2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip

A Novel Flit Serialization Strategy to Utilize Partially Faulty Links in Networks-on-Chip

Changlin Chen*, Ye Lu†, Sorin D. Cotofana*
*Computer Science and Engineering, Software and Computer Technology
Delft University of Technology, Delft, the Netherlands
Email: {c.chen-2, S.D.Cotofana}@tudelft.nl
†The institute of Electronic, Communication, and Information Technology (ECIT)
Queen's University of Belfast, Northen Ireland, UK
Email: ylu10@qub.ac.uk

***Abstract*—Aggressive MOS transistor size scaling substantially increase the probability of faults in NoC links due to manufacturing defects, process variations, and chip wire-out effects. Strategies have been proposed to tolerate faulty wires by replacing them with spare ones or by partially using the defective links. However, these strategies either suffer from high area and power overheads, or significantly increase the average network latency. In this paper, we propose a novel flit serialization method, which divides the links and flits into several sections, and serializes flit sections of adjacent flits to transmit them on all available fault-free link sections to avoid the complete waste of defective links bandwidth. Experimental results indicate that our method reduces the latency overhead significantly and enables graceful performance degradation, when compared with related partially faulty link usage proposals, and saves area and power overheads by up to 29% and 43.1%, respectively, when compared with spare wire replacement methods.**

***Keywords*-Networks-on-Chip; partially faulty link usage; permanent error; flit serialization;**

I. Introduction

While the aggressive transistor size shrinking improves the chip capability, it also makes the manufacturing yield and chip dependability increasingly serious concerns. Links in Networks-on-Chip (NoC) are becoming more prone to various kinds of failures caused by manufacturing defects [1], chip wear-out effects [2], or Process Parameter Variations (PPV) [3] [4]. As a single permanent fault in a link may degrade the system's performance dramatically or even render the chip useless, defective links need to be tolerated.

The most intuitive way to deal with defective links is making use of fault-tolerant adaptive routing protocols [5] [6]. Defective links are discarded and alternative fault-free routes are chosen by the routing function to forward packets. This ineffective usage of link bandwidth increases packet delivery time if the route is not on the minimal path, and decreases network throughput due to the congestion around the faulty links.

For a given permanent wire fault rate, the fault probability in a defective link is typically low, thus rather than completely discarding a defective link, a more effective approach is to isolate the faulty wires in the defective links and to keep on use the fault-free ones to transmit packets. While wires with small frequency deviation due to PPV can be dealt with by the methods described in [7] or [8], this paper proposes a novel flit serialization method to tolerate permanent faulty wires and to utilize partially faulty links.

In order to achieve the maximum utilization of the link bandwidth, links and flits are divided into several sections, and novel fault-tolerant transmitters and receivers, which are placed inside the output and input ports of NoC routers, respectively, are proposed to efficiently utilize the fault-free sections. Adjacent flits are serialized at the transmitter side to fit the narrowed link and are deserialized at the receiver side. The proposed transmitter and receiver are transparent to the router such that their utilization is not constrained by the router architecture and implementation, nor by the network topology.

The proposed link fault-tolerant architecture is compared with equivalent state of the art solutions described in [9], [10], and [11] in the context of a baseline NoC system. The experimental results indicate that the proposed method has following advantages:

1) The proposed method utilize the remaining link bandwidth more efficiently than other partially faulty link usage strategies and enables graceful performance degradation of the NoC system.
2) When compared with the spare wire replacement method, which has a similar fault-tolerant capability, our approach reduce the area and the dynamic power overheads by up to 29% and 43.1%, respectively.

The rest of the paper is organized as follows. A brief survey of related work is presented in Section II. Architecture and detailed implementation of the proposed transmitter and receiver are described in Section III. Two options to combine our proposal with Error Control Coding (ECC) logic are illustrated in Section IV. Section V presents the simulation result and Section VI concludes the presentation.

978-0-7695-4677-3/12 $26.00 © 2012 IEEE
DOI 10.1109/NOCS.2012.22

II. Related Work

Grecu et al. [1] proposed a simple link fault-tolerant method that uses pre-fabricated spare wires to replace faulty wires due to manufacturing, which can effectively enhance the NoC interconnect yield. However, this approach has the drawback of high silicon cost. Since wires do not scale with transistors, and as the feature size of transistor further decreases, the overhead brought by using spare wires is expected to increase [12]. Lehtonen et al. [13] proposed a cost effective method that divides the link into four sections and provides each section with one spare wire. However, this approach cannot tolerate the case in which more than one faulty wire exist in the same section. To address this drawback, an improved method was proposed in [9], where the 4 spare wires are shared rather than exclusively owned by each section. As the number of cross-points used for sharing spare wires increases with the link width and the count of spare wires, this approach is constrained by the link width and wire error probability. To maintain a fault-free link, more spare wires and cross-points are needed resulting in a wider link and a higher permanent error probability.

Palesi et al. [11] and Lehtonen et al. [9] proposed the method of using flit splitting to tolerate faulty wires. As opposed to the spare wires replacement method discussed above, extra transmission latency is introduced by using the flit splitting method. In this approach, a link is divided into four sections, and fault-free sections can be used to transmit flits. However, the available bandwidth is reduced to half even if only one section is defective, and if faulty wires exist in every section, the link has to be discarded. In the remainder of this paper, we name this method as Simple Flit Quad Splitting (SFQS).

A packet rebuilding/restoring algorithm is proposed by Yu et al. [14] to utilize the links with reduced bandwidth. To be transmitted on the defective link, every flit is split into two parts (a big part and a small part). The first part (the big part) of each flit is transmitted first in the narrowed link, followed by a reassembled flit at the end of the packet containing the rest (the small part) of the flits. The saved fault free wires are used to replace the broken wires. This method can efficiently utilize the remaining link bandwidth. However, because an integral flit can only be rebuilt after the reassembled flit has been received, this method cannot be used in low latency routers with wormhole or Virtual-Cut-Through (VCT) switching technology.

By noticing that the faulty wires are typically evenly distributed, Vitkovskiy et al. [10] introduced a link recovery mechanism named PFLRM, which is mainly comprised of a flit shifter, a de-shifter, and a flit re-assembler. Using this approach, the flits are first rotated at the transmitter side according to the fault vector of the link and then transmitted. At the receiver side, the flits are de-rotated and re-assembled. The number of iterations of shift operation and transmission to transmit a flit correctly is decided by the maximum width of the faulty wire cluster in the link. At least two cycles (one iteration) are needed to transmit a flit successfully even if only a single faulty wire exists in the link. If m broken wires are clustered together, m iterations are required. Although this approach can theoretically be tolerant to a defective link with an arbitrary number of faulty wires, the latency overhead introduced by this approach can be significantly high.

In summary, spare wires can preserve the performance but introduce a high silicon overhead. By contrast, flit splitting and PFLRM have low area overhead but introduce high extra latency. The method proposed in this paper use a novel serialization strategy to utilize the partially faulty links. It significantly reduce the latency, when compared with flit splitting and PFLRM, while maintaining a more reasonable silicon cost, when compared with faulty wire replacement method.

III. The Proposed Architecture

A high-level block diagram of the proposed partially faulty link utilization strategy is depicted in Fig. 1. Similar with the work proposed in [11], we divide the data link into several sections, for example 4 or 8 link sections. Accordingly, flits are divided into the same number of flit sections. A Test Data Generator (TDG) and a Test Error Detector (TED) are used to diagnose the link and generate a fault vector to indicate the faulty link sections. Two transmitter (TX) and receiver (RX) pairs with unidirectional links are used to connect adjacent routers. TX and RX are made aware about the health status of the link via the fault vector. When the link is fault-free, flits can be transmitted according to the normal protocol applicable to a healthy link, bypassing the proposed flit deserialization unit at the receiver side. When faulty wires exist in some link sections, flit sections of adjacent flits are first serialized by the flit serialization unit, and then transmitted via the fault-free link sections. At RX side, the flit deserialization unit deserializes all flit sections to reconstruct the original flits. We note in here that since the number of control lines is much smaller than the number of data lines, they can be protected by Triple Modular Redundancy (TMR) method with a marginal overhead. Moreover, ECC logic (not shown in Fig. 1) can be added before (after) the flit serialization (deserialization) unit to protect data from transient errors.

A. Link Diagnosis

TDG and TED diagnose the link and provide the fault vector to the flit serialization and deserialization modules. Unlike spare wire replacement or PFLRM, which require the precise status of each wire, our method just need link fault vectors at the section level. For example, if the third section of a link (with four sections in total) contains faulty wires, the fault vector of the link is marked as '1101'. Thus

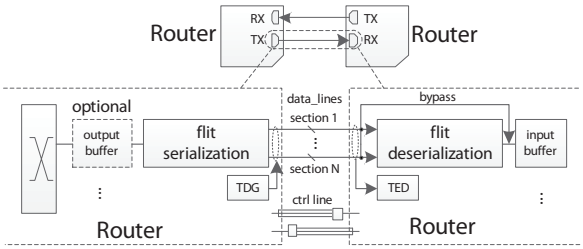


Figure 1. Proposed link fault-tolerant architecture.

the effort to do link diagnosis is reduced, when compared with the complicated bit level link diagnosis methods in [13] and [15].

Link testing is performed periodically and can be triggered by an uncorrectable error detected by the ECC logic. Each section is tested independently while data transmission is still going on other sections. Test vectors are generated by TDG at TX side and are transmitted on the section under test. Received test vectors are analyzed by TED at RX side. This process is repeated on each section until the link fault vector is obtained. Because intermittent errors may have the same syndrome as permanent errors when they happen, sections which are marked as faulty in the previous test are also tested, to prevent situations when vanished intermittent errors are still disabling sections. At the end of the diagnosis process the achieved fault vector is sent to the transmitter and receiver control units.

B. Flit Serialization

The flit serialization unit is presented in Fig. 2. For the sake of simplicity, we assume that both the flit and link are divided into four sections, and the link width is 32-bit. We note that the proposed principle is more general and can be applied to wider links with more sections. To serialize the flits, a link register (*link_reg_TX*) with a width of two flits is used. The link register is divided into eight sections and each section can be read and write independently. The register is designed in this way such that a new flit can be registered in the Least Significant Half (LSH) of the register if there are flit sections in the Most Significant Half (MSH) still waiting to be transmitted, and vice versa. If the link is fault-free, only the MSH of the register is used, acting as a conventional link register. Otherwise, flits are serialized in *link_reg_TX* under the control of the *flit_serialize_ctrl* unit. The serialization process is presented in more detail in Section III-D. Multiplexers are used to select the flit sections to be transmitted. The link sections containing faulty wires that are indicated by the fault vector are not used for data transmission. The signal *flit_type* indicates the presence of a new flit from the crossbar (output buffers are not used).

With a narrowed link, the flit is transmitted at a lower rate on the link than on the crossbar within the router, thus a *data_acceptable* signal is added to disable the transmission

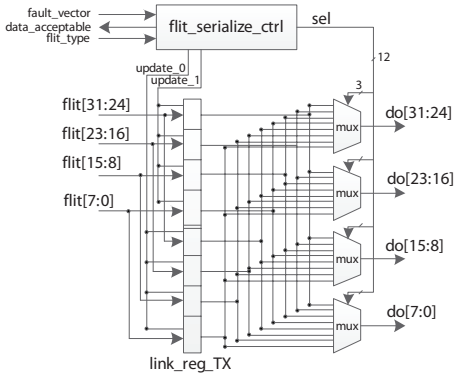


Figure 2. Flit serialization unit - TX.

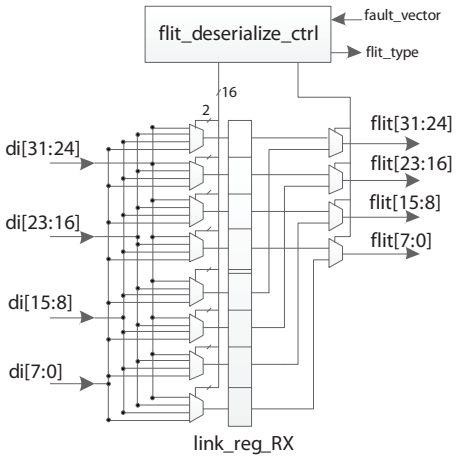


Figure 3. Flit deserialization unit - RX.

requests when the buffer space in the serialization unit is full and cannot accept a new flit. For networks that employ STALL/GO and ACK/NACK flow-control mechanisms, *data_acceptable* can be directly connected to the flow-control signals. For networks that use credit-based flow-control mechanism, a Virtual Channel (VC) can only request for flit transmission when the credit in its downstream VC is larger than zero, so the cancellation of requests does not affect the implementation of the flow-control mechanism.

C. Flit Deserialization

The flit deserialization unit is depicted in Fig. 3. Similar with the flit serialization unit, a link register (*link_reg_RX*) is also employed. Multiplexers are used to select the valid sections from the link under the control of *flit_deserialize_ctrl* unit. Relative sections of the link register are updated when new flit sections are received. When the MSH or LSH of the link register is full, one flit is recovered and can be read out by the router. A *flit_type* signal indicates the availability of a new valid flit.

D. Flit Transmission Process

Fig. 4 graphically depicts the timing diagrams capturing the flit transmission process specific to our method. When the link is fault-free, a flit from the crossbar is loaded into the MSH of *link_reg_TX* and then transmitted to the input buffers of the downstream router directly (see Fig. 4(a)). At the RX side, the flit deserialization unit is bypassed.

Fig. 4(b) presents the situation when one of the sections is affected by faults. At TX side, flit a floats at the output port of the crossbar at the rising edge of $clk1$ and is written into the MSH of *link_reg_TX* at the rising edge of $clk2$. During $clk2$ cycle, the first three sections of the flit a (a_3, a_2, a_1) are transmitted via the three fault-free sections of the link. At the rising edge of $clk3$, flit b is written into the LSH of *link_reg_TX*. Flit sections a_0, b_3, and b_2 are transmitted in the same cycle. The signal *data_acceptable* is set to '0' in $clk3$ such that no new flit may appear at the output port of crossbar in $clk4$. A wait cycle is inserted in this way to wait for the last three sections of flit c to be sent in $clk5$. The signals *high_reg_state* and *low_reg_state* are used to indicate the status of *link_reg_TX* MSH and LSH, respectively. Each signal is asserted once a flit is written into the corresponding register part, and de-asserted in the clock cycle when the data are read out.

At the receiver side (see Fig. 4(c)), flit sections are de-serialized and reassembled into integral flits in *link_reg_RX*. Valid flit sections are selected by input side multiplexers and be written into the correct positions in *link_reg_RX*. Once the MSH or LSH of the register is full, an integral flit is recovered. The signals *flit_1_recovered* and *flit_2_recovered* indicate the availability of recovered flits and control the output side multiplexers to select the corresponding register sections.

In principle, we can use finer section granularity to achieve better performance. However this implies that more and larger multiplexers are required, which may have a negative impact on area and power consumption overheads. The number of sections can be determined via a trade-off process, which takes into consideration the targeted fault-tolerant capability and the available silicon real estate.

E. Link Latency

The link latency when the proposed method is used to continuously transmit different number of flits (*flit_number*) can be expressed as:

$$latency = \left\lceil \frac{section\_number \times flit\_number}{fault\_free\_section\_number} \right\rceil . \quad (1)$$

Where *section_number* is the number of sections in the link. For example, if 10 flits are waiting to be transmitted via a partially faulty link, which has one defective link section, the link latency is 14 cycles when the link is divided into four sections, and 12 cycles when the link is divided into eight sections.

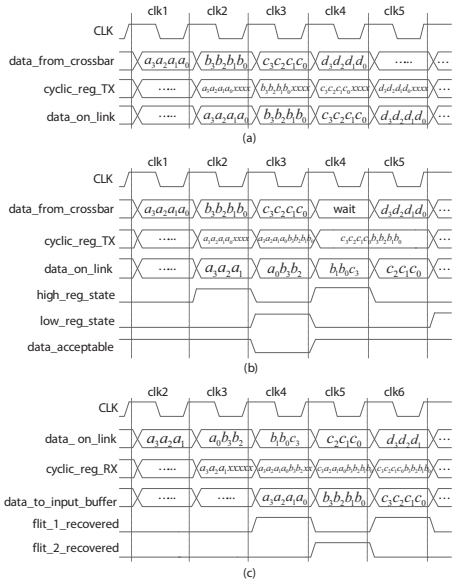


Figure 4. Timing diagram of proposed mechanism (a) Timing diagram for a fault-free link; (b) Transmitter side when one section contains faulty wires; (c) Receiver side when one section contains faulty wires.

By comparison, the link latencies of PFLRM and SFQS are expressed as in (2) and (3), respectively.

$$latency_{pflrm} = (cluster\_size + 1) \times flit\_number \quad (2)$$

$$latency_{sfqs} = \frac{section\_number \times flit\_number}{available\_section\_number} \quad (3)$$

Where *cluster_size* in (2) is the fault cluster size in the link, and *available_section_number* in (3) can have value 1, 2, or 4 according to the number of fault-free sections in the link.

A comparison of link latency overheads when the proposed method, PFLRM, and SFQS are used to continuously transmit flits via a defective link is presented in Table I. The link latency overhead is 0% if the link is fault free. Note that the fault number in the table indicates the fault cluster size in the PFLRM worst scenario, and the number of faulty sections for the proposed method and SFQS. In the best PFLRM scenario, the cluster size is always one. From the Table we can observe that only when more than half of the link sections are broken, the link latency overhead of the proposed method reaches 100%. By comparison, the overheads are at least 100% for both PFLRM and SFQS methods.

If we assume that each wire has the same probability (p_e) to be permanently fault, the probability that the number of faulty wires (N_e) equals k in an n-bit wide link can be calculated using (4).

Table I
LINK LATENCY OVERHEADS WHEN FLITS ARE TRANSMITTED CONTINUOUSLY

Fault number	Proposed 4 sections	Proposed 8 sections	PFLRM Best situation	PFLRM Worst situation	SFQS
0	0%	0%	0%	0%	0%
1	33.3%	14.3%	100%	100%	100%
2	100%	33.3%	100%	200%	100%
3	300%	60.0%	100%	300%	300%
4	–	100%	100%	400%	–
5	–	167%	100%	500%	–
6	–	300%	100%	600%	–
7	–	700%	100%	700%	–

$$P_{N_e=k} = \binom{n}{k} {p_e}^k (1-p_e)^{n-k} \quad (4)$$

In a 32-bit wide link, if $p_e = 10^{-5}$ [9], the probability of $N_e = 2$ is $P_{N_e=2} = 4.96 \times 10^{-8}$, while the probability of $N_e = 4$ is $P_{N_e=2} = 3.60 \times 10^{-16}$. This means that while it is possible to have links with high fault degrees in an NoC, the probability for this to happen in practical implementations is rather low. Thus, in practice, we can expect that most of the links are affected by a low number of faults in the same time, in which case the proposed method is quite effective.

In extreme cases corresponding to large physical defects multiple adjacent wires may get faulty. In such a case PFLRM requires as many iterations as faulty wires to successfully transmit a flit, which results in a large latency overhead. Alternatively, to be able to face such defects, a spare wire replacement method has to make use of multiple spare wires, which results in a large area overhead. In the case of the proposed method such a large defect will most likely affect one or two link sections, thus it can better face such cases at the expense of less area overhead.

IV. ECC Integration

ECC and retransmission are widely used effective methods to tolerate transient errors in communication. In a router which embeds the proposed link fault-tolerant architecture, ECC and retransmission logic can be used either outside or inside of the proposed architecture.

If ECC is used outside of the link fault-tolerant architecture, i.e., the error coding logic is placed before the flit serialization unit and the error decoding logic is placed after the flit deserialization unit (see Fig. 5(a)), a powerful ECC is needed to detect and correct transient errors in, not only the data link, but also in the transmitter and the receiver. Because our proposed architecture is transparent to the rest of the router, ECC logic can be implemented in a conventional way. The detection of transient errors in a flit that cannot be corrected by ECC will trigger a request for retransmission and the link diagnosis process.

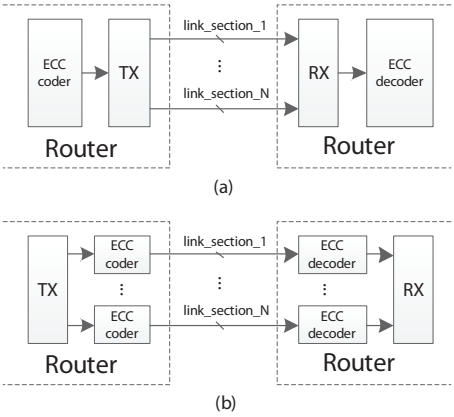


Figure 5. Proposed architecture collaborate with ECC (a) ECC logic before TX and after RX; (b) ECC logic after TX and before RX.

If ECC is used inside of the proposed architecture, i.e., the error coding logic is placed after flit serialization unit and the error decoding logic is placed before the flit deserialization unit (see Fig. 5(b)), only transient errors that might appear on the link are covered. Error coding and decoding logics are added to each link section. If the error in a flit section cannot be corrected by ECC, the flit section is marked as invalid. As a consequence the flit it belongs to is invalidated and the retransmission process is invoked. The drawback of this approach is that the transmitter and receiver cannot be protected by ECC.

V. Performance Evaluation

To put the implications of our link fault-tolerant architecture in a better practical prospective, we evaluate and compare it with other three tightly related proposals presented in [9], [10], and [11], namely spare wire replacement, PFLRM, and SFQS, respectively. To this end, we implemented all these four link fault-tolerant methods at RTL level by using Verilog HDL, and applied them in the context of the NoC platform developed by Lu et al. [16]. The baseline router has 2 pipeline stages: look-ahead Routing Computation (RC) and combined VC/Switch Allocation

Table II
POWER AND AREA OVERHEAD OF DIFFERENT LINK FAULT-TOLERANT METHODS

Link fault-tolerant methods		Dynamic Power (mW)	Leakage Power (mW)	Area (μm^2)
Basic router		18.20 / 0%	0.5269 / 0%	69560 / 0%
Proposed	4 sections	20.81 / 14.3%	0.7994 / 51.7%	89567 / 28.8%
	8 sections	21.18 / 16.4%	0.9294 / 76.4%	96538 / 38.8%
Spare wires	4	26.71 / 46.8%	0.9959 / 89%	102383 / 47.2%
	8	29.03 / 59.5%	1.0442 / 98.2%	116789 / 67.9%
PFLRM		19.30 / 6.0%	0.5789 / 9.9%	83326 / 19.8%
SFQS		20.27 / 11.4%	0.6807 / 29.1%	81288 / 16.9%

(VA/SA) in the first stage, and Switch Traversal (ST) in the second stage. Link Traversal (LT), where the proposed fault-tolerant architecture is applied, is implemented as a separated pipeline stage. Each baseline router has 5 Physical Channels (PC), and each PC is facilitated with 5 Virtual Channels (VC), and a 4-flit deep, 32-bit wide buffer is applied in each VC. Both the width of flit and link are 32, which is most widely used in NoC proposals [17]. The router and the link fault-tolerant modules are synthesized using the Synopsys Design Compiler with TSMC 65-nm standard cell as target technology. The target frequency is 500MHz.

A. Area and Power Overhead

The power consumptions and area overheads of all these four different link fault-tolerant methods are presented in Table. II. The proposed method is evaluated with two versions containing 4 and 8 link sections, and the spare wire is evaluated with two versions containing 4, and 8 spare wires. From the Table we can observe that, the area and power overheads of the proposed architecture are lower than the ones of spare wire replacement, but higher than the ones of PFLRM and SFQS. For example, with 8 link sections, the area overhead of our method is 38.8%, which is 29% less than the one of the spare wire replacement method (8 wires), but 19% higher than PFLRM and 21.9% higher than SFQS. The dynamic power overhead of the proposed architecture with 8-section version is 16.4%, which is 43.1% lower than of using 8 spare wires (59.5%), but 5% higher than SFQS (11.4%) and 10.4% higher than PFLRM (6.0%). The leakage power consumption of proposed method also falls between spare wire replacement and other two partially faulty link usage methods.

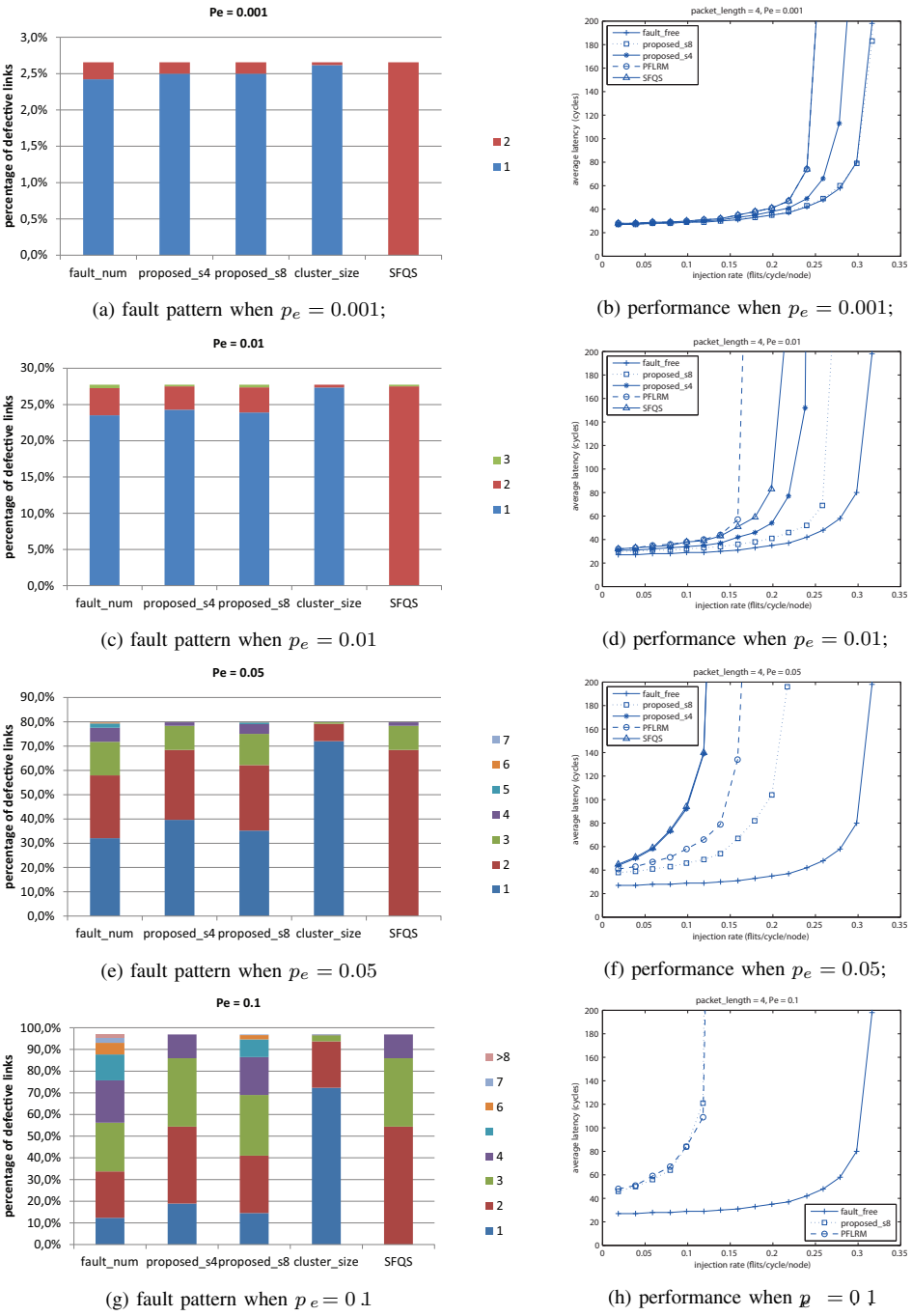
For all the fault-tolerant methods we have evaluated, the main parts of the power and area overheads are caused by multiplexers and registers. When compared with SFQS, the proposed method with four link sections use twice more multiplexers and registers in the transmitter and receiver, and with eight sections the ratio increases to four. Therefore, the area and power cost of our proposed method is higher than SFQS. When the spare wire replacement method is used, an $N + 1 : 1$ multiplexer is used at each end of a link wire (N is the number of faulty wires can be tolerated) , leading to higher area and power consumptions than all the other methods.

B. System performance

To evaluate the performance of our proposed architecture, we applied different fault patterns of the links with a range of permanent wire fault rates (0.001, 0.01, 0.05, and 0.1) to an 8×8 2D mesh NoC system. We assume that each wire has the same fault rate p_e and faults are uniformly distributed across the links. For comparison and illustration purpose, the fault rate p_e is set higher than its typical value [9]. The three partially faulty link usage strategies, i.e., the proposed method with four (proposed_s4) and eight (proposed_s8) link sections, PFLRM, and SFQS, are applied to the NoC system and simulated with each fault pattern. Synthetic uniform random traffic pattern and XY routing protocol are used in the simulation.

Fig. 6(a), (c), (e), and (g) present the fault distribution in each fault pattern and Fig. 6(b), (d), (f), and (h) plot the corresponding performance curves of the NoC system in terms of average latency when different strategies are utilized. In the fault distribution figures, the height of each column represents the proportion of defective links in the NoC. In each column, different colors represent the proportions of defective links with different fault degrees. For example, the red parts of the columns denote the proportions of links with two faulty wires (*fault_num*), or links with two unusable link sections in proposed_s4, proposed_s8, and SFQS, or links with a fault cluster size of two in PFLRM.

As we can observe in Fig. 6(a), when the permanent wire fault rate is low ($p_e = 0.001$), the percentage of defective links in the NoC is 2.7%, out of each 91% contain one faulty wire while the rest of them have two. For the proposed method, this means that 94% of the defective links contain one unusable section and 6% contain two. As it is indicated in Table I, the link latency overheads on this defective links are very low in proposed_s8. Therefore, proposed_s8 achieves the best performance as its average latency is very close to the fault-free case. The performance of proposed_s4 is lower than proposed_s8 but still better than PFLRM and SFQS. This can be explained by the fact that in links with one defective section, both PFLRM and SFQS will double



(a) fault pattern when $p_e = 0.001$;

(b) performance when $p_e = 0.001$;

(c) fault pattern when $p_e = 0.01$

(d) performance when $p_e = 0.01$;

(e) fault pattern when $p_e = 0.05$

(f) performance when $p_e = 0.05$;

(g) fault pattern when $p_e = 0.1$

(h) performance when $p_e = 0.1$

Figure 6. Fault Patterns and Corresponding Performance of the NoC.

the link latency at least, while proposed_s4 can keep the latency overheads as low as 33.3%.

With the increase of p_e, more links contain faults and the average faulty wire number becomes larger, leading to more unusable sections and bigger fault cluster size in a link (see Fig. 6(c), (e), and (g)). The average latencies increase for all partially faulty link usage strategies. But the proposed method still outperforms PFLRM and SFQS (see Fig. 6(d)) when the fault rate is not very high ($p_e = 0.01$).

When p_e further increases to 0.05 (see Fig. 6(e) and Fig. 6(f)), proposed_s8 still has the best performance because the link latency overhead on more than 99% of the defective links is less than 100%. As the number of links which have 200% plus link latency in proposed_s4

and SFQS (more than two sections are broken) surpasses PFLRM (cluster size larger than one), PFLRM achieves a better performance than proposed_s4 and SFQS but still worse than proposed_s8. We note that at such p_e value, totally broken links exist in proposed_s4 and SFQS (all link sections are broken). Because fault-tolerant routing is not considered in this paper, the totally broken links are still given one available section to maintain the simulation integrity. This cannot fundamentally affect the results because only 1 or 2 such links may exist in the NoC at this fault rate.

When the permanent wire fault rate is as high as 0.1, 97% of the links are defective and the average fault degree is very high, as dipected in Fig. 6(g). Under this extreme condition, proposed_s8 exhibits equivalent performance as PFLRM (see Fig. 6(h)). Proposed_s4 and SFQS have so many totally broken links that they are not considered. If the fault rate keeps on increasing, totally broken links appear in proposed_s8 and as a consequence its performance gets worse than the PFLRM one.

The number of faulty wires in a link equals the number of spare wires required in spare wire replacement method. From Fig. 6(g) we can observe that when the permanent wire fault rate is 0.1, spare wire replacement method can maintain 98% of links available with 8 spare wires, while proposed_s8 can achieve the same fault-tolerant capability with reduced performance. As the fault degrees of links increase to the fault pattern presented in Fig. 6(g), the performance of proposed_s8 degrades gracefully.

VI. Conclusions

A novel structure which can utilize partially faulty links with graceful performance degradation has been presented. The flits and links are divided into several sections according to the required fault-tolerant capability and silicon budget. When faulty wires exist in some of the link sections, flit sections of adjacent flits are serialized at the transmitter side, transmitted via the remaining fault-free link sections, and recovered at the receiver side before they are written into the input buffers. The proposed transmitter and receiver are transparent to other parts of the routers thus other function units, e.g., the ECC logic, can be implemented conventionally. Experimental results demonstrate that the latency overhead is significantly reduced by our method when compared with related partially faulty link usage methods, i.e., PFLRM and SFQS, while the area and power overheads can be diminished by up to 29% and 43.1%, respectively, when compared with the spare wire replacement method with similar fault-tolerant capability. Our experiments also suggest that for links which are too much affected by faults, it might be a better choice to deem them as totally broken and to route packets on alternative low latency links. The combination of fault-tolerant routing and our proposed partially faulty link usage method to achieve an intelligent path selection constitute a future work subject.

References

[1] C. Grecu, A. Ivanov, R. Saleh, and P. P. Pande, "NoC interconnect yield improvement using crosspoint redundancy," IEEE DFT, pp.457-465, October 2006.

[2] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," IEEE Micro, vol.25, pp.10-16, November-December, 2005.

[3] O. Unsal, et al, "Impact of Parameter Variations on Circuits and Microarchitecture," IEEE Micro, Vol. 26, pp.30-39, November-December, 2006

[4] C. Hernandez, F. Silla, V. Santonja, and J. Duato, "A new mechanism to deal with process variability in NoC links," in IEEE IPDPS, pp.1-11, May 2009.

[5] C. Chen, and G. Chiu, "A fault-tolerant routing scheme for meshes with nonconvex faults," IEEE Trans. Parallel and Distributed Systems, vol. 12, no. 5, May 2001.

[6] P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic communication: A new paradigm for fault-tolerant Networks-on-Chip", VLSI Design, vol. 2007.

[7] R. Tamhankar, et al, "Timing-error-tolerant Network-on-Chip design methodology," IEEE Trans. CAD of Integrated Circuits and Systems, vol.26, no.7, pp.1297-1309, July 2007.

[8] A. Strano, C. Hernandez, F. Silla, D. Bertozzi, "Process variation and layout mismatch tolerant design of source synchronous links for GALS Networks-on-Chip," SoC, pp. 43-48, September 2010.

[9] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu, "Self-adaptive system for addressing permanent errors in on-chip interconnects," IEEE Trans. VLSI Systems, vol.18, no.4, pp.527-540, April 2010.

[10] A. Vitkovskiy, V. Soteriou, and C. Nicopoulos, "A fine-grained link-level fault-tolerant mechanism for Networks-on-Chip," ICCD, pp.447-454, October 2010.

[11] M. Palesi, S. Kumar, and V. Catania, "Leveraging partially faulty links usage for enhancing yield and performance in Network-on-Chip," IEEE Trans. CAD of Integrated Circuits and Systems, vol.29, no.3, pp.426-440, March 2010.

[12] T. Bjerregaard, and S. Mahadevan, "A survey of research and practices of Network-on-Chip," ACM Computing Surveys, Vol.38, March 2006.

[13] T. Lehtonen, P. Liljeberg, and J. Plosila, "Online reconfigurable self-timed links for fault tolerant NoC," VLSI Design, vol. 2007.

[14] Q. Yu, and P. Ampadu, "Transient and permanent error co-management method for reliable Networks-on-Chip," NOCS, pp. 145-154, May 2011.

[15] C. Grecu, A. Ivanov, R. Saleh, and P. P. Pande, "Testing Network-on-Chip communication fabrics," IEEE Trans. CAD of Integrated Circuits and Systems, vol. 26, no. 12, pp.2201-2213, December 2007.

[16] Y. Lu, J. McCanny, and S. Sezer, "Exploring virtual-channel architecture in FPGA based Networks-on-Chip," in Proceedings of the 24th IEEE International SOC Conference (SOCC), pp. 302-307, September 2011.

[17] E. Salminen, A. Kulmala, T. D. Hamalainen, "Survey of Network-on-Chip proposals," white paper, OCP-IP, March 2008.