Handling Physical-Layer Deadlock Caused by Permanent Faults in Quasi-Delay-Insensitive Networks-on-Chip

Guangda Zhang, Wei Song, Jim Garside, Javier Navaridas, and Zhiying Wang

***Abstract*— Networks-on-Chip (NoCs) are promising fabrics to provide scalable and efficient on-chip communication for large-scale many-core systems. In place of the well-studied synchronous NoCs, the event-driven asynchronous ones have emerged as promising replacement thanks to their strong timing robustness especially when implemented in quasi-delay-insensitive (QDI) circuits. However, their fault tolerance has rarely been studied. The QDI NoCs show complicated failure scenarios and behave differently from synchronous ones. As the scaling semiconductor technology is expected with the accelerated aging process, permanent faults become more likely to happen at runtime. These faults can break the handshake, leading to physical-layer deadlocks which can spread and paralyze the whole QDI NoC. This physical-layer deadlock cannot be resolved using conventional fault-tolerant or deadlock management techniques. This paper systematically studies the impact of permanent faults on QDI NoCs, and presents novel deadlock detection and recovery techniques to handle the fault-caused physical-layer deadlock. The proposed detection technique has been implemented to protect the NoC data paths that occupy ∼90% of the logic. Employing the detection and recovery techniques to protect interrouter links (∼60% of the logic), a permanently faulty link is precisely located and the network function can be recovered with graceful performance degradation.**

***Index Terms*— Deadlock, network-on-chip (NoC), permanent fault, quasi-delay-insensitive (QDI), spatial division multiplexing (SDM).**

I. Introduction

NETWORKS-ON-CHIP (NoCs) are a promising infrastructure to support on-chip communication of large-scale multicore systems due to their efficiency and scalability requirements [1]. Most existing NoCs are built synchronously using global clocks. Synchronous NoCs need to distribute the global clock with little skew over long distances, which may cross multiple timing domains belonging to different intellectual property (IP) cores. Costly custom calibration of the huge global clock tree is usually required in high-performance designs [2]. These become increasingly difficult and expensive as the network scales. As an alternative, NoCs can be implemented using asynchronous circuits [3] controlled by handshake protocols. As a result, problems inherent to the distribution of the global clock are alleviated or removed [4], [5]. The asynchronous NoC partitions a chip into multiple synchronous islands, leading to a globally asynchronous, locally synchronous (GALS) system [6]. This naturally enables individual frequency/voltage control and simplifies the chip-level timing closure [7]. With the absence of clock, an event-driven and asynchronous NoC could convey data in a speedy and energy-efficient fashion [5]. The timing robustness brought by a quasi-delay-insensitive (QDI) [3] NoC is also attractive.

Fault tolerance has been extensively studied in synchronous NoCs [8] but rarely in asynchronous NoCs, especially in QDI ones. Protecting QDI NoCs from faults is more difficult than protecting synchronous ones. One reason is that it is difficult to detect a fault and locate its position without using a clock signal as a timing reference. If the receiver fails to get the right data bit, this may be caused by a fault or a transmission delay. The delay can be tolerated in QDI NoCs and the receiver keeps waiting for the lost bit, which may never come due to a fault, proposing a challenge. Most existing QDI NoCs lack fault-tolerance capability [5], [9].

Faults on QDI NoCs can be classified into transient and permanent faults [10]. Transient faults last only for a short period but may be long enough to trigger a transient error of corrupted data. Such an error can be corrected or filtered using fault-tolerant codes or duplication-based techniques, preserving the functionality of the victim component. This has been studied in our previous work [11], [12]. Permanent faults are damages caused by the circuit aging process [10]. Although they are rare, permanent errors caused by observable permanent faults have serious effect on QDI NoCs. Beside data errors, they can halt the handshaking process, resulting in *physical-layer deadlocks*. These deadlocks are different from network-layer ones caused by the cyclic dependence of packets [13]. With the handshake protocol destroyed, the usual deadlock avoidance methods like turn models and fault-tolerant routings cannot

Manuscript received November 13, 2016; revised March 4, 2017 and May 31, 2017; accepted July 5, 2017. Date of publication August 15, 2017; date of current version October 23, 2017. This work was supported in part by the Engineering and Physical Sciences Research Council under Grant EP/I038306/1 and Grant EP/K015699/1, in part by the European Commission Horizon 2020 Programme under Grant 671553, in part by the China Scholarship Council, and in part by the National Natural Science Foundation of China under Grant 61272144, Grant 61402497, and Grant 61402501. *(Corresponding author: Wei Song.)*

G. Zhang was with the School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. He is now with AMS, Beijing 100091, China (e-mail: zhanggd_nudt@hotmail.com).

W. Song was with the School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. He is now with the Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, U.K. (e-mail: wei.song@cl.cam.ac.uk).

J. Garside and J. Navaridas are with the School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K. (e-mail: james.garside@manchester.ac.uk; javier.navaridas@manchester.ac.uk).

Z. Wang is with the School of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: zywang@nudt.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2729081

work without locating and isolating the faulty component. Handling runtime permanent faults on QDI NoCs in such a deadlock state is more difficult than on synchronous NoCs. In the era of deep submicrometer when reliability becomes one of the critical challenges for digital systems [14], it is important to keep specific, critical or ultraexpensive systems working even with some performance loss, proposing a demand for permanent-fault-tolerant QDI NoCs.

This paper handles physical-layer deadlocks caused by permanent faults on QDI NoCs. Its contribution includes the following.

1) A generic deadlock detection mechanism is proposed. It identifies deadlocks caused by faults apart from those due to cyclic dependence. The defective section is precisely located.
2) Instead of initiating an expensive system reboot in the presence of a fault, a *Drain&Release* technique is proposed to release the fault free but deadlocked resources while isolating the defective component.
3) Spatial division multiplexing (SDM) [9], [15] is adopted to divide each interrouter link into independent sublinks. The network function degrades gracefully by masking faulty sublinks and allowing succeeding traffic to go through fault-free ones.
4) For intermittent faults (early symptom of permanent faults) that are long enough to cause a deadlock, the recovery mechanism automatically resumes the isolated pipeline stages once the fault disappears.

II. Background

A. Asynchronous Protocols

Current digital systems are dominated by synchronous circuits, which are governed by one or more global clocks. In the deep submicrometer era, billions of transistors can be integrated on a single chip, where multiple IP cores run at their own clock frequencies, dividing the whole chip into multiple separate timing domains [4], [5], [16]. Besides, the shrinking semiconductor geometry makes circuits more sensitive to process and environmental variations, resulting in delay variations potentially corrupting timing requirements [10]. It becomes increasingly difficult to deliver global clocks across the whole chip with acceptable clock skew.

Asynchronous circuits [3] use handshake protocols rather than clocks, removing the issues caused by clocks. The level-triggered four-phase (return-to-zero) handshake protocol is widely used due to its simplicity in implementation. An illustrative waveform of the four-phase protocol is shown in Fig. 1(a). A full handshake cycle comprises four transitions: transmission of data (*data*+), acknowledge of data (*ack*+), transmission of a spacer (*data*−), and acknowledge of the spacer (*ack*−). Here, the spacer is an all-zero word used to separate two data words. The data bus holds a complete data word (a solid circle) during the transition of *ack*+ while it holds a spacer (an empty circle) during *ack*−. In between the complete data word and the spacer, the bus holds incomplete data (a half full circle). The first two transitions are also called the *set* phase while the rest two are called the *reset* phase.

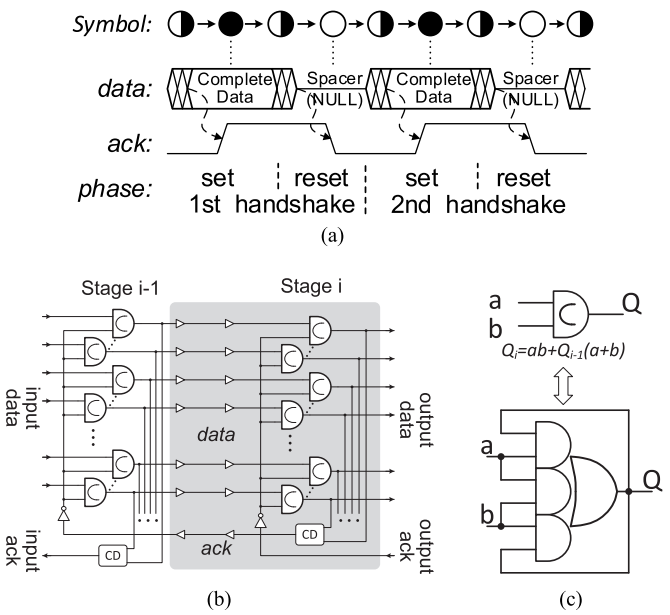


Fig. 1. Asynchronous protocol and pipeline model. (a) Four-phase 1-of-n protocol. (b) Pipeline model. (c) C-element.

QDI circuits are a family of timing-robust asynchronous circuits, which assume that both delays of gates and wires are positive and arbitrary. No delay assumptions other than the isochronic-forks [3] are used so that QDI circuits could tolerate delay variations. In QDI circuits, data are encoded with the timing information through using DI codes [17], such as the 1-of-n codes applied in this paper. A four-phase QDI pipeline is shown in Fig. 1(b). It comprises one or more 1-of-n channels, corresponding to the number of 1-of-n symbols that can be transmitted in parallel. Each pipeline stage contains multiple 1-of-n slices, each of which is a latch built from n C-elements to store a single 1-of-n symbol [3] (a one-hot formatted number between 0 and $n-1$). As the commonly used basic asynchronous element, a C-element outputs "1" (or "0") when both of the inputs are "1"s (or "0"s). Its symbol and standard-cell implementation are shown in Fig. 1(c). A completion detection (CD) circuit synchronizes all 1-of-n channels and produces the *ack* to the preceding stage. A *complete* data word on an N-channel stage is composed of N 1-of-n symbols, while a spacer contains N spacer symbols. During transmission, a stage may hold an *incomplete* data word containing less than N 1-of-n symbols. Note that channel slicing [18] can be used to improve the pipeline performance where all the N 1-of-n channels are divided into several groups, each of which runs independently with its own CDs before the resynchronization point of all groups. This paper employs the basic pipeline with one CD synchronizing all 1-of-n channels at each stage [3], which helps the proposed fault detection mechanism afterward.

B. Permanent Faults on Synchronous Versus QDI Circuits

Permanent faults occur with the aging process [19], [20] and threaten the lifetime of electronics. Some manufacturing imperfections are so tiny that they are not detected during the manufacturing test but become evident after a long period

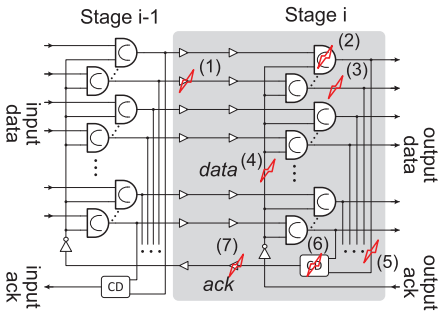


Fig. 2. Pipeline fault model.

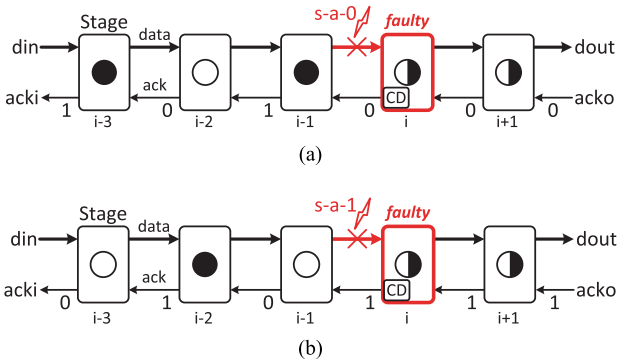


Fig. 3. Deadlocked pipelines due to stuck-at faults on the forward data path. (a) Data stuck-at-0 fault. (b) Data stuck-at-1 fault.

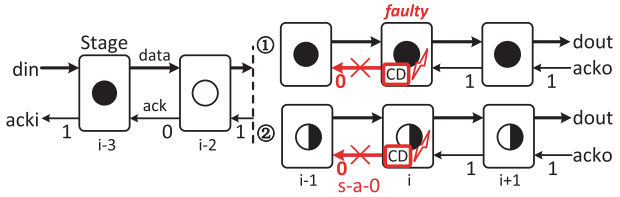


Fig. 4. Deadlocked pipelines due to stuck-at-0 faults on CD or *ack* wires.

of operation, leading to runtime permanent faults [20]. The scaling effects with the advancing semiconductor technology have a sustaining negative impact on long-term chip lifetime [21] and accelerate the aging process. Electronics become more susceptible to runtime permanent faults. This paper uses stuck-at faults to model permanent faults, which has been widely accepted in the semiconductor industry [22]. The state of a wire is either stuck at "0" or "1," masking details of the fault behavior and making it simple to analyze the fault impact.

Permanent faults on synchronous circuits typically lead to persistent data errors demonstrating fixed patterns. Assuming the clock network stays intact, the faulty circuit continuously produces data errors, which can be used to detect faults with accumulated history statistics, i.e., error syndromes, which are usually obtained by using transient-fault-tolerant techniques like fault-tolerant codes [12], [23], [24]. If the error syndromes satisfy specific patterns or timeout conditions, the fault is taken as permanent and the recovery process is invoked; otherwise, it is transient or intermittent.

Differently, QDI circuits are event-driven and controlled by handshake protocols. A permanent fault, which locks a signal at one logical level, not only corrupts data, but breaks the handshake process, deadlocking the circuit. Most existing transient-fault-tolerant QDI designs [12], [25] are deadlocked as well when the handshake is stalled; they cannot easily extract error syndromes, making most conventional fault-tolerant techniques fail [23], [24]. In a NoC, this deadlock occurs in the physical layer, which is different from the network-layer one due to the packet cyclic dependence [13]. Like all deadlocks, physical-layer deadlocks reserve network resources and spread, potentially leading to a paralyzed NoC.

III. Modeling Fault-Caused Deadlocks

Before analyzing the fault-caused deadlocks on QDI pipelines, a simple pipeline fault model is built to reduce all 1-bit permanent faults[1] into two types of faults: "*Data*" and "*Ack*" faults. The pipeline fault model is shown in Fig. 2. The interpipeline interconnects or combinational circuits are considered a part of the receiving pipeline stage. A fault occurring on (1) data interconnects, (2) the C-element or (3) the immediate output of the C-element, or even (4) the *ack* fanouts is considered as a "*Data*" fault, because it alters the data word visible to the downstream stages. A fault that occurs on (5) the CD inputs, (6) the CD, or (7) the *ack* interconnect to the preceding stage is considered as an "*Ack*" fault, because it does not alter a visible data word but obstructs necessary data transitions. For all faults shown in Fig. 2, pipeline *Stage* i is termed the "faulty" stage. The impact of a permanent fault on a QDI pipeline can be classified into four classes (this paper does not support PCHB-implemented QDI circuits [3]).

1) *Data Stuck-at-0 [Fig. 3(a)]:* The input of a latch can get stuck at "0" when a *Data* stuck-at-0 fault strikes on the forward data path, preventing "1" from propagating to downstream stages. 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 1-of-n symbol.
2) *Data Stuck-at-1 [Fig. 3(b)]:* Due to a stuck-at-1 fault on the forward data path, the latch of the faulty stage gets stuck at "1," preventing all downstream stages from being reset as they keep holding a 1-of-n symbol with the invalid "1." As a result, all their *ack* signals are "1"s.
3) *Ack Stuck-at-0 (Fig. 4):* The *ack* signal to the preceding *Stage* i-1 could get stuck at "0" due to a permanent fault on the backward *ack* wire or the CD of the current stage (*Stage* i). As a result, *Stage* i-1 may hold either a complete data word if the faulty *ack* arrives 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, all pipeline stages downstream of the faulty stage cannot be fully reset. Their *ack* signals are all "1"s. The *ack* signal back to the prefault *Stage* i-1 has two faulty scenarios depending on the fault position.

[1]The proposed technique does not handle simultaneous multibit faults. However, if the simultaneous multibit fault causes the same deadlock that is undifferentiated from the one caused by a 1-bit fault, it is handled as a 1-bit fault. Multiple uncorrelated faults can be handled as individual 1-bit faults.

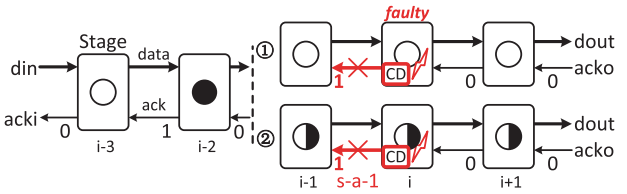


Fig. 5. Deadlocked pipelines due to stuck-at-1 faults on CD or *ack* wires.

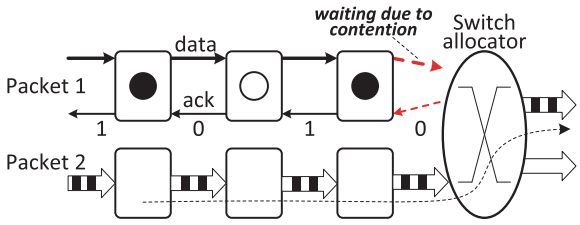


Fig. 6. Fault-free pipeline stall due to contention.

4) *Ack stuck-at-1 (Fig. 5):* The *ack* wire to the preceding stage may get stuck at "1" due to a fault on the backward path, so that incoming "1"s or 1-of-n symbols to the prefault stage cannot be latched. All stages downstream of the faulty stage then hold spacers or incomplete data words, getting stuck at the *set* phase with low *ack*s.

For all cases, all prefault pipeline stages are fault free and would eventually alternately hold complete data words and spacers according to the four-phase handshake protocol [3]. The following observation presents a key pattern that is shared by all the above faulty cases.

Observation 1: A permanent fault leads to a physical-layer deadlock when the faulty wire or gate obstructs the normal handshake process. With sufficient time, the deadlocked pipeline always freezes into a steady state, where all stages downstream of the faulty one have the same *ack* while the *ack* signals of upstream stages are alternately valued.

It should be noticed that some pipeline stages may be stalled for a long time due to traffic contention, which is common in NoCs. Fig. 6 shows an example that two packets in separate pipelines compete for the output, while packet 2 wins the arbitration and gets outputted but packet 1 is stalled before the switch, resulting traffic congestion. This fault-free stall due to contention is different from the aforementioned fault-caused deadlock. Using the 4-phase 1-of-n protocol, the pipeline with the stalled packet presents alternately valued *ack* signals on upstream stages when data wires are stable and there are no matched (allocated) downstream stages, while a deadlocked faulty pipeline holds the same *ack* value on all downstream stages. Thus, the fault-free stalled pipeline will not be mistaken as deadlocked. The presence of consecutive pipelines with the same *ack* value is the key in differentiating fault-caused physical-layer deadlocks from network contention.

IV. General Deadlock Detection Strategy

A. Baseline QDI NoC Router

Fig. 7(a) shows a mesh NoC built from QDI routers and links. The whole chip is a GALS system, where the NoC

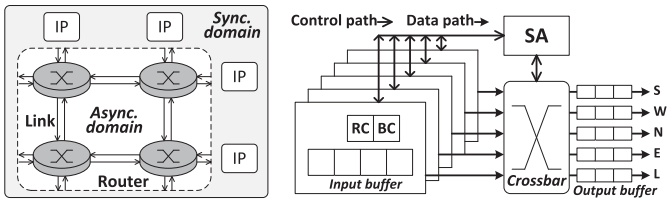


Fig. 7. Baseline QDI NoC and its router. (a) GALS system. (b) Generic router structure.

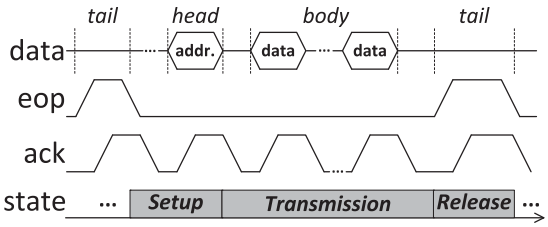


Fig. 8. Flit sequence and router state classification.

forms the asynchronous domain. Four-phase 1-of-4 protocol is applied to both routers and links [3]. A generic router with five bidirectional ports is shown in Fig. 7(b). Four of the ports (south, west, north, and east) communicate with adjacent routers and the fifth connects to the local processing element (PE). A central crossbar provides multiple connection paths from input to output. Pipelined buffers at the input/output 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). Each input buffer has a BC and an RC unit. The BC unit regulates the incoming flit flow while the RC unit computes the output direction of each packet using its head flit. The central crossbar is controlled by the SA unit, which receives routing requests from all RC units and accordingly allocates available output ports to requesting input ports. As Fig. 8 shows, a packet is divided into head, body, and tail flits. The destination address is stored in the head followed by a sequence of body flits. A tail flit, indicated by an end-of-packet (*eop*) signal, separates consecutive packets.

The baseline QDI NoC employs an XY-dimension-ordered routing and wormhole switching [1], so that the control logic deals with only head and tail flits. Once a path through the network is built, it starts carrying a high-density flit flow while control logic keeps in a stable state. Considering that the aging caused permanent faults is believed to be strongly related to circuit activities [26], control logic is expected to have a significant longer life time than the data path [27]. This paper focuses on the fault tolerance of the data path, which can be abstracted to a QDI pipeline.

B. Deadlock Management Strategies for QDI NoCs

Traditional (nonfaulty) deadlocks occur when a NoC cannot resolve cyclic dependence, which is normally triggered by an unrestricted routing algorithm or limited network resources [28]. The deadlocked network would then suffer significant performance degradation or even system failure.

Two strategies can deal with the deadlock: deadlock avoidance and recovery.

1) Deadlock avoidance has been extensively used to prevent cyclic dependence, making NoCs deadlock-free in a fault-free environment. The well-known turn models prohibiting some turns belong to the simplest avoidance methods. Virtual channels can be used to avoid deadlocks by specifying escape channels [1].
2) Deadlock recovery tries to resolve a deadlock if it ever occurs [29]. Therefore, full adaptive routing can be used to improve network performance. Differentiating the real deadlock from congestion is the main challenge. Considering the fact that deadlock rarely occurs if the network is below its saturation point, this method is attractive due to its lower hardware cost compared to deadlock avoidance.

A permanent fault on a QDI NoC stalls the normal handshake process and locks up the pipeline stage, which then causes a physical-layer deadlock. Since this type of deadlock is not caused by cyclic dependence and its occurrence is hardly predictable, traditional deadlock recovery does not work. This paper proposes a new deadlock recovery strategy that comprises two phases. 1) **Deadlock detection**, when the fault-caused deadlock is precisely located and differentiated from traditional deadlocks or network congestion and 2) **deadlock recovery**, when the defective components are bypassed to resume the deadlocked network.

C. General Detection Strategy for Deadlocked QDI NoCs

The physical-layer deadlock can be easily differentiated from network-layer deadlock or network congestion by detecting contiguous *ack* signals (Observation 1) that are unique to fault-caused physical-layer deadlock. The pipelined data path in a NoC can be partitioned into link and router pieces, which are protected separately. A fault on the data path is thus either a link fault or a router data-path fault. It may corrupt the routing request, affect the control logic, and cause data errors. Adding a pair of detection circuits to the ends of each piece to monitor their activities, we can determine that the monitored piece is defective if the following conditions are satisfied.

1) No transition is detected for a long time, implying the pipeline piece is either idle, or blocked by congestion, or deadlocked.
2) The pipeline piece demonstrates one of the following fault patterns.
   a) *Link Fault*: The *ack* signals before the fault at the output of the prefault router are alternately valued (the link is not idle). For the input of the postfault router, all pipeline stages on the reserved path after the fault have the same *ack* (ruling out congestion and network-layer deadlock).
   b) *Router Data-Path Fault*: A path has been built across the router. The *ack* signals at the router input are alternately valued and the granted output has the same *ack*.

V. Deadlock Detection on QDI NoCs

In this section, protected link pieces are first used to demonstrate the fault impact on the baseline QDI NoCs (Section IV-A) and the proposed deadlock detection mechanism. Then, the detection of router data-path faults is discussed. Note that, the control signals *eop* and *ack* are considered as an inseparable part of the data path of a NoC. They are also protected by the proposed fault-tolerance mechanism.

A. Impact of Permanently Faulty Links

Faults may happen on any gates or wires. In a NoC, a fault may affect the body data, the *eop* indicator, or the *ack* signal, leading to six types of faults: stuck-at-1 (*s-a-1*) or stuck-at-0 (*s-a-0*) faults on each of them. When a link fault strikes, the control logic of the input buffer gets stuck at a specific state. Identifying this state is important in detecting the fault. According to wormhole switching, an input buffer can be deadlocked into three states: Setup, Transmission, and Release (Fig. 8).

1) *Setup*: A head flit is arriving and getting blocked in an input buffer, waiting for being granted an output. RC samples the address information from the head flit, generates a routing request, and waits for the grant.
   a) *Data s-a-0* fault: The head flit contains the address information in 1-of-n codes. This fault may corrupt the "1" of a 1-of-n symbol to "0," preventing a complete head flit from arriving.
   b) *Ack s-a-1* fault: When an *ack s-a-1* fault strikes on the head flit, it prevents the head flit from arriving at the input buffer.

   In both the cases, the head is incomplete so no routing request is generated. The incoming packet is stalled at the front of this input buffer.
   a) *Data s-a-1* or *ack s-a-0* faults stall handshake by preventing the received head flit from being reset. Since the routing request is successfully generated and will be granted, the input buffer eventually enters the Transmission state. The Setup *data s-a-1* or *ack s-a-0* fault is equivalent to corresponding Transmission faults.
   b) An *eop s-a-0* fault will not manifest since the *eop* indicator is low in this state.
   c) An *eop s-a-1* fault creates a fake tail flit. Depending on its timing, if the fake tail flit causes a premature *ack* before a complete head flit is received, this *ack* blocks the remaining head flit. Otherwise, if the fake tail flit arrives after the head flit, a routing request is produced and an output would be allocated. The fake tail flit then prematurely pushes the input buffer into Release state.
2) *Transmission*: SA has allocated an output to the requesting input. The head or body flits are traversing the router. All *data* and *ack* stuck-at faults can deadlock the reserved packet path into this state.

State	Fault type	Description	Boolean Expression
Setup	data s-a-1	Persistent head flit, equiv. to the corresponding **Transmission** faults	
	ack s-a-0		
	eop s-a-0	Equiv. to **Transmission** eop s-a-0 fault	
	data s-a-0	Incomplete head flit	1. !rt_ack & !ipia & !ipoa
	ack s-a-1		
	eop s-a-1	Incomplete head flit, persistent tail flit	2. !rt_ack & ipeop & !ipoa
		Complete head flit, persistent tail flit	
Trans-mission	eop s-a-1	Fake tail, equiv. to **Release** eop s-a-1 fault	
	data s-a-0	Deadlocked head/body flit	3. *rt_ack* & (*ipia* == *ipoa*)
	ack s-a-1		
	data s-a-1		
	ack s-a-0		
	eop s-a-0		
Release	ack s-a-0	Persistent tail flit	
	eop s-a-1		
	data s-a-0	Equiv. to the corresponding **Setup** or **Transmission** faults	
	ack s-a-1		
	data s-a-1		
	eop s-a-0		

Fig. 9. Fault types on QDI NoCs.

a) *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. The fault-free "1" gets reset later, leaving the faulty "1" traversing the downstream pipeline stages.
b) *Ack s-a-*0/1 faults prevent data symbols from being set or reset.
c) An *eop s-a-*0 fault prevents the tail flit from arriving.
d) An *eop s-a-*1 fault creates a fake tail flit, which forces the input buffer into Release.

3) *Release*: At the end of Transmission, the input buffer enters Release when it receives a high *eop*. BC releases the reserved output after the *eop* is reset. The input buffer then returns to Setup.
a) *Data s-a-*0/1 and *ack s-a-*1 faults have no effect in this phase, since they cannot prevent the resetting operation of the *eop* indicator.
b) *Ack s-a-*0 and *eop s-a-*1 faults prevent the *eop* from resetting.
c) An *eop s-a-*0 fault has no effect in Release but would block future tail flits.

Fig. 9 summarizes the impact of all faults. Note that a fault may not manifest itself immediately and strike the postfault router. Such a fault is combined into the state that it affects. Fig. 10 shows the protected NoC data path divided into link and router pieces. The *rt_ack* signal denotes when an output is allocated by SA and *ipeop* notifies the arrival of *eop*. These signals also indicate the state when the router is deadlocked: Setup (*rt_ack*− & *eop*−), Transmission (*rt_ack*+ & *eop*−), and Release (*rt_ack*+ & *eop*+). By sampling the *ack* signals of two contiguous stages (*ipia* and *ipoa*), the *eop* indicator (*ipeop*) and the grant indicator (*rt_ack*), a pattern checker at each input can detect and classify a deadlock into three scenarios.

Case 1: A *data s-a-*0 or *ack s-a-*1 fault in the Setup state prevents 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*−). No routing request is generated or granted (*rt_ack*−). This case is indicated by (!*rt_ack* & !*ipia* & !*ipoa*).

Case 2: A Setup *eop s-a-*1 fault happens when the link is idle. It creates a fake tail flit (*ipeop*+) which blocks all incoming packets; therefore, no routing request is generated (*rt_ack*−). The fault is denoted by (!*rt_ack* & *ipeop* & !*ipoa*).

Case 3: All stuck-at faults in Transmission and Release states can be combined into the same scenario. A route is allocated (*rt_ack*+) but a fault strikes during data transmission. All downstream stages have the same *ack* value. This case can be identified by (*rt_ack* & *ipia*==*ipoa*).

A deadlock caused by cyclic dependence in network layer or temporary blockage due to congestion is impossible to trigger any of the above cases. In Section V-B, a timeout mechanism will be proposed to locate the faulty link piece.

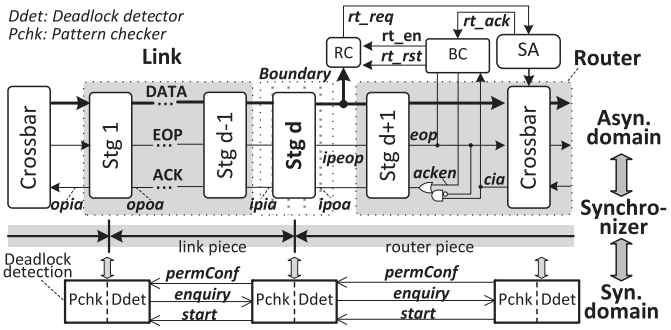


Fig. 10. Protected pipeline pieces with deadlock detection circuits.

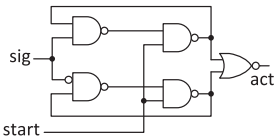


Fig. 11. Transition detector.

B. Detect Deadlock Caused by a Permanently Faulty Link

A general timeout strategy is proposed to detect the deadlock caused by permanently faulty interrouter links in a QDI NoC. A link starts at the output port of the preceding router and ends at the input of the succeeding router, including pipelined input/output buffers and link wires as Fig. 10 shows. A deadlock detector is added at the start of the link (*Stg* 1) and a pattern checker is added at the end of the link (*Stg d*) under control of the deadlock detector.

The key component in the deadlock detector is a transition detector (Fig. 11) monitoring the transition on a certain signal [30]. The *start* is an active-high enable signal. During active detection (*start*+), a positive transition on the detector output (*act*+) denotes that transitions are detected at the monitored signal (*sig*). Resetting *start* withdraws the output to "0" (*act*−). If no transition is detected (*act*−) for a long time, the link is either idle or blocked. The pattern checker

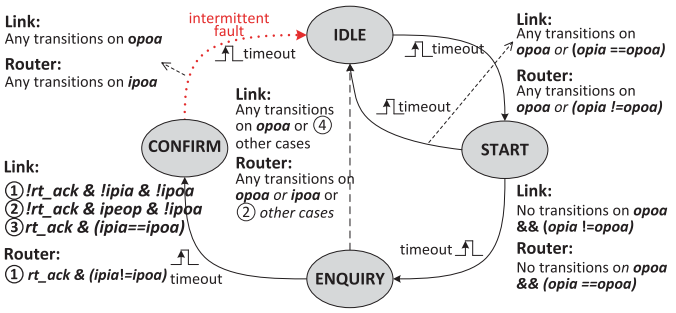


Fig. 12. State machine in the deadlock detector for faulty links and routers.

at the end of the link would check the *ack* sequence using Observation 1 to determine whether the link is struck by a permanent fault.

As Fig. 12 shows, the detection process is controlled by a state machine in the deadlock detector which monitors an *ack* signal at the output buffer (*opoa*). The state transition is partially triggered by a timeout signal produced by a shared synchronous counter in each router. The deadlock detector is, therefore, a sync./async. hybrid circuit with potential metastability issues. Although it is considered safe to directly sample the asynchronous signals because sampling happens only when the link is assumed idle or blocked, synchronizers are added to further reduce the risk of metastability (Fig. 10) [31]. The state machine has four states: IDLE, START, ENQUIRY, and CONFIRM. Excepting the return from ENQUIRY to IDLE, all state transitions are triggered by *timeout*.

1) IDLE is the default state after reset.
2) In START state, the transition detector monitors the transitions on the *ack* (*opoa*) of the output buffer. If no transition is detected and the two contiguous *ack* signals at the output match the deadlock pattern defined in Observation 1 (*opia* != *opoa*), the state transits to ENQUIRY; otherwise, it transits to IDLE as no deadlock is detected.
3) In ENQUIRY state, the deadlock detector sends an *enquiry* signal to the pattern checker in the input buffer of the succeeding router. This checker samples three asynchronous signals from the headmost pipeline stage (*Stg d*). If the values of these signals match one of the three deadlock scenarios described in Section V-A and the transition detector still finds no transition on *opoa* for a whole timeout period, the pattern checker confirms the deadlock and the state transits to CONFIRM; otherwise, the state transits to IDLE immediately.
4) State CONFIRM triggers the network recovery procedure, which isolates the defective pipeline stages between the transition detector and the pattern checker. The state is stuck at CONFIRM for permanent faults.

Considering long lasting intermittent faults, 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 (Fig. 12). The disappearance of the intermittent fault will bring signal transitions on *data*, *eop*, or *ack* wires, which further leads to the transition of the *ack* signal at the output buffer (*opoa*) detected by the transition detector. Consequently, when the next *timeout* arrives, the state machine is reset so that the blocked link can be reused.

C. Detect Deadlock Caused by a Permanently Faulty Router

A fault may happen inside a router, making the *data* path defective as well. The main component belonging to the *data* path in the baseline router is a crossbar connecting inputs to outputs. Similar to handling link faults, a pair of detection circuits are added before and after the crossbar to monitor activities of the input and output buffers. The deadlock detector is put at the output buffer side as well containing a state machine, which is very similar to the one for link faults except for some conditions according to Section IV-C. However, to detect a router data-path fault rather than a link one, the deadlock detector at the output buffer needs to interrogate the corresponding input buffer of the "same" router (the SA has allocated the output buffer to this input and a path through the crossbar has been built) in the ENQUIRY state about the deadlock pattern, instead of enquiring the succeeding router in detecting a link fault.

As Fig. 12 shows, in START, the deadlock detector 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*), the output buffer is downstream of the fault. The state machine transits to ENQUIRY at the end of the second timeout period to interrogate the input of the same router to check if two contiguous *ack* signals are complementary as a potential prefault stage. If no transitions are detected and the two *ack* signals are different (*ipia*!=*ipoa*) during the third timeout period, a permanent fault is confirmed to be detected and the state machine transits to CONFIRM. Note 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 (*rt_ack*+), will the fault exhibit and affect the NoC. The fault deadlocks the reserved packet path.

VI. Recovery From Permanently Faulty Links

Section V provides means to detect a physical-layer deadlock in a QDI NoC and locate the faulty stages on links or intrarouter data-paths. As long as a permanent (intermittent) 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. This section proposes a recovery scheme to resume deadlocked NoCs from faulty links. Recovery from router data-path faults usually requires adaptive routing support from the routing layer, which is left as the future work. The recovery from a faulty link contains two processes: 1) deadlock removal: release deadlocked but fault-free network resources using a *Drain&Release* technique and 2) faulty link isolation: by applying the SDM [9], [15], every link is divided into parallel sublinks. The faulty sublink is disabled to prevent it from causing further deadlock.

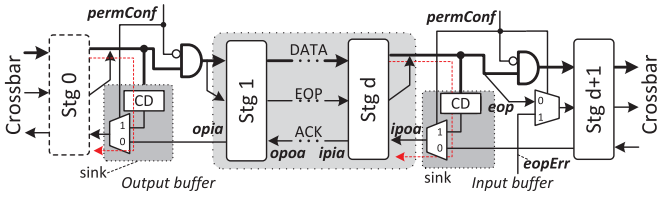


Fig. 13. Modified input/output buffers for network recovery.

A. Deadlock Removal

A fault affects a NoC by halting an in-flight packet, which in turn blocks all other packets awaiting the occupied resources. This deadlock must be resolved to resume the delivery of packets in waiting. A *Drain&Release* technique is proposed to resume the network by dropping the halted packet from the pipeline segments upstream and downstream to the fault separately (the resulting packet loss can be managed through retransmissions or other techniques in the upper data-link or routing layer [32], which is out of the scope of this paper).

1) *Drain*: Draining fault-free flits from the *upstream* segment at the output of the prefault router to resume the deadlocked resources upstream of the fault.
2) *Release*: Producing a fake tail flit at the postfault router input to resume the *downstream* deadlocked resources.

Fig. 13 shows the modified input/output buffers, where the interrouter link is protected (the central shaded region). The circuit for deadlock recovery is added at both ends of the protected link. At the output buffer of the router, a new stage *Stg* 0 is added, so that the extra circuit for recovery can reuse its CD (discussed later) which saves circuit area and avoids extending the critical path. *Stg* 0 is not mandatory, and if added, it can be taken as a part of the router data-path. At the input buffer of the router, the extra circuit is carefully placed before *Stg* $d+1$ (considered in the router data-path). Thus, the whole link (*Stg* 1 to *d*) is protected while the intrarouter critical path (the paths across the crossbar) is mostly unaffected.

1) Drain Operation: According to the four-phase protocol, complete data symbols and spacers are alternately distributed along the upstream pipeline. Thus, adding a sink at the start of the faulty interrouter link (output buffer of the prefault router) drops the blocked packet remaining in the upstream pipeline. When the tail flit of the blocked packet is dropped, all resources in the upstream network resume operation.

As Fig. 13 shows, the sink comprises a CD and a multiplexer. During recovery (*permConf*+), the sink is enabled by driving the *ack* to *Stg* 0 with the added sink CD. The CD can reuse the one belonging to the preceding pipeline stage (*Stg* 0). The *Drain* operation should not affect the faulty status. An AND gate is, therefore, added on the data path to block transitions when draining.

2) Release Operation: According to Section V-A, a router input may be deadlocked at either Setup, Transmission, or Release. Correspondingly, the BC of an input buffer [Fig. 14(a)] would be stuck before transition *rt_ack*+, *eop*+, or *eop*−, as shown in the signal transition

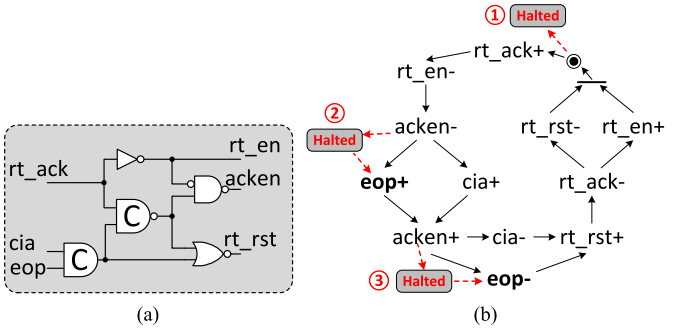


Fig. 14. BC at the input buffer of a router. (a) Circuit. (b) STG.

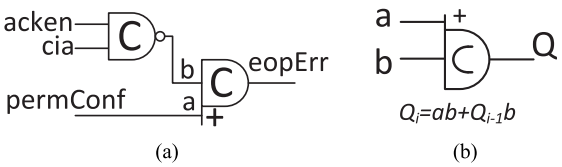


Fig. 15. Tail generator added to the BC. (a) Tail generator. (b) Asymmetric C-element.

graph (STG) [3] shown in Fig. 14(b) [9]. Using extra circuits to bridge the halted transition in the STG, BC generates a fake tail flit, which then releases reserved network resources downstream of the fault.

The transmission of a packet starts with the Setup state. Initially, a head flit is blocked in the input buffer. According to its destination address, RC generates a routing request to SA. When the requested output port is available, SA grants a path to the requesting RC (*rt_ack*+). Then, RC latches/disables the address sampling (*rt_en*-) and allows the head flit, along with all following flits, to traverse the crossbar, which starts the Transmission state. When a tail flit is received (*eop*+) and delivered to the output buffer (*cia*+), a whole packet is transmitted and the Release state begins to release the reserved path. BC starts the release process by blocking the input buffer (*acken*+) in preparation for the next packet. Consequently, the front pipeline stage of the input buffer is reset (*eop*− and *cia*−), which then triggers the reset of RC (*rt_rst*+) to withdraw the routing request to SA. SA, therefore, releases the reserved path in the crossbar (*rt_ack*−). Finally, BC reactivates RC (*rt_rst*− and *rt_en*+) for the address decoding of the next packet. Router enters Setup again.

1) If a permanent fault strikes the interrouter link and causes the router to be stuck in the Setup state, no routing request is generated. Since SA has not allocated a path, no downstream resource is deadlocked. No further operation is thus required.
2) Alternatively, if the router is stuck at Transmission or Release, a path is allocated and all downstream routers are deadlocked. A fake tail is then generated by a local tail generator to release all downstream routers.

Fig. 15 shows the tail generator added to the BC, where an asymmetric C-element with a plus input is used (the plus input "a" affects only the *set* operation) [3]. When a fault-caused deadlock is detected (*permConf*+), the tail generator

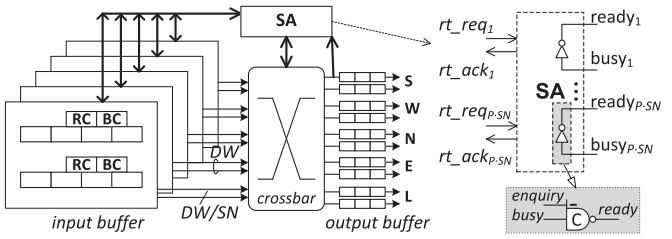


Fig. 16. SDM router structure [9].

produces a tail flag (*eopErr*), replacing the original *eop* wire connected to the front of the input buffer. This replacement is safe, since the pipeline has been confirmed deadlocked. The *cia* acknowledges that the fake tail flit is accepted by the output. Then, *acken* goes high to finish the tail transmission and release the reserved path.

1) If the input is deadlocked at Transmission (*acken*-), no tail flit has yet arrived at the *Stg* $d+1$ in Fig. 13. A fake tail flit needs to be produced. The added AND gate at the input of the router (Fig. 13) enforces a low *ack* (*cia*−). Thus, this allows the tail generator to produce a high *eopErr*, which is equivalent to the *eop*+ transition in the STG [Fig. 14(b)]. A fake tail flit is thus generated. This tail flit is withdrawn (*eopErr*−) when it is accepted by the output (*cia*+ and *acken*+).

2) Otherwise if the input is deadlocked at Release, BC gets stuck at *acken*+ [Fig. 14(b)] constantly waiting for the withdrawal of the tail flit (*eop*-). The *cia* is also kept high, constantly waiting for the withdrawal of the tail flit. When the permanent fault is confirmed (*permConf*+), the tail generator outputs a low *eopErr* (a fake withdrawal of the tail flit). Consequently, all downstream routers are sequentially released.

For all cases, after the *Release* operation, the input buffer is eventually halted at the Setup state 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 resumed available. The fault-caused physical-layer deadlock is eliminated.

B. Faulty Link Isolation

For a system with permanent faults, the usual recovery method is to block the defective component and use redundancy to compensate for its function. When it comes to NoCs, fault-tolerant adaptive routings [24], [33] could be used to bypass the defective link. This research uses spatial division multiplexing (SDM) [9], [15] to implement recovery, which does not rely on adaptive routing but is compatible with it.

Employing SDM to a generic *DW* bits wormhole NoC (Fig. 16), the link between two SDM routers can be divided into *SN* sublinks, each of which is *DW*/*SN* bits wide. The crossbar of a *P*-port SDM router becomes $P{\cdot}SN \times P{\cdot}SN \times (DW/SN)$ bits wide. An SDM link with *SN* sublinks transfers *SN* packets in parallel, but each requires a longer latency to reach its destination due to the narrowed bandwidth. SDM has been used to improve the network overall performance as it was proved that SDM NoCs have advantages in energy consumption, area overhead, and flexibility [9], [15]. Since a link is physically divided into multiple sublinks, a fault affects only one sublink, while leaves other sublinks in the same link untouched. This inherent redundancy supports gradual performance degradation from permanent faults. By reconfiguring the SA [9], [34] to block the defective sublink and direct packets to the remaining fault-free sublinks, a fault link retains its function with a reduced aggregated bandwidth.

As Fig. 16 shows, the availability of each output sublink is indicated by a high *ready* while *ready*− denotes that the sublink has been allocated. Thus, a faulty sublink can be isolated by locking its *ready* signal to low. To achieve this, the inverter generating *ready* is replaced with an asymmetric C-element whose *enquiry* pin is controlled by the deadlock detector. This *enquiry* signal is driven to high during ENQUIRY and CONFIRM states (Figs. 10 and 12). Since a fault-caused deadlock is confirmed only when the sublink is allocated, *ready* becomes low before CONFIRM. The high *enquiry*, therefore, prohibits *ready* to return high during CONFIRM, consequently avoiding any new packet from being allocated to the sublink even after the deadlocked resources are drained (*busy*-). For a permanent fault, the state is stuck at CONFIRM so that the faulty sublink is safely isolated.

VII. Technical Issues

In a QDI NoC, a fault can cause a physical-layer deadlock, which not only blocks the packet that utilizes the faulty link but also stalls all packets that wait the resources occupied by the deadlocked packet. Such stall can spread out the whole network very quickly with heavy traffic, which leads to significant performance drop. This is much more harmful than merely causing data errors and packet loss. It is crucial to detect this type of deadlock and try to recover from it.

To detect faults, deadlock detection circuits and two extra wires (*start* and *enquiry*, Fig. 10) are added to network data path. To implement the recovery, the deadlock confirmation signal (*permConf*) generated at the output buffer needs to be transmitted to the input buffer of the postfault router, leading to three redundant wires added for each sublink. These circuits are not protected. The occurrence of permanent faults is strongly related to the activity on individual circuit segments [26], [35]. Since the extra circuits added for fault tolerance operate at a far lower frequency than the data interconnects, they are less prone to permanent faults.

The deadlock detection circuit needs a clock (T_{clk}) to drive its state machine. There is no requirement on its skew, jitter, and frequency but a slow clock is usually preferred for small detection energy. Designers can use any clock sources, such as the clock of the local PE or a slow global clock, to keep the minimal impact on the global clock tree. The timeout period (T_t) decides the detection delay (T_{det}). It takes two to four timeout periods to confirm fault-caused deadlock ($2T_t \leq T_{det} \leq 4T_t$). The lower bound is achieved when the deadlock forms just before the state machine transits to START. The upper bound is reached when the deadlock occurs before IDLE. Considering the timeout period, the following hold.

1) If only a permanent fault or only deadlock detection is considered, the timeout should be longer than the clock

period ($T_t > T_{clk}$) and the latency of transmitting one flit through one pipeline stage (usually several nanoseconds, to allow a stable sample from transition detectors).

2) If the network recovery is implemented and intermittent faults are considered, the timeout period should be longer than the transmission latency for the longest packet (defined by higher layers) traveling through a single router to ensure that all recovery operations are finished in one timeout period. Thus, the permanent fault will not be mistaken into an intermittent one (resetting the state machine to IDLE directly) during the recovery process where signal transitions happen. The deadlock detector can safely transit to the initial state after the recovery.

This paper divides the data path into link and router pieces to implement the deadlock detection (Fig. 10). *Stage d* at the input buffer of the router is the boundary of the two protected pieces, where two adjacent *ack* signals are sampled to support deadlock detection. Currently, this boundary is not fully protected. Its protection can be implemented through overlapping the link and router pieces, or sampling more *ack* signals next to the boundary, so as to provide a full protection of the data path. In addition, this paper presents the recovery from deadlock caused by a permanently faulty link. Faulty routers are detected but recovering from deadlock caused by a faulty router is difficult. One potential solution is to carefully isolate the faulty router and then use adaptive routings to detour traffic from it. These are left as future work. Nevertheless, this is the first QDI NoC that tolerates permanent faulty links to our best knowledge.

VIII. Area, Energy, and Speed Evaluation

A SystemC/Verilog mixed environment was used to evaluate the performance of a 4×4 2-D-mesh network. Asynchronous cells, such as C-elements and Mutex Elements [3], were built using standard cells. All routers were implemented in gate-level Verilog and synthesized using the UMC 130-*nm* standard cell library. PEs, network interfaces (NIs), and the interconnection between routers were implemented in SystemC. The network was injected with random uniform traffic. Each packet is 64 B long and divided into flits matching the width of links (or sublinks for SDM). PEs and the network are connected by NIs, which convert packets into flits complying with the four-phase 1-of-4 handshake protocol and reassemble received flits into packets.

Four levels of protection have been evaluated, including *unprotected*, *protected links with fault detection*, *protected data path with fault detection*, and *protected links with fault recovery*. Table I reveals the performance of the SDM router and the whole NoC under various levels of protection. The overhead of adding protection on the baseline of unprotected design is noted in parentheses. Each link is *DW* bits wide and divided into *SN* sublinks.

The area overhead is small. Detecting faulty interrouter links in 32-bit SDM routers ($DW = 32$ and $SN = 2$) incurs an area increase of 2%. Adding fault detection for the intrarouter data path as well would increase the area by 4.8%.

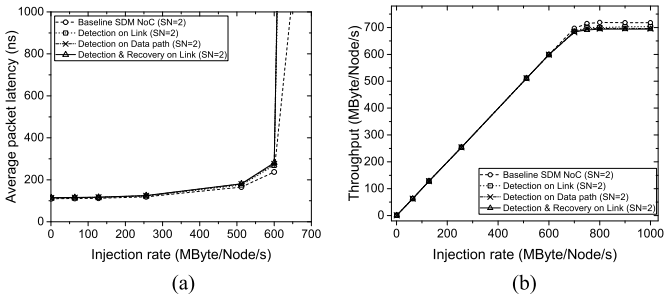
TABLE I
Comparison of NoCs With Incremental Fault Tolerance

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 (4.1%)
	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%)

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%)

TABLE II
Area of Main Components in Protected SDM Routers

Area (μm^2)	Switch Allocator	Crossbar	Input buffers	Output buffers	Total
Protected data path with fault detection	6,408	21,678	23,030	23,409	74,525
Protected links with fault recovery	6,408	20,758	23,828	23,044	74,038



(a) (b)

Fig. 17. Throughput and latency of 32-bit NoCs with different injection loads. (a) Average network latency. (b) Network throughput.

Alternatively supporting recovery from faulty links needs 4.1% extra area. Applying similar protection to 64-bit SDM routers ($DW = 64$ and SN=2) leads to area overhead of 1.6%, 4.9%, and 4.8%, respectively, for the three different levels. Table II gives an overview of the area of main components in two classes of protected SDM routers ($DW = 32$ and $SN = 2$). The SA is the main control logic, while the input/output buffers and crossbars belong to the data path. Therefore, it can be generally estimated that: 1) for the SDM router with fault detection on the whole data path, about 90% of the router is protected by the detection circuit and 2) for the router with fault detection and recovery on links, about 60% of the logic is protected.

The protection circuit has marginal impact on the speed performance. Fig. 17 reveals the average packet transmission latency and the network throughput of 32-bit SDM NoCs injected with gradually increased load. When the network is lightly loaded, there is little congestion so that the packet transmission latency remains low and the network throughput is proportional to the injection rate. As the injection rate

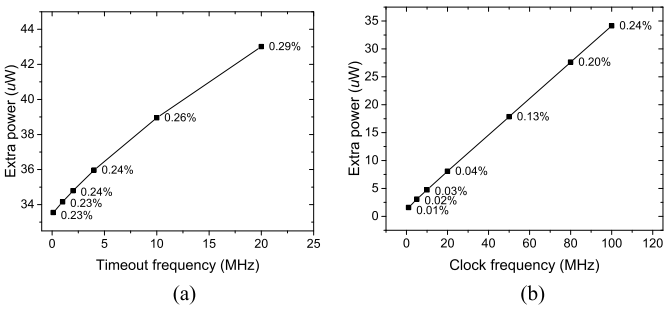


Fig. 18. Extra power consumption with various timeout and clock frequencies. (a) Timeout. (b) Clock.

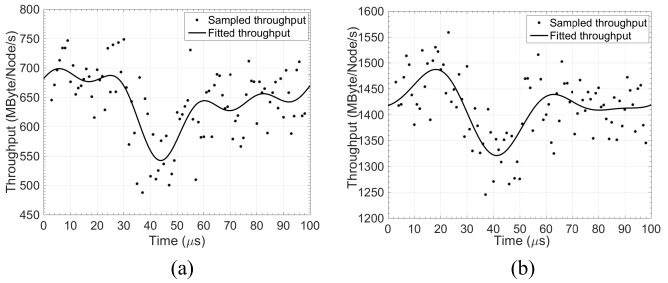


Fig. 19. Throughput of SDM NoCs with a stuck-at-1 fault. (a) 32-bit SDM NoC (SN = 2). (b) 64-bit SDM NoC (SN = 4).

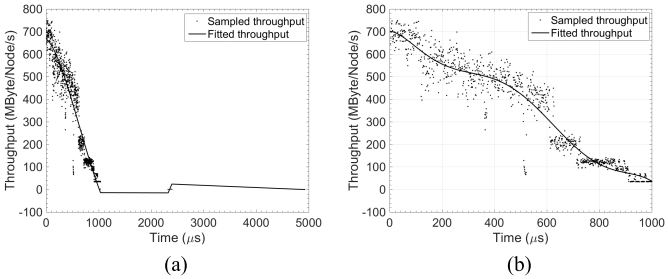


Fig. 20. Throughput of SDM NoCs with multiple faults injected. (a) 5 ms. (b) 1 ms.

increases, the network becomes congested. Although the network throughput remains roughly proportional to the injection rate until the network is saturated, the packet transmission latency rises dramatically long before saturation. According to the figures in Table I, adding fault protection to the 32-bit NoCs prolongs the minimal packet transmission latency (latency in an unloaded network) by no more than 4.5% and reduces the throughput by less than 3.6%. The same for 64-bit NoCs are 2.5% for latency and 2.4% for throughput, which are even less noticeable.

The energy performance is evaluated in networks that are near saturated. The results are presented in the form of energy per transmitting 1 bit of data through a single router. For the protected network, the deadlock detection circuit uses a 100-MHz clock to form a 1-μs timeout period (1 MHz). For the 32-bit SDM routers ($DW = 32$ and $SN = 2$), detecting faulty links consumes 0.7% more energy than the baseline case. It increases to 1.8% if faulty intrarouter paths are also detected. Adding recovery from faulty links incurs 1.3% extra energy. As for the 64-bit SDM routers ($DW = 64$ and $SN = 4$), the similar energy overhead increases to 1.7%, 1.8%, and 2.2%, respectively, for the cases of detecting faulty links, detecting data path, and recovering from faulty links. Nonetheless, the energy overhead of supporting fault protection remains marginal.

The power consumed by the synchronous deadlock detection circuit (the timeout counter and the state machine) has been estimated in a saturated 32-bit NoC. Fig. 18(a) shows the power of the synchronous circuit driven by a fixed 100-MHz clock with increasing timeout frequencies, while Fig. 18(b) reveals the power of different clock frequencies with a fixed timeout frequency of 1 MHz. The power of the synchronous circuit is generally linear with both the timeout and the clock frequency. A slow clock with a long timeout period should be used for low power consumption, which also reduces the transitions on the protection circuit.

IX. Fault Tolerance Evaluation and Comparison

Random 1-bit faults are injected to random locations on the data path of QDI NoCs, including links, buffers, and crossbars, to evaluate the fault-tolerance capability. Test results show that all the faults occurring on the protected region of NoC data path and causing physical-layer deadlock have been detected and precisely located. It is also shown that network function successfully recovers from faulty links using the fault recovery design and an isolated link can be reused if the fault is intermittent.

The dynamic deadlock detection and recovery process are shown in Fig. 19. The clock frequency is set to 100 MHz. Faults are injected to a saturated network, where they exert the maximum damage. To differentiate the detection and recovery process, the timeout period is set "long enough" (10 μs), which is far longer than the packet latency through a router (less than 70 *ns*). A fault is inserted to the *West* input link to router (3,3) at 30 μs, leading to a steep decrease of the network throughput. For an unprotected network, it eventually fails to work as the deadlock spreads. As for the protected network [an SDM NoC using two 16-bit sublinks, $SN = 2$ in Fig. 19(a)], initially the throughput drops sharply to 550 *MB/Node/s* at around 44 μs but starts to recover after the deadlock is detected at around 50 μs. After the recovery process, the throughput bounces back to 650 *MB/Node/s*, which is only 7.2% less than the prefault throughput of 694 *MB/Node/s*. Fig. 19(b) shows the same fault on a 64-bit NoC. The network throughput drops from 1480 *MB/Node/s* to 1,320 *MB/Node/s* rapidly and then returns to 1,410 *MB/Node/s* after the network is recovered. This clearly demonstrates the effectiveness of the proposed techniques.

The fault recovery design can tolerate multiple faults until all sublinks of a link are isolated. A protected 32-bit SDM NoC (SN=2) is injected with a random fault every 50 μs. The clock and timeout frequencies are set to 100 and 0.2 MHz, respectively. Fig. 20(a) shows the network throughput drops to around zero after the first 20 faults. Zooming in to the first 1 *ms*, Fig. 20(b) shows the gradually decreasing throughput. Although it still hangs around 500 *MB/Node/s* after eight faults, it drops quickly when more faults are injected due to the loss of link connections.

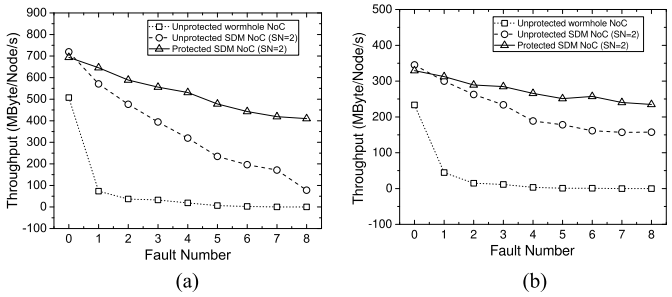


Fig. 21. Throughputs of 32-bit NoCs with an increasing number of faults. (a) 4×4 mesh. (b) 8×8 mesh.

Fig. 21 compares the throughputs of unprotected wormhole NoCs, unprotected SDM NoCs ($SN = 2$), and protected SDM NoCs ($SN = 2$, with fault detection and recovery on links). The throughput results are averaged from 50 independent runs of injecting a fixed number of faults to random interrouter links. It can be found that, in both the 4×4 and 8×8 mesh networks, the throughput of unprotected wormhole NoCs drops dramatically when one fault occurs. When two faults are inserted, the final throughput is close to (or equal) zero. Fault-free SDM networks present around 50% throughput improvement against wormhole. SDM NoCs are also robust, because each link has multiple independent sublinks and a single fault can break only one of them. The network remains working reasonably well with a small number of faults. However, as the number of faults increases, the resulting physical-layer deadlocks reserve more network resources and spread. The network throughput drops significantly and the network is paralyzed. The unprotected SDM NoCs show an approximate linear throughput drop with the number of faults. Using the proposed fault detection and recovery techniques to protect the interrouter links, the deadlocked network resource caused by a permanent fault is limited to one unidirectional link, avoiding the spread of the deadlock. As a result, the protected SDM NoC shows strong tolerance to permanent link faults. When there is only one fault, the protected SDM NoCs achieve 13.2% (4×4) or 4.2% (8×8) higher throughput than unprotected NoCs. When the number of faults increases to 8, the throughput of the protected SDM NoCs is $\sim$400% (4×4) and $\sim$50% (8×8) higher than unprotected ones. Comparing with fault-free networks, throughput degrades 45% (4×4) or 28.7% (8×8) for the protected SDM NoCs and 89.2% (4×4) or 54.4% (8×8) for the unprotected SDM NoCs, while the wormhole NoCs are dead. The proposed fault detection and recovery techniques improve fault tolerance significantly.

X. Comparison With Related Work

Existing research on permanent faults is mostly on synchronous circuits and NoCs [32]. There is rarely any research on managing permanent faults in QDI NoCs. The management of permanent faults on synchronous NoCs usually relies on transient-fault-tolerant techniques, using accumulated error syndromes to determine fault types. Similar to conventional transient-fault-tolerant techniques for synchronous circuits, the protection of asynchronous circuits from transient faults can be achieved by using information redundancy [12], [36]–[41], physical redundancy [25], [42]–[46], or temporal redundancy techniques [47], [48]. Although these techniques seem promising, they are not able to easily identify the fault location, making them difficult to be adopted for protecting QDI links or QDI NoCs. More seriously, most existing transient-fault-tolerant techniques cannot avoid or recover from the physical-layer deadlock caused by permanent faults in QDI NoCs, as described in Section II-B.

Many existing asynchronous NoCs use bundled-data routers connected by QDI links, such as four-phase bundled-data routers with four-phase 1-of-n links [4], [49] or four-phase bundled-data routers with two-phase 1-of-n links [50]. These NoCs are not fully QDI, because QDI links are isolated by bundled-data routers. Only the NoCs using both QDI routers and QDI links [9], [51]–[53] are fully QDI NoCs, which tolerate delay variations on both routers and links. This paper aims to provide fault tolerance to the fully QDI NoCs.

Most of existing QDI NoCs have no or very limited fault-tolerant capabilities. The ANoC [53] uses a design-for-test design to detect off-line stuck-at faults, which is unsuitable for tolerating on-line faults. No recovery mechanism is proposed. The transient-fault-tolerance of interchip communication was studied on the SpiNNaker system [30], [54]. A transient-fault-tolerant phase converter (between two-phase and four-phase) at the chip interface was designed to reduce transient-fault-caused deadlocks. However, the design cannot be used to protect four-phase 1-of-n QDI NoCs from permanent faults. An SDM router [9] is able to drop packets at the input buffer when the decoded routing request signal is detected invalid. The provided fault tolerance is very limited.

Some study has been done for fault-caused deadlock in asynchronous circuits [25], [53], [55]–[57] but not in the context of NoCs. Different from this research, Peng [58] proposed to add appropriate redundant logic at transistor level to force asynchronous circuits to deadlock in the presence of faults. Using a timeout mechanism to detect the deadlock and trigger the reconfiguration of the faulty module, the faulty circuit function can be resumed. A timeout mechanism using delay lines has been proposed to detect faults on the level-encoded 2-phase dual-rail [59] encoded QDI links of a self-timed NoC [33], [60], whose routers, however, are bundled-data designs. Since self-timed pipelines are used inside the router and they do not latch incomplete data, the fault-caused partial data (which leads to large skew) are isolated in the QDI interrouter link, making the fault locating possible. In a pure QDI NoC studied in this paper, this partial data will propagate to all downstream stages as long as they are ready (Section III), which will cause multiple deadlocks reported along the deadlocked packet path if their technique is used, failing to locate the fault position. Their technique is incompatible with fully QDI NoCs. In terms of recovery, conventional techniques, including spare wires replacement [37], splitting transmission [61], and fault-tolerant routings [24], [33], have been used to recover the network from permanent faults. Without relying on these techniques, this paper proposes techniques to remove the fault-caused deadlock and makes use of SDM to

recover the network function, which can be used along with those conventional techniques to implement network recovery at different layers.

In addition, timeout-based methods have been used to detect the network-layer deadlock [29]. Theoretically, they generally detect any deadlocks caused by cyclic dependence or faults. However, the deadlock detection technique proposed in this paper not only detects the deadlock, more importantly, it locates the faulty pipeline stages, allowing for accurate isolation and network recovery. Resolving network-layer deadlocks is out of the scope of this paper.

Our previous work has explored the impact of permanent faults on link wires and implemented the online detection of deadlock due to permanently faulty links [27], [62]. Recovery of the network function has also been studied [62]. As an extension, this paper details the fault model considering a complete generic asynchronous pipeline (including latches and completion detectors). All possible deadlock scenarios due to different faults on the pipelined interrouter links and router data paths are discussed. A general fault detection strategy is proposed. Compared with previous work, this paper originally explores the detection of permanent faults on router logic, so that permanent faults occurring on data paths of NoCs can be detected. Detailed experiments are implemented to evaluate both the performance and the fault-tolerance capability of QDI NoCs using different protection strategies.

To the best of our knowledge, this is the first research providing a thorough analysis on the impact of permanent faults exerting on the data paths of QDI NoCs and a solution to accurately locate the faulty pipeline stages. This paper has also partially resolved the issue by recovering faulty NoCs from faulty link paths, which provides a promising foundation for future fully fault-tolerant QDI NoCs.

XI. Conclusion

Caused by the aging process, a permanent fault striking on a QDI circuit can break the handshake protocol and then trigger a physical-layer deadlock. This deadlock is different from network-layer ones provoked by cyclically dependent packets. Conventional deadlock management techniques are incapable of correctly detecting and resolving these fault-caused deadlocks. This paper studied the impact of faults on QDI NoC data paths. A new detection technique can precisely locate the faulty link and router logic on the data path triggering the deadlock. Utilizing SDM, a fine-grained recovery strategy was proposed to resume a deadlocked network from a permanent fault. In the recovery process, faulty links are precisely isolated to achieve a graceful performance degradation with $\sim$60% of the logic being protected. Detailed experimental results are given.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. IEEE DAC*, Jun. 2001, pp. 684–689.

[2] S. R. Vangal *et al.*, "An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[3] J. Sparsø and S. B. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*. Norwell, MA, USA: Kluwer, 2001.

[4] R. Dobkin, R. Ginosar, and A. Kolodny, "QNoC asynchronous router," *Integr. VLSI J.*, vol. 42, no. 2, pp. 103–115, 2009.

[5] F. Clermidy *et al.*, "A 477 mW NoC-based digital baseband for MIMO 4G SDR," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2010, pp. 278–279.

[6] D. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 1984.

[7] E. Beigne, F. Clermidy, S. Miermont, and P. Vivet, "Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC," in *Proc. NoCS*, Apr. 2008, pp. 129–138.

[8] S. Werner, J. Navaridas, and M. Luján, "A survey on design approaches to circumvent permanent faults in Networks-on-Chip," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 59:1–59:36, 2016.

[9] W. Song, "Spatial parallelism in the routers of asynchronous on-chip networks," Ph.D. dissertation, School Comput. Sci., Univ. Manchester, Manchester, U.K., 2011.

[10] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul./Aug. 2003.

[11] G. Zhang, W. Song, J. D. Garside, J. Navaridas, and Z. Wang, "Transient fault tolerant QDI interconnects using redundant check code," in *Proc. DSD*, Sep. 2013, pp. 3–10.

[12] G. Zhang, W. Song, J. D. Garside, J. Navaridas, and Z. Wang, "Protecting QDI interconnects from transient faults using delay-insensitive redundant check codes," *Microprocess. Microsyst.*, vol. 38, no. 8, pp. 826–842, 2014.

[13] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, no. 5, pp. 547–553, May 1987.

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

[15] 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 Trans. Comput.*, vol. 57, no. 9, pp. 1182–1195, Sep. 2008.

[16] R. Riedlinger *et al.*, "A 32 nm, 3.1 billion transistor, 12 wide issue Itanium processor for mission-critical servers," *IEEE J. Solid-State Circuits*, vol. 47, no. 1, pp. 177–193, Jan. 2012.

[17] T. Verhoeff, "Delay-insensitive codes—An overview," *Distrib. Comput.*, vol. 3, no. 1, pp. 1–8, 1988.

[18] W. Song and D. Edwards, "A low latency wormhole router for asynchronous on-chip networks," in *Proc. ASP-DAC*, 2010, pp. 437–443.

[19] F. A. Bower, D. J. Sorin, and S. Ozev, "A mechanism for online diagnosis of hard faults in microprocessors," in *Proc. MICRO*, 2005, pp. 197–208.

[20] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Runtime logic and interconnect fault recovery on diverse FPGA architectures," in *Proc. Military Aerosp. Appl. Program. Devices Technol. Int. Conf.*, 1999, pp. 1–8.

[21] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *Proc. DSN*, Jun. 2004, pp. 177–186.

[22] S. Al-Arian and D. Agrawal, "Physical failures and fault models of CMOS circuits," *IEEE Trans. Circuits Syst.*, vol. CS-34, no. 3, pp. 269–279, Mar. 1987.

[23] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu, "Self-adaptive system for addressing permanent errors in on-chip interconnects," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 4, pp. 527–540, Apr. 2010.

[24] 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 Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 1053–1066, Jun. 2013.

[25] W. Jang and A. J. Martin, "SEU-tolerant QDI circuits," in *Proc. ASYNC*, 2005, pp. 156–165.

[26] R. Aitken, G. Fey, Z. T. Kalbarczyk, F. Reichenbach, and M. S. Reorda, "Reliability analysis reloaded: How will we survive?" in *Proc. DATE*, 2013, pp. 358–367.

[27] 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 *Proc. GLSVLSI*, 2014, pp. 211–216.

[28] 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 *Proc. DATE*, Mar. 2011, pp. 1–6.

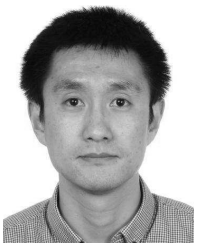
[29] S. Lee, "A deadlock detection mechanism for true fully adaptive routing in regular wormhole networks," *Comput. Commun.*, vol. 30, no. 8, pp. 1826–1840, Jun. 2007.

[30] Y. Shi, S. B. Furber, J. Garside, and L. A. Plana, "Fault tolerant delay insensitive inter-chip communication," in *Proc. ASYNC*, May 2009, pp. 77–84.

[31] J. Sparsø, "Asynchronous design of Networks-on-Chip," in *Proc. Norchip*, 2007, pp. 1–4.
[32] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in Networks-on-Chip," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 8:1–8:38, Jul. 2013.
[33] M. Imai and T. Yoneda, "Improving dependability and performance of fully asynchronous on-chip networks," in *Proc. ASYNC*, Apr. 2011, pp. 65–76.
[34] S. Golubcovs, D. Shang, F. Xia, A. Mokhov, and A. Yakovlev, "Modular approach to multi-resource arbiter design," in *Proc. ASYNC*, 2009, pp. 107–116.
[35] M. Bohr and K. Mistry, "Intel's revolutionary 22 nm transistor technology," *Rob Willoner Innov.*, 2011. [Online]. Available: http://download.intel.com/newsroom/kits/22nm/pdfs/22nm-Details_Presentation.pdf
[36] F.-C. Cheng and S.-L. Ho, "Efficient systematic error-correcting codes for semi-delay-insensitive data transmission," in *Proc. ICCD*, Sep. 2001, pp. 24–29.
[37] T. Lehtonen, P. Liljeberg, and J. Plosila, "Online reconfigurable self-timed links for fault tolerant NoC," *VLSI Des.*, vol. 2007, May 2007, Art. no. 94676.
[38] J. Pontes, "Soft error mitigation in asynchronous networks on chip," Ph.D. dissertation, Faculty Inform. Pontifical Catholic Univ. Rio Grande do Sul, Porto Alegre, Brazil, 2012.
[39] M. Y. Agyekum and S. M. Nowick, "Error-correcting unordered codes and hardware support for robust asynchronous global communication," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 1, pp. 75–88, Jun. 2012.
[40] J. Pontes, N. Calazans, and P. Vivet, "Parity check for m-of-n delay insensitive codes," in *Proc. IOLTS*, Jul. 2013, pp. 157–162.
[41] J. Lechner, A. Steininger, and F. Huemer, "Methods for analysing and improving the fault resilience of delay-insensitive codes," in *Proc. ICCD*, Oct. 2015, pp. 519–526.
[42] Y. Monnet, M. Renaudin, and R. Leveugle, "Designing resistant circuits against malicious faults injection using asynchronous logic," *IEEE Trans. Comput.*, vol. 55, no. 9, pp. 1104–1115, Sep. 2006.
[43] S. Almukhaizim, F. Shi, E. Love, and Y. Makris, "Soft-error tolerance and mitigation in asynchronous burst-mode circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 7, pp. 869–882, Jul. 2009.
[44] W. Friesenbichler and A. Steininger, "Soft error tolerant asynchronous circuits based on dual redundant four state logic," in *Proc. DSD*, 2009, pp. 100–107.
[45] W. Friesenbichler, T. Panhofer, and M. Delvai, "A comprehensive approach for soft error tolerant four state logic," in *Proc. DDECS*, Apr. 2009, pp. 214–217.
[46] N. Julai, A. Yakovlev, and A. Bystrov, "Error detection and correction of Single Event Upset (SEU) tolerant latch," in *Proc. IOLTS*, 2012, pp. 1–6.
[47] S. Ogg, B. Al-Hashimi, and A. Yakovlev, "Asynchronous transient resilient links for NoC," in *Proc. CODES+ISSS*, 2008, pp. 209–214.
[48] J. Pontes, N. Calazans, and P. Vivet, "Adding temporal redundancy to Delay Insensitive codes to mitigate single event effects," in *Proc. ASYNC*, 2012, pp. 142–149.
[49] A. Sheibanyrad and A. Greiner, "Two efficient synchronous asynchronous converters well-suited for Networks-on-Chip in GALS architectures," *Integr. VLSI J.*, vol. 41, no. 1, pp. 17–26, 2008.
[50] T. Bjerregaard and J. Sparso, "Implementation of guaranteed services in the MANGO clockless Network-on-Chip," *IEE Proc.-Comput. Digit. Techn.*, vol. 153, no. 4, pp. 217–229, Jul. 2006.
[51] J. Bainbridge and S. Furber, "Chain: A delay-insensitive chip area interconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, Sep./Oct. 2002.
[52] T. Felicijan and S. B. Furber, "An asynchronous on-chip network router with quality-of-service (QoS) support," in *Proc. SOCC*, 2004, pp. 274–277.
[53] Y. Thonnart, P. Vivet, and F. Clermidy, "A fully-asynchronous low-power framework for GALS NoC integration," in *Proc. DATE*, 2010, pp. 33–38.
[54] Y. Shi, "Fault-tolerant delay-insensitive communication," Ph.D. dissertation, School Comput. Sci., Univ. Manchester, Manchester, U.K., 2010.
[55] A. J. Martin and P. J. Hazewindus, "Testing delay-insensitive circuits," in *Proc. Univ. California/Santa Cruz Conf. Adv. Res. VLSI*, 1991, pp. 118–132.
[56] I. David, R. Ginosar, and M. Yoeli, "Self-timed is self-checking," *J. Electron. Test.*, vol. 6, no. 2, pp. 219–228, 1995.
[57] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing asynchronous circuits: A survey," *Integr. VLSI J*, vol. 19, no. 3, pp. 111–131, 1995.
[58] S. Peng, "Implementing self-healing behavior in quasi delay-insensitive circuits," Ph.D. dissertation, Comput. Syst. Lab., Cornell Univ., Ithaca, NY, USA, 2006.
[59] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (LEDR)," in *Proc. Univ. California/Santa Cruz Conf. Adv. Res. VLSI*, 1991, pp. 55–70.
[60] T. Yoneda, M. Imai, N. Onizawa, A. Matsumoto, and T. Hanyu, "Multi-chip NoCs for automotive applications," in *Proc. PRDC*, 2012, pp. 105–110.
[61] M. Zhang, Q. Yu, and P. Ampadu, "Fine-grained splitting methods to address permanent errors in Network-on-Chip links," in *Proc. ISCAS*, 2012, pp. 2717–2720.
[62] G. Zhang, W. Song, J. Garside, J. Navaridas, and Z. Wang, "An asynchronous SDM Network-on-Chip tolerating permanent faults," in *Proc. ASYNC*, May 2014, pp. 9–16.

Guangda Zhang received the Ph.D. degree in computer science from The University of Manchester, Manchester, U.K., in 2016, for work in fault-tolerant techniques for asynchronous networks-on-chip.

His current research interests include fault tolerance, asynchronous circuits, on-chip communication, computer architecture, neural networks, and air traffic management.



Wei Song received the Ph.D. degree in computer science from The University of Manchester, Manchester, U.K., in 2011, and spent three years afterward researching areas related to asynchronous circuit.

He is currently a Research Associate with the Computer Laboratory, University of Cambridge, Cambridge, U.K. His current research interests include computer architecture, high-speed asynchronous circuit, on-chip communication, system-on-a-chip, and reconfigurable logic.

Jim Garside received the Ph.D. degree in computer science from The University of Manchester, Manchester, U.K., in 1987, for work in signal processing architecture.

Post-doctoral work on parallel processing systems based on Inmos Transputers was followed by a spell in industry writing air traffic control software. Returning to academia gave an opportunity for integrated circuit design work, dominated by design and construction work on asynchronous microprocessors in the 1990s. He has been involved with dynamic hardware compilation, GALS interconnection, and the development of the hardware and software of the SpiNNaker neural network simulator.

Javier Navaridas received the Ph.D. degree in computer engineering from the University of the Basque Country, Leioa, Spain, in 2009, which was rewarded with an Extraordinary Doctorate Award (Top 5%).

He joined The University of Manchester, Manchester, U.K., with a prestigious Newton Fellowship in 2010, where he is currently a Lecturer in computer architecture. His current research interests include interconnection networks for parallel and distributed systems, networks on chip for SoCs and multiprocessors, and characterization of applications behavior.

Zhiying Wang received the Ph.D. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 1989.

He is currently a Professor with the National University of Defense Technology. His current research projects include asynchronous microprocessor design, nanotechnology circuits and systems based on optoelectronic technology, and virtual computer system. His current research interests include computer architecture, asynchronous circuit, computer security, VLSI design, and multicore memory system.