. The input buffer is activated (*acken*-), allowing the flit to traverse the crossbar. The data path enters the **Data Transmission** phase and waits for the tail flit indicating the end of the packet. When a tail flit is detected by the BC (*eop*+) and latched by the output buffer (*cia*+), the packet enters into the **Route Release**. BC blocks the input buffer (*acken*+), withdraws the tail flit (*eop*- and *cia*-) and resets the RC (*rt_rst*+) to clear the old routing request (*rt_req*-). As a result, the allocated packet path through this router is released (*rt_ack*-). The input enables the RC unit to sample the data path (*rt_en*+ and *rt_rst*-) and starts waiting for the next packet arrival. This process strictly follows the wormhole switching.

6.1.3 *Release* operation at the router

A permanent link fault may occur at any time, polluting the head, body or tail flit of a packet, deadlocking the post-fault router or the buffer controller into a specific state. By analysing the deadlocked state of the buffer controller and resume the handshake process, a fake tail flit can be generated at the input of the post-fault router and transmitted to release all reserved downstream routers and links, which is the main idea of the proposed ***Release*** operation.

According to Section 5.3, a router input may be deadlocked in one of the three states (**Route Setup**, **Data Transmission** and **Route Release**) as shown in Figure 6.3b. Correspondingly, the STG of the buffer controller at the input may halt before one of the three signal transitions (*rt_ack*+, *eop*+ and *eop*-), which are the deadlocked states as Figure 6.4a shows. By enabling the halted signal transition in this STG, the deadlock can be eliminated to release the packet path downstream of the fault.

1. A permanent fault may begin before or when the head flit comes. No routing request was generated. As a result, the input of the post-fault router is deadlocked at the **Route Setup** phase. Since the switch allocator has not allocated



(a) Deadlocked STG (b) Tail flit generator

Figure 6.4: Modified buffer controller for network recovery

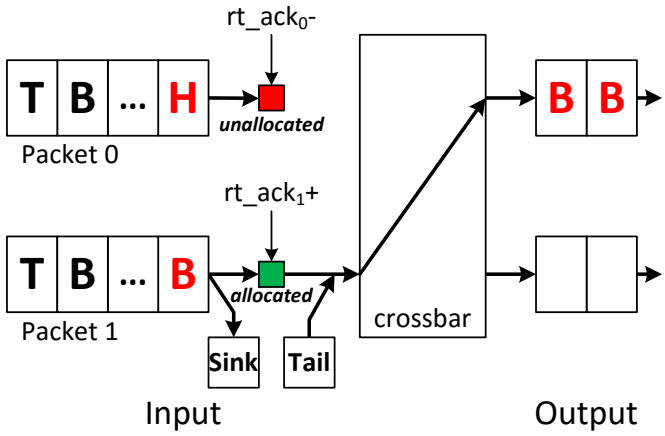


Figure 6.5: Abstracted release operation

any outputs to this incoming packet (Packet 0 in Figure 6.5), this input buffer is the end of the deadlocked packet path and this post-fault router is the only one *downstream* of the fault. No further recovery operations are required.

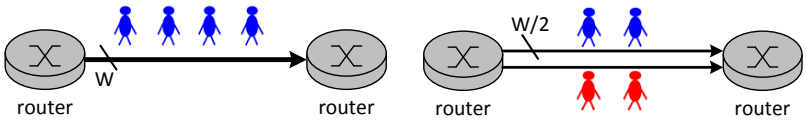
2. If a link fault pollutes a packet which has been allocated a data path through the post-fault router (Packet 1 in Figure 6.5), the input of the post-fault router is deadlocked at the **Data Transmission** or **Route Release** phase, as Figure 6.4a shows. The routing request has been granted by the switch allocator (*rt_ack*+). There may be multiple routers *downstream* of the fault which will not receive the tail flit due to the fault. A fake tail can be generated by a tail generator circuit locally at the input of this post-fault router to release all *downstream* routers.

Figure 6.4b depicts the implementation of a tail generator added to the buffer controller at the router input. To enable the fake tail (*eopErr*) to transfer along the reserved packet path, its generation should follow the 4-phase handshake protocol and the flow control protocol. When a fault-caused deadlock is detected (*permConf*+), the tail generator produces a tail flag (*eopErr*), replacing the original *eop* wire and arriving at the front of the input buffer. Note that the replacement is safe since the pipeline has been confirmed to be deadlocked (*permConf*+). The *cia* is the *ack* signal back to the input buffer from the crossbar to acknowledge the input that a complete flit (it is the fake tail flit here) has been accepted by the output. The *acken* goes high when detecting a tail flit and closes the built path through the router crossbar.

- If the input is deadlocked at **Data Transmission** (*acken*-), the real tail flit cannot reach the front of the input buffer (*Stg d*+1 at the post-fault router input in Figure 6.2). The added AND gate at the input of the router (Figure 6.2) can ensure a low *ack* (*cia*) from the crossbar. Thus, the tail generator will output a high *eopErr* which could be latched and transmitted to all deadlocked stages downstream of the fault. In other words, this tail generator enables the transition of *eop*+ in the STG of the buffer controller (Figure 6.4a), which is equivalent to creating a tail flit. This fake tail flit will traverse through all *downstream* routers and release them one by one. This tail flit is withdrawn (*eopErr*-) when it is accepted by the output (*cia*- and *acken*+).

- If the input is deadlocked at **Route Release** where the *EoP* wire keeps high in all pipeline stages downstream of the fault (leading to a high *cia*), the buffer controller gets stuck at *acken*+ (Figure 6.4a), allowing spacers rather that data words to traverse the router. It keeps waiting for the withdrawal of the tail flit. When the permanent fault is confirmed (*permConf*+), the tail generator outputs a low *eopErr* at the router input which crosses this post-fault router and transfers along the deadlocked data path to the destination. This enables the transition of the buffer controller back to the *eop*-. With the withdrawal of the tail flit (*eopErr*-), deadlocked routers downstream of the fault are released in sequence.

For all these cases, after the *Release* operation, the input buffer of the post-fault router is halted at the **Route Setup** phase and keeps waiting for a new head flit – which will never come for a permanent fault case. All the other fault-free routers and links *downstream* of the faulty link are released and recovered to use. Until now, the fault-caused physical-layer deadlock has been eliminated from the network. The



(a) Wormhole (b) SDM

Figure 6.6: Link connections using wormhole and SDM

next step is to isolate the defective link from the network to prevent it from affecting the future packets, which can be implemented by using Spatial Division Multiplexing (SDM) [LMV+08, SE11b].

6.2 Faulty link isolation based on SDM

For a system with permanent faults, the usual recovery method is to block the defective component and use redundancy to compensate for its function, which may incur a large hardware overhead. When it comes to NoCs, fault-tolerant adaptive routings [IY11, FLJ+13] could be used to bypass the defective link. The recovery method proposed in this research is based on the Spatial Division Multiplexing (SDM) [LMV+08, SE11b] which does not rely on any routing algorithms or duplication. Thus, the switch allocator of a SDM router may have a choice of sub-link for the output. If one sub-link is faulty, it is removed as a possible choice, reducing the link aggregate bandwidth but retaining the link.

6.2.1 Spatial Division Multiplexing (SDM)

Figure 6.6a shows a w-bit link between two wormhole routers, which can carry a w-bit data unit (or flit). Alternatively, the link between two SDM routers can be physically divided into m sub-links, each of which transfers (w/m)-bit wide flits separately (Figure 6.6b). The crossbar of a p-port SDM router becomes $pm \times pm$ (w/m)-bit wide [LMV+08]. In other words, a SDM link with m sub-links can transfer m flits of m different packets in parallel, but each requires a longer latency to reach its destination due to the narrowed bandwidth. From the whole system point of view, the performance of the SDM NoC is competitive with a network without SDM. SDM has been used to improve the network overall performance and it was proved that the SDM NoCs have advantages in energy consumption, area overhead and flexibility [LMV+08, SE11b]. This research utilises the permanent-fault-tolerance feature of SDM NoCs rather than going to the implementation details.

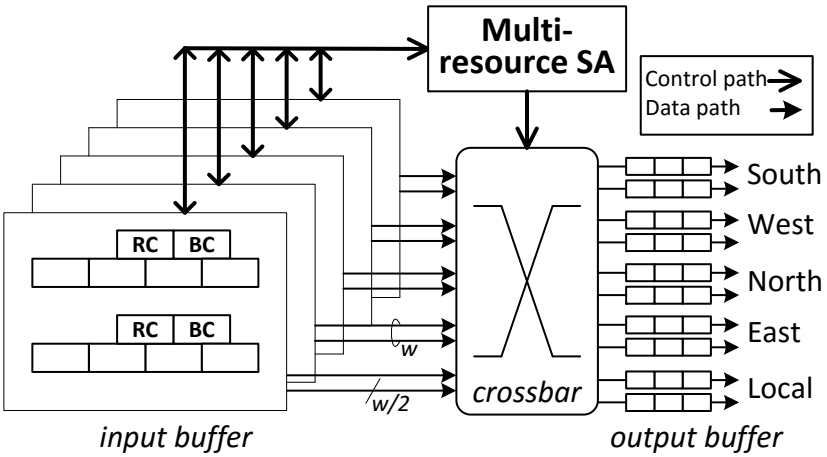


Figure 6.7: SDM router

Since a link is physically divided into multiple sub-links, a single fault will affect only one sub-link; the other sub-links of the same link can still be used to transfer packets to the same direction. This feature is inherent to SDM NoCs which supports the recovery from permanent faults. By reconfiguring the switch allocator of the SDM router to block the defective sub-link, following packets to the same output direction will go through fault-free sub-links and the network function is recovered.

It has been mentioned in Section 5.3 that, two redundant wires are added to each inter-router link to detect and locate a permanently faulty link. Adding network recovery, the fault or deadlock confirmation signal (*permConf*) generated at the deadlock detector of the pre-fault router output needs to be transmitted to the input of the post-fault router. Furthermore, since SDM divides every link into several physically independent sub-links, each sub-link requires its own deadlock detection and recovery techniques. Therefore, besides the redundant detection and recovery logic, three extra wires are added to each sub-link.

6.2.2 Switch allocator reconfiguration

Figure 6.7 illustrates a five-port SDM router example with each link divided into two sub-links, corresponding to halved input/output buffers at each port. The crossbar becomes 10×10 ($w/2$)-bit wide [LMV$^+$08]. The switch allocator becomes a multi-resource one, which needs to allocate one of the multiple output sub-links to each request from the multiple input sub-links each time.

Figure 6.8 shows a multi-resource arbiter example [GSX$^+$09, SE11b] used to implement the multi-resource switch allocator where multiple requests compete for one

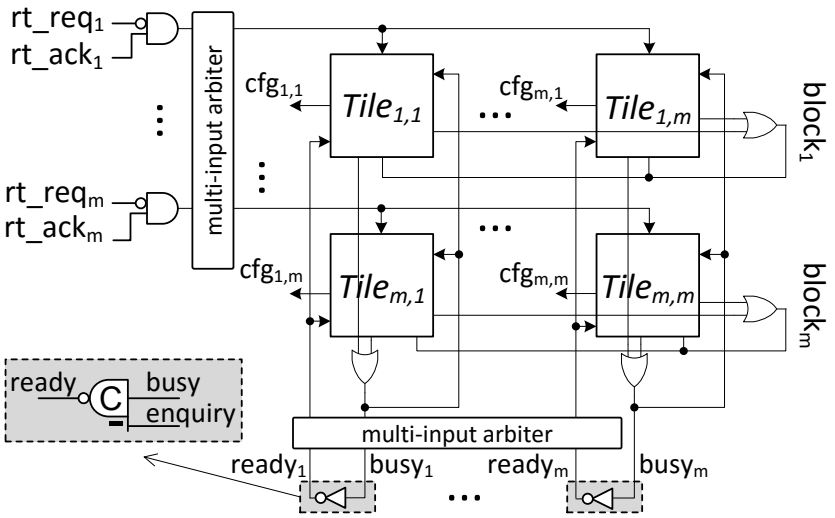


Figure 6.8: Multi-resource switch allocator [SE11b]

of the *m* resources (i.e. output sub-links). Two multi-input arbiters are used to select one routing request and one output sub-link respectively, constructing a match pair. The state of the output sub-link is indicated by a *ready* signal. A low *ready* indicates the output sub-link has been allocated to a request or it is transmitting a packet. Only an idle output with a high *ready* can be granted to the incoming winning request.

By keeping the *ready* indication flag low, the faulty output sub-link is isolated from the network. One way to implement this isolation for a permanent fault case is to ensure that, when the recovery process starts (or this deadlock is detected), the faulty link is "busy" and will never be "ready" (or idle) to use, so that this defective sub-link is isolated. To make this possible, the inverter shown in Figure 6.8 is replaced by an asymmetric C-element to generate the *ready* from the *busy* signal. In the detection circuits, a physical-layer deadlock can be confirmed only when the pre-fault router has allocated a packet to this faulty output sub-link (if this sub-link is idle and not transmitting any packets when the fault happens, it does not need to be recovered), so that the *busy* flag is high (*ready* is low). Using this asymmetric C-element, its minus input, *enquiry*, which went high when the sub-link was suspected to be deadlocked (the **ENQUIRY** state in Figure 5.9), will lock the *ready* to low level permanently for a permanent fault, even after the *Drain* operation when the router path is withdrawn (*busy*-), thus preventing following packets from using this defective link again. The succeeding packets requesting the same direction will be directed to other, fault-free non-busy sub-links. The network function is maintained.

6.3 Recovery of intermittent and transient faults

As Chapter 3 mentioned, intermittent faults may happen and appear after some time. In terms of intermittent faults, transitions will be detected at the input/output of the post/pre-fault router when the fault disappears. The state machine of the deadlock detector is afterwards reset to **IDLE** where the *enquiry* signal is low, unlocking the added asymmetric C-element in the switch allocator (Figure 6.8). The blocked sub-link is recovered to use again. This process has been depicted in Figure 5.9 with a dotted line. It should be noted that, one time-out period is required before recovering the blocked faulty sub-link to let the deadlock recovery operation finish so that the fault impact is cleared. Thus, when the intermittent fault is considered, the time-out period should be longer than the packet transmission delay through a router at least.

Transient faults can also break the handshake protocol and deadlock the QDI NoC (Chapter 5). Differently from a permanent fault, which leads to the isolation of the defective sub-link, a transient fault will not break a wire or gate, so that the logic is healthy and can be used again. Some existing fault-tolerant techniques focusing on transient faults on QDI circuits [JM05, ZSG+14b] can avoid the occurrence of the deadlock, but the requirement is all pipeline stages are continuously protected by extra circuits or duplication, which is highly expensive and impractical. A system or chip reboot can eliminate the transient-fault-caused deadlock, but is expensive as well and may be not allowed in some critical equipments [AIKR13]. Therefore, it is beneficial to propose a fine-grained recovery mechanism.

A sub-link (when SDM is used) deadlocked by a transient fault is not defective and should be reused. When the state machine in the deadlock detection circuit (Figure 5.15) goes to **CONFIRM** where the deadlock is detected and the fault type is known, the fault deadlocking the sub-link is regarded as permanent by default. All recovery operations dealing with a permanently faulty link (including *Drain&Release* and the blockage of the sub-link) are executed. It is reasonable to assume that they can finish in one long time-out period. If the fault is transient, the state machine will transit to **TF_CONFIRM** where the blocked sub-link is reset and resumed to use. Otherwise, the fault is permanent (**CONFIRM**) and the defective sub-link is blocked forever.

According to Section 5.4.2, deadlock detection circuits require three wires for each sub-link (using SDM) when both transient and permanent link faults are considered. Adding the network recovery, the fault confirmation signal needs to be transmitted to the post-fault router to start the *Release* operation. As a result, four redundant wires are added to each sub-link to implement the network detection and recovery.

6.4 Technical issues in deadlock detection & recovery

In a QDI NoC, the fault-caused physical-layer deadlock could cause more packet stalls at the network layer, which can spread across the whole network, decreasing its performance and destroying its communication function; this is much more harmful than data errors or packet loss. Chapters 5 and 6 proposed fault-tolerant techniques to detect the fault-caused physical-layer deadlock and then recover the network function. Redundant deadlock detection and recovery circuits are added to sample signals from the asynchronous NoC and control the flit flow during the recovery process, providing a complete fault-tolerant architecture.

It was demonstrated that, a pair of detection circuits (including a deadlock detector and a pattern checker) are added to network data path to detect the fault-caused physical-layer deadlock. When SDM is employed to support the network recovery from a link fault, each sub-link requires a pair of detection circuits. Two to four redundant wires are added to each link (or sub-link when SDM is used) to implement different scenarios of deadlock detection or recovery. Currently these redundant circuits are not protected, which increase the circuit area and could have a negative impact on the chip reliability. Given most runtime permanent faults are strongly related to the workload of the device [BM11, AFK+13], it is reasonable to assume that the overall reliability against permanent faults largely increases through protecting the links since there are far fewer activities on these extra circuits compared with the usual router logic, especially when a long time-out period is used.

Different from permanent faults caused by the ageing process, transient faults can happen randomly on the network. Their occurrence does not necessarily rely on signal activities. Considering the fault-tolerant designs proposed in this research, the state machine in the deadlock detector is critical and should be safe. Most of transient faults happening on detection circuits could be masked [KH04]. A transient fault may cause a wrong deadlock indication (*permConf*+) and start the network recovery process. This could result data errors, which can be tolerated using other fault-tolerant techniques [ZSG+14b, JM05]. The state machine controlling the detection and recovery process will keep running under the control of the clock and finally resets to the initial state due to detected flit flow. Another possible risk is that a transient-fault-caused physical-layer deadlock is mistaken into a permanent one due to a transient fault on the signal indicating the fault type. Under the assumption of a 1-bit fault during a detection cycle, this cannot happen. Since the physical-layer deadlock caused by

transient faults can paralyse the function of the whole network, using reasonable redundancy for deadlock detection and recovery could avoid system reboot and make the network more reliable. Some techniques such as scan chains [TTD+09] can be used to monitor and protect the behaviour of these redundant circuits periodically. In addition, a fault or the deadlock recovery operation can destroy a transmitting packet, resulting in packet loss. Extra retransmission mechanisms should be employed to redeliver the lost packet. These are left as future work, but are fairly "conventional" techniques.

Another technical issue is about the implementation of the synchronous deadlock detection circuits. The clock needs to drive only the state machine in deadlock detection circuits. There is no requirement on its skew, jitter and frequency. Different routers may use different clocks, producing different time-out periods. In fact, designers can use any clock sources in the NoC, such as the local clock from the synchronous processing element or a slow global clock. Also since every deadlock detector works independently, it can have its own counter to generate a *timeout* signal at arbitrary frequencies or share the counter (*timeout* as well) among arbitrary number of neighbours for area efficiency. The total latency T_{det} of detecting the fault that causes a physical-layer deadlock from the fault occurrence has two parts:

$$T_{det} = T_{deadlock} + T_{report} \tag{6.1}$$

where $T_{deadlock}$ denotes the time to form a deadlock and T_{report} is delay of reporting or confirming this deadlock (at state **CONFIRM**, in Figure 5.9, 5.11 and 5.15). $T_{deadlock}$ is out of control of the deadlock detection and recovery. It is short on a busy link or internal router path but infinite on an idle channel. Nevertheless, no damage is made if no deadlock is formed.

The latency reporting the confirmation of the fault-caused deadlock T_{report} is an important factor in evaluating the detection speed. The upper and lower bounds of T_{report} can be described as (6.2):

$$2T_t \leq T_{report} \leq 4T_t \tag{6.2}$$

where T_t is the time-out period. The lower bound is achieved when the deadlock is formed just before the state machine in the detection circuits transits to **START**. The upper bound is reached when the deadlock occurs before **IDLE**. There is no strong constraint on the period of the clock (T_{clk}) and the time-out period (T_t). Obviously reducing T_t decreases the report latency T_{report}, but several issues should be considered:

- If only a permanent fault or only deadlock detection is considered, T_t should be larger than T_{clk} and the latency of transmitting one flit through one pipeline stage (usually several nanoseconds, to allow a stable sample from transition detectors).
- If the proposed network recovery technique is employed to deal with deadlocks caused by intermittent or transient faults, T_t should be long enough (at least longer than a packet transmission latency through a router) to ensure that all recovery operations are finished in one time-out period, so that the state machine can safely transit to the initial state **IDLE**. This case is employed in the implemented NoC.

As mentioned in Section 3.1.3, an intermittent fault can be taken as a long transient fault or a permanent fault depending its length. Both the intermittent and transient faults will disappear, so that the previously blocked faulty sub-link should be recovered to use after the default recovery process (for a permanent fault). Their recovery methods are different. An intermittent fault is assumed to disappear after the recovery process (if it disappears during the detection process, no deadlocks happen) and the resulting deadlock pattern is the same as the one caused by permanent faults, so that the resulting signal transition can be captured by the deadlock detector and recover the blocked sub-link after the default recovery process (Figure 5.9). Differently, a transient fault disappears during the deadlock detection process, which will not cause detectable transitions once the physical-layer deadlock has been formed, so that the faulty link is recovered only if the fault is diagnosed as transient after the deadlock detection (Figure 5.15).

In the implemented NoC with both deadlock detection and recovery, a long time-out period is employed. On the one hand, it ensures the network function has enough

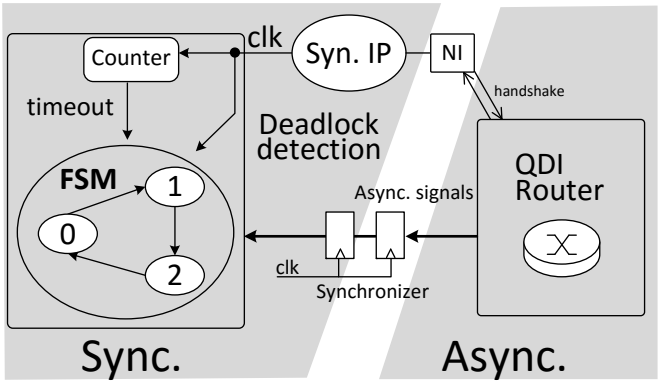


Figure 6.9: Interface between the detection circuits and the QDI NoC

time to recover after the deadlock detection; on the other hand, a long time-out period can reduce the possibility of the metastability caused by the sampling of asynchronous signals by synchronous circuits every one time-out period. Figure 6.9 shows the interface between the synchronous deadlock detection circuits and the asynchronous router. Transition detectors are enabled to monitor activities of some asynchronous signals from the second time-out period (the **START** state, Figure 5.9). If a transition is detected, the sampling stops and the state machine will be reset afterwards. Otherwise, asynchronous signals are sampled every one time-out period to check against the deadlock patterns, where the metastability could happen. The metastability possibility decreases after each long time-out since the asynchronous data path becomes more like to be deadlocked. A synchronizer [Spa07, Gin11] is added between the synchronous detection circuit and the asynchronous router to further reduce the metastability effects. Therefore, it can be claimed that this async./sync. interface design is highly robust against metastability.

This chapter presents the recovery of a link fault that causes a physical-layer deadlock. A fault on the router logic could also deadlock the NoC, which has been able to be detected to some extent (Section 5.3.5). However, its recovery is difficult if no redundancy or spares are used. If a permanent fault is confirmed to be happening in a router, the usual way is using fault-tolerant routings to detour the whole router [IY11]. To tolerate transient faults in the router logic, error detection or correction is usually added to the router ports to correct transient faults on the fly so that no deadlock could happen [RFZJ13]. The recovery of the network function from a router fault is left as future work. To our best knowledge, there is no previous publication which fully tolerates a permanently faulty link of QDI NoCs. The proposed detection and recovery techniques are novel; they first solve the serious fault-caused physical-layer deadlock problems in QDI NoCs.

6.5 Summary

It can be concluded that the network recovery from a fault-caused physical-layer deadlock can be easily implemented using the SDM. In terms of a permanently faulty sub-link, it is shut down so that all the following packets requesting the same direction will go through fault-free ones. Though this removal of the faulty sub-link leads to a moderate performance loss, it is acceptable for specific critical digital systems since keeping them working is important. Note that most redundant circuits are used to

eliminate the deadlock and clean the flits remaining in the reserved fault-free network resources (*Drain&Release* operation), which is unavoidable for a fine-grained network recovery in QDI NoCs (a coarse-grained recovery strategy such like a system reboot does not need this process, but it cannot locate the faulty component without further overhead such as built-in-self-test circuits [McC85]). The reconfiguration of the route connection depends on the natural design of the SDM router, which induces little area overhead. When a transient or intermittent link fault is also considered, the previously blocked sub-link can be reused again so that the network function is fully recovered. Detailed experimental results will be presented in Chapter 7.

These proposed recovery methods do not rely on any specific routings. Other traditional recovery techniques, such as fault-tolerant routings, can be further used to release the traffic pressure of the damaged link, avoiding making hot spots in the network and compensating for the performance loss, which is left as future work. In addition, the proposed fault detection and recovery techniques do not try to correct faults, so that packet loss is possible, which can be detected in a higher network layer. Using retransmission or other fault-tolerant techniques, the lost packet can be routed to the destination or recovered, which is outside the scope of this research and left as future work as well.

Chapter 7

Performance and fault tolerance evaluation

Previous chapters proposed several techniques to improve the fault-tolerance of QDI communication and NoCs. This chapter describes implementations of both unprotected and protected QDI routers and NoCs to provide a detailed performance and fault-tolerance evaluation. The UMC 130 *nm* standard cell library is used for hardware implementation. Asynchronous cells, such as C-elements and Mutex Elements (MEs) [SF01], are built using standard cells (Appendix A). This chapter first presents designs of main router components, from which a wormhole, and further a Spatial Division Multiplexing (SDM), router is built. Adding detection and recovery circuits to SDM routers, the physical-layer deadlock on the network data path can be detected and a link fault can be recovered. To evaluate the network performance and its fault tolerance capability, a SystemC/Verilog [IEE12, TM02] mixed environment is constructed where the attached Processing Elements (PEs) are SystemC models which can inject/receive packets into/from the QDI NoC built from gate-level post-synthesis routers. Thus, by simulating unprotected/protected NoCs, the implementation of proposed fault-tolerant techniques can be evaluated. By constructing a fault environment over the NoC, the network fault-tolerances are analysed.

7.1 Router and NoC implementation

The structure of a five-port asynchronous wormhole router was proposed in Section 2.2.7 (Figure 2.23), which employs a 4-phase, 1-of-4 asynchronous protocol and can be used to build a 2D-mesh network. Its working manner was depicted in Section 5.1.

Spatial Division Multiplexing (SDM) was proposed to improve the overall network performance due to its advantages in energy consumption, area overhead and flexibility [LMV+08, SE11b]. A SDM router built from the wormhole one was depicted in Figure 6.7. This chapter uses SDM QDI NoCs to evaluate the proposed fault-tolerant techniques because of their inherent fault-tolerance capability against a link fault: a link fault affects only one sub-link in a SDM NoC while the other healthy sub-links of the same link can still work. Thus, as a design case, fault detection and recovery circuits are added to SDM routers to provide protection from the fault-caused physical-layer deadlocks. It should be noted that the proposed fault-tolerant techniques in this research are general for 4-phase 1-of-n QDI designs (or QDI NoCs), they do not have specific requirement on the router implementation.

Figure 7.1 illustrates the data path through a router, containing input/output buffers at the ends of inter-router links and the crossbar (Figure 7.2). Half buffers built from C-elements are used to construct the input/output buffers, forming a 4-phase, 1-of-4 QDI pipeline. Depending on the link width, the pipeline may contain multiple 1-of-4 sub-channels, which are synchronized by a Completion Detector (CD) at each pipeline stage. Both the completion signal indicating a complete 1-of-4 word and the End-of-Packet (EoP) signal indicating the end of a packet (i.e. the tail flit) could trigger the backward *ack* signal high, starting a return-to-zero phase. The flit flow is controlled

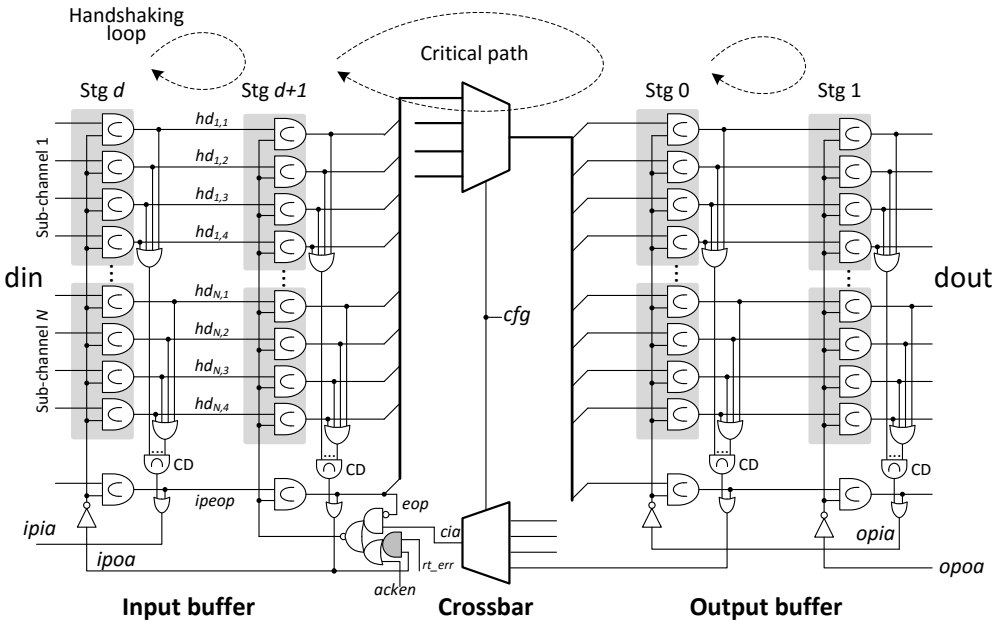


Figure 7.1: Data path through a wormhole router

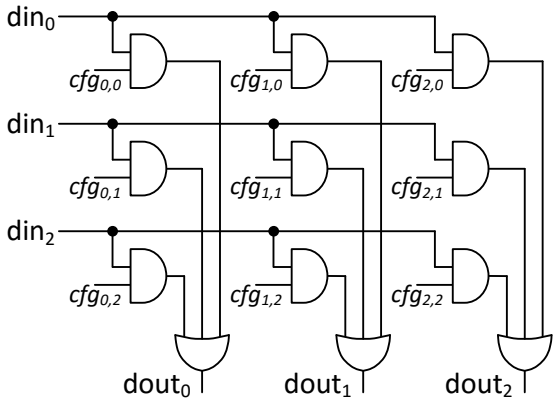
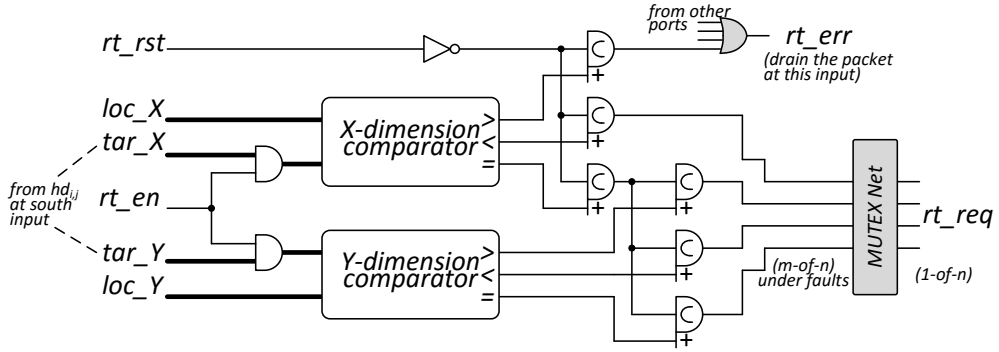
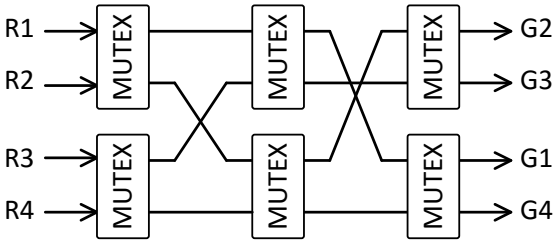


Figure 7.2: Router crossbar (3×3)



(a) RC at router input (X-dimension) [SE11b]



(b) MUTEX net [DGK09]

Figure 7.3: Routing Computation unit for XY-dimension ordered routing

by a Buffer Controller (BC) at input, which has been introduced in Section 6.1. According to the wormhole switching [DT03], when a packet arrives at a router, its head flit (denoted by $hd_{i,j}$) is first blocked before the headmost stage of the input buffer (*Stg d*+1), waiting to be granted an output. It can be found from Figure 7.1 that the longest handshaking loop is between the pipeline stages at the input and output sides of the crossbar. Without the global clock, this handshaking loop is the critical path that decides the equivalent period or saturation throughput of a built packet path.

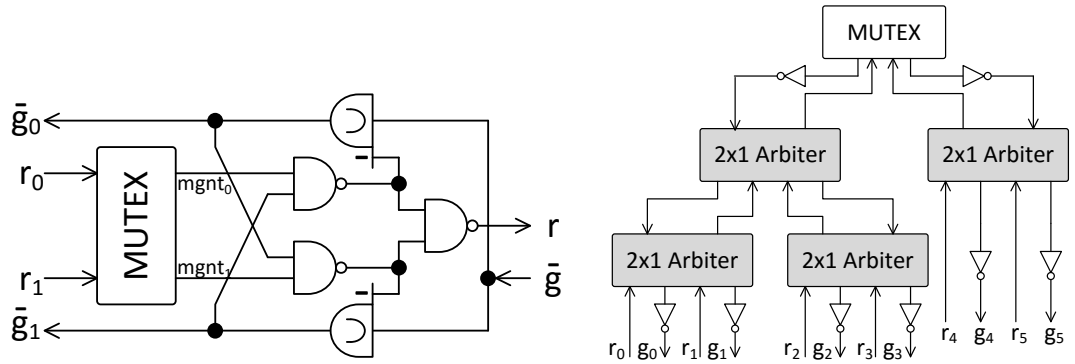
Figure 7.3a depicts the Routing Computation (RC) unit at the input port of the

router [SE11b]. The main components of RC are comparators, which compare the detected packet destination address (indicated by *tar_X* and *tar_Y*) with the local router address (indicated by *loc_X* and *loc_Y*), and calculate the routing request (*rt_req*) to the switch allocator to ask for an available output channel. Take the RC at the X-dimension input for example. XY-Dimension-Ordered Routing (XY-DOR) is used in the mesh network (a packet must transfer along the X-dimension first before it goes to the Y-dimension), so that the X-dimension comparator first compares *loc_X* with *tar_X* to decide the next hop direction. If *loc_X* = *tar_X*, it means the packet has arrived at the last router at the X-dimension and needs to turn to the Y-dimension, where the Y-dimension addresses (*loc_Y* and *tar_Y*) will be compared. If *loc_Y*=*tar_Y*, this node is the destination and the packet goes to the router *Local* output.

In a fault-free case, the generated routing request *rt_req* is 1-of-n. Under a 1-bit fault environment, a fault may mutate the head flit, resulting a 2-of-n routing request. In this case, a MUTEX net (Figure 7.3b) built from C_n^2 Mutex Elements [DGK09] is added to the RC unit to force the routing request back to 1-of-n. As a result, most faulty scenarios can be filtered. In some cases, if the final request is 1-of-n but directs to an incorrect direction, the packet may transmit to the wrong destination and cause packet loss, which can be detected in upper network layers [Sor09]. Under the XY-DOR, a faulty packet may eventually get blocked at one router input since the decoded destination address cannot be reached, violating the routing protocol. This scenario can be indicated by a high *rt_err* signal (*acken* is low, Figure 7.1) at the innermost pipeline stage of the router input buffer (*Stg d*+1) which builds a *sink* at this router input to drop the packet [SE11b].

For example, an incoming packet from the *South* input link should not request a *South* output according to the applied network protocol. Therefore, under a fault environment, if the decoded destination address directs to the *South* output, a fault indication signal *rt_err* is set high (*acken* is low) to drop the packet immediately from this input buffer (Figure 7.1). In this way, most of the general impact of a fault on the NoC data path (Section 5.2.2), which leads to a packet blockage at the network layer, can be tolerated. The fault-caused physical-layer deadlock in a QDI NoC cannot be managed by these conventional techniques, which is more harmful to the network than packet loss. This chapter will give detailed experimental results about the implementation of the proposed fault-tolerant techniques.

The most complicated control logic in a router is the Switch Allocator (SA), which controls the allocation of an output to an input. Without using SDM, the allocator



(a) Arbiter element (2×1) [JY96] (b) 6×1 tree arbiter

Figure 7.4: Arbiter implementation [JY96, SE11b]

in a wormhole router can use the tree arbiter shown in Figure 7.4. The tree arbiter contains multiple 2×1 arbiter elements, in which the Mutex Element (ME) is the main component. As Figure 7.4a shows, if neither input requests have been granted ($\overline{g}_0$ and $\overline{g}_2$ are '1's), the ME grants one of the requests and its corresponding grant signal is set high. After the granted request is withdrawn, the other request will be granted. Different from a usual wormhole router, where one direction is corresponding to one output channel, a SDM router has multiple narrower output channels to the same direction. The switch allocator in a SDM router needs to allocate these multiple output resources to inputs, requiring a multi-resource arbiter (Figure 6.8) which has been introduced in Section 6.2.

The baseline router without fault-tolerance is built from the above main components. Adding redundant fault-tolerant circuits as Chapters 5 and 6 introduced, the resulting SDM router can tolerate the fault-caused physical-layer deadlocks.

7.2 Area, energy and speed evaluation

A SystemC/Verilog [IEE12, TM02] mixed environment was built to evaluate the network performance, as Figure 7.5 shows. The protected and unprotected routers have been implemented using the UMC 130 *nm* standard cell library and synthesized by the Synopsys Design Compiler [Syn13] with default wire load models. Since this is the first thorough research on the fault-caused deadlock in QDI NoCs to the best knowledge of the author, instead of exploring absolute performance results or comparing with irrelevant baselines, the experiments target on comparing the protected NoCs with unprotected ones to evaluate the performance and hardware overhead brought

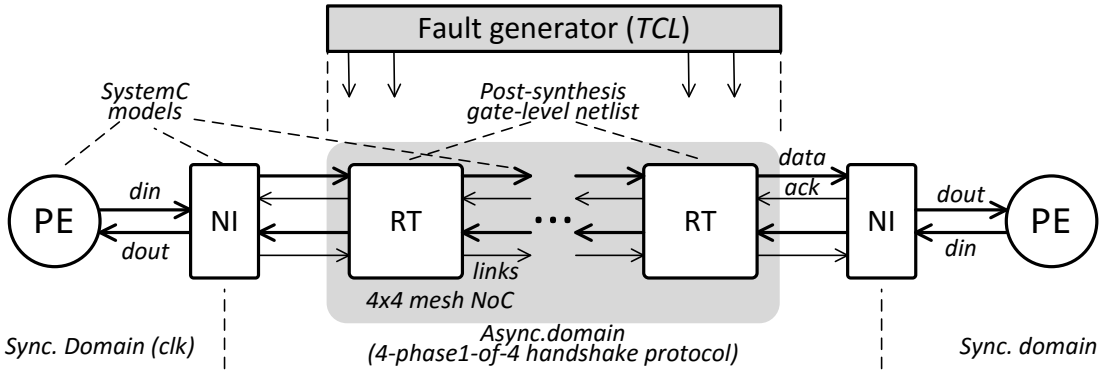


Figure 7.5: NoC simulation model

by the proposed fault-tolerant techniques. Note that, although circuits implemented using more advanced semiconductor technologies exhibit different performance, the UMC 130 *nm* library is currently available to use, which is convenient for verifying the proposed techniques and evaluating the general performance overhead brought by the added fault-tolerance.

In the experiments, both the protected and unprotected routers share the same configuration of five ports (South, West, North, East and Local in Figure 6.7) – two pipeline stages in input buffers and three in output buffers. Similar to the approach employed in Chapter 4, representative pipeline stages were created using standard cells. Results were collected from post-synthesis gate-level netlists (i.e. no place and route was performed). The interconnection "wiring" that includes periodic asynchronous buffers was modelled using SystemC to insert synthesised gate-level stages (annotated with gate latency). The number of stages depends on the practical application, but with an asynchronous pipeline they can be inserted arbitrarily. The logically more complex stages in the router are slower than simple repeaters in the link and, for modelling purposes, it was assumed that the delay due to long wires could be factored into that at layout time. In other words, since the handshaking loop between the pipeline stages at the input and output sides of the crossbar is the critical path (Figure 7.1), which determines the saturation throughput of a built packet path (or pipeline), using deeper input/output buffers will not affect the pipeline throughput but reduce the network contention. As a result, the network throughput increases. The number of stages in input/output buffers can be flexibly adjusted according to the practical design requirement. Therefore, the inter-router link connecting two post-synthesis gate-level routers includes gate-level pipelined buffers and SystemC-modelled link wires.

The experiments target on comparing the protected NoCs with unprotected ones

to evaluate the fault-tolerant techniques, rather than exploring absolute performance results. The router area can be achieved from the synthesis report. To achieve the speed and energy information of the network, 16 gate-level, post-synthesis routers were connected to build a 4×4 2D-mesh network. Each router was annotated with the gate latency. Synchronous Processing Elements (PEs), Network Interfaces (NIs) and the interconnection between routers were implemented in SystemC. All PEs kept injecting random packets to and receiving packets from the network, so that the network traffic was uniformly distributed. Since a 64-byte-long packet payload has been adopted in many existing NoCs to fit the size of cache blocks [Kes99, BW04, GKS+07, MM09, GXZ09, SMK10], the packet payload was fixed to 64 bytes in simulations to evaluate the network performance. The flit width was set to the width of the inter-router links (or sub-links when SDM was used). NIs implement the protocol transformation between the synchronous domain of PEs and the asynchronous domain of the NoC. Packets are encoded into 1-of-4 words when they enter into the asynchronous network and are decoded when they leave the NoC. By simulating the unprotected baseline QDI SDM NoC and the protected one, the performance overhead brought by the added different fault-tolerance can be evaluated.

The area (μm^2), throughput (*MByte/s/node*), energy (*pJ/bit*) and minimum transmission latency (*ns*) of different SDM routers or NoCs under incremental protection from permanent faults are compared, including *unprotected*, *protected links with fault detection*, *protected data path with fault detection*, and *protected links with fault recovery*. The percentage inside the parentheses denotes the overhead brought by the fault-tolerance compared with the *unprotected* baseline, which is calculated using (7.1).

$$Overhead(\%) = (\#(protected) - \#(unprotected))/\#(unprotected) \tag{7.1}$$

The presented results at the *fault detection* columns illustrate the hardware and performance overhead brought by the deadlock detection circuits, which are the same for all kinds of faults (including a permanent, transient or intermittent fault; they all can break the handshake protocol). *DW* represents the data width of the inter-router link (in *bits*), each of which is divided into *SN* sub-links using SDM. Thus the width of each sub-link is *DW/SN*-bit (16-bit in this case), which is the flit width as well.

It can be found from Table 7.1 that, the area of a 32-bit baseline SDM router (*DW*=32, *SN*=2) is 71,123 μm^2. Adding deadlock detection circuits to protect the inter-router links (Figure 5.6a), the router area increases by 2.0% and reaches 72,580 μm^2. If the router logic is also protected through adding one more pair of detection circuits

Table 7.1: Comparison of 32-bit NoCs with incremental permanent-fault-tolerance

32-bit 4-phase 1-of-4 QDI SDM NoC with each link divided into 2 sub-links (DW=32, SN=2)		**Unprotected baseline NoC**	**Protected links/data paths with *fault detection***		**Protected links with *fault recovery***
			protected link	**protected data path**	
Router	Area (μm^2)	71,123	72,580 (2.0%)	74,525 (4.8%)	74,038 (4.9%)
	Average energy (*pJ/bit*)	0.544	0.548 (0.7%)	0.552 (1.8%)	0.551 (1.3%)
NoC	Average minimum Latency (*ns*)	110.0	113.0 (2.7%)	114.0 (3.6%)	115.0 (4.5%)
	Saturation throughput (*MByte/Node/s*)	720	700 (-2.7%)	696 (-3.3%)	694 (-3.6%)

Table 7.2: Comparison of 64-bit NoCs with incremental permanent-fault-tolerance

64-bit 4-phase 1-of-4 QDI SDM NoC with each link divided into 4 sub-links (DW=64, SN=4)		**Unprotected baseline NoC**	**Protected links/data paths with *fault detection***		**Protected links with *fault recovery***
			protected link	**protected data path**	
Router	Area (μm^2)	203,241	206,477 (1.6%)	213,103 (4.9%)	212,983 (4.8%)
	Average energy (*pJ/bit*)	0.599	0.609 (1.7%)	0.610 (1.8%)	0.612 (2.2%)
NoC	Average minimum Latency (*ns*)	124.9	127.6 (2.1%)	128.2 (2.6%)	128.0 (2.5%)
	Saturation throughput (*MByte/Node/s*)	1,500	1,472 (-1.9%)	1,461 (-2.6%)	1,465 (-2.4%)

across the router crossbar (Figure 5.10a), the physical-layer deadlock on the NoC data path is detectable; the router area increases by 4.8% and reaches 74,525 μm^2. Since the protection of the router logic requires an extra crossbar to exchange information between the router input and output, it brings more overhead to the router area than the protection of the inter-router link despite its pattern checker is simpler. Besides the deadlock detection, when the network recovery from a permanently faulty sub-link is implemented (Figure 6.2), the router area reaches 74,038 μm^2, an increase of 4.9% compared with the unprotected one.

The area results of 64-bit routers (*DW*=64, *SN*=2) are presented in Table 7.2. It can be found that, adding deadlock detection circuits to protect the inter-router links, the router area increases by less than 2.0% compared with the unprotected baseline. Adding deadlock detection to protect the whole data path (including the inter-router link and the router logic) or implementing the deadlock detection and network recovery from a permanently faulty sub-link, increases the router area by less than 5.0%.

Table 7.3 gives an overview of the area of main components in a SDM router protected with deadlock detection and recovery. The multi-resource switch allocator takes about 10% of the total router area in both routers. The crossbar takes the most area in

Table 7.3: Area of main components in a protected SDM router

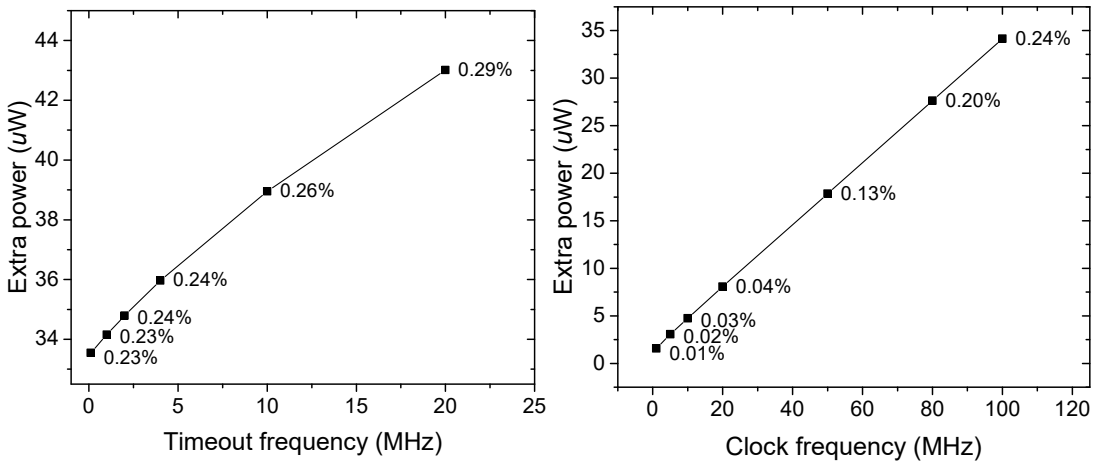
Area (μm^2)	Switch Allocator	Crossbar	Input buffers	Output buffers	Total
DW=32, SN=2	6,408	20,758	23,828	23,044	74,038
DW=64, SN=4	21,806	98,369	46,439	46,369	21,2983

the 64-bit SDM router (*DW*=64, *SN*=2), which is as high as 46%. In the implemented routers, the input buffer has two pipeline stages while the output buffer has three. Due to the Routing Computation unit and the Buffer Controller (for each sub-link of a SDM router) being included in the input buffers, the input and output buffers have a similar area. This router design and optimization is not the topic of this research. Details about implementing a SDM router have been published [LMV+08,SE11b]. This chapter concerns more about the hardware overhead brought by the fault-tolerance.

To evaluate the energy and throughput of the network, all PEs keep injecting random packets to the network at the maximum rate, achieving the saturation throughput. This means as long as there is space in a router, the attached PE will send out a packet. As long as a packet arrives at a PE, the packet will get accepted. By counting the received packets during a period of simulation time, the network throughput can be determined. The clock and time-out frequencies used by the deadlock detection and recovery circuits are set to 100 *MHz* and 1 *MHz* respectively. The network saturation throughput of a 32-bit unprotected SDM NoC (*DW*=32, *SN*=2) is 720 *MByte/Node/s*. When including deadlock detection to inter-router links, the saturation throughput decreases by 2.7% to 700 *MByte/Node/s*. If deadlock detection circuits are added to both links and routers, the saturation throughput decreases to 696 *MByte/Node/s*. To implement the network recovery, most redundant circuits are added to the data path of the network (Figure 6.2), increasing the packet transmission latency. As Section 6.1 mentions, since the router crossbar belongs to the critical path limiting the network throughput, the redundant circuits for network recovery are added to different pipeline stages which are not in the critical path, reducing the performance decrease accompanied with the fault tolerance. As a result, implementing deadlock detection and recovery on inter-router links incurs a throughput decrease by 3.6%.

The average energy of a NoC can be achieved by using (7.2).

$$Energy = power \cdot (simulation\ time)/((payload\ received) \cdot (router\ number)) \quad (7.2)$$



(a) Timeout

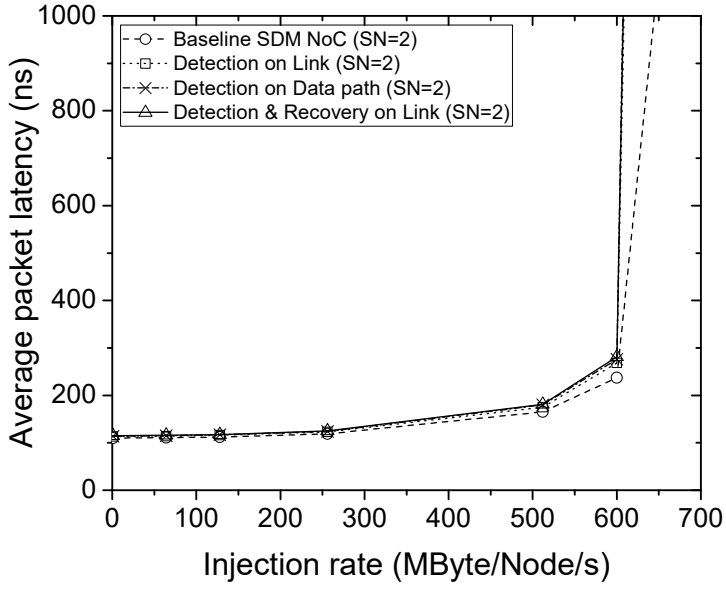
(b) Clock

Figure 7.6: Extra power consumption with various time-out and clock frequencies

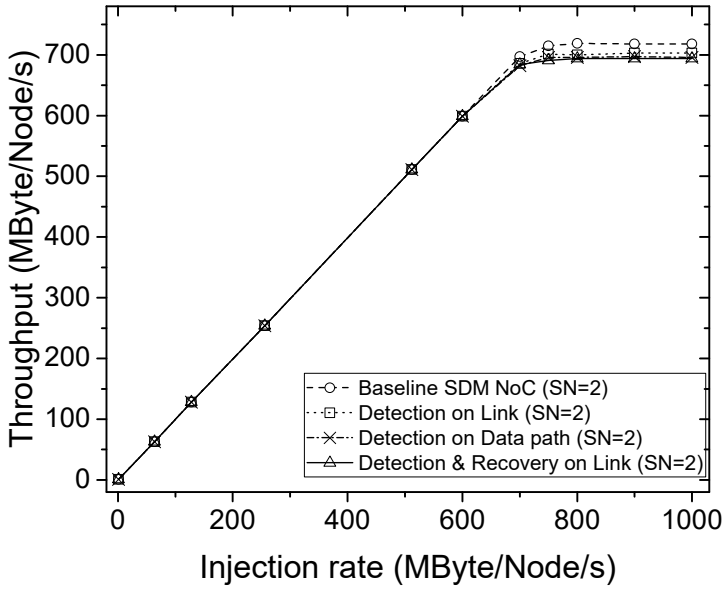
The power is calculated from the recorded signal transition activities when the network is saturated. For a 32-bit NoC, using deadlock detection on the inter-router links and the network data paths, the average energy of the protected NoC increases by 0.7% and 1.8% respectively compared with the unprotected one. Protecting the inter-router link using both detection and recovery circuits, the energy increases 1.3% on average.

It is found that the power overhead of the deadlock detection and recovery circuits is generally linear with the time-out and the clock frequencies. Figure 7.6a shows the router power with various time-out frequencies in a saturated 32-bit network (clock frequency set to 100 MHz) and Figure 7.6b reveals the increased power with increased clock frequency also in a saturated network (time-out frequency set to 1 MHz). The percentage labelling the curves denotes the ratio of the power of clocks and registers in detection and recovery circuits to the power of the whole router. According to the results, a slow clock and long time-out period should be used for low power consumption. This low ratio of the power of detection and recovery circuits also reflects the few signal transitions in the redundant circuits for fault-tolerance compared with the other main router components. Currently these redundant circuits, which are expected to have an extremely long life compared with other components (discussed in Section 6.4), are unprotected. Their protection is left as future work.

The energy and saturation throughput results for 64-bit NoCs are revealed in Table 7.2. It can be concluded that, comparing with the unprotected NoCs, adding deadlock detection circuits to inter-router links or the whole data path will decrease the throughput by less than 3.5% and increase the energy by less than 2.0%. Applying



(a) Average network latency



(b) Network throughput

Figure 7.7: Throughput and latency of 32-bit NoCs with different injection loads

network recovery for a permanently faulty link, at most the saturation throughput decreases by 3.6% and the average energy increases by 2.2%.

Another important metric to evaluate the network performance is the average packet transmission latency, or network latency. The packet transmission latency is the time that a packet needs to transit the whole network serially. It starts from the time when the head of a packet arrives at the network and ends at the time when the tail flit leaves

Table 7.4: NoCs protected from both transient and permanent faults

4-phase 1-of-4 QDI SDM NoC	Area (μm^2)	Average energy (*pJ/bit*)	Average minimum Latency (*ns*)	Saturation throughput (*MByte/Node/s*)
DW=32, SN=2	77,144 (8.5%)	0.61 (11.8%)	120.8 (9.8%)	661.5 (-8.5%)
DW=64, SN=4	220,565 (8.5%)	0.64 (6.6%)	137.0 (9.7%)	1,288.54 (-14.1%)

the network. Figure 7.7a depicts the network latency, which increases with increasing the packet injection rate. The minimum network latency can be achieved when the injection rate is close to zero. Packets were injected when the network was under zero load. As Table 7.1 and 7.2 show, the average minimum latencies of 32-bit and 64-bit unprotected SDM NoCs are about 110.0 *ns* and 125.0 *ns* respectively. When applying fault detection, the minimum network latency increases by no more than 3.6%. After incorporating fault recovery, it increases by 4.5% at most. Figure 7.7b illustrates the network throughput along with increasing injection rate. It can be found that, in the beginning when the injection rate is low, the packet congestion rarely happens so that the network throughput equals the injection rate generally. As the injection rate keeps increasing, the network traffic load becomes heavy and network congestion could happen. As a result, the network latency increases. After some point the network throughput will not increase any more, which means the network is saturated. The relatively steady throughput represents the saturation throughput, which has been reported in Table 7.1 and 7.2. The network latency, however, will increase dramatically after the network is saturated due to the heavy traffic load and severe congestion.

Table 7.4 presents the experimental results of SDM QDI routers or NoCs which can manage the physical-layer deadlocks caused by both a transient and permanent link fault. The percentage inside the parentheses denotes the overhead (calculated using (7.1)) compared with the baseline SDM NoC (or router) without any protection (Table 7.1 and 7.2). Compared with the network which deals with only permanent faults or the physical-layer deadlocks, fault-diagnosis circuits are added to the router inputs to diagnose the fault type after the deadlock is detected and located. The state machine based on a time-out mechanism requires one more state to recover the network function from a transient fault than the state machine optimized for one type of fault (Figure 5.15). As a result, compared with the baseline SDM NoCs without any fault tolerance (Table 7.1 and 7.2), the router area increases by 8.5% on average. The average energy of a router increases by 11.8% at most. Under light traffic where congestion can rarely happen, the average minimum latency of a packet in the protected

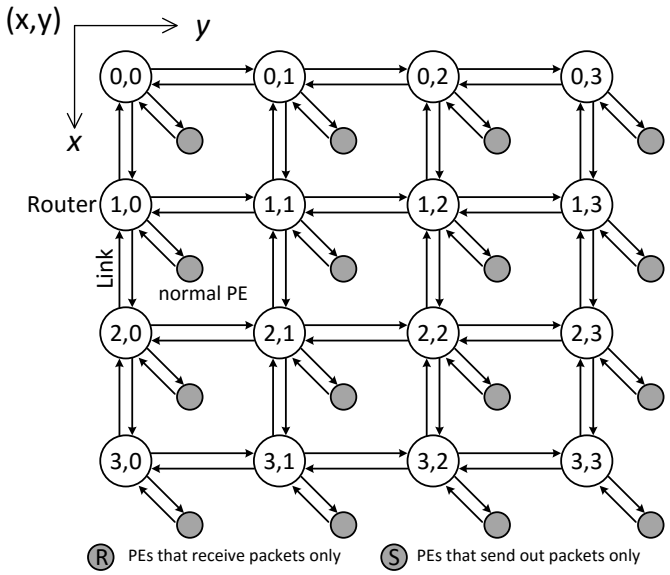


Figure 7.8: Fault-free 4×4 2D-mesh NoC

SDM NoC is 120.8 *ns* for the 32-bit network and 137.0 *ns* for the 64-bit one, which increases less than 10% compared with unprotected SDM NoC. Injecting packets into the network using the maximum speed, the obtained saturation throughput of the 32-bit NoC decreases by 8.5% while the 64-bit NoC runs 14.1% slower.

7.3 Fault tolerance evaluation

7.3.1 Effect of the physical-layer deadlock

Before evaluating the fault-tolerance capability of the network, the spread behaviour of a physical-layer deadlock in a network is discussed first. Figure 7.8 illustrates a fault-free 4×4 2D-mesh NoC where all nodes (routers and PEs) are indexed by their coordinates (x,y). There are 16 PEs, which attach to 16 routers through 32 unidirectional links and, 48 unidirectional inter-router links. XY-Dimension-Order Routing (XY-DOR) [DT03] is employed so that a packet would traverse the X-dimension first and then the Y-dimension. With wormhole switching, if a link fault breaks the handshake protocol and causes a physical-layer deadlock, a packet containing a sequence of flits may get blocked across multiple routers and links (Section 2.2.7). This physical-layer deadlock will cause more packet stalls, which spread over the network. As a result, more network resources will be blocked. The spread of this deadlock depends on the fault occurrence time, the fault position and the traffic load.

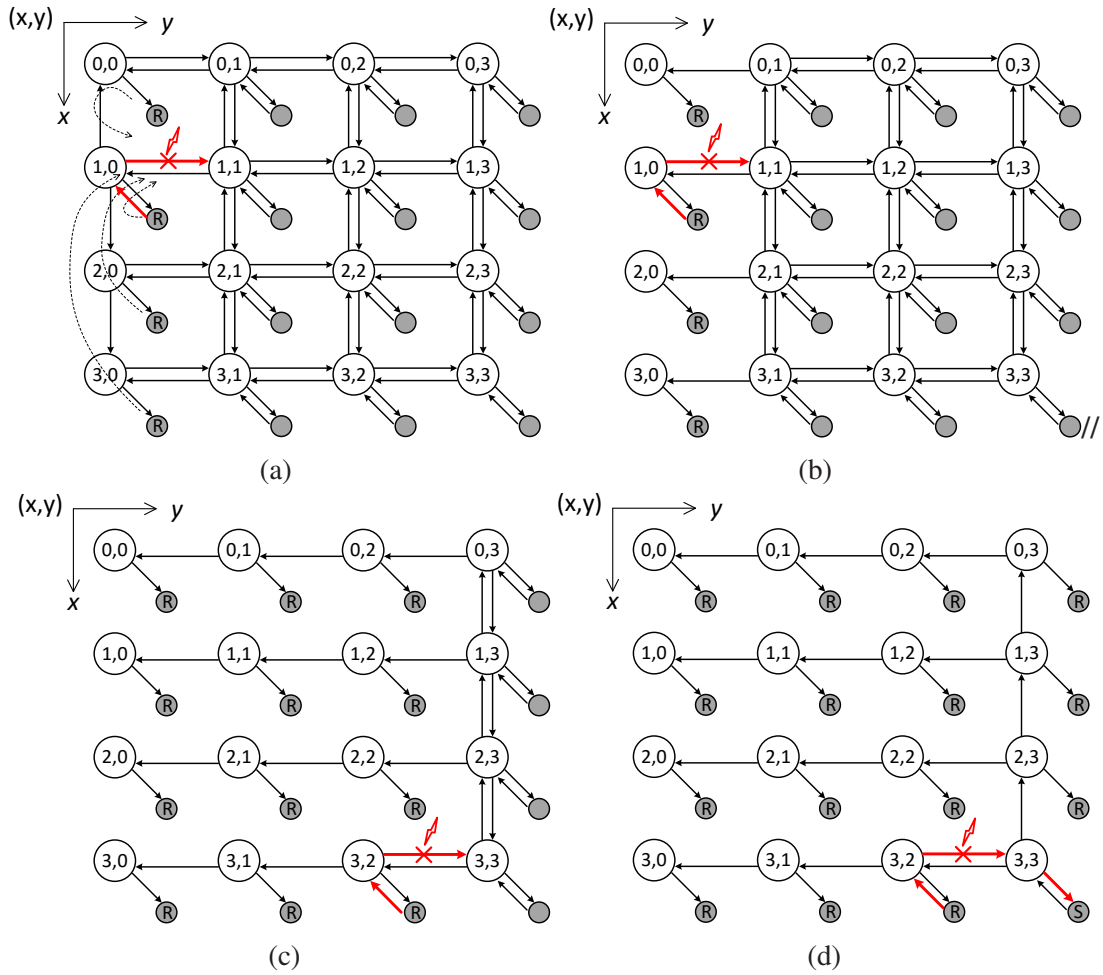


Figure 7.9: Incomplete network due to a permanently faulty link at the Y-dimension

A faulty link on the Y-axis

Figure 7.9a demonstrates a case that a permanently faulty link on the Y-axis prevents a packet from arriving at Router (1,1). The packet starts from PE (1,0) and directs to PE (1,j) ($1 \leq j \leq 3$). Its head flit is blocked before Router (1,1) due to the fault. As a result, all possible routes that need to traverse the defective link (1,0)→(1,1) are blocked. Since packets that need to traverse the faulty link come from PEs left of the fault (containing PEs with a coordinate of (i,0), $0 \leq i \leq 3$), these packets are all blocked in routers and links on the routes from those PEs to the faulty link. As a result, PEs (PE (i,0), $0 \leq i \leq 3$) left of the faulty link can only receive packets. Figure 7.9a presents the network removing the "dead" link connections (dotted lines represent the possible packet routes in a fault-free case). According to the employed XY-DOR, inter-router links (i,0) →(i,1) ($0 \leq i \leq 3$) at the Y-axis, and links at the X-axis that y=0

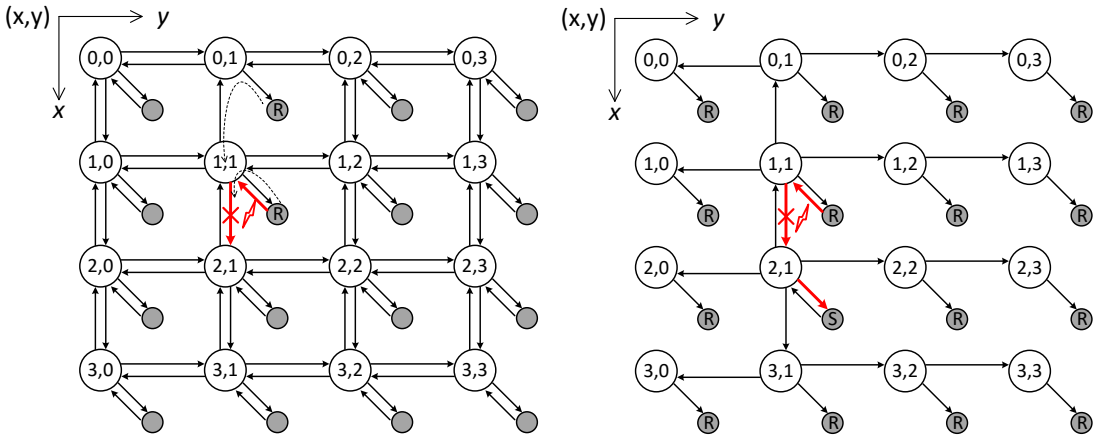


Figure 7.10: Incomplete network due to a permanently faulty link at the X-dimension

(Link (1,0)→(0,0), (1,0)→(2,0) and (2,0)→(3,0)) can only carry packets sent from PEs whose *y* equals '0', which cannot send packets any more. Thus these links can be removed from the network as well. Figure 7.9b presents the final incomplete network. It can be found that, 4 out of the 16 PEs can receive packets only. 14 out of the 80 link connections are removed from the network.

Figure 7.9c shows another case that a fault occurs on the link (3,2)→(3,3) and causes a deadlock, which can affect more network resources. The packet is produced at PE (3,2) and directs to PE (3,3). Due to the defective link, it is blocked before Router (3,3). As a result, all PEs left of the faulty link (PE with a coordinate of *y* smaller than three), which can send packets to the faulty link, will be blocked and cannot send out packets any more. Eventually, only 4 out of 16 PEs are normal which can receive and send out packets. The network function is seriously disrupted. If the link fault happens on a body flit which is passing through the link, the result could be more serious. As depicted in Figure 7.9d, the physical-layer deadlock happens when a packet route has been built from the source PE (3,2) to the destination PE (3,3). Since all the other PEs could send packets to the destination (3,3), they eventually cannot inject packets to the network any more. Only the PE (3,3) can send out packets. In a word, a fault is enough to paralyse a whole QDI NoC.

A faulty link on the X-axis

Figure 7.10a illustrates a case that a fault happens on link (1,1)→(2,1) and deadlocks the network. As a result, a packet originating from PE (1,1) cannot reach Router (2,1).

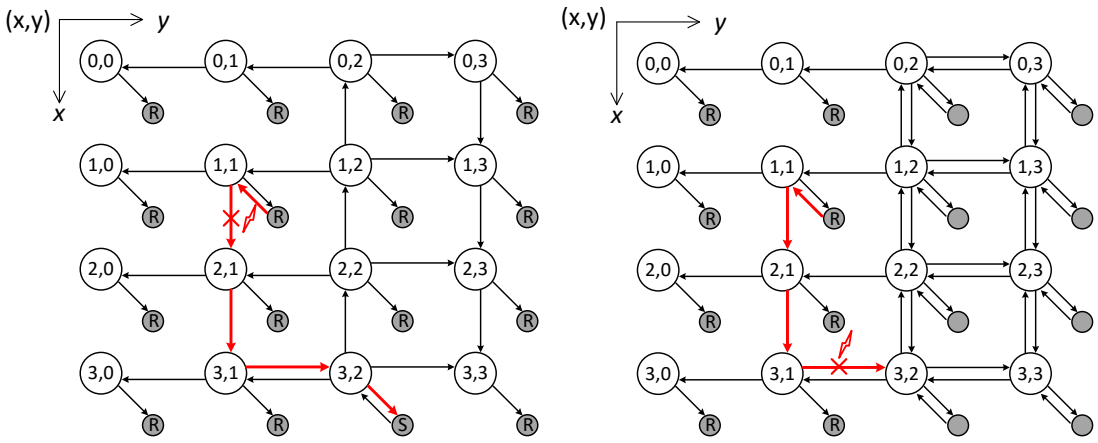


Figure 7.11: Incomplete network due to a fault on the built packet path

According to the XY-DOR, PE (0,1) can also send packets through the faulty link. As a result, PE (0,1) and (1,1) can only receive packets from the network. If a packet route through the faulty link has been built, as Figure 7.10b shows, the destination PE (2,1) of the faulty packet is the only node that can send out packets eventually while all the other PEs can only receive packets. Figure 7.11 illustrates cases that long built packet paths are deadlocked by faults.

It can be concluded that, **one fault that breaks the handshake protocol and causes a physical-layer deadlock can easily paralyse a QDI NoC**. This physical-layer deadlock cannot be recovered using upper-layer fault-tolerant techniques (see Chapter 3). Thus the proposed fault detection and recovery techniques targeting the physical-layer deadlock are important to the practical use of QDI NoCs.

7.3.2 Fault tolerance evaluation and comparison

To evaluate the fault-tolerance capability of a protected NoC, a fault environment is built over the NoC system(Figure 7.5). Using scripts written in Tool Command Language [Ous90], random faults and delays can be inserted to the network, including the inter-router links and post-synthesis routers, at any time during the simulation. Values of some specific signals in the network can also be probed. Assume PEs and the links between PEs and routers are fault-free. To verify the detection and recovery circuits, faults are randomly inserted to the network data path, including the router logic (crossbar) and inter-router links.

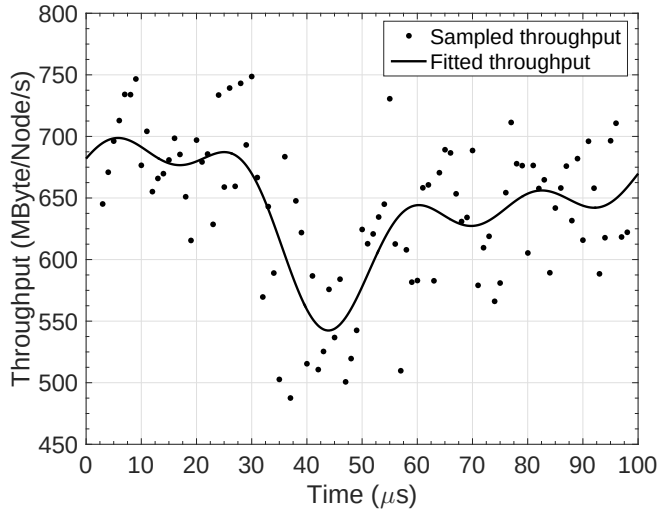
Test results show that, under a 1-bit fault environment, the fault-detection circuits

achieve 100% detection accuracy. As long as a fault occurs on the defined data path region of the NoC (Chapter 5) and causes a physical-layer deadlock, it can be detected and located precisely. The fault recovery is implemented only for link faults since the recovery of a router fault usually requires support from upper network layers (such as fault-tolerant routings to bypass the faulty router) [IY11], which is not the focus of this research. The simulation shows that, for a permanently faulty link, the network function is successfully recovered by isolating the defective sub-link from the network and releasing the other healthy network resources deadlocked previously. For intermittent faults, the previously isolated link can be reused. In the face of a transient-fault-caused deadlock (assume the transient fault happens on the forward path; the transient fault on the backward path cannot be differentiated from a permanent fault, which is left as future work), the fault diagnosis circuit could tell the right fault type and then the deadlock is removed from the network. It should be noticed that, the generation of a transient-fault-caused physical-layer deadlock requires data skew between two parallel 1-of-n sub-channels (belonging to one sub-link when SDM is used). This can be easily implemented by sampling the incoming transition of a signal and delaying its value change, which is like inserting a transient fault to the same level of the old signal value. The fault duration is the length of the delay. Afterwards, by inserting a different transient fault to the corresponding wire or gate output at the next pipeline stage, a physical-layer deadlock can be produced.

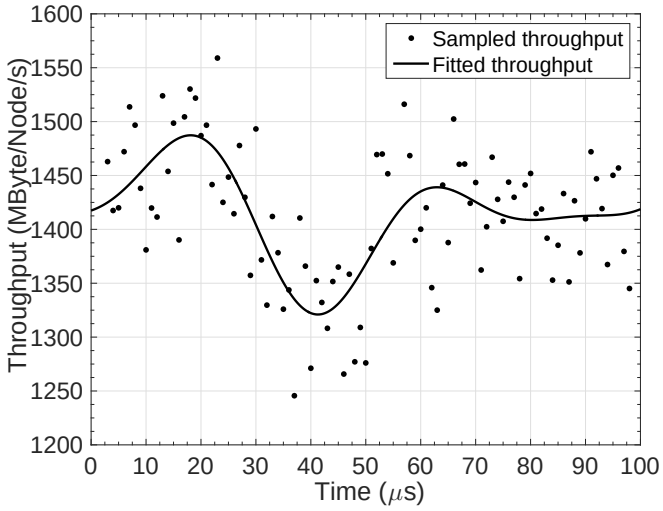
In practical designs, if tolerating transient fault is the target, some fault-tolerant techniques must be used to avoid data errors or packet loss. In a specific circuit requiring tolerance of permanent faults, keeping the system working can be the design target. Both cases share the point that a fault may break the handshake and cause a physical-layer deadlock. Thus the following experimental results concentrate on reflecting the management process of a physical-layer deadlock rather than differentiating the fault type. Toleration of transient faults, for example, can utilise the proposed Delay-Insensitive Redundant Check (DIRC) coding scheme (Chapter 4) or other redundancy techniques [JM05]. They can be combined with the proposed deadlock management techniques to provide a multi-level protection, which is future work.

The management of a fault-caused physical-layer deadlock in this research contains the detection and recovery process. This process can be obviously reflected in Figure 7.12. Figure 7.12 illustrates the change of the network throughput along the time after a fault deadlocks the network. The clock frequency is set to 100 MHz. Packets are injected to the network using the maximum rate so that the network runs

in a saturated state. To clearly differentiate the fault detection and recovery process, the time-out period is set "long enough" (10 μs) which is far longer than the packet latency through a router (less than 70 ns). The fault is inserted to the *West* input link to router (3,3) at 30 μs, leading to a steep decrease of the network throughput. If no SDM and fault-tolerant techniques are used, 12 out of the total 16 PEs will lose the injection function according to Figure 7.9c, so that the system function fails. For the protected 32-bit SDM NOC (SN=2, Figure 7.12a), its throughput steadies at around



(a) 32-bit SDM NoC (SN=2)



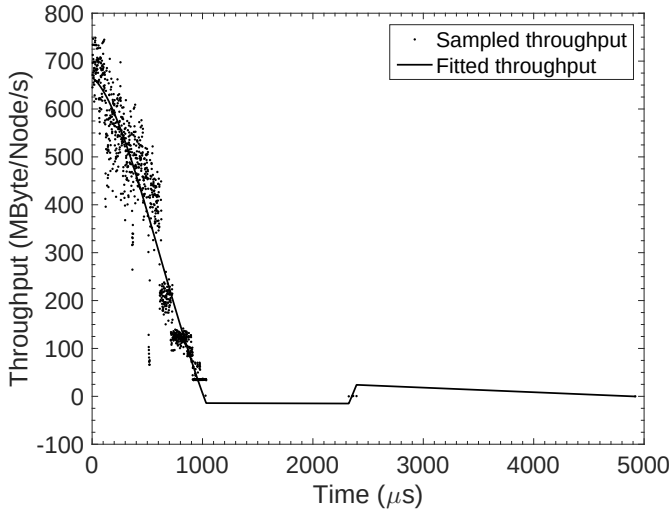
(b) 64-bit SDM NoC (SN=4)

Figure 7.12: Throughput of SDM NoCs with a stuck-at-1 fault

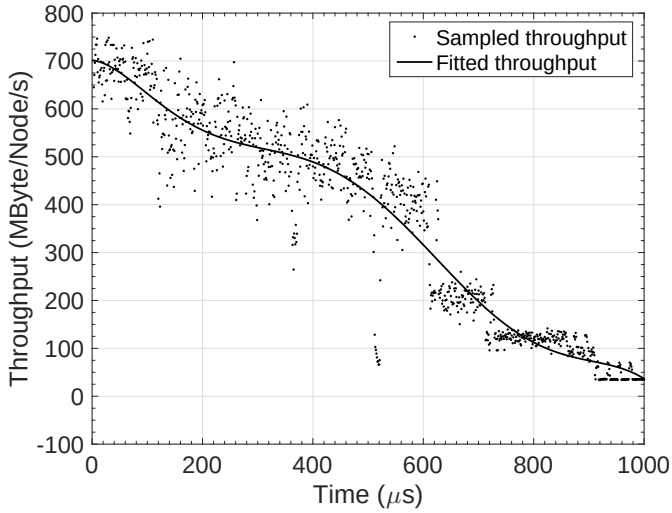
694 *MByte/Node/s* in the beginning. After the fault insertion which causes a physical-layer deadlock, the network throughput decreases rapidly to ∼550 *MByte/Node/s* at 44 μs. The deadlock is detected at 50.18 μs and then the network recovery starts. After the recovery process, the defective link is isolated. The network throughput returns to ∼650 *MByte/Node/s*, decreased by around 7.2% compared with the original saturation throughput without faults. Figure 7.12b illustrates the same fault on a 64-bit NoC. The network throughput drops from 1,480 *MByte/Node/s* to ∼1,320 *MByte/Node/s* rapidly and then returns to ∼1,410 *MByte/Node/s* after the network is recovered. As a result, only the defective link is removed from the network, which can behave properly with a slightly decreased performance. This clearly demonstrates the effectiveness of the proposed protection techniques.

The proposed deadlock management techniques can tolerate some multi-bit faults if the resulting deadlocked packet paths have no intersections and the faults do not block a whole link. When multi-bit faults happen, multiple sub-links of a link may be broken, leading to a fully blocked link. This is equivalent to the case that one permanent fault occurs on a wormhole NoC link. Using fault-tolerant routings [IY11], the network can be recovered. Figure 7.13 illustrates the case that multiple permanent faults are injected to a protected 32-bit SDM NoC (*SN*=2). Faults are inserted every 50 μs. The clock and time-out frequencies are set to 100 *MHz* and 0.2 *MHz* respectively. The simulation runs 10 *ms* so that in total 200 faults are randomly inserted into the network. Figure 7.13a shows the network throughput trend in the first 5 *ms*. It drops to around zero after 20 faults are inserted. Figure 7.13b picks the first 1 *ms* of the simulation. After 8 faults are inserted (400 μs), the network throughput can still be as high as 500 *MByte/Node/s*. When 10 or more faults are inserted, the network throughput drops quickly due to the lost link connections.

Figure 7.14 compares the throughputs of basic unprotected wormhole NoCs, unprotected SDM NoCs (*SN*=2), and protected SDM NoCs (*SN*=2, with fault detection and recovery on links). A fixed number of permanent faults are randomly inserted to the inter-router links in simulation. Every sample point is achieved by averaging the final saturation throughputs of 50 simulations. It can be found that, in both the 4×4 and 8×8 mesh networks, the throughput of the unprotected wormhole NoC drops dramatically when one fault happens. When two permanent faults are inserted, the final throughput is close to (or equal) zero. Almost all nodes of the network are stalled. This has been explained in Section 7.3.1, which demonstrates that one fault is enough to disrupt the whole NoC system.



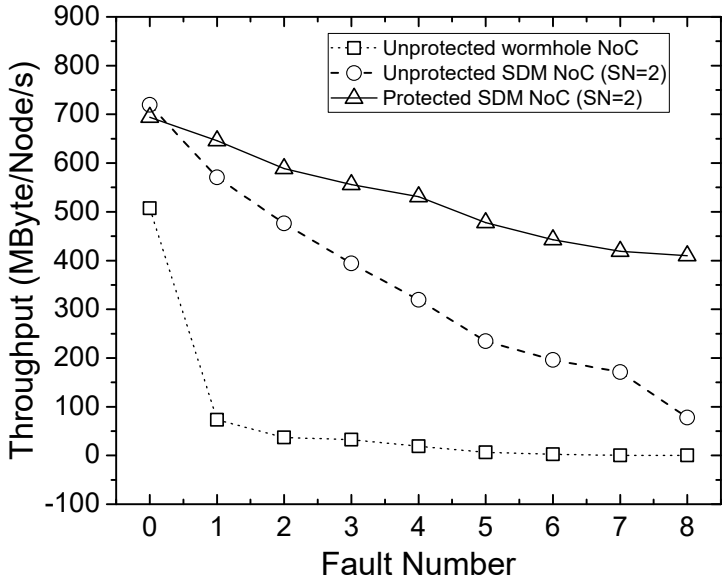
(a) 5 *ms*



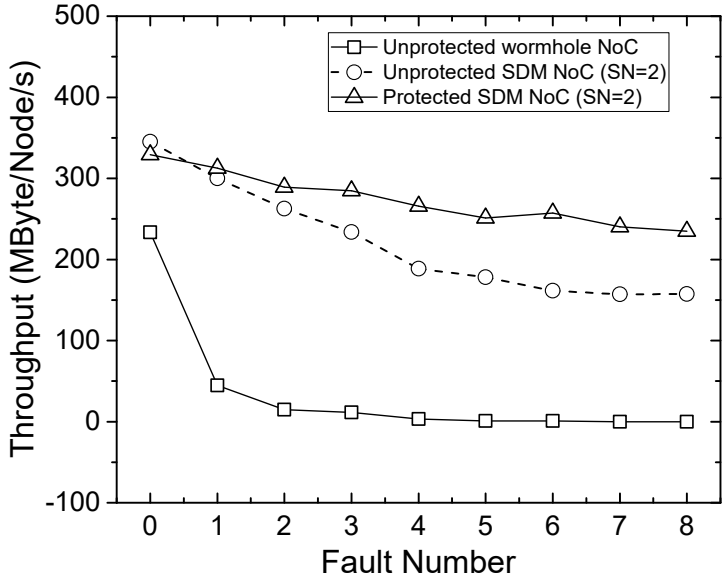
(b) 1 *ms*

Figure 7.13: Throughput of SDM NoCs with multiple faults

When SDM is employed (*SN*=2), the network saturation largely increases compared with the wormhole NoC (∼50% increase), demonstrating the performance advantage of SDM NoCs [LMV$^+$08, SE11b]. This research concerns about the fault-tolerance capability. The SDM network is robust since each inter-router link has two physically independent sub-links. One link fault will not isolate a whole link. If all inserted faults happen on one sub-link and stall a short packet route, the network can work properly with a reasonable throughput. However, as the number of faults increases, one whole link becomes more likely to be broken. The network throughput



(a) 4×4 mesh



(b) 8×8 mesh

Figure 7.14: Throughputs of 32-bit NoCs with an increasing number of faults

drops significantly. The averaged throughput in an unprotected SDM NoC shows a generally linearly dropping trend with the fault count. It should be noted that, without any fault-tolerance, one fault could stall a long packet route (Figure 7.11), significantly reducing the network performance. If multiple faults happen, PEs or routers can be removed from the network. Even losing one node in the network can be fatal which may paralyse the whole network. Thus, it is important to isolate the faulty component from the network and make the healthy network resources continue to work.

Using the fault (or deadlock) detection and recovery techniques proposed in this research, the protect SDM NoC shows a robust feature against permanently faulty links. One fault could remove only one sub-link from the network, which could reduce the network performance slightly but the network function behaves properly. It can be found from Figure 7.14 that though the saturation throughput of protected SDM NoCs is less than the unprotected ones in a fault-free case due to the redundant circuits for fault-tolerance, when one fault is inserted to the network, the protected SDM NoCs achieve a throughput which is, on average, 13.2% (4×4) and 4.2% (8×8) higher than unprotected ones, demonstrating the high robustness obtained from the proposed techniques. In the case that 8 faults are inserted, the throughput of the protected SDM NoCs are $\sim$400% (4×4) and $\sim$50% (8×8) higher than the unprotected ones. Comparably, the wormhole NoC is completely stalled. Compared with the saturation throughput in a fault-free environment, the final throughput of protected SDM NoC decreases 45.0% (4×4) and 28.7% (8×8), while for the unprotected SDM NoCs, the throughput decreases 89.2% (4×4) and 54.4% (8×8). Thus, using the proposed fault detection and recovery techniques, the robustness of the network improves significantly.

7.4 Summary

This chapter demonstrated detailed experimental results and made comparisons between different network implementations, to evaluate the hardware and performance overhead brought by the deadlock detection and recovery circuits proposed in this research. The achieved fault-tolerance capability was systematically analysed. It can be summarised that, the overhead caused by the management of the fault-caused physical-layer deadlocks is acceptable and reasonable for systems requiring a high fault-tolerance. Under the statement that only one type of faults (transient or permanent; an intermittent fault is taken as a permanent one as long as it causes a deadlock which can be detected) could cause the physical-layer deadlock in a QDI NoC, using the proposed deadlock detection and recovery techniques, the router area increases less than 5% and the network saturation throughput decreases less than 4%. The average energy increases around 2%. When both transient and permanent faults are considered to be able to deadlock a QDI NoC in a severe fault environment, fault-diagnosis circuits and recovery circuits for transient faults are added, but the area overhead is only around 8.5% compared with unprotected NoCs, while the throughput decreases less than 15%.

It was demonstrated that one fault that breaks the handshake protocol and causes a physical-layer deadlock can easily paralyse a QDI NoC without any fault-tolerance. The physical-layer deadlock cannot be recovered using upper-layer fault-tolerant techniques (Chapter 3). Thus the proposed fault detection and recovery techniques targeting the physical-layer deadlock are significantly important to the practical use of QDI NoCs. The built fault environment can inject random faults to the network, demonstrating a 100% detection accuracy under a 1-bit fault precondition. When multiple faults are injected into the network, the unprotected wormhole NoCs fail immediately while the unprotected SDM NoCs could sustain for some time, but also face a significant function loss. Adding detection and recovery circuits, the protected SDM NoCs could work properly even when multiple permanent faults are injected, showing a strong robustness. To further improve the reliability of the QDI NoC, fault-tolerant codes can be applied to reduce data errors. The proposed Delay-Insensitive Redundant Coding (DIRC) scheme is a competitive alternative due to its flexibility (Chapter 4), which is left as the future work.

Chapter 8

Conclusions and future work

8.1 Summary of the thesis

The advancing semiconductor technology makes it possible to integrate more and more processing cores on a single chip to achieve continuously increasing chip performance, posing a growing demand for scalable and efficient interconnection. Networks-on-Chip (NoCs) have emerged as a promising candidate to support large-scale on-chip communication. Most existing NoCs are built synchronously, which could be restricted by issues induced by the growing clock distribution as the network scales (Chapter 1). As an alternative, event-driven asynchronous circuits which are controlled by handshake protocols rather than global clocks, can be employed to implement NoCs. Removing the clock, asynchronous NoCs have many attractive advantages over synchronous ones.

In the deep sub-micron era, reliability has become a challenge faced by the scaling electronics. Accompanied with the shrinking device dimensions, factors like the lowering voltage supply, the increasing clock frequency and the growing density of chips, have a negative impact on the chip reliability. Electronic systems are more susceptible to *faults*. *Fault-tolerance* has become an essential objective for critical digital systems, especially in specialized fields such as aerospace, military and medical equipment.

Fault tolerance has been systematically studied in traditional synchronous NoCs, but rarely in asynchronous ones. Using one timing-robust class of asynchronous circuits – the Quasi-Delay-Insensitive (QDI) circuits – to implement the NoC, QDI NoCs can naturally tolerate delay variation, which is attractive for large-scale NoCs. Faults have more complicated and devastated impact on QDI NoCs compared with synchronous NoCs, which is a challenging issue needed to be resolved.

This research studied the impact of different faults on QDI NoCs and presented thorough fault-tolerant solutions at the circuit level to protect them. This is the first systematic research on the deadlock effect in QDI NoCs caused by different faults. The 4-phase 1-of-n synchronous protocol is employed in all QDI implementations in this thesis. This protocol is extensively used in the state-of-the-art asynchronous NoCs, thanks to its simple and comparatively efficient implementation. The trade-off of achievable fault-tolerance and the incurred hardware overhead has been carefully evaluated. In addition, since the studied fault effects and fault-tolerant techniques are specifically for asynchronous circuits, instead of comparing with synchronous baselines (which has been explored [SE11b]), the protected asynchronous pipelines or NoCs were compared with unprotected ones in this research to evaluate proposed fault-tolerant techniques. This research targets on modelling the fault effects on QDI NoCs and providing a general and thorough fault-tolerant architecture. Optimization of the network performance and relevant techniques are outside the scope of this thesis.

8.1.1 Analysis of fault impact on QDI pipelines

The end-to-end path delivering a packet in a QDI NoC can be abstracted as a QDI pipeline. Without the clock as a timing reference, a fault can cause complicated faulty results in QDI pipelines, which behave quite differently from synchronous ones. QDI pipelines were modelled to analyse the impact of different fault scenarios. This analysis is used as the basis of all fault-tolerant techniques proposed in this research. The model was presented in Chapter 3.

Data transmitted in QDI pipelines are encoded as Delay-Insensitive (DI) symbols. It was found that faults on QDI pipelines not only cause data errors, including symbol insertion, deletion and corruption, but also break the handshake protocol, leading to physical-layer deadlock. It is well-known that a permanent fault can deadlock the QDI pipeline. This research demonstrated that a transient fault can also deadlock a QDI pipeline, which has been ignored by asynchronous circuit researchers. This research analysed the deadlock pattern abstracted from all possible fault scenarios, and based on that knowledge, to detect faults, diagnose fault types and recover the network.

8.1.2 Delay-Insensitive Redundant Check codes

The large number of long wires used in NoC links are subjected to significant delay variations and transient faults due to various noises. Implemented in a QDI fashion,

the QDI interconnects are timing-robust, but still vulnerable to faults. This research proposed a Delay-Insensitive Redundant Check (DIRC) coding scheme (Chapter 4) to protect QDI link wires from transient faults.

DIRC coding significantly improves the tolerance of 4-phase 1-of-n QDI interconnects to transient faults. It adds a 1-of-n *check* word to each group of 1-of-n code words, making a DIRC code, which tolerates all 1-bit and some multi-bit transient faults. The check generation and error correction processes follow the simple-to-implement arithmetic rules of 1-of-n codes. Since the DIRC code is systematic, it can easily be adopted in existing 1-of-n QDI pipelines for flexible fault-tolerance. DIRC pipelines using different construction patterns can provide different fault-tolerance for interconnection with a moderate hardware overhead, making DIRC especially attractive to large-scale, communication-centred fabrics such as NoCs and buses. The Redundant Protection of Acknowledge wires (RPA) was also proposed to protect acknowledge wires similarly, with little hardware overhead. Experimental results show that DIRC pipelines achieve thousands-fold improvement on the fault-tolerance capability even in a severe fault environment. The hardware overhead of DIRC pipelines is moderate and can be further reduced using different construction patterns. In most cases, the DIRC pipeline is less than 1.5$\times$ slower than the basic unprotected one, while the fault-tolerance is measured to be more than 1,000$\times$ stronger.

8.1.3 Detection of fault-caused physical-layer deadlocks

According to the knowledge gained in Chapter 3, a fault can break the handshake protocol and result in a deadlock in QDI NoCs. This physical-layer deadlock is different from the one in the network layer caused by cyclic dependence. It cannot be resolved using traditional deadlock management techniques for synchronous NoCs. As a result, all pipeline stages that are downstream to the fault have the same *ack* while the *ack* signals between upstream stages are alternately valued. This is the key to detect a fault-caused physical-layer deadlock in Chapter 5.

In the presence of faults, the timing-robust QDI circuit never knows if an unreceived signal is caused by faults or just delayed. It is the adaptability of a QDI circuit to timing variation which makes it more vulnerable to this kind of deadlock-type fault. Thus, it is unavoidable to bring a timing reference to differentiate the fault and delay. A pair of deadlock detection circuits based on a new time-out mechanism are attached to the ends of the protected region to monitor the asynchronous data path. By detecting activities of several critical asynchronous signals, the detection circuits

are able to tell if the fault occurs in the intermediate region, and the faulty component is found. This method can detect the physical-layer deadlocks caused by both transient and permanent faults. If the two faults are considered at the same time, a fault diagnosis operation is required to tell the fault type. Since this detection needs to use synchronous circuits to sample the asynchronous network, synchronizers are required at the interface to reduce the metastability issue, which can be further alleviated by using a long time-out period. Experimental results show that, using detection circuits to protect different components on the network data path, the router area increases less than 5% and the network saturation throughput decreases less than 4% compared with the unprotected one. The average energy increases by around 2%. This makes the proposed mechanism very attractive due to the small overhead.

8.1.4 Recovery of deadlocked network from faulty links

A fine-grained recovery strategy was presented in Chapter 6 to eliminate the fault-caused physical-layer deadlock and isolate the faulty link, so as to recover the network.

A link fault in a QDI NoC can deadlock a long reserved packet path: only one link is faulty while the other network resources are healthy but deadlocked. Without resetting the network, the fault-free flits remaining in the section upstream of the fault can be dropped from a "sink" at the faulty link, so that the upstream fault-free deadlocked network resources are released and can be reused, which is the proposed *Drain* operation. The section downstream of the fault can be released by creating a tail flit and letting it go through the reserved path to the destination, which is the *Release* operation. All fault-free network resources on the deadlocked packet path will be released by the *Drain&Release* operation.

To isolate the faulty wire, Spatial Division Multiplexing (SDM) is employed where each physical link is divided into several independent sub-links, which can transmit different packets in parallel. Thus a fault breaks only one sub-link. By configuring the switch allocator of the pre-fault router, the defective sub-link can be isolated from the network. The following traffic will go through the faulty link using the other healthy sub-links and the network function is recovered with a graceful performance degradation. For the deadlock caused by transient faults, the isolated sub-link will be resumed after the recovery. Upper network-layer techniques, such as fault-tolerant routings, can be further used to reduce the performance loss and balance the traffic over the network. Using retransmission or other fault-tolerant techniques, the lost packet can be routed to the destination or recovered, which is outside the scope of this thesis.

8.1.5 Overall remarks

This research proposed multiple fault-tolerant techniques to protect QDI NoCs from different faults.

- The fault models provided a systematic analysis of the impact that faults exert on QDI pipelines. These models particularly studied and classified the characteristics of physical-layer deadlocks caused by different faults, which provides a foundation of the following fault-tolerant designs that detect, diagnose and recover from faults.

- Link protection is important not only for NoCs but all communication fabrics that utilise long wires. The proposed DIRC encoding scheme provided a flexible protection to QDI links. The level of protection can be tuned and scaled according to the practical fault-tolerance demands in order to reduce unnecessary hardware overhead. DIRC adds fault-tolerance to the timing-robust QDI links in a non-intrusive way that most existing QDI designs can easily adopt it.

- It is well-known that a permanent fault can cause a physical-layer deadlock. It was revealed in this research that even a transient fault can disrupt the handshake process and deadlock the QDI NoC as well. Keeping the QDI NoC and the whole system working in the presence of different faults is overly important in some critical systems even with some performance overhead. However, this research of fault-caused deadlocks in asynchronous circuits was barely studied. The proposed deadlock management techniques provide a systematic strategy to locate the fault, diagnose the fault type and then recover the NoC by isolating the faulty link to later traffic. For deadlocks caused by transient faults, the isolated links can be reused after the network recovery. This is the first thorough research on the fault-caused deadlock in QDI NoCs to the best knowledge of the author.

8.2 Lessons learnt

NoCs are promising fabrics to provide scalable and efficient on-chip communication for large-scale many-core systems. They can be implemented using synchronous or asynchronous circuits. Most existing NoCs are built synchronously with a clock distribution network. Faults on synchronous NoCs typically cause data errors, which can be detected and corrected using conventional well-studied, fault-tolerant techniques.

Asynchronous NoCs have many potential advantages over the synchronous ones due to their clockless feature, which are promising for the future large-scale multi-core systems. Faults occurring on asynchronous NoCs, however, can not only corrupt data, but disrupt the handshake process and cause physical-layer deadlocks. In a wormhole network, this deadlock may break or stall a transmitting packet. The broken or stalled packet may straddle multiple routers, which will cause more packet stalls. As a result, the deadlock spreads over the network and the network function is paralysed. Due to the broken handshake protocol, this fault-caused, physical-layer deadlock is different from those due to the cyclic dependence of packet transmissions in the network layer, and cannot be handled using conventional deadlock management techniques. Resolving this deadlock in the physical-layer is mandatory to restore the network operation.

For the first time the fault-caused physical-layer deadlock was systematically studied in this research. It demonstrated that both transient and permanent faults can cause physical-layer deadlocks, and produce different syndromes or patterns which help to differentiate physical-layer deadlocks from network-layer ones or temporary network congestion. Thus the faulty component can be detected and located precisely. Meanwhile, these deadlock patterns often determine the fault type so that fault diagnosis circuits were implemented to tell which fault (transient or permanent) deadlocks the packet path. After the deadlock detection and diagnosis, the deadlocked network needs to be recovered.

A packet may be stalled or broken by a fault during its transmission across the asynchronous network. As a result, the reserved packet path is jammed with fault-free (upstream of the fault) and polluted (downstream of the fault) flits when the deadlock is detected. To restore the network operation, a *Drain&Release* mechanism was demonstrated which drains remaining flits in the deadlocked packet path and releases healthy network resources. Depending on the fault type, the faulty component (inter-router link in this research) should be either permanently blocked (for permanent faults) or reused (for transient faults). Spatial Division Multiplexing (SDM) provides a natural way to restore the network function from a permanently faulty link wire. By locking the faulty sub-link belonging to one physical link, the other healthy sub-links can still be used to transmit packets in the same direction. The deadlock network is recovered.

These proposed deadlock management techniques are essential to maintain the function of asynchronous NoCs in the presence of faults. Keeping the network running is essential even with some graceful performance degradation. It was demonstrated

that, applying these fault-tolerant techniques to different baseline QDI NoCs may incur around 1.6%∼8.5% area overhead and 2%∼14% speed overhead. It is believed that, considering the largely enhanced fault-tolerance capability achieved by resolving the harmful deadlock issue, these accompanied penalties are moderate and acceptable in practical asynchronous NoC designs. The proposed fault-tolerant techniques are feasible to protect asynchronous NoCs and provide a realistic fault-tolerance solution for future large-scale many-core SoCs.

As a complementary measure to enhance the reliability of asynchronous NoCs, the proposed systematic DIRC coding scheme can be flexibly employed to protect the large number of long link wires in QDI NoCs from transient faults, reducing the data error rate. The protected QDI interconnects achieve both timing-robustness and fault-tolerance. The incurred overhead varies depending on the construction manner. It was demonstrated that, the area of a complete DIRC pipeline stage can be around four times larger than the unprotected one, while the speed overhead can be less than 50%. If the area of link wires is considered and these wires consume significantly large area, the area overhead will approach to $(1/CN)$, which is 50% when CN=2 (CN denotes the number of 1-of-n data words in a DIRC code). The area-consuming 1-of-n adders can be designed in the transistor level to reduce the hardware overhead. A trade-off can be made between the achieved fault-tolerance capability and the hardware overhead according to the practical design requirement, which has been discussed in Chapter 4. Protecting links in only critical regions can further reduce the hardware cost.

It can be inferred from this research that, compared with synchronous NoCs where faults typically cause data errors, asynchronous or QDI NoCs are more vulnerable to faults since the possibly fault-caused physical-layer deadlocks can easily paralyse the network function. Besides permanent faults which can stall the handshake process, even a transient fault can deadlock the network, which is a new challenging topic deeply studied in this research. The occurrence of this kind of deadlock is indirectly due to the adaptability of asynchronous circuits to timing variations, though this timing-robustness is an attractive feature that synchronous circuits do not have. Therefore, it has to be admitted that, *from this deadlock point of view, without any fault-tolerant techniques, asynchronous NoCs are less reliable than conventional synchronous ones*.

Besides the fault-tolerance challenge studied in this research, it should also be kept in mind that, due to the additional circuits controlling the handshake process, asynchronous circuits often have drawbacks in area and speed in practical designs. As the

leakage currents become a major contributor to the power consumption in nanoscale technologies, controlling the circuit area below a specific level is necessary for low-power designs [HB13]. The hardware cost incurred by the fault-tolerance should also be balanced. Meanwhile, the lack of mature Electronic Design Automation (EDA) tools and design methodology makes it difficult to implement high-performance large-scale asynchronous circuits effectively, deterring the wide use of asynchronous designs in commercial fields [SF01]. However, as has been widely accepted by asynchronous circuit designers, asynchronous NoCs are promising for the future large-scale many-core Systems-on-Chip (SoCs) where distributing the global clock and the chip-level timing closure becomes increasingly difficult. The resulting Globally Asynchronous, Locally Synchronous (GALS) system achieves promising advantages over the synchronous counterparts, including the simplified chip-level timing closure, the decreased clock distribution network, the good support for modularity and design reuse, and the independent frequency/voltage control in different synchronous domains. The static power consumption can be reduced by adding power gating circuits [OTM10] or using leakage control techniques [HB13] while the performance can be improved through carefully designing the asynchronous pipeline structure [CBL$^+$10]. These all drive the research and usage of asynchronous circuits in implementing the future large-scale NoCs.

This research proposed a holistic, efficient, resilient and cost effective solution to resolve the deadlock effect caused by faults, providing an opportunity for asynchronous NoCs to cater for the future on-chip interconnections requiring high reliability. The proposed fault-tolerant techniques can maintain the function of asynchronous NoCs in the presence of different faults, which is significant for critical electronic devices since keeping them running is essential even with some performance degradation. In addition, though this research focuses on studying the fault-tolerance of asynchronous circuits or NoCs (which is a relatively new topic that has not been thoroughly studied) rather than exploring how to optimize asynchronous designs and improve the circuit performance, the trade-off between the achieved fault-tolerance capability and the incurred hardware cost is necessary and has been explored. Taking the proposed DIRC coding scheme protecting QDI links for example, the systematic feature of the DIRC codes makes it possible to place DIRC pipeline stages arbitrarily in an unprotected QDI pipeline, thus some certain pipeline segments can be protected. According to the practical fault-tolerance requirement, the DIRC pipeline using different construction patterns (Figure 4.12) can provide required fault-tolerance with a moderate

and reasonable hardware overhead. More evaluation of the hardware cost of DIRC has been explored in Appendix B. For the proposed deadlock management techniques, redundant circuits can be placed at a specific pipeline segment to monitor its activities, so as to detect the fault-caused deadlock and recover the network function (Section 3.5). This general strategy allows designers to flexibly apply the deadlock management circuits to protect asynchronous NoC or general asynchronous pipelines from the fatal deadlock, while the hardware cost is under control.

Fault-tolerance is a concept over a whole system, which needs protection from multiple different layers to ensure all system functions are correct under a specific fault environment. The redundancy required by the fault-tolerance brings performance and hardware overhead. Thus it is valuable to be aware of practical fault-tolerant requirement, and from a system view to achieve required fault-tolerance from different network layers with cost-effective designs. As an example, to protect QDI NoCs, the proposed techniques managing the fault-caused physical-layer deadlocks in this research are mandatory in the bottom circuit level to keep the network functioning. It is impossible to protect all components in a network. The relatively expensive point-to-point protection (DIRC) can be applied to some critical network region or components depending on the practical designs to filter a large portion of transient faults on-the-fly. These methods mainly protect the network from the bottom physical or data link layer (Section 2.2.2). At higher layers, the same techniques employed in synchronous NoCs can still be used, such as resending corrupted packets, fault-tolerant routings, end-to-end protection [IY11, CH01, Pon12], etc. to provide protection.

Although the proposed fault-tolerant techniques are based on 4-phase, 1-of-n asynchronous protocols in this research, they provide a basic clue to resolve the destructive deadlock effect caused by different faults and can be generalized to provide protection for other types of QDI NoCs.

8.3 Future work

This research explored the fault-tolerance of asynchronous NoCs, proposed a systematic protection strategy against different faults, which can deal with many intractable faulty scenarios. However, there is still lots of work to do to provide full protection. Several topics are left as future work:

- *Protection of the router control logic*

  The proposed fault-tolerant techniques for QDI NoCs in this research concentrate on protecting the network data path, including the input/output buffers of the router, crossbar and inter-router link wires in a generic NoC. The data path carrying packets is assumed to be more susceptible to faults than the router control logic (the control logic mainly deals with the head and tail of a flit, and is responsible for the packet path reservation and cancellation). Currently the router control logic is not protected. Faults on the control logic can also cause severe results. In the presence of permanent faults which can fail the circuit function, a spare is needed to replace the faulty control component with the support from upper layer techniques (such as fault-tolerant routings [IY11]). Compared to permanent faults, transient faults are relatively random and they may result in wrong control signals, or even deadlocked control components [Pon12]. Duplication-based fault-tolerant techniques [JM05, Pon12] could be used to protect control logic from transient faults.

- *Protection of added redundant circuits for fault-tolerance*

  Currently the redundant circuits added to protect the network are unprotected. Given most runtime permanent faults are strongly related to the workload of the device [BM11, AFK+13], it is reasonable to assume that these circuits have a long life-time, especially when a long time-out is used. Techniques such as scan chains [TTD+09] can be used to monitor these redundant circuits periodically to ensure they are working properly. For transient faults, "conventional" fault-tolerant techniques can protect these circuits. The main questions are if these circuits really need to be protected and what is the required level of the protection. Note that protecting these circuits would cause further redundancy. Therefore, a simple protection which can detect a "dead" circuit in the presence of permanent faults is a proper approach, which also ensures transient faults on these circuits do not stall the network.

- *Multi-layer protection and systematic fault-tolerance evaluation*

  The proposed network recovery techniques utilise SDM in the physical layer to isolate the faulty component and recover the network function. They do not rely on any specific routings. Upper-layer recovery techniques, such as fault-tolerant routings [IY11], can be further used to release the traffic pressure of the damaged link, avoiding hot spots in the network and compensating for the

performance loss. If a router is found permanently faulty, it can also be detoured using fault-tolerant routings. Besides, the proposed fault detection and recovery techniques concentrate on recovering the deadlocked NoC to work. They do not correct faults, so that packet loss could happen. Using retransmission or other fault-tolerant techniques [Sor09], the lost packet can be routed to the destination or recovered.

In addition, in a NoC structured system, some blocks or regions may have different fault-tolerance requirement. Therefore, the critical region can use an enhanced protection compared with other sections. The proposed fault-tolerant techniques can be flexibly configured to provide different fault-tolerance, which will be studied in the future.

- *Other designs*

  The proposed detection technique uses synchronous circuits to sample the asynchronous network, requiring synchronizers at the interface to reduce the metastability. Synchronization [Gin03, MM07, NS15] is a classical issue faced by circuit designers, and a long-lasting popular topic for the asynchronous community. Though it is out of the scope of this research, it is important to study the possible metastability risk and ensure a safe sampling operation.

  Fault-diagnosis is required when co-managing transient and permanent faults. The implemented diagnosis circuits can determine the type of faults on the forward data path (including the large number of data wires and asynchronous latches) of a QDI pipeline. The backward *ack* path, including the completion detector and a single *ack* wire can be protected using duplication-based techniques, preventing transient faults on the backward path from deadlocking the pipeline. Thus, on the backward path, only permanent faults can cause the physical-layer deadlock.

Bibliography

[AAA87] S. A. Al-Arian and D. P. Agrawal. Physical failures and fault models of CMOS circuits. *IEEE Transactions on Circuits and Systems*, 34(3):269–279, 1987.

[ACG+03] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. SPIN: a scalable, packet switched, on-chip micro-network. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 70–73, 2003.

[ADMX+11] R. Al-Dujaily, T. Mak, F. Xia, A. Yakovlev, and M. Palesi. Run-time deadlock detection in Networks-on-Chip using coupled transitive closure networks. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, Mar. 2011.

[AFEH12] M. A. Al Faruque, T. Ebi, and J. Henkel. AdNoC: Runtime adaptive Network-on-Chip architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(2):257–269, Feb. 2012.

[AFK+13] R. Aitken, G. Fey, Z. T. Kalbarczyk, F. Reichenbach, and M. Sonza Reorda. Reliability analysis reloaded: How will we survive? In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 358–367, 2013.

[AIKR13] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security Privacy*, 11(4):14–26, Jul. 2013.

[AMDI15] AMD Advanced Micro Devices Inc. AMD Product Resource Center. `http://products.amd.com/`, 2015. Online; accessed: 23/11/2015.

[AMM+15] J. L. Autran, D. Munteanu, S. Moindjie, T. Saad Saoud, S. Sauze, G. Gasiot, and P. Roche. Astep (20052015): Ten years of soft error and atmospheric radiation characterization on the Plateau de Bure. *Microelectronics Reliability*, 55(910):1506–1511, 2015.

[AN10] M. Y. Agyekum and S. M. Nowick. An error-correcting unordered code and hardware support for robust asynchronous global communication. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 765–770, 2010.

[AN12] M. Y. Agyekum and S. M. Nowick. Error-correcting unordered codes and hardware support for robust asynchronous global communication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(1):75–88, 2012.

[ASLM09] S. Almukhaizim, F. Shi, E. Love, and Y Makris. Soft-error tolerance and mitigation in asynchronous burst-mode circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(7):869–882, 2009.

[Bai00] W. J. Bainbridge. *Asynchronous system-on-chip interconnect*. PhD thesis, University of Manchester, 2000.

[Bau01] R. C. Baumann. Soft errors in advanced semiconductor devices-part I: the three radiation sources. *IEEE Transactions on Device and Materials Reliability*, 1(1):17–22, 2001.

[Bau05] R. C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

[BB92] M. Blaum and J. Bruck. Unordered error-correcting codes and their applications. In *Proceedings of Twenty-Second International Symposium on Fault-Tolerant Computing (FTCS)*, pages 486–493, 1992.

[BBF+01] S. Behling, R. Bell, P. Farrell, H. Holthoff, F. O'Connell, and W. Weir. The POWER4 processor introduction and tuning guide. *IBM redbooks*, 2001.

[BCMV08] E. Beigne, F. Clermidy, S. Miermont, and P. Vivet. Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC.

In *Proceedings of ACM/IEEE International Symposium On Networks-on-Chip (NOCS)*, pages 129–138, Apr. 2008.

[BDF+14] J. F. Bulzacchelli, T. O. Dickson, F. D. Ferraiolo, R. J. Reese, and M. B. Spear. Calibration of multiple parallel data communications lines for high skew conditions, Mar. 2014. US Patent 8,681,839.

[BDM09] G. Blake, R. G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, Nov. 2009.

[BF02] W. J. Bainbridge and S. B. Furber. Chain: a delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, 2002.

[BHS+95] R. Baumann, T. Hossain, E. Smith, S. Murata, and H. Kitagawa. Boron as a primary source of radiation in high density DRAMs. In *Proceedings of 1995 Symposium on VLSI Technology*, pages 81–82, 1995.

[BKY00] A. Bystrov, D. J. Kinniment, and A. Yakovlev. Priority arbiters. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 128–137, 2000.

[BM06] T. Bjerregaard and S. Mahadevan. A survey of research and practices of Network-on-Chip. *ACM Computing Surveys (CSUR)*, 38(1), Jun. 2006.

[BM11] M. Bohr and K. Mistry. Intel's Revolutionary 22 nm Transistor Technology. *Rob Willoner at Innovation*, 2011.

[BN98] P. D. Bradley and E. Normand. Single event upsets in implantable cardioverter defibrillators. *IEEE Transactions on Nuclear Science*, 45(6):2929–2940, 1998.

[Bor05] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, Nov. 2005.

[Bos91] B. Bose. On unordered codes. *IEEE Transactions on Computers*, 40(2):125–131, 1991.

[Bot15] W. R. Bottoms. A roadmap for heterogeneous integration in electronics. `http://www.itrs2.net/itrs-reports.html`, May 2015. Online; accessed: 28/01/2016.

[BS05] T. Bjerregaard and J. Sparsø. A router architecture for connection-oriented service guarantees in the MANGO clockless Network-on-Chip. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, volume 2, pages 1226–1231, 2005.

[BS06] T. Bjerregaard and J. Sparsø. Implementation of guaranteed services in the MANGO clockless Network-on-Chip. *IEE Proceedings Computers and Digital Techniques*, 153(4):217–229, 2006.

[BS09] W. J. Bainbridge and S. J. Salisbury. Glitch sensitivity and defense of quasi delay-insensitive Network-on-Chip links. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 35–44, May 2009.

[BSH75] D. Binder, E. C. Smith, and A. B. Holman. Satellite anomalies from galactic cosmic rays. *IEEE Transactions on Nuclear Science*, 22(6):2675–2680, 1975.

[BSK+10] R. P. Bastos, G. Sicard, F. Kastensmidt, M. Renaudin, and R. Reis. Evaluating transient-fault effects on traditional C-element's implementations. In *Proceedings of IEEE International On-Line Testing Symposium (IOLTS)*, pages 35–40, 2010.

[BSN+15] B. Bowhill, B. Stackhouse, N. Nassif, Zibing Yang, A. Raghavan, and et al. The Xeon ®processor E5-2600 v3: A 22nm 18-core product family. In *Proceedings of IEEE International Solid - State Circuits Conference (ISSCC)*, pages 1–3, Feb. 2015.

[BSO05] F. A. Bower, D. J. Sorin, and S. Ozev. A mechanism for online diagnosis of hard faults in microprocessors. In *Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 197–208, 2005.

[BTEF03] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber. Delay-insensitive, point-to-point interconnect using m-of-n codes. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 132–140, 2003.

[BW04] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proceedings of 37th International Symposium on Microarchitecture (MICRO)*, pages 319–330, Dec. 2004.

[BY92] R. V. Balakrishnan and D. W. L. Young. System for reducing skew in the parallel transmission of multi-bit data slices, Mar. 1992. US Patent 5,101,347.

[CBL+10] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, and N. Wehn. A 477mW NoC-based digital baseband for MIMO 4G SDR. In *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, pages 278–279, 2010.

[CC15] N. Chatterjee and S. Chattopadhyay. Fault tolerant mesh based Network-on-Chip architecture. In *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, pages 417–420, May 2015.

[CH01] F.-C. Cheng and S.-L. Ho. Efficient systematic error-correcting codes for semi-delay-insensitive data transmission. In *Proceedings of IEEE International Conference on Computer Design (ICCD)*, pages 24–29, 2001.

[Con03] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19, 2003.

[Dal90] W. J. Dally. Virtual-channel flow control. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 60–68, 1990.

[DGK09] R. Dobkin, R. Ginosar, and A. Kolodny. QNoC asynchronous router. *Integration, the VLSI Journal*, 42(2):103–115, 2009.

[DGY95] I. David, R. Ginosar, and M. Yoeli. Self-timed is self-checking. *Journal of Electronic Testing*, 6(2):219–228, 1995.

[DMB06] G. De Micheli and L. Benini. *Networks on chips: technology and tools*. Morgan Kaufmann, 2006.

[Don84] Hao Dong. Modified berger codes for detection of unidirectional errors. *IEEE Transactions on Computers*, C-33(6):572–575, Jun. 1984.

[DS87] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, 1987.

[DT01] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of Design Automation Conference (DAC)*, pages 684–689. IEEE, 2001.

[DT03] W. J. Dally and B. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann Publishers, San Francisco, 2003.

[DWD91] M. E. Dean, T. E. Williams, and D. L. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). In *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*, pages 55–70, 1991.

[EJP09] N. Enright Jerger and L.-S. Peh. *On-chip Networks*. Morgan & Claypool Publishers, 2009.

[EKD+03] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, and et al. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceedings of 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 7–18, Dec. 2003.

[FF04] T. Felicijan and S. B. Furber. An asynchronous on-chip network router with Quality-of-Service (QoS) support. In *Proceedings of IEEE International SOC Conference (SOCC)*, pages 274–277, 2004.

[FFD+14] E.J. Fluhr, J. Friedrich, D. Dreps, V. Zyuban, G. Still, and et al. POWER8^{TM}: A 12-core server-class processor in 22nm SOI with 7.6Tb/s off-chip bandwidth. In *Proceedings of IEEE International Solid - State Circuits Conference (ISSCC)*, pages 96–97, Feb. 2014.

[FLJ+13] 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*, 21(6):1053–1066, 2013.

[FLP+13] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, Dec. 2013.

[FPD09] W. Friesenbichler, T. Panhofer, and M. Delvai. A comprehensive approach for soft error tolerant four state logic. In *Proceedings of International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 214–217, Apr. 2009.

[FS09] W. Friesenbichler and A. Steininger. Soft error tolerant asynchronous circuits based on dual redundant four state logic. In *Proceedings of Euromicro Conference on Digital System Design (DSD)*, pages 100–107, 2009.

[Gee05] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, May 2005.

[GGC+13] G. Gasiot, M. Glorieux, S. Clerc, D. Soussan, F. Abouzeid, and P. Roche. Experimental soft error rate of several flip-flop designs representative of production chip in 32 nm CMOS technology. *IEEE Transactions on Nuclear Science*, 60(6):4226–4231, Dec. 2013.

[Gin03] R. Ginosar. Fourteen ways to fool your synchronizer. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 89–96, May 2003.

[Gin11] R. Ginosar. Metastability and synchronizers: A tutorial. *IEEE Design & Test of Computers*, 28(5):23–35, 2011.

[GKS+07] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger. On-chip interconnection networks of the TRIPS chip. *IEEE Micro*, 27(5):41–50, Sep. 2007.

[GSX+09] S. Golubcovs, D. Shang, F. Xia, A. Mokhov, and A. Yakovlev. Modular approach to multi-resource arbiter design. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 107–116, 2009.

[GXZ09] H. Gu, J. Xu, and W. Zhang. A low-power fat tree-based optical network-on-chip for multiprocessor system-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 3–8, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[GYB07] K. T. Gardiner, A. Yakovlev, and A. Bystrov. A C-element latch scheme with increased transient fault tolerance for asynchronous circuits. In *Proceedings of IEEE International On-Line Testing Symposium (IOLTS)*, pages 223–230, Jul. 2007.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[HB13] J. Hamon and E. Beigne. Automatic leakage control for wide range performance QDI asynchronous circuits in FD-SOI technology. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 142–149, May 2013.

[HBB95] H. Hulgaard, S. M. Burns, and G. Borriello. Testing asynchronous circuits: a survey. *Integration, the VLSI Journal*, 19(3):111 – 131, 1995.

[HBC+13] J. Hart, S. Butler, H. Cho, Y. Ge, G. Gruber, and et al. 3.6GHz 16-core SPARC SoC processor in 28nm. In *Proceedings of IEEE International Solid - State Circuits Conference (ISSCC)*, pages 48–49, Feb. 2013.

[HKM+03] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar. Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-um to 90-nm generation. In *Proceedings of IEEE International Electron Devices Meeting (IEDM)*, pages 21.5.1–21.5.4, Dec. 2003.

[HR04] P. D. Hyde and G. Russell. A comparative study of the design of synchronous and asynchronous self-checking RISC processors. In *Proceedings of IEEE International On-Line Testing Symposium (IOLTS)*, pages 89–94, Jul. 2004.

[HS12] J. Hussein and G. Swift. Xilinx WP395 (v1.0) Mitigating Single-Event Upsets, White Paper. http://www.xilinx.com/support/documentation/white_papers/wp395-Mitigating-SEUs.pdf, May 2012. Online; accessed 16/01/2016.

[HVS+07] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz mesh interconnect for a Teraflops processor. *IEEE Micro*, 27(5):51–61, Sep. 2007.

[IEE12] IEEE. IEEE Standard for Standard SystemC Language Reference Manual. *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pages 1–638, Jan. 2012.

[Int09] Intel. An Introduction to the Intel© QuickPath Interconnect. http://www.intel.com/, Jan. 2009. Online; accessed 23/11/2015.

[Int13] Intel. Intel Xeon ® Phi™ Coprocessor x100 Product Family: Specification Update. http://www.intel.com/, May 2013. Online; accessed: 10/01/2016.

[Int15] Intel. Intel© Product Specifications. http://ark.intel.com/, 2015. Online; accessed 23/11/2015.

[ITR11] ITRS. International technology roadmap for semiconductors 2011 edition: Table DESN4 logical/circuit/physical design technology requirements. http://www.itrs2.net/2013-itrs.html, 2011. Online; accessed: 10/01/2016.

[ITY+10] E. Ibe, H. Taniguchi, Y. Yahagi, K. i. Shimbo, and T. Toba. Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. *IEEE Transactions on Electron Devices*, 57(7):1527–1538, Jul. 2010.

[IY11] M. Imai and T. Yoneda. Improving dependability and performance of fully asynchronous on-chip networks. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 65–76, 2011.

[Jan08] W. Jang. *Soft-error tolerant quasi delay-insensitive circuits*. PhD thesis, California Institute of Technology, 2008.

[Jha89] N. K. Jha. Separable codes for detecting unidirectional errors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(5):571–574, 1989.

[JM05] W. Jang and A. J. Martin. SEU-tolerant QDI circuits. In *Proceedings of 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 156–165, 2005.

[JY96] M. B. Josephs and J. T. Yantchev. CMOS design of the tree arbiter element. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(4):472–476, Dec. 1996.

[JYB12] N. Julai, A. Yakovlev, and A. Bystrov. Error detection and correction of Single Event Upset (SEU) tolerant latch. In *Proceedings of the IEEE 18th International On-Line Testing Symposium (IOLTS)*, pages 1–6, Washington, DC, USA, 2012. IEEE Computer Society.

[Kes99] R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, Mar. 1999.

[KGGV07] M. Krstic, E. Grass, F. K. Gurkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of Computers*, 24(5):430–441, Sep. 2007.

[KH04] T. Karnik and P. Hazucha. Characterization of soft errors caused by single event upsets in CMOS processes. *IEEE Transactions on Dependable and Secure Computing*, 1(2):128–143, 2004.

[Kin07] D. J. Kinniment. *Synchronization and Arbitration in Digital Systems*. Wiley Online Library, 2007.

[KK98] I. Koren and Z. Koren. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, 1998.

[KM04] S. Krishnamohan and N. R. Mahapatra. A highly-efficient technique for reducing soft errors in static CMOS circuits. In *Proceedings of IEEE International Conference on Computer Design (ICCD)*, pages 126–131, 2004.

[KSDH00] J. L. Knighten, N. W. Smith, II. DiBene, J. T., and L. O. Hoeft. EMI common-mode current dependence on delay skew imbalance in high speed differential transmission lines operating at 1 gigabit/second data rates. In *Proceedings of IEEE First International Symposium on Quality Electronic Design (ISQED)*, pages 309–313, 2000.

[KXIZ07] W. Kuang, E. Xiao, C. M. Ibarra, and P. Zhao. Design asynchronous circuits for soft error tolerance. In *Proceedings of IEEE International Conference on Integrated Circuit Design and Technology (ICICDT)*, pages 1–5, 2007.

[KZT05] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 408–419, Jun. 2005.

[KZYD10] W. Kuang, P. Zhao, J. S. Yuan, and R. F. DeMara. Design of asynchronous circuits for high soft error tolerance in deep submicrometer CMOS circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(3):410–422, Mar. 2010.

[LBTW97] S.-H. Lo, D. A. Buchanan, Y. Taur, and W. I. Wang. Quantum-mechanical modeling of electron tunneling current from the inversion layer of ultra-thin-oxide nMOSFET's. *IEEE Electron Device Letters*, 18(5):209–211, 1997.

[LCYH14] Hong-Ting Lin, Yi-Lin Chuang, Zong-Han Yang, and Tsung-Yi Ho. Pulsed-latch utilization for clock-tree power optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(4):721–733, Apr. 2014.

[Lec14] J. Lechner. *Building Robust GALS Circuits: Fault-Tolerant and Variation-Aware Design Techniques for Reliable Circuit Operation*. PhD thesis, Technische Universität Wien, 2014.

[Lee07] S. Lee. A deadlock detection mechanism for true fully adaptive routing in regular wormhole networks. *Computer Communications*, 30(8):1826–1840, Jun. 2007.

[LHHA14] F. K. Lodhi, S. R. Hasan, O. Hasan, and F. Awwad. Low power soft error tolerant Macro Synchronous Micro Asynchronous (MSMA) pipeline. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 601–606, Jul. 2014.

[LK08] R. K. Lawrence and A. T. Kelly. Single event effect induced multiple-cell upsets in a commercial 90 nm CMOS digital technology. *IEEE Transactions on Nuclear Science*, 55(6):3367–3374, 2008.

[LLP07] T. Lehtonen, P. Liljeberg, and J. Plosila. Online reconfigurable self-timed links for fault tolerant NoC. *VLSI Design*, 2007:1–13, 2007.

[LLP12] J. Lechner, M. Lampacher, and T. Polzer. A robust asynchronous interfacing scheme with four-phase dual-rail coding. In *Proceedings of International Conference on Application of Concurrency to System Design (ACSD)*, pages 122–131, 2012.

[LM04] C. LaFrieda and R. Manohar. Fault detection and isolation techniques for quasi delay-insensitive circuits. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, pages 41–50, 2004.

[LMSP99] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Runtime logic and interconnect fault recovery on diverse FPGA architectures. In *Proceedings of Military and Aerospace Applications of Programmable Devices and Technologies International Conference*, 1999.

[LMV+08] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor. Concepts and implementation of spatial division multiplexing for guaranteed throughput in Networks-on-Chip. *IEEE Transactions on Computers*, 57(9):1182–1195, 2008.

[LPP08] B. Li, L.-S. Peh, and P. Patra. Impact of process and temperature variations on Network-on-Chip design exploration. In *Proceedings of ACM/IEEE International Symposium On Networks-on-Chip (NOCS)*, pages 117–126, 2008.

[LSH15] J. Lechner, A. Steininger, and F. Huemer. Methods for analysing and improving the fault resilience of delay-insensitive codes. In *Proceedings*

of IEEE International Conference on Computer Design (ICCD), pages 519–526, Oct. 2015.

[LWL+10] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu. Self-adaptive system for addressing permanent errors in on-chip interconnects. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(4):527–540, 2010.

[MAMN08] P. B. McGee, M. Y. Agyekum, M. A. Mohamed, and S. M. Nowick. A level-encoded transition signaling protocol for high-throughput asynchronous global communication. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 116–127. IEEE, 2008.

[Mar90] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Proceedings of the sixth MIT conference on Advanced research in VLSI*, pages 263–278, 1990.

[Mar91] A. J. Martin. *Synthesis of asynchronous VLSI circuits*. No. CALTECH-CS-TR-93-28, California Institute of Technology, 1991.

[MB59] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of International Symposium on the Theory of Switching*, volume 29, pages 204–243, 1959.

[McC85] E. J. McCluskey. Built-in self-test techniques. *IEEE Design & Test of Computers*, 2(2):21–28, Apr. 1985.

[McP12] J. W. McPherson. Time dependent dielectric breakdown physics models revisited. *Microelectronics Reliability*, 52(910):1753 – 1760, 2012.

[MG03] M. Mishra and S. C. Goldstein. Defect tolerance at the end of the roadmap. In *Proceedings of International Test Conference (ITC)*, volume 1, pages 1201–1210, 2003.

[MH91] A. J. Martin and P. J. Hazewindus. Testing delay-insensitive circuits. In *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*, pages 118–132. MIT Press, 1991.

[MHH+05] S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala, and S. A. Wender. Predicting the number of fatal soft errors in Los Alamos national laboratory's ASC Q supercomputer. *IEEE Transactions on Device and Materials Reliability*, 5(3):329–335, 2005.

[MJM11] M. Mosaffa, F. Jafari, and S. Mohammadi. Designing robust threshold gates against soft errors. *Microelectronics Journal*, 42(11):1276–1289, 2011.

[MM07] R. Mullins and S. Moore. Demystifying data-driven and pausible clocking schemes. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 175–185, Mar. 2007.

[MM09] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, pages 196–207, New York, NY, USA, 2009. ACM.

[MMC15] R. Maddah, R. Melhem, and S. Cho. Rdis: Tolerating many stuck-at faults in resistive memory. *IEEE Transactions on Computers*, 64(3):847–861, Mar. 2015.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, Apr. 1965.

[MRL04] Y. Monnet, M. Renaudin, and R. Leveugle. Asynchronous circuits sensitivity to fault injection. In *Proceedings of IEEE International On-Line Testing Symposium (IOLTS)*, pages 121–126, Jul. 2004.

[MRL05a] Y. Monnet, M. Renaudin, and R. Leveugle. Asynchronous circuits transient faults sensitivity evaluation. In *Proceedings of Design Automation Conference (DAC)*, pages 863–868, Jun. 2005.

[MRL05b] Y. Monnet, M. Renaudin, and R. Leveugle. Hardening techniques against transient faults for asynchronous circuits. In *Proceedings of IEEE International On-Line Testing Symposium (IOLTS)*, pages 129–134, 2005.

[MRL06] Y. Monnet, M. Renaudin, and R. Leveugle. Designing resistant circuits against malicious faults injection using asynchronous logic. *IEEE Transactions on Computers*, 55(9):1104–1115, 2006.

[MRLD01] J. M. Martínez-Rubio, P. López, and J. Duato. A cost-effective approach to deadlock handling in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(7):716–729, Jul. 2001.

[MT03] K. Mohanram and N. A. Touba. Cost-effective approach for reducing soft error failure rate in logic circuits. In *Proceedings of International Test Conference (ITC)*, volume 1, pages 893–901, 2003.

[Muk08] S. Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers, 2008.

[MVF00] J. Muttersbach, T. Villiger, and W. Fichtner. Practical design of globally-asynchronous locally-synchronous systems. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 52–59, 2000.

[MVK+99] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner. Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems. In *Proceedings of International ASIC/-SOC Conference*, pages 317–321, 1999.

[MW78] T. C. May and M. H. Woods. A new physical mechanism for soft errors in dynamic memories. In *Proceedings of 16th Annual Reliability Physics Symposium*, pages 33–40, 1978.

[MW79] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Transactions on Electron Devices*, 26(1):2–9, 1979.

[Nas00] S. R. Nassif. Modeling and forecasting of manufacturing variations. In *Proceedings of 5th International Workshop on Statistical Metrology*, pages 2–10, 2000.

[NCL+15] J. Noh, V. Correas, S. Lee, J. Jeon, and I. Nofal et al. Study of neutron soft error rate (SER) sensitivity: Investigation of upset mechanisms by

comparative simulation of FinFET and planar MOSFET SRAMs. *IEEE Transactions on Nuclear Science*, 62(4):1642–1649, Aug. 2015.

[NKM10] A. Nieuwoudt, J. Kawa, and Y. Massoud. Crosstalk-induced delay, noise, and interconnect planarization implications of fill metal in nanoscale process technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(3):378–391, Mar. 2010.

[Now96] S. M. Nowick. Design of a low-latency asynchronous adder using speculative completion. *IEE Proceedings, Computers and Digital Techniques*, 143(5):301–307, Sep. 1996.

[NS15] R. Najvirt and A. Steininger. How to synchronize a pausible clock to a reference. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 9–16, May 2015.

[OAHY08] S. Ogg, B. Al-Hashimi, and A. Yakovlev. Asynchronous transient resilient links for NoC. In *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 209–214, 2008.

[OBF+93] J. Olsen, P. E. Becher, P. B. Fynbo, P. Raaby, and J. Schultz. Neutron-induced single event upsets in static RAMS observed a 10 km flight attitude. *IEEE Transactions on Nuclear Science*, 40(2):74–77, 1993.

[OTM10] C. Ortega, J. Tse, and R. Manohar. Static power reduction techniques for asynchronous circuits. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 52–61, May 2010.

[Ous90] J. K. Ousterhout. Tcl: An embeddable command language. In *Proceedings of the 1990 Winter USENIX Conference*. Citeseer, 1990.

[PBP15] J. Pak, Y. Bei, and D. Z. Pan. Electromigration-aware redundant via insertion. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 544–549, Jan. 2015.

[PCC+14] R. Pawlowski, J. Crop, M. Cho, J. Tschanz, V. De, T. Fairbanks, H. Quinn, S. Borkar, and P. Y. Chiang. Characterization of radiation-induced SRAM and logic soft errors from 0.33V to 1.0V in 65nm

CMOS. In *Proceedings of IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, Sep. 2014.

[PCD+11] L. A. Plana, D. Clark, S. Davidson, S. B. Furber, J. Garside, E. Painkras, J. Pepper, S. Temple, and W. J. Bainbridge. SpiNNaker: design and implementation of a GALS multicore system-on-chip. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 7(4):17:1–17:18, Dec. 2011.

[PCV12] J. Pontes, N. Calazans, and P. Vivet. Adding temporal redundancy to Delay Insensitive codes to mitigate single event effects. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 142–149, 2012.

[PDF+98] N. C. Paver, P. Day, C. Farnsworth, D. L. Jackson, W. A. Lien, and J. Liu. A low-power, low-noise, configurable self-timed DSP. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 32–42, Washington, DC, USA, 1998. IEEE Computer Society.

[Pen06] S. Peng. *Implementing self-healing behavior in Quasi Delay-Insensitive circuits*. PhD thesis, Cornell University, 2006.

[PFS10] T. Panhofer, W. Friesenbichler, and A. Steininger. Reliability estimation and experimental results of a self-healing asynchronous circuit: A case study. In *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 91–98, Jun. 2010.

[PFT+07] L. A. Plana, S. B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang. A GALS infrastructure for a massively parallel multiprocessor. *IEEE Design & Test of Computers*, 24(5):454–463, Sep. 2007.

[PM05] S. Peng and R. Manohar. Efficient failure detection in pipelined asynchronous circuits. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 484–493, 2005.

[PM06] S. Peng and R. Manohar. Self-healing asynchronous arrays. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 34–45, Mar. 2006.

[PMMC10] J. Pontes, M. Moreira, F. Moraes, and N. Calazans. Hermes-AA: A 65nm asynchronous NoC router with adaptive routing. In *Proceedings of IEEE International SOC Conference (SOCC)*, pages 493–498, 2010.

[PMMC11] J. Pontes, M. Moreira, F. Moraes, and N. Calazans. Hermes-A - an asynchronous NoC router with distributed routing. In *Proceedings of International Conference on Integrated Circuit and System Design: Power and Timing modeling, Optimization and Simulation (PATMOS)*, pages 150–159, 2011.

[PN95] S. J. Piestrak and T. Nanya. Towards totally self-checking delay-insensitive systems. In *Proceedings of International Symposium on Fault-Tolerant Computing (FTCS)*, pages 228–237, Jun. 1995.

[Pon12] J. Pontes. *Soft Error Mitigation in Asynchronosu Networks on Chip*. PhD thesis, PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL, 2012.

[PS02] J. Pangjun and S. S. Sapatnekar. Low-power clock distribution using multiple voltages and reduced swings. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(3):309–318, Jun. 2002.

[Ram11] C. Ramey. TILE-Gx100 ManyCore Processor: Acceleration Interfaces and Architecture. *Presented at: Hot Chips (HC23)*, 2011.

[RFR+10] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Proceedings of ACM/IEEE International Symposium On Networks-on-Chip (NOCS)*, pages 25–32, 2010.

[RFZJ13] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch. Methods for fault tolerance in Networks-on-Chip. *ACM Computing Surveys*, 46(1):8:1–8:38, Jul. 2013.

[RJDC98] P. J. Restle, K. A. Jenkins, A. Deutsch, and P. W. Cook. Measurement and modeling of on-chip transmission line effects in a 400 MHz microprocessor. *IEEE Journal of Solid-State Circuits*, 33(4):662–665, Apr. 1998.

[RK94] D. A. Rennels and Hyeongil Kim. Concurrent error detection in self-timed VLSI. In *Proceedings of International Symposium on Fault-Tolerant Computing (FTCS)*, pages 96–105, Jun. 1994.

[RMA+14] S. Rusu, H. Muljono, D. Ayers, S. Tam, Wei Chen, and et al. Ivytown: A 22nm 15-core enterprise Xeon ®; processor family. In *Proceedings of IEEE International Solid - State Circuits Conference (ISSCC)*, pages 102–103, Feb. 2014.

[Saa61] T. L. Saaty. *Elements of queueing theory: with applications*. McGraw-Hill New York, 1961.

[SABR04] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, pages 177–186, 2004.

[Sar03] K. Saraswat. Interconnections: Aluminum metallization. `http://www.stanford.edu/class/ee311/NOTES/Interconnect_Al.pdf`, 2003. Online; accessed 16/01/2016.

[SE10a] W. Song and D. Edwards. An asynchronous routing algorithm for Clos networks. In *Proceedings of International Conference on Application of Concurrency to System Design (ACSD)*, pages 67–76, 2010.

[SE10b] W. Song and D. Edwards. A low latency wormhole router for asynchronous on-chip networks. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 437–443, 2010.

[SE11a] W. Song and D. Edwards. Asynchronous spatial division multiplexing router. *Microprocessors and Microsystems*, 35(2):85–97, 2011.

[SE11b] W. Song and D. Edwards. *Spatial parallelism in the routers of asynchronous on-chip networks*. PhD thesis, University of Manchester, 2011.

[SEE96] M. Shams, J. C. Ebergen, and M. I. Elmasry. A comparison of CMOS implementations of an asynchronous circuits primitive: the C-element. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 93–96, Aug. 1996.

[SF01] J. Sparsø and S. B. Furber. *Principles of Asynchronous Circuit Design: a Systems Perspective*. Kluwer Academic Publishers, 2001.

[SFGP09] Y. Shi, S. B. Furber, J. Garside, and L. A. Plana. Fault tolerant delay insensitive inter-chip communication. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 77–84, 2009.

[SG08] A. Sheibanyrad and A. Greiner. Two efficient synchronous asynchronous converters well-suited for Networks-on-Chip in GALS architectures. *Integration, the VLSI Journal*, 41(1):17 – 26, 2008.

[SGH+07] J. C. Smolens, B. T. Gold, J. C. Hoe, B. Falsafi, and K. Mai. Detecting emerging wearout faults. In *Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects*, 2007.

[Shi10] Y. Shi. *Fault-Tolerant Delay-Insensitive Communication*. PhD thesis, University of Manchester, 2010.

[Sla11] Charles Slayman. Soft error trends and mitigation techniques in memory devices. In *Proceedings of Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–5. IEEE, 2011.

[SMK10] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis. An analysis of on-chip interconnection networks for large-scale chip multiprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(1):4:1–4:28, May 2010.

[SN96] K. L. Shepard and V. Narayanan. Noise in deep submicron digital design. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 524–531, 1996.

[SNI06] Y. Sasaki, K. Namba, and H. Ito. Soft error masking circuit and latch using Schmitt Trigger circuit. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 327–335, Oct. 2006.

[Sor09] D. Sorin. *Fault Tolerant Computer Architecture*. Morgan & Claypool Publishers, 2009.

[Spa07] J. Sparsø. Asynchronous design of Networks-on-Chip. In *Proceedings of Norchip*, pages 1–4, 2007.

[SS93] J. Sparsø and J. Staunstrup. Delay-insensitive multi-ring structures. *Integration, the VLSI Journal*, 15(3):313 – 340, 1993. Special Issue on asynchronous systems.

[SSC08] E. Stott, P. Sedcole, and P. Y. K. Cheung. Fault tolerant methods for reliability in FPGAs. In *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, pages 415–420, 2008.

[Sut89] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[Syn13] Synopsys. Design Compiler ® User Guide, Version I-2013.12. `www.synopsys.com`, Dec. 2013.

[Syn14] Synopsys. PrimeTime ® PX User Guide, Version I-2013.12-SP2. `www.synopsys.com`, Mar. 2014.

[SZG14] W. Song, G. Zhang, and J. Garside. On-line detection of the deadlocks caused by permanently faulty links in Quasi-Delay Insensitive Networks on Chip. In *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 211–216. ACM, 2014.

[TCL+07] A. Taubin, J. Cortadella, L. Lavagno, A. Kondratyev, and A. Peeters. Design automation of real-life asynchronous devices and systems. *Foundations and Trends in Electronic Design Automation*, 2(1):1–133, Jan. 2007.

[TKM+02] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, and et al. The Raw microprocessor: a computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, Mar. 2002.

[TM02] D. E. Thomas and P. R. Moorby. *The Verilog® Hardware Description Language*, volume 2. Springer Science & Business Media, 2002.

[TM04] M. B. Tahoori and S. Mitra. Defect and fault tolerance of reconfigurable molecular computing. In *Proceedings of Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 176–185, 2004.

[TN93] A. Taber and E. Normand. Single event upset in avionics. *IEEE Transactions on Nuclear Science*, 40(2):120–126, 1993.

[TS83] E. Takeda and N. Suzuki. An empirical model for device degradation due to hot-carrier injection. *IEEE Electron Device Letters*, 4(4):111–113, Apr. 1983.

[TSM14] TSMC. Annual Reports: R&D Organization and Investment. `http://www.tsmc.com/download/ir/annualReports/2014/english/e_5_2.html`, 2014. Online; accessed: 28/01/2016.

[TSR+98] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proceedings of Design Automation Conference (DAC)*, pages 732–737, 1998.

[TTD+09] X.-T. Tran, Y. Thonnart, J. Durupt, V. Beroulle, and C. Robach. Design-for-test approach of an asynchronous Network-on-Chip architecture and its associated test pattern generation and application. *IET Computers Digital Techniques*, 3(5):487–500, 2009.

[Tu03] K. N. Tu. Recent advances on electromigration in very-large-scale-integration of interconnects. *Journal of Applied Physics*, 94(9):5451–5473, 2003.

[TVC10] Y. Thonnart, P. Vivet, and F. Clermidy. A fully-asynchronous low-power framework for GALS NoC integration. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 33–38, 2010.

[TW10] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks 5th Edition*. Prentice Hall, 2010.

[Vaj11] A. Vajda. *Programming Many-core Chips*. Springer, 2011.

[Ver88] T. Verhoeff. Delay-insensitive codes—an overview. *Distributed Computing*, 3(1):1–8, 1988.

[VHR+08] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, and et al. An 80-Tile Sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, Jan. 2008.

[VM02] T. Verdel and Y. Makris. Duplication-based concurrent error detection in asynchronous circuits: shortcomings and remedies. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFTVS)*, pages 345–353, 2002.

[WCB+15] J. Warnock, B. Curran, J. Badar, G. Fredeman, D. Plass, and et al. 22nm Next-generation IBM System z microprocessor. In *Proceedings of IEEE International Solid - State Circuits Conference (ISSCC)*, pages 1–3, Feb. 2015.

[WJM08] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor system-on-chip (MPSoC) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.

[YA10] Q. Yu and P. Ampadu. Transient and permanent error co-management method for reliable Networks-on-Chip. In *Proceedings of ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 145–154, 2010.

[YA12] Q. Yu and P. Ampadu. Dual-layer adaptive error control for Network-on-Chip links. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(7):1304–1317, Jul. 2012.

[YCCP03] J. Yang, C. Choy, C. Chan, and K. Pun. Design for self-checking and self-timed datapath. In *Proceedings of VLSI Test Symposium (VTEST)*, pages 417–422, Apr. 2003.

[YIO+12] T. Yoneda, M. Imai, N. Onizawa, A. Matsumoto, and T. Hanyu. Multi-Chip NoCs for automotive applications. In *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 105–110, 2012.

[Yu11] Q. Yu. *Transient and permanent error management for Networks-on-Chip*. PhD thesis, University of Rochester, 2011.

[YYL+12] J. Yao, Z. Ye, M. Li, Y. Li, R. D. Schrimpf, D. M. Fleetwood, and Y. Wang. Statistical analysis of soft error rate in digital logic design including process variations. *IEEE Transactions on Nuclear Science*, 59(6):2811–2817, Dec. 2012.

[ZGS+15] G. Zhang, J. D. Garside, W. Song, J. Navaridas, and Z. Wang. Deadlock recovery in asynchronous Networks on Chip in the presence of transient faults. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 100–107, May 2015.

[ZPJ+15] Y. Zhang, Z. Peng, J. Jiang, H. Li, and M. Fujita. Temperature-aware software-based self-testing for delay faults. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 423–428, Mar. 2015.

[ZSG+13] G. Zhang, W. Song, J. D. Garside, J. Navaridas, and Z. Wang. Transient fault tolerant QDI interconnects using redundant check code. In *Proceedings of Euromicro Conference on Digital System Design (DSD)*, pages 3–10, 2013.

[ZSG+14a] G. Zhang, W. Song, J. D. Garside, J. Navaridas, and Z. Wang. An asynchronous SDM Network-on-Chip tolerating permanent faults. In *Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 9–16, May 2014.

[ZSG+14b] G. Zhang, W. Song, J. D. Garside, J. Navaridas, and Z Wang. Protecting QDI interconnects from transient faults using delay-insensitive redundant check codes. *Microprocessors and Microsystems*, 38(8, Part A):826 – 842, 2014.

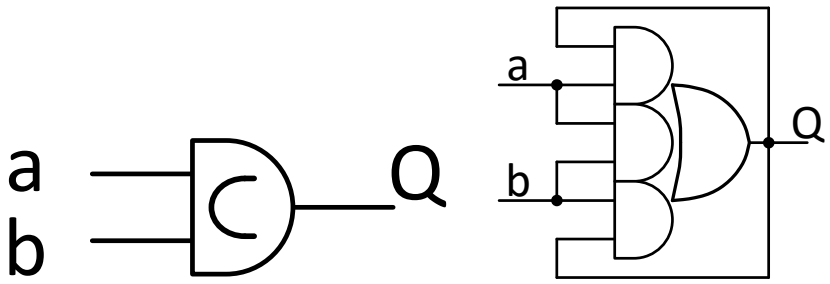
[ZYA12] M. Zhang, Q. Yu, and P. Ampadu. Fine-grained splitting methods to address permanent errors in Network-on-Chip links. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2717–2720, 2012.

Appendix A

Implementation of asynchronous cells

Using the UMC 0.13 μm standard cell library, all asynchronous elements are built from standard cells. This chapter presents the implementation of several basic asynchronous elements used in asynchronous circuits.

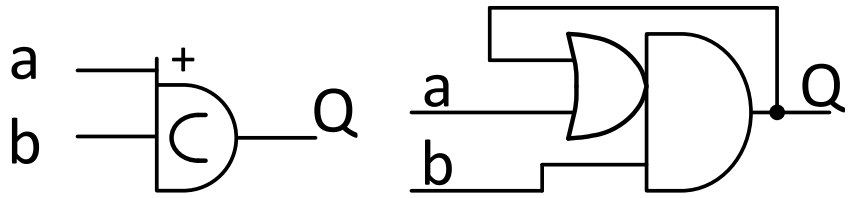
2-input symmetric C-element



(a) Symbol (b) Schematic

Figure A.1: 2-input symmetric Muller C-element

```
// Verilog implementation
module c2 (a, b, Q);
input a, b;
output Q;
AO222EHD U1 (.A1(Q), .A2(a), .B1(Q), .B2(b), .C1(a), .C2(b), .O(Q));
endmodule
```



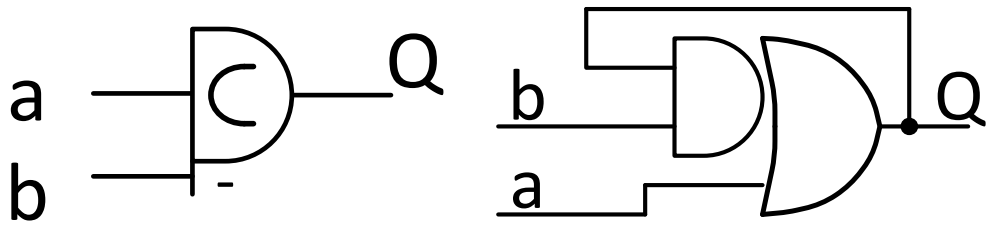
(a) Symbol (b) Schematic

Figure A.2: 2-input asymmetric C-element with a plus input

2-input asymmetric C-element with a plus input

```
// Verilog implementation
module c2p (a, b, Q);
input a, b;
output Q;
OA12EHD U1 (.B1(a), .B2(Q), .A1(b), .O(Q));
endmodule
```

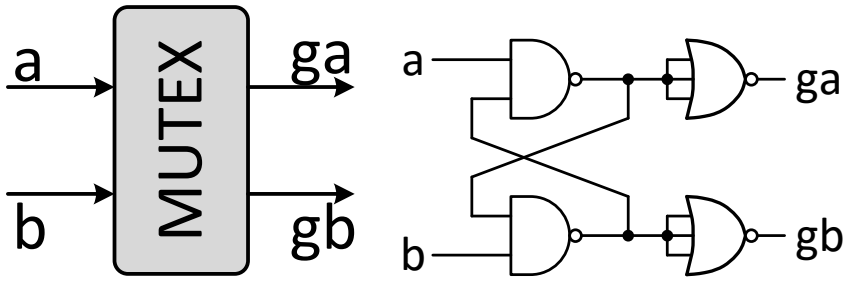
2-input asymmetric C-element with a minus input



(a) Symbol (b) Schematic

Figure A.3: 2-input asymmetric C-element with a minus input

```
// Verilog implementation
module c2n (a, b, Q);
input a, b;
output Q;
AO12EHD U1 (.B1(b), .B2(Q), .A1(a), .O(Q));
endmodule
```



(a) Symbol and logic function (b) Truth table

Figure A.4: Mutex element

Mutex Element

```
// Verilog implementation
module mutex (a, b, ga, gb);
input    a, b;
output ga, gb;
wire     gan, gbn;
ND2HHD U1 (.I1(a), .I2(gbn), .O(gan));
NR3HHD U2 (.I1(gbn), .I2(gbn), .I3(gbn), .O(gb));
NR3HHD U3 (.I1(gan), .I2(gan), .I3(gan), .O(ga));
ND2KHD U4 (.I1(b), .I2(gan), .O(gbn));
endmodule
```

Appendix B

Latency and area models of DIRC pipelines

Delay-Insensitive Redundant Check (DIRC) coding scheme has been proposed to protect QDI communication from transient faults (Chapter 4). Applying DIRC codes and the Redundant Protection of Acknowledge (RPA) wires, the resulting DIRC pipelines can be constructed in a flexible way to satisfy different fault-tolerance requirement. This chapter builds latency and area analytical models to evaluate the hardware overhead of different DIRC pipelines with different construction patterns.

To make the following analytical models easy to follow, some constants and parameters used through this chapter are listed below:

- ***N*** and ***n***: An unprotected basic QDI pipeline contains N parallel 1-of-n channels.
- ***CN***: The number of 1-of-n data words in a DIRC code, or the number of 1-of-n data channels in a DIRC channel.
- ***GN***: The number of DIRC channels, or the number of groups of data words after using DIRC. For simplicity, it is assumed $N = CN \cdot GN$.
- α: Ratio of the area of 3-input to 2-input C-element.
- β: Ratio of the area of 2-input OR gate to C-element.
- ***k***: Ratio of the area of long link wires (A_w) between two basic pipeline stages to the area of one basic asynchronous latch (A_{Basic}).

B.1 Latency analysis

The saturation throughput of a QDI pipeline is determined by the equivalent period of two contiguous pipeline stages where a forward data path and a backward *ack* path form a loop (Figure 3.9). A latency model is built to evaluate the speed overhead of DIRC pipelines. For

the sake of simplicity, it is assumed that cell delays for positive/negative transitions are the same while the wire delay is zero. The loop latency T can be expressed as (B.1), where t_{Latch}, t_{CD} and t_{Comb} are the propagation latency of an asynchronous latch, the CD and other combinational circuits respectively. The equivalent period is approximated to $2T$.

$$T = 2t_{Latch} + t_{CD} + t_{Comb} \tag{B.1}$$

Assume a basic pipeline stage (denoted by a subscript of *Basic*) contains N 1-of-n slices. According to the implementation shown in Figure 2.12, the asynchronous latch is built from 2-input C-elements whose propagation latency is t_{c2}. CD is a tree constructed by n-input OR-gates and 2-input C-elements. Its delay $t_{CD,Basic}$ is determined by the propagation latency of an n-input OR-gate (t_{OR}) and the tree depth ($\lceil \log_2 N \rceil$). The loop latency of a basic pipeline T_{Basic} can be estimated using (B.2) (the delay of the inverter, used to invert the *ack* signal at the latch input, is ignored).

$$t_{Latch,Basic} = t_{c2} \quad t_{CD,Basic} = t_{OR} + \lceil \log_2 N \rceil t_{c2} \quad t_{Comb,Basic} = 0 \tag{B.2}$$

For a DIRC pipeline using both DIRC and RPA (Figure 4.5), 1-of-n adders are added to the data path. The asynchronous latch for data words uses 3-input C-elements instead of 2-input ones. The redundant check words increase the tree depth of the CD (the new depth is $\lceil \log_2 (N + GN) \rceil \approx \lceil \log_2 N \rceil + 1$). Using RPA does not change the tree depth but only induces a delay of a 3-input C-element at the input side. Therefore, the loop latency of a DIRC pipeline T_{DIRC} can be estimated using (B.3):

$$\begin{aligned} t_{Latch,DIRC} = t_{c3} \quad t_{Comb,DIRC} = t_{adder} \\ t_{CD,DIRC} \approx t_{OR} + (\lceil \log_2 N \rceil + 1) t_{c2} + t_{c3} \end{aligned} \tag{B.3}$$

where t_{c3} and t_{adder} are the propagation latency of a 3-input C-element and 1-of-n adders respectively. t_{c3} and t_{c2} depend on the cell library used and the implementation of C-elements. Their delays can be similar. According to the implementation of 1-of-n adders (Section 4.3.1), t_{adder} can be estimated as (B.4).

$$t_{adder} = \lceil \log_2 CN \rceil (t_{c2} + t_{OR}) \tag{B.4}$$

Therefore, the loop latency of a DIRC pipeline can be expressed as (B.5).

$$T_{DIRC} = T_{Basic} + (3t_{c3} - t_{c2}) + t_{adder} \tag{B.5}$$

It can be concluded that, using DIRC plus RPA, t_{adder} is the only variable determining the pipeline speed overhead while the other items in (B.5) are constants. The propagation latency

of the adder is proportional to $\lceil \log_2 CN \rceil$. Usually, the number of data channels in a DIRC code (CN) is far less than the total data channel number for a wide pipeline ($CN << N$, typically CN=2, while the value of N depends on the link width), since a large CN could induce a large area overhead (discussed in the next Section B.2). It can be concluded that $t_{adder} << T_{Basic}$. In this case, the loop latency of a DIRC pipeline, T_{DIRC} is generally less than $2T_{Basic}$. In other words, the period of a DIRC pipeline is more than half the speed of the unprotected basic pipeline, demonstrating the low speed overhead brought by DIRC and RPA.

B.2 Area analysis

As a systematic code, DIRC keeps the original data words so that the internal pipeline stages or combinational circuits between two DIRC stages can obtain the original data directly. This feature makes it possible to construct DIRC pipelines using different patterns, providing designers with flexible choices. Since different DIRC pipelines may have quite different area overhead, area models are built to analyse the area overhead of DIRC pipelines.

B.2.1 Area model for a pipeline stage

An area model for a single pipeline stage is first built to analyse the area of different pipeline stages, including the basic (Figure 2.12), DIRC (Figure 4.5), sDIRC and rDIRC pipeline stages (Section 4.3.4, sDIRC and rDIRC are incomplete DIRC pipeline stages as the sender and receiver respectively, Figure 4.11). Since the proposed RPA technique protecting acknowledge wires causes little overhead compared with DIRC and can be used independently, this area model evaluates only the area overhead brought by DIRC.

The area of a single pipeline stage A can be expressed as (B.6), where A_{Latch}, A_{CD} and A_{Comb} is the area of the asynchronous latch, the CD and other combinational circuits respectively. Wire area is assumed to be zero.

$$A = A_{Latch} + A_{CD} + A_{Comb} \tag{B.6}$$

For a basic N-word-wide 1-of-n pipeline, the area of the asynchronous latch built from 2-input C-elements, $A_{Latch,Basic}$, can be estimated as (B.7) where A_{c2} is the area of a 2-input C-element. The CD area (Figure 4.9b) can be approximated to $A_{CD,Basic}$ shown in (B.8) where A_{OR} is the area of an n-input OR-gate. For the basic pipeline, $A_{Comb,Basic} = 0$. The area of a basic pipeline stage A_{Basic} can be estimated as (B.9) (the area of the inverter used to invert the *ack* signal is small and ignored).

$$A_{Latch,Basic} = N \cdot n \cdot A_{c2} \tag{B.7}$$

$$A_{CD,Basic} = N \cdot A_{OR} + (N-1) \cdot A_{c2} \approx N \cdot (A_{OR} + A_{c2}) \tag{B.8}$$

$$A_{Basic} = A_{Latch,Basic} + A_{CD,Basic} \tag{B.9}$$

Applying DIRC, all N 1-of-n data channels are divided into GN groups, each of which contains CN 1-of-n data channels ($CN = N/GN$) and one check channel. As a result, each DIRC stage contains $(N+GN)$ slices. As described in Section 4.3.1, 3-input C-elements are used to latch data and 2-input C-elements are used to latch check words in DIRC stages. The latch area $A_{Latch,DIRC}$ can be estimated as (B.10) where A_{c3} is the area of a 3-input C-element. The CD of a DIRC stage needs to detect $(N+GN)$ 1-of-n codes. Its area $A_{CD,DIRC}$ can be estimated as (B.11).

$$A_{Latch,DIRC} = N \cdot n \cdot A_{c3} + GN \cdot n \cdot A_{c2} = (A_{c3}/A_{c2} + 1/CN) \cdot A_{Latch,Basic} \tag{B.10}$$

$$A_{CD,DIRC} = (N+GN) \cdot A_{OR} + (N+GN-1) \cdot A_{c2} \approx (1+1/CN) \cdot A_{CD,Basic} \tag{B.11}$$

A complete DIRC stage has $(N+GN)$ 1-of-n adders where N adders are used to regenerate data words and GN adders are used to generate check words (Figure 4.5). The area of a single 1-of-n adder unit A_a increases with n and can be estimated as (B.12). When $CN > 2$, each adder becomes an adder tree containing $(CN-1)$ 1-of-n adder units (Figure 4.7). It is assumed that the area of adders regenerating data words and producing check words is $A_{adder,data}$ (i.e. $A_{Comb,rDIRC}$) and $A_{adder,check}$ (i.e. $A_{Comb,sDIRC}$) respectively. $A_{adder,data}$, and $A_{adder,check}$ can be expressed as (B.13) while their sum is A_{adder} (i.e. $A_{Comb,DIRC}$).

$$A_a = n^2 \cdot A_{c2} + n \cdot A_{OR} \tag{B.12}$$

$$\begin{cases} A_{Comb,rDIRC} = A_{adder,data} = N \cdot (CN-1) \cdot A_a \\ A_{Comb,sDIRC} = A_{adder,check} = N \cdot (1-1/CN) \cdot A_a \\ A_{Comb,DIRC} = A_{adder} = N \cdot (CN-1/CN) \cdot A_a \end{cases} \tag{B.13}$$

Assuming $A_{c3} = \alpha A_{c2}$ and $A_{OR} = \beta A_{c2}$, (B.13) can be rewritten into (B.14). The area of a complete DIRC stage A_{DIRC} is depicted in (B.15).

$$\begin{cases} A_{adder,data} = (CN-1)(n+\beta) \cdot A_{Latch,Basic} \\ A_{adder,check} = (1-1/CN)(n+\beta) \cdot A_{Latch,Basic} \\ A_{adder} = (CN-1/CN)(n+\beta) \cdot A_{Latch,Basic} \end{cases} \tag{B.14}$$

$$\begin{aligned} A_{DIRC} = &(\alpha + 1/CN) \cdot A_{Latch,Basic} + (1+1/CN) \cdot A_{CD,Basic} + \\ &(CN-1/CN)(n+\beta) \cdot A_{Latch,Basic} \end{aligned} \tag{B.15}$$

Comparing (B.15) with (B.9), it can be found that 1-of-n adders contributions most to the

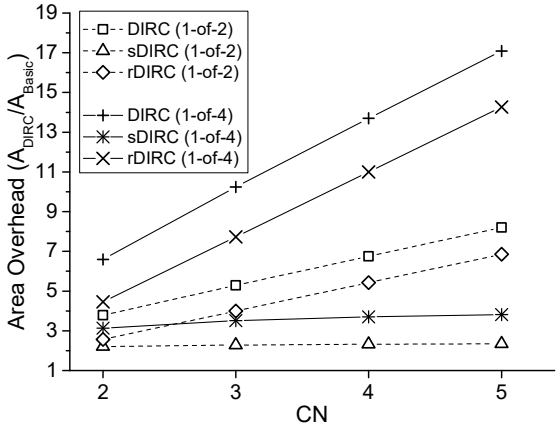


Figure B.1: Area overhead estimation of a single pipeline stage

area overhead (the third term in (B.15)). According to (B.12), the area of a single 1-of-n adder unit A_a increases with n^2 so that 1-of-4 DIRC pipelines have larger area overhead than 1-of-2 ones with the same width. For a 1-of-n DIRC pipeline stage with a fixed data width, A_{adder} increases approximately linearly with CN while the area overhead brought by the latch and CD decreases slightly according to (B.15). As a result, the area of a DIRC stage increases with CN. Similarly, area of the incomplete DIRC stages, sDIRC and rDIRC, can be estimated as (B.16).

$$\begin{cases} A_{sDIRC} = (1+1/CN)\cdot A_{Latch,Basic} + (1+1/CN)\cdot A_{CD,Basic} + \\ \qquad\qquad (1-1/CN)(n+\beta)\cdot A_{Latch,Basic} \\ A_{rDIRC} = \alpha A_{Latch,Basic} + A_{CD,Basic} + (CN-1)(n+\beta)\cdot A_{Latch,Basic} \end{cases} \tag{B.16}$$

According to their transistor counts [SEE96], the area ratio of 3-input to 2-input C-element (α) and the area ratio of 2-input OR gate to 2-input C-element (β) can be estimated as (B.17).

$$\alpha \approx 1.25; \quad \beta \approx 0.5 \tag{B.17}$$

Figure B.1 presents the area overhead estimation of different DIRC pipeline stages with different configurations. The baseline is the area of basic pipeline stages. The Y-axis is the ratio of the area of DIRC (or sDIRC, rDIRC) stages to the area of the basic pipeline stages, measuring the area overhead. CN represents the number of data words in a DIRC code. As Figure B.1 shows, for a complete 1-of-2 DIRC pipeline stage that $CN = 2$, A_{DIRC}/A_{Basic} is approximately 3.8; when $CN = 5$, the ratio rises to 8.2. For a 1-of-4 DIRC pipeline stage, A_{DIRC}/A_{Basic} is approximately 6.6 when $CN = 2$, and 17.0 when $CN = 5$. 1-of-4 pipeline stages have a steeper gradient compared with 1-of-2 ones. Thus, considering only complete DIRC stages, applying DIRC to 1-of-2 pipeline with $CN=2$ incurs the least area overhead.

Compared with complete DIRC pipeline stages, the incomplete sDIRC and rDIRC stages incur less area overhead. The sDIRC stage has the lowest gradient with CN. The ratio A_{sDIRC}/A_{Basic} increases slightly with a rising CN, from 2.2 to 2.3 for 1-of-2 pipelines and from 3.1 to 3.8 for 1-of-4 pipelines.

B.2.2 Area model for a pipeline with different construction

The area of a pipeline can be estimated as the total area of all pipeline stages and long wires. For the purpose of simplicity, it is assumed that all long inter-stage wires have the same area (short internal wires are ignored). A_{wire} is the area of $N \cdot n$ wires between two contiguous basic pipeline stages. Applying DIRC, the wire number becomes $N \cdot n \cdot (1 + 1/CN)$. Thus, in a protected pipeline segment, the wire area between two contiguous stages is $(1 + 1/CN) \cdot A_{wire}$ (*ack* wires are ignored due to their small number). The original basic pipeline stages between the sDIRC and rDIRC are expanded due to the check words. The area of an expanded basic pipeline stage (A_{eBasic}) can be estimated as (B.18). The area of a whole pipeline (AP) is estimated in (B.19):

$$A_{eBasic} = (1 + 1/CN) \cdot A_{Latch} + A_{CD,DIRC} \approx (1 + 1/CN) \cdot A_{Basic} \tag{B.18}$$

$$AP \approx a \cdot A_{Basic} + b \cdot A_{eBasic} + c \cdot A_{sDIRC} + d \cdot A_{DIRC} + e \cdot A_{rDIRC} + f \cdot A_{wire} \tag{B.19}$$

where a, b, c, d, e, f are constants for a specific pipeline.

Taking a 5-stage pipeline for example (Figure 4.12a), the area of a basic pipeline AP_{Basic} can be estimated as (B.20):

$$AP_{Basic} = 5A_{Basic} + 4A_{wire} \tag{B.20}$$

If the pipeline is fully protected (using three complete DIRC stages and two incomplete DIRC stages) as shown in Figure 4.12b, the pipeline area $AP_{DIRC,P0}$ can be estimated as (B.21):

$$AP_{DIRC,P0} = 3A_{DIRC} + A_{sDIRC} + A_{rDIRC} + 4(1 + 1/CN) \cdot A_{wire} \tag{B.21}$$

Protecting alternate stages (Figure 4.12c), three DIRC stages are separated by two expanded basic stages. The area of this pipeline $AP_{DIRC,P1}$ is shown in (B.22).

$$AP_{DIRC,P1} = 2A_{eBasic} + A_{DIRC} + A_{sDIRC} + A_{rDIRC} + 4(1 + 1/CN) \cdot A_{wire} \tag{B.22}$$

The area of pipelines with point-to-point protection ($AP_{DIRC,P2}$) in Figure 4.12d and a critical-stage protection ($AP_{DIRC,P3}$) in Figure 4.12e can be estimated as (B.23) and (B.24).

$$AP_{DIRC,P2} = 3A_{eBasic} + A_{sDIRC} + A_{rDIRC} + 4(1 + 1/CN) \cdot A_{wire} \tag{B.23}$$

$$AP_{DIRC,P3} = 3A_{Basic} + A_{sDIRC} + A_{rDIRC} + (4 + 1/CN) \cdot A_{wire} \quad \text{(B.24)}$$

The area overhead of a whole 5-stage pipeline can be divided into two cases ($A_{wire} = 0$ and $A_{wire} > 0$) to discuss.

(1) The wire area is assumed to be zero ($A_{wire} = 0$)

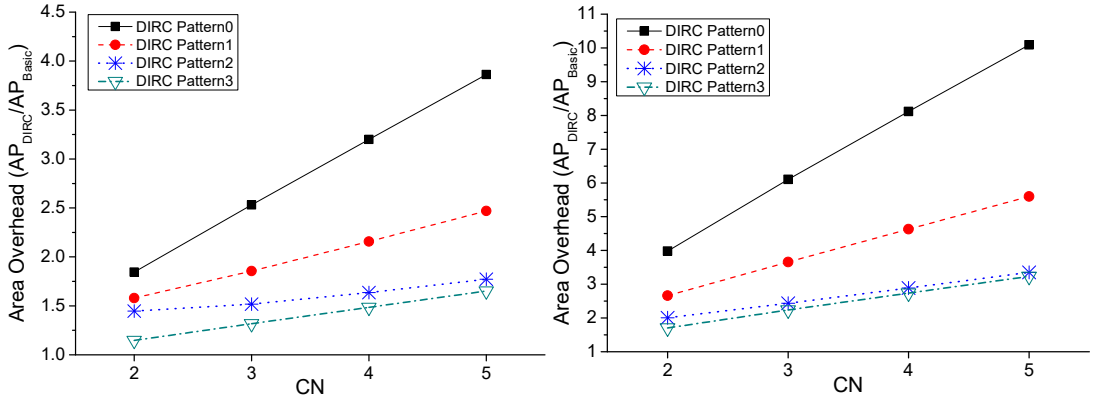
Using (B.18) ˜ (B.24), an area overhead estimation of the pipelines using different patterns is summarised in Figure B.2. The area of the basic pipeline is used as the baseline. The Y-axis represents the area overhead brought by DIRC, which is the ratio of the area of DIRC pipelines to the corresponding basic pipelines (AP_{DIRC}/AP_{Basic}). It can be found that, for all pipelines, the area overhead increases approximately linearly with CN. The pipeline protecting only critical stages (pattern 3) brings the least redundant circuit.

Therefore, DIRC pipelines using different construction patterns may have a quite different area overhead. If DIRC is only used to protect some specific or critical pipeline segments, the incurred area overhead can be greatly decreased.

(2) The wire area is non-zero ($A_{wire} = k \cdot A_{Basic}, k > 0$)

In practical pipelines, the data wires between two contiguous stages may be long and occupy a lot of area due to the large number of inserted wire-buffers. Assuming that $A_{wire} = k \cdot A_{Basic}$ $(k > 0)$, k is the ratio of the area of long wires to the area of a basic pipeline stage as (B.9) shows. Its value is decided by the practical circuit after fabrication which can vary. The area overhead of different pipelines is re-estimated using the previous model. The fully-protected DIRC pipeline (pattern 0), which has the largest area overhead than other pipelines, is first presented to analyse the effect of k on the worst-case area overhead (Figure B.3). For the fully-protected DIRC pipeline, the number of long wires between two contiguous DIRC stages is $N \cdot (1 + 1/CN) \cdot n$.

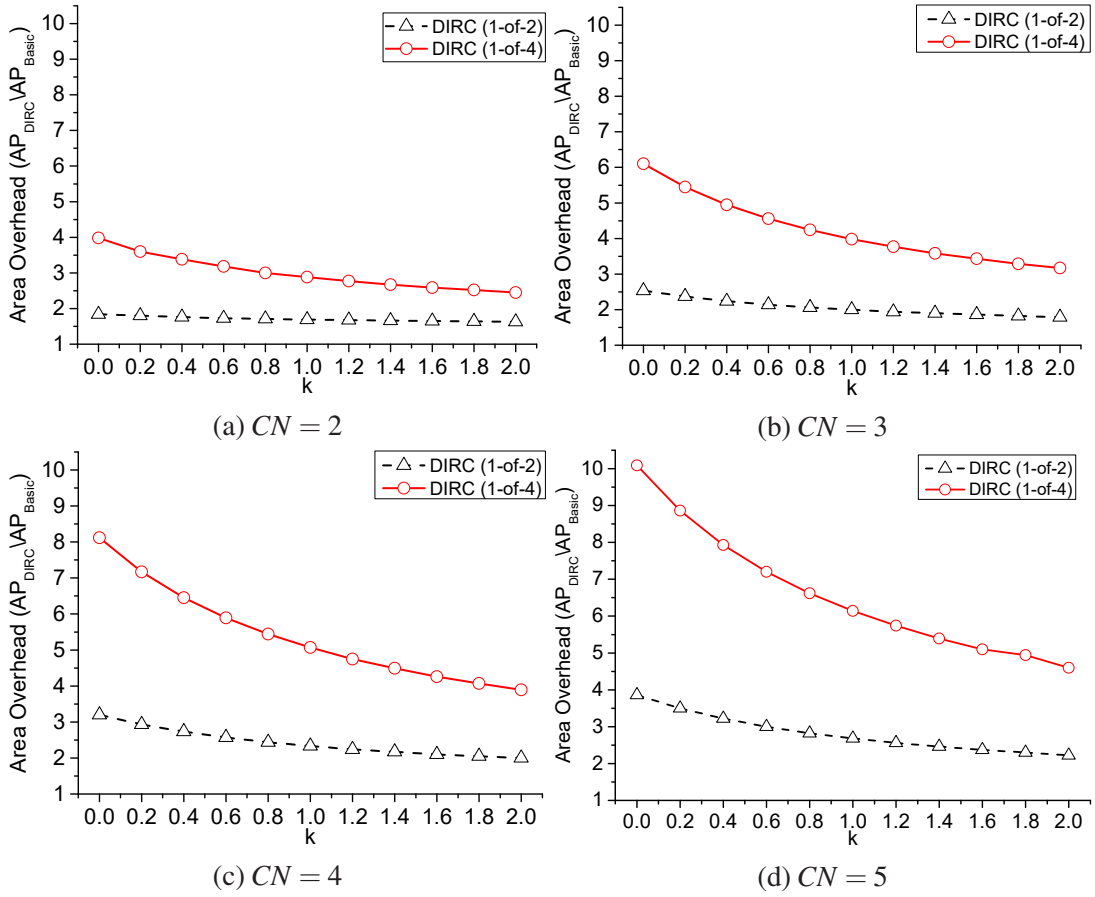
In Figure B.3, the Y-axis still denotes the area ratio AP_{DIRC}/AP_{Basic}. 1-of-4 pipelines have



(a) 1-of-2 pipelines

(b) 1-of-4 pipelines

Figure B.2: Area overhead of different pipelines (wire area $A_{wire} = 0$)



(a) $CN = 2$ (b) $CN = 3$

(c) $CN = 4$ (d) $CN = 5$

Figure B.3: Area overhead estimation of fully-protected pipelines ($A_{wire} = k \cdot A_{Basic}$)

a larger area overhead than 1-of-2 pipelines. The area overhead decreases with k. In an extreme condition when wires consume a significantly larger area than pipeline stages, the ratio will approach to $(1+1/CN)$. A small k denotes that the pipeline stages are the main contributor of the whole area. Since the area of complete DIRC stages increases quickly with CN (Figure B.1), a small CN ($CN = 2$) is more acceptable than a large one. Although the wire number can be reduced by increasing CN, the area of the whole pipeline still increases with CN.

DIRC and basic pipeline stages are mutually exchangeable. Arbitrary basic stages can be replaced by DIRC ones to strengthen fault-tolerance. This feature permits designers to use DIRC flexibly according to the practical design requirement. If the pipeline is constructed using other patterns (Figure 4.12c $\sim$ 4.12e), the area overhead can be further reduced. As an example, Figure B.4 presents the area overhead of 1-of-2 pipelines using the point-to-point protection pattern (pattern 2 in Figure 4.12d). Since the area overhead of incomplete DIRC stages is lower than that of a complete DIRC stage (Figure B.1), increasing CN may not lead to a very large area of pipeline stages but reduces the wire area a lot. As a result, the pipeline area can be reduced. Figure B.4 shows cases where a large CN may reduce the area ratio (CN=3

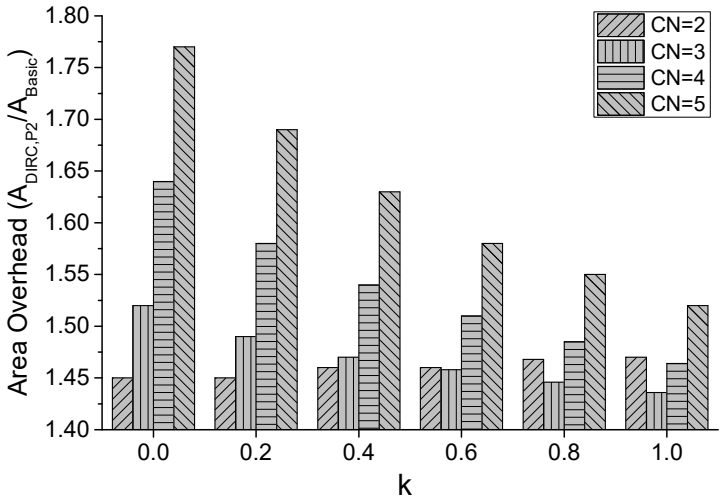


Figure B.4: Area overhead of 1-of-2 pipelines (point-to-point protection)

when $k \geq 0.6$). This feature provides flexible choices for designers to design a fault-tolerant system with an acceptable overhead according to the practical requirement.

B.3 Summary

In practical designs, designers should choose suitable configuration or patterns to build the pipeline to satisfy the fault-tolerance requirement. Simultaneously, the hardware and performance overhead brought by the redundancy is considered. *CN* represents the number of data words in a DIRC code. According to above models, the speed of a DIRC pipeline can be more than half the speed of an unprotected basic pipeline if a small *CN* is used (*CN*=2 for example), showing lower speed overhead compared with existing designs [JM05]. Without accounting for the inter-stage wire area, the area of a complete DIRC pipeline stage can be approximately 4× (1-of-2) or 7× (1-of-4) compared to the unprotected stages. For incomplete DIRC pipeline stages, the area overhead reduces to about 2× and 3× respectively for 1-of-2 and 1-of-4 pipelines respectively. Considering a whole pipeline, the pipeline area including the wire area can approach to (1+1/*CN*). The models also prove that if DIRC is only used to protect some specific or critical pipeline segments, the incurred area overhead can be greatly decreased. These models give a quick estimation on the overhead of DIRC implementation, which are also reflected in the experimental results (Section 4.4).