Addressing Transient and Permanent Faults in NoC With Efficient Fault-Tolerant Deflection Router

Chaochao Feng, Zhonghai Lu, *Member, IEEE*, Axel Jantsch, *Member, IEEE*, Minxuan Zhang, and Zuocheng Xing

***Abstract*— Continuing decrease in the feature size of integrated circuits leads to increases in susceptibility to transient and permanent faults. This paper proposes a fault-tolerant solution for a bufferless network-on-chip, including an on-line fault-diagnosis mechanism to detect both transient and permanent faults, a hybrid automatic repeat request, and forward error correction link-level error control scheme to handle transient faults and a reinforcement-learning-based fault-tolerant deflection routing (FTDR) algorithm to tolerate permanent faults without deadlock and livelock. A hierarchical-routing-table-based algorithm (FTDR-H) is also presented to reduce the area overhead of the FTDR router. Synthesized results show that, compared with the FTDR router, the FTDR-H router can reduce the area by 27% in an 8 × 8 network. Simulation results demonstrate that under synthetic workloads, in the presence of permanent link faults, the throughput of an 8 × 8 network with FTDR and FTDR-H algorithms are 14% and 23% higher on average than that with the fault-on-neighbor (FoN) aware deflection routing algorithm and the cost-based deflection routing algorithm, respectively. Under real application workloads, the FTDR-H algorithm achieves 20% less hop counts on average than that of the FoN algorithm. For transient faults, the performance of the FTDR router can achieve graceful degradation even at a high fault rate. We also implement the fault-tolerant deflection router which can achieve 400 MHz in TSMC 65-nm technology.**

***Index Terms*— Deflection routing, fault-tolerance, on-line fault diagnosis, permanent fault, transient fault.**

I. Introduction

NETWORK-ON-CHIP (NoC) approach has emerged as a promising solution for on-chip communications to enable integrating various processors and on-chip memories into a single chip [1]. However, with the technology scaling down to the nanometer domain, shrinking transistor sizes, lower power voltages, and higher operating frequencies seriously affect the reliability of CMOS VLSI circuits [2]. Two kinds of faults (permanent and transient) need to be addressed in NoC architectures. There are two methods to cope with transient and permanent faults in NoC. Flow-control-based methods (e.g., [3] and [4]), combine the error control code with the retransmission mechanism to tolerate transient faults occurring in transmission; the other is fault-tolerant routing which utilizes the inherent structure redundancy of NoC to route packets around the permanent faulty routers or links to achieve fault-tolerance. A good fault-tolerant routing algorithm should ensure "0 lost packet" in whatever fault patterns as long as a path exists. However, many of them (e.g., [5] and [6]) can only improve the successful arrival rate of the packet.

Recently, bufferless router has been studied in NoC to achieve higher speed and lower cost than a wormhole or virtual channel router [7]–[9]. Except one input register for each input port, there are no other buffers in the bufferless router. Due to the lack of buffers, deflection routing is utilized in the bufferless router to route packets to neighboring routers immediately without buffering in the router. The fully adaptive feature of deflection routing provides the potential to route packets to avoid faulty links/routers and achieve fault-tolerance. In our previous works [10], a reconfigurable fault-tolerant deflection routing (FTDR) algorithm based on reinforcement learning has been proposed for 2-D mesh NoC. The advantage of the FTDR algorithm is the topology-agnostic feature, which is insensitive to the shape of the faulty region. The routing table can be reconfigured during packets transmission. In [10], we only focus on the routing algorithm to handle permanent faults. In this paper, a fault-tolerant solution, including an on-line fault diagnosis mechanism, a link-level error control scheme, and a fault-tolerant routing algorithm, is proposed for the bufferless NoC to tolerate both transient and permanent faults. The fault diagnosis mechanism uses the single-error-correcting and double-error-detecting (SECDED) Hamming code [11] to detect both transient and permanent link faults. A hybrid automatic repeat request (ARQ) and forward error correction (FEC) link-level error control scheme using retransmission is proposed to handle transient faults. The FTDR algorithm guarantees "0 lost packet" as long as the fault pattern does not cut the network into two or more disconnected parts. The algorithm is also modified to still work when some links are in the test mode. A hierarchical-routing-table-based algorithm (FTDR-H) is also presented to reduce the area overhead of the router. Synthesized results show that, compared with the FTDR router, the FTDR-H router can reduce the area by 27% in an 8 × 8 network. Simulation results demonstrate that under synthetic workloads, in the presence of permanent link faults, the throughput of an 8 × 8 network with FTDR and

Manuscript received July 23, 2011; revised March 21, 2012; accepted May 18, 2012. Date of publication July 31, 2012; date of current version May 20, 2013. This work was supported in part by the National Natural Science Foundation of China under Grant 60970036, Grant 61003301, and Grant 61170083.

C. Feng, M. Zhang, and Z. Xing are with the School of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: fengchaochao@nudt.edu.cn; mxzhang@nudt.edu.cn; zcxing@nudt.edu.cn).

Z. Lu and A. Jantsch are with the Department of Electronic Systems, Royal Institute of Technology, Stockholm 16440, Sweden (e-mail: zhonghai@kth.se; axel@kth.se).

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.2012.2204909

FTDR-H algorithms are 14% and 23% higher on average than that with the fault-on-neighbor (FoN) aware deflection routing algorithm [12] and the cost-based deflection routing algorithm [13], respectively, and achieve almost 2 × less hop counts on average than that with the cost-based algorithm. Under real application workloads, the FTDR-H algorithm achieves 20% less hop counts on average than that of the FoN algorithm. For transient faults, the performance of the FTDR router can achieve graceful degradation even at a high fault rate. We also implement the FTDR and FTDR-H routers, which can achieve 400 MHz in TSMC 65-nm technology.

Few previous works provide a fault-tolerant solution to handle both transient and permanent faults for NoC. This paper proposes a fault-tolerant solution for a bufferless NoC. We have significantly extended our previous works [10] in the following aspects.

1) A link-level error control scheme handling transient faults through a hybrid ARQ/FEC method is introduced in the fault-tolerant solution. Experiment results show that this scheme can achieve graceful degradation at various transient fault rates. The previous works have only proposed the FTDR algorithm to handle permanent faults.
2) An on-line fault diagnosis mechanism using blocked SECDED Hamming code is proposed to distinguish transient faults from permanent faults, while the previous works do not refer to this topic. The encoding scheme with interleaving can also handle burst errors during packets transmission.
3) A test process is proposed to check if a faulty link contains a real permanent fault. The test process does not interfere with regular packets transmission. The FTDR algorithm is modified to still work well when the link is in the test mode. In addition, we discuss more details about the FTDR-H algorithm.
4) For experiments, we add quantitative comparison for the learning process of the FTDR and FTDR-H algorithms, and evaluate the effect of faults on the performance of the real application, and evaluate the performance of the FTDR router under transient faults.

The rest of this paper is organized as follows. Related works are reviewed in Section II. Section III describes the NoC architecture and fault diagnosis mechanism. In Section IV, a link-level error control scheme is proposed to handle transient faults. The reinforcement-learning-based FTDR algorithm and the hardware implementation of the router are proposed in Section V. In Section VI, simulation experimental results under both synthetic and real application workloads are discussed, followed by the conclusion in Section VII.

II. Related Works

A. Fault Detection Mechanisms for NoC

The precondition for a fault-tolerant system running smoothly is to detect the location of the faults first. The fault detection mechanism should also distinguish transient faults from permanent faults. Transient link errors can be detected via error coding techniques, such as cyclic redundancy check and parity codes [14]. A code disjoint fault detection scheme is used for online NoC fault diagnosis to locate the position of the faulty link/router precisely [15]. A diagnosis method that injects test patterns at the boundaries of a 2-D mesh network has been proposed in [16], to locate faults of routers and links based on different test patterns.

For detecting permanent errors in NoC, Lehtonen *et al.* propose an in-line test method to test each adjacent pair of wires and a syndrome storing-based error detection method based on evaluation of consecutive code syndromes at the receiver [17]. Few works focus on detecting transient faults and permanent faults at the meantime. A transient and permanent error co-management method for NoC links is described in [18]. However, the error control coding (ECC) scheme which is selected based on the noise condition can only detect transient errors.

B. Techniques to Handle Transient Faults for NoC

Transient faults in NoC can be handled at both link-level and transport level by three schemes: ARQ, FEC, and hybrid ARQ/FEC [19].

A simple and cost-effective end-to-end packet-based communication protocol to address transient faults for NoC has been proposed in [20]. Go-back-n retransmission policy is used, which means the receiver generates a single acknowledgement for a pre-defined number of packets that it receives. A link-level ARQ scheme, which is called hop-by-hop [21], utilizes a three-flit-deep retransmission buffer per virtual channel to handle transient errors in virtual channel router. In [4], the authors have compared the end-to-end flow control with the link-level flow control in terms of latency and power consumption.

Due to the lack of buffers in the deflection router, implementing link-level error control scheme on it is different from the wormhole/virtual channel router with more input buffers. Kohler *et al.* have proposed a link-level ARQ mechanism that uses only one additional retransmission register to handle both router-internal transient faults and transient link faults in deflection router [13]. However, because of only one retransmission buffer, packets will be lost if the second transient link fault occurs one cycle after the first one or the transient link fault lasts for two cycles.

C. Fault-Tolerant Routing Algorithms to Handle Permanent Faults for NoC

Two kinds of fault-tolerant routing, which are known as stochastic and deterministic, have been proposed for NoC to handle permanent faults. Stochastic communication transfers redundant packets through different paths to avoid faults. Dumitras *et al.* have proposed a probabilistic broadcast scheme that is derived from the randomized gossip protocol [22]. In order to limit the number of packet copies, redundant random walk [23] only replicates packets at the source node. Although the stochastic communication can be highly fault-tolerant, it wastes much bandwidth to transfer redundant packets, which reduces the throughput of the network significantly.

Depending on the shape of the fault region, deterministic fault-tolerant routing algorithms can be categorized into two classes: one can handle regular fault regions (e.g., convex and concave shapes) and the other, which is also known as topology-agnostic, can handle irregular fault regions. A reconfigurable routing algorithm is proposed to route packets surrounding a faulty router in a 2-D mesh NoC [24]. But it can only be used in one faulty router topology and extended to one faulty region topology. In [25], a deadlock-free routing algorithm is presented to handle irregular mesh topology with rectangular regions. This algorithm uses the concept of faulty-rings and faulty-chains to isolate the faulty nodes from the rest of the network. A resilient routing algorithm for fault-tolerant NoC based on turn model is described in [26]. However, the routing table of the router is reconfigured to avoid faulty components in an offline process. Although the regular nature of a faulty region can facilitate to route packets around it, it is not practical to assume the shape of the faulty region since the occurrence of the fault is not predictable.

For deflection routing, a fault adaptive routing algorithm, based on a cost function to make routing decision, has been proposed in [13]. The cost function takes the route length and local fault status into consideration. Because the routing decision is only based on the fault information of the current router, the hop count field of the packet can easily overflow even in some simple fault patterns, which can lead to livelock. A FoN aware deflection router [12], which can tolerate convex and concave fault regions without deadlock and livelock, makes routing decision based on the two-hop fault information and the fault region shape without using a routing table.

The network with irregular fault regions can be viewed as the network topology changing from regular to irregular topology. Some routing algorithms used in irregular interconnection network are based on finding the spanning tree embedded in the network, such as up*/ down* [27] and IMMUNET [28]. For NoC, few works focus on handling network with irregular fault regions. A fully adaptive fault-tolerant routing (called force-directed wormhole routing) has been proposed to handle completely broken links and routers [29]. FDWR exchanges routing information between routers to configure the routing table. The problem of this approach is that additional flits rather than data flits are needed to check neighboring router states for updating routing table. A region-based routing has been proposed to handle irregular NoC [30]. This algorithm groups destinations into regions to make routing decision, however, it needs an additional region computation process.

III. NoC Architecture and Fault Diagnosis

A. NoC Topology and Packet Format

The NoC architecture is based on a 2-D mesh topology, Nostrum NoC [31]. Each processing element is attached to a router (R), as shown in Fig. 1. The difference from the ordinary 2-D mesh is that the boundary output is connected to the input of the same router. This can be viewed as an additional packet buffer. All incoming packets are prioritized according to their hop counts, which record the number of hops the packet has been routed. The router makes routing decision for each arriving packet from the highest priority to the lowest. If a desired output port has already been occupied by a higher priority packet, a free port with the smallest stress value will be chosen, which means the packet has to be deflected. The stress value, which can be used to balance traffic load, is the number of packets processed by neighboring routers in the last four cycles.

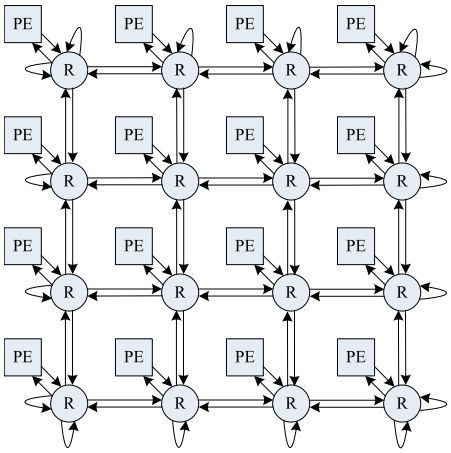


Fig. 1. NoC architecture.

The basic data transfer unit in this paper is a packet. The original packet format, which is compatible with a multicore NoC platform [32], is shown in Fig. 2(a). A packet, which has 114 bits, contains a 34-bit head and an 80-bit payload. A valid bit (V) is used to mark a packet valid or not. Relative addressing is used for the source and destination address fields (SA and DA) which have 12 bits (six bits for row/column address), respectively. The hop counter field (HC, nine bits) records the number of hops the packet has been routed. In order to detect and correct errors in transmission, SECDED Hamming codes are used to encode the head and payload, respectively. The encoded packet format, which has 156 bits, is shown in Fig. 2(b). The head is divided into two parts and Hamming (23, 17) code is used to encode each part. The payload is divided into five parts, each of which is encoded with Hamming (22, 16) code. The details of the fault diagnosis mechanism will be discussed in Section III-C.

B. Fault Model

In this paper, faults are considered as faulty links which can be both transient and permanent faults. For deflection router, the number of input ports should be equal to the number of output ports, so permanent link failures are assumed to be bidirectional. In each router, a four-bit fault vector is used to represent the fault status of its four links (North, East, South, and West). A "1" in the fault vector represents the corresponding bidirectional links are broken. The faulty region can be any shape as long as it does not disconnect the network. Transient faults are often caused by single-event upset. It has been stated that single-event transient pulsewidths vary from about 900 ps to 3 ns for a 1.5-μm technology and rapidly scale down with the technology [33]. We implement the fault-

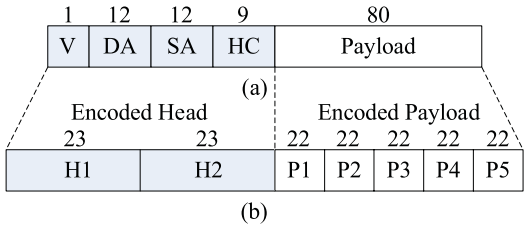


Fig. 2. (a) Original packet format. (b) Encoded packet format.

tolerant deflection router with TSMC 65-nm technology, which can achieve 400 MHz (2.5 ns per cycle). So it is reasonable to assume that most of transient faults last for one clock cycle in our works. If a transient fault lasts for two clock cycles, the router will enter into a test model to test whether the link contains a real permanent fault. Besides faults on links, partial transient and permanent faults on input registers of the router can also be detected after decoding the packet. Faults on the control circuit and routing table of the router are beyond the scope of this paper.

C. Link-Level Fault Detection and Protection

SECDED Hamming code, which can correct single error and detect double errors, is used to encode the packet to perform fault diagnosis. To make a compromise among performance, area and power consumption, we compare two ECC strategies: 1) encode the whole packet with Hamming (122, 114) code and 2) encode the head with two Hamming (23, 17) codes and the payload with five Hamming (22, 16) codes. For the first encoding strategy, eight parity bits are used to encode the 114 bits packet into 122 bits. It can only correct one-bit error and detect two-bit error in the packet. The second strategy divides the packet into seven parts: two for head and five for payload, encoded with Hamming (23, 17) and Hamming (22, 16) codes, respectively. The encoded packet length has 156 bits with 42 parity bits. It can correct seven simultaneous single-bit errors in each part and detect at most 14 error bits in the case of a two-bit error in each part. If any of the seven parts contains a two-bit error for one cycle, it will lead to a retransmission. If the retransmitted packet has the same two-bit error in any part, the link will be tested to check if it is considered as a permanent faulty link. The input register is followed by the decoder, so the ECC mechanism can detect and correct both link and input register errors.

Table I shows the latency, area, and power comparison of encoders and decoders for Hamming (122, 114) and (22, 16) codes, respectively (developed in VHDL and synthesized with TSMC 65-nm technology). The results for Hamming (23, 17) code are similar as Hamming (22, 16) code. As the table illustrates, the latency, area, and power consumption of the encoder and decoder for Hamming (122, 114) code are much larger than those of Hamming (22, 16) code. Partitioning the packet into smaller blocks and encoding separately is clearly a better strategy for improving performance and lowering area and power consumption. From the synthesize results, the number of logic levels for the encoding and decoding circuits of H(122, 144) code is 14 and 16, respectively, while the number of logic levels for the encoding and decoding circuits of H(22, 16) is 7 and 10, respectively. Using smaller blocks can reduce the number of logic levels for the encoder and decoder. In addition, different parts of packet in the ECC strategy 2) are encoded with interleaving, which can also protect against burst errors. To correct multiple error bits, the Bose, Chaudhuri and Hocquenghem (BCH) code [34] can be constructed, however, as the number of bits to be corrected increases, the hardware complexity and decoding time will increase significantly. As illustrated in [35], the hardware overhead for BCH code is much larger than that of the Hamming code with interleaving. Table II also shows the code rate, which is the width of codeword (n) divided by the width of dataword (k), for three different FEC coding strategies. The ECC strategy 1) has the highest code rate but lowest reliability, while the strategy using BCH(170, 114) code to correct at most seven error bits has the lowest code rate. Considering the hardware overhead and communication performance, the ECC strategy 2) is chosen as our final coding method.

TABLE I
LATENCY, AREA, AND POWER COMPARISON OF ENCODERS AND DECODERS FOR DIFFERENT HAMMING CODES

	Latency (ns)		Area (μm^2)		Power (mW)	
	enc	dec	enc	dec	enc	dec
H(122, 114)	0.28	0.31	3304	5436	2.3	3.8
H(22, 16)	0.13	0.21	437	1086	0.21	0.67

TABLE II
CODE RATE COMPARISON FOR DIFFERENT FEC CODING STRATEGIES

Coding strategy	n	k	Code rate	Notes
ECC (1)	122	114	0.93	H(122,114)
ECC (2)	156	114	0.73	2×H(23,17)+5×H(22,16)
BCH	170	114	0.67	B(170,114)

In order to distinguish transient faults from permanent faults, the fault diagnosis process is shown in Fig. 3. The decoder will generate a syndrome, which contains the error information of the packet. If the decoder detects a single-bit error in any one part of the encoding packet, it will correct it no matter which kind of faults it is. If it detects a two-bit error in any one part of the encoding packet for one cycle, which is considered as a transient fault, it will require the upstream router to retransmit the packet. If the syndromes of two consecutive received packets are the same, which means the retransmitted packet contains the same two-bit error, in order to check whether the link contains real permanent faults, the router enters into a test mode to test this faulty link by applying four test vectors ($\{0\}^n$, $\{1\}^n$, $\{01\}^{n/2}$, $\{10\}^{n/2}$, where n is the number of bit width). The test process will be conducted at most twice. In the test mode, the link being tested is temporarily disabled. The downstream router requires the upstream router to send the test vectors by setting a *test_initial* signal to "1". If the downstream router detects any one of the four test vectors still containing the same error, the test process will be conducted again. If the second test still fails, the link is marked as a permanent faulty link. If all tests pass,

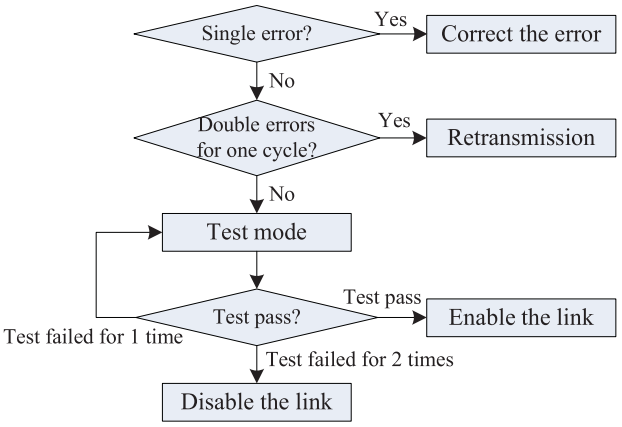


Fig. 3. Fault diagnosis process.

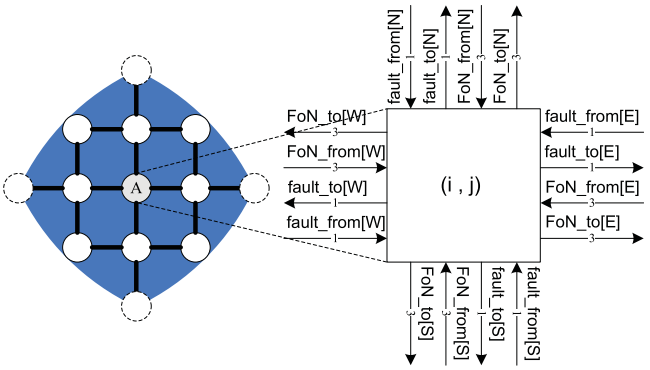


Fig. 4. Fault information transmission mechanism.

the link will be enabled again. In the test mode, the other links of the upstream and downstream routers can still transmit packets as normal. From this test process, transient faults can be distinguished from permanent faults.

D. Two-Hop Fault Information Transmission

A two-hop fault information transmission mechanism [12] has been used to reduce the average hop counts. In the two-hop fault information transmission mechanism, each router is responsible for not only transmitting its own link state to four neighbors but also collecting the link states from its three neighbors and transmitting them to the fourth neighbor with a three-bit vector FoN_to[d]. For example, router A can get the states of 16 links within two hops, as shown in Fig. 4.

Four dedicated signals [fault_from[d] (one bit), fault_to[d] (one bit), FoN_from[d] (three bits), FoN_to[d] (three bits)], which have eight bits in total for each direction, are used to transmit the fault information, as shown in Fig. 4. For each direction $d \in \{North,\ East,\ South,\ West\}$, the four signals are defined as follows:

1) fault_from[d] (one bit): the input link state along the direction d, receiving from the neighbor along d;
2) fault_to[d] (one bit): the output link state along the direction d, sending to the neighbor along d;
3) FoN_from[d] (three bits): three link states (except d) of the neighbor along d;
4) FoN_to[d] (three bits): link states along three directions except d, transmitting to the neighbor along d.

The dedicated fault information signals are critical for the fault-tolerant deflection router. To protect them against transient and permanent faults, we use two Hamming (7,4) codes, which can detect and correct one-bit error, to encode the eight-bit signal for each direction.

IV. Link-Level Error Control Scheme

The ARQ scheme using retransmission performs well without incurring much latency for low error rates, however, at higher error rate the hybrid ARQ/FEC scheme with slight hardware overhead provides better performance than the pure ARQ scheme [19]. We propose a hybrid ARQ/FEC scheme to perform error control to tolerate transient faults during packets transmission. In the case of a single-bit error in any part of the packet, the error can be corrected after the packet has been decoded. If any part of the packet contains a two-bit error for one cycle, the router which receives the packet will require the router, which sends the packet, to retransmit the packet. The hardware structure of the ARQ scheme for one input port i ($i \in \{North,East,South,West,Local\}$) is shown in Fig. 5. Each input port of the router has an input buffer (IB_i) with two entries instead of the original one and the boundary input port of the boundary router has an input buffer with three entries. Additionally, a retransmission buffer (RB_i) is used to buffer the packet which may be retransmitted. After decoding, the packet will be written to RB_i. A 2-to-1 multiplexer is used to select to send a new packet or retransmit the last packet. A request signal *arq* is introduced between two neighboring routers to indicate whether the last packet should be retransmitted or not. The fault information transmission signal (fault_to[i]) is used to disable the outgoing link i of the upstream router temporarily.

The signal timing to handle transient link faults is shown in Fig. 6. In this case, we assume a transient fault is on the output link j between router n and n+1, affecting the transmitting packet *P1* at Cycle 2 when the router n+1 detects the error. At Cycle 3, the router n+1 sets the signal *arq* to initiate the retransmission. At the same cycle, *P1* has already been written to RB_i and retransmitted to OUT_j. At Cycle 2 and *3*, *P2* and *P3* arrive at IN_i and are written to IB_i^2 and IB_i^1 at Cycle 3 and *4*, respectively. At Cycle 4, in order to avoid another packet from IN_i to overwrite *P2* in IB_i^2, the router n sets the signal fault_to[i]. After detecting this signal valid, the router n–1 will disable the output i temporarily for one cycle to stop sending a packet to router n. If the router n-1 is fully loaded, which means it has four packets to handle, it will also set the signal fault_to[i] to disable the output i of the router n–2 for one cycle. This process will be repeated along the direction i until finding a router, which is not fully loaded or reaching the boundary router. If it reaches the boundary router, because of the boundary input buffer with three entries, it can buffer an additional packet in the boundary input port buffer. From the experiment results under real application workloads, it is reasonable to assume that the network is never fully loaded to guarantee the process

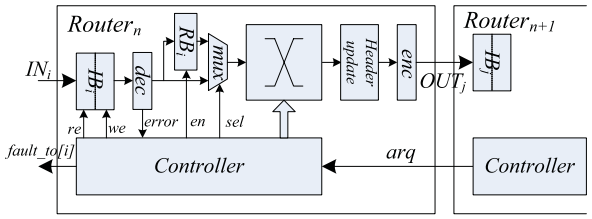


Fig. 5. Hardware structure of link-level error control scheme.

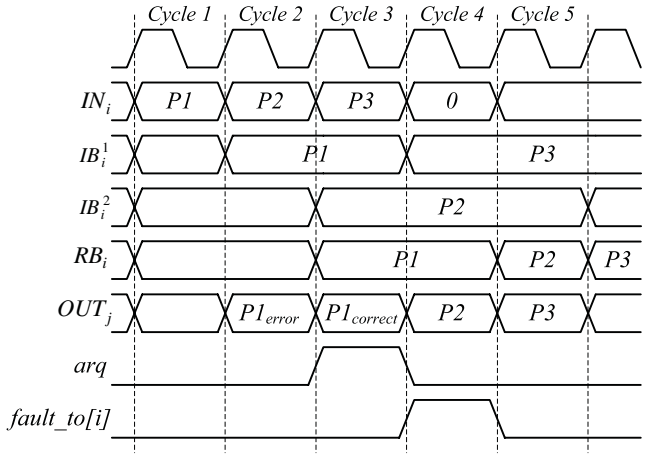


Fig. 6. Signal timing of ARQ scheme.

to be finished in a nonfully-loaded router. If a fault occurs in two consecutive cycles (such as Cycle 2 and *3*), the router n will drop this packet and test its output link j through the method discussed in Section III.B. If all tests have been passed, the link j will be enabled again, otherwise it will be marked as a permanent faulty link. Different from the ARQ scheme described in [13], if *P2* at Cycle 4 contains a transient fault, our scheme can retransmit it at Cycle 5. However, the ARQ scheme mentioned in [13] cannot handle this situation and the packet will be lost, because it uses only one input register and one retransmission register. Actually, our proposed error control scheme can also be extended to contain more input buffer entries to handle transient faults lasting for more than one cycle easily. Without loss of generality, to handle transient faults lasting for n cycles, the bufferless router needs an input buffer with at least $n + 1$ entries.

V. Reconfigurable FTDR

ARQ scheme fails to work in the presence of permanent faults. FEC scheme in this paper can only handle one-bit permanent fault for each encoding part of the packet. In this section, we propose a reconfigurable FTDR algorithm based on reinforcement learning to route packets avoiding permanent faulty links.

A. Basic Q-Routing

Q-routing is an adaptive routing algorithm based on a variant of the reinforcement learning—Q-learning, which makes routing decision using only local information without having to know the network topology in advance [36]. Because of its topology-agnostic characteristic, it can be modified to achieve fault-tolerance.

Q-routing is a table-based routing algorithm. The 2-D routing table stores the lowest estimated delivery time to each router in the network from each output of the current router. For example, the routing table entry $Q^x(d, y)$ denotes the lowest estimated delivery time from x to d through neighbor y. If the router x sends a packet to d through y, x receives the minimum estimated delivery time from y to d ($\min_z Q^y_{t-1}(d, z)$) after the packet is sent to y. Then the estimated delivery time $Q^x(d, y)$ can be updated, as shown in

$$Q^x_t(d, y) = (1-\alpha)Q^x_{t-1}(d, y)+\alpha(b^x_t+\min_z Q^y_{t-1}(d, z)) \quad (1)$$

where b^x_t is the time the packet spent in the buffer of router x, and α is the learning rate, which determines to what extent the newly acquired information will override the old information. A factor of 0 means no learning, while a factor of 1 considers only the most recent information.

B. FTDR Algorithm

For deflection routing, we use the number of hops to destination as Q-value instead of the estimated delivery time to build the routing table Q in each router. $Q^x(d, y)$ is defined as the minimum number of hops from x to d through y. Different from the estimated delivery time Q-value of the original Q-routing, $Q^x(d, y)$ is a deterministic value which is equal to one hop plus the minimum number of hops from y to d (defined as $\min_z Q^y(d, z)$). So (2) is used to update the routing table. It can be explained as follow: when a router x sends a packet to d through y, y returns the minimum number of hops from y to d back to x. Then x updates the corresponding entry with one hop plus the minimum number of hops from y to d ($\min_z Q^y_{t-1}(d, z)$). Equation (2) can also be derived from (1) (when $b^x_t = 1$ and $\alpha = 1$). Our deflection router has only one input register, which means the packet does not have to wait in the router, so b^x_t is 1. Because the Q-value ($Q^x(d, y)$) is not an estimate value, which is relevant to the previous Q-value in the original Q-routing, α is equal to 1 instead of a value between 0 and 1. Different from the FDWR algorithm [29] using additional packets to update the routing table, Q-value in the FTDR algorithm is transmitted with hard wires, which does not interfere with regular packets transmission

$$Q^x_t(d, y) = 1 + \min_z Q^y_{t-1}(d, z). \quad (2)$$

There are $n \times m$ entries in the routing table, where n is the number of routers in the network and m is the number of neighboring routers. For 2-D mesh, m is four. For a given topology, the initial routing table is fixed. For example, Fig. 7 shows a routing table of router 5 in a 3×3 2-D mesh. The router chooses a direction with the minimum number of hops to destination to send a packet. In the case of several directions with equal number of hops to destination, the router will choose one of them with the smallest stress value. For instance, both North and West directions have the minimum number of hops from router 5 to router 1, so the router will choose one of them with the smallest traffic load to route the packet.

	North	East	South	West
Number of hops to R1	2	4	4	2
Number of hops to R2	1	3	3	3
Number of hops to R3	2	2	4	4
Number of hops to R4	3	3	3	1
Number of hops to R5	0	0	0	0
Number of hops to R6	3	1	3	3
Number of hops to R7	4	4	2	2
Number of hops to R8	3	3	1	3
Number of hops to R9	4	2	2	4

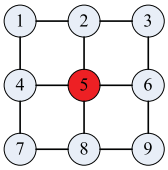


Fig. 7. Routing table of router 5 in a 3 × 3 mesh.

The routing table update function is shown in Algorithm 1. If there is no fault in the network, the routing table cannot be updated. If one link of the router is broken or temporarily disabled during testing, all table entries corresponding to this direction are set to "∞" (steps 2–4). After a learning period, the table entries will converge to a fixed value which denotes the minimum number of hops from each port to each destination. Additionally, we use the two-hop fault information to reduce the average hop counts. If a router detects that one of its neighbors y along direction d has only one link not faulty based on the two-hop fault information, the table entries from d to all destinations except y are set to "∞" (steps 5–7). If a two-hop link is faulty (FoN_from[d][j] = 1, $j \in$ {North, East, South, West}), the table entries from d to all destinations along j are updated with the previous table entry plus 2 (steps 8–11). Fig. 8 shows an example of reconfiguring the routing table of router 4 in a 3 × 3 2-D mesh with two faulty links. For example, router 4 finds FoN_from[East][North]= 1 and FoN_from[South][East]= 1, then the routing table entries from East to router 2 and from South to router 8 and 9 are reconfigured by the previous table entry plus 2. If the temporarily disabled link is enabled again, all entries corresponding to this direction are updated from "∞" to the minimum number of hops from this direction to the destination (steps 12–14), which is a new feature different from [10]. If the Q-value is not zero, the corresponding table entry will be updated with the Q-value (steps 15 and 16). In addition, the routing table can tolerate transient faults inherently. If one entry of the routing table is corrupted by the transient fault temporarily, the router will make wrong routing decision for a short while according to the duration of the transient fault. After the transient fault disappears, the faulty entry can be reconfigured through the updated Q-value.

The pseudo code of the algorithm is shown in Algorithm 2. There are at most four packets reaching a router at the same time. The router makes routing decision from the highest priority packet to the lowest. The router first calculates the destination ID of the packet and looks up the routing table to check if the packet has reached the destination (steps 1–4). If the packet has not reached the destination, the router looks up the productive direction(s) with the minimum number of hops to destination from the routing table and then chooses a free productive port with the smallest stress value to route the packet (steps 7–9). If there is no free productive port, the router chooses a free port with the smallest stress value to route the packet, which can balance the network traffic loads (steps 10 and 11). The router also sends the minimum number of hops to destination back to the neighboring router, which sends the packet, to update its routing table (step 6).

Algorithm 1 Routing Table Update Function

RoutingTableUpdate(*dest_ID, Q_value_from, FoN_from*)
1: ***for*** *dir* ∈ {*North*, *East*, *South*, *West*}
2: ***if*** *link*(*dir*) is faulty ***then***
3: ***for*** *i* = 1 to *N* ***and*** *i* ≠ *Swich_ID*
4: *table_entry*(*i*)(*dir*) ← ∞
5: ***if*** *Neighbor*(*dir*) has only one link not faulty ***then***
6: ***for*** *i* = 1 to *N* ***and*** (*i* ≠ *Swich_ID* ***and*** *i* ≠ *Neighbor_ID*(*dir*)) then
7: *table_entry*(*i*)(*dir*) ← ∞
8: ***for*** *j* ∈ {*North*, *East*, *South*, *West*}
9: ***if*** *FoN_from(dir)(j)*=1 ***then***
10: ***for*** *n* ∈ {all switches along *j* through *Neighbor*(*dir*)}
11: *table_entry*(*n*)(*dir*)=*table_entry*(*n*)(*dir*) + 2;
12: ***if*** *link(dir)* from disable to enable ***then***
13: ***for*** *i* = 1 to *N* ***and*** *i* ≠ *Swich_ID*
14: *table_entry*(*i*)(*dir*) ← *min_hops(swich_ID, i, dir)*
15: ***if*** *Q_value_from*(*dir*, *dest_ID*) ≠ 0 ***and*** *dest_ID* ≠ *Swich_ID* ***then***
16: *table_entry*(*dest_ID*)(*dir*) ← *Q_value_from*(*dir*, *dest_ID*)

	North	East	South	West
R1	1	3	3	∞
R2	2	2	4	∞
R3	3	3	5	∞
R4	0	0	0	0
R5	3	1	3	∞
R6	4	2	4	∞
R7	3	3	1	∞
R8	4	2	2	∞
R9	5	3	3	∞

(a)

	North	East	South	West
R1	1	3	3	∞
R2	2	4	4	∞
R3	3	3	5	∞
R4	0	0	0	0
R5	3	1	3	∞
R6	4	2	4	∞
R7	3	3	1	∞
R8	4	2	4	∞
R9	5	3	5	∞

(b)

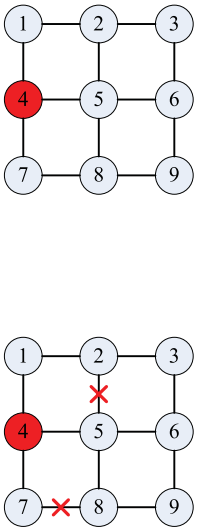


Fig. 8. Routing table update example. (a) Initial routing table. (b) Reconfigured routing table.

C. Hierarchical FTDR Algorithm

The routing table is the main overheads of the FTDR algorithm. Each router contains an $N \times 4$ routing table, where N is the number of routers in the network. Each entry of the routing table contains m bits, so the size of the whole table is $N \times m \times 4$ bits. As the size of the network increases, the routing table size will increase significantly. In order to reduce the table size, we also propose a hierarchical-routing-table-based deflection routing algorithm (called FTDR-H). The $n \times n$ mesh can be divided into several sub-regions with equal size. Each router contains a local and a region routing table. The local routing table stores the minimum number of hops from all ports of the current router to other routers in the same region, while the region routing table stores the

Algorithm 2 FTDR Algorithm

For each input packet (from the highest priority to the lowest priority)
1: *dest_ID*← get_dest_ID(d_x, d_y)
2: *Hops_to_Dest*←table_lookup(*dest_ID*)
3: ***if*** *Hops_to_Dest = (0,0,0,0)* ***then***
4: Route packet to local port
5: ***else***
6: *Q_value_to*(*input_Dir, dest_ID*) ←1 + min(*Hops_to_Dest*) //send Q-value
7: $\{d_{productive}\}$←get_prefer_Dir(*Hops_to_Dest*)
8: ***if*** there are free ports in $\{d_{productive}\}$ ***then***
9: Choose a free productive port with the smallest stress value to route the packet
10: ***else*** //all productive ports are not free
11: Choose a free port with the smallest stress value to route the packet

minimum number of hops from all ports of the current router to the nearest boundary routers in other regions. When a packet reaches a router, the router first checks whether the destination is in the same region as the current router or not. If it is, the router makes routing decision according to the local routing table. If the destination is not in the same region, the router makes routing decision according to the region routing table. The local and region routing tables are also updated by (2). Here, an 8×8 2-D mesh, which is divided into four 4×4 regions, is used as an example, shown in Fig. 9. Each router contains a local routing table of 16×4 entries and a region routing table of 4×4 entries. So the total size of the routing table is reduced from 64×4 to 20×4 entries.

D. Deadlock and Livelock Avoidance

FTDR (FTDR-H) algorithm is based on deflection routing, which is inherently deadlock-free since packets never have to wait in a router. Unlike store-and-forward and virtual cut-through switching techniques which provide buffer queue(s) to store blocked packets, packets are never blocked in deflection routing due to the fact that there is no buffer queue to store the packet. In deflection router, the number of input ports is equal to the number of output ports. Thus, an incoming packet will always find a free output port to go. It may be deflected to a nonpreferred path but will never be blocked. Packets are always on the run cycle by cycle until arriving at their destinations and being ejected from the network.

However, livelock has to be avoided by limiting the number of misroutings. The FTDR algorithm makes routing decision according to the packet priority and routing table. First, the algorithm always gives the highest priority to the oldest packet. If the packet wins the link arbitration, it will always be routed through the productive direction to the destination. Since a higher priority packet wins link arbitration, the packet can eventually advance toward its destination deterministically. Given a network size and different fault patterns, the length of the hop count field must be long enough guaranteeing the priority cannot saturate.

Second, it can be proved that the routing table entry will converge to the minimum hops to each destination from the following proof. Packet priority and converged routing table, which limit the number of misroutings, guarantee that a packet can eventually advance toward its destination thus livelock is avoided.

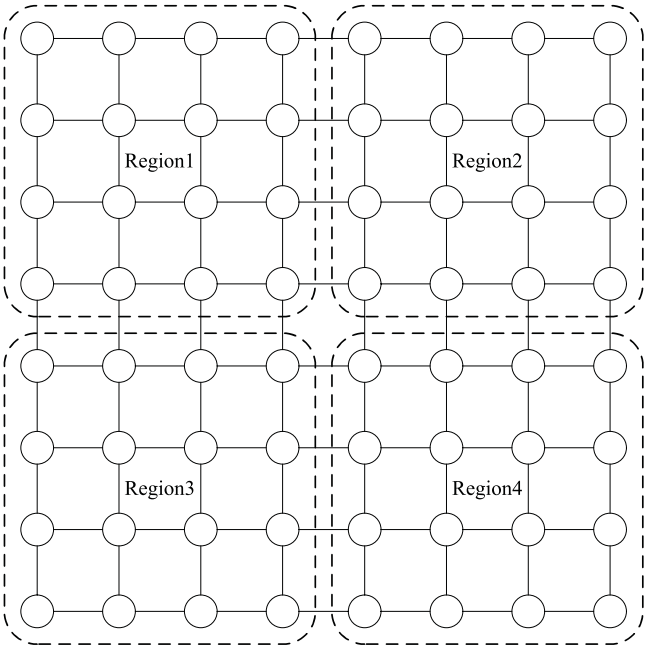


Fig. 9. Region partition example.

Theorem 1: With the FTDR algorithm, the routing table entry will converge to the minimum number of hops to destination within a limited time in the presence of fault regions, which do not disconnect the network.

Proof: Without loss of generality, assume router x sends packets to router d through neighboring router y. The initial Q-value $Q_0^x(d, y)$ is the minimum number of hops from x to d through y. The proof is divided into two parts based on the fault status of the network.

1) No faults: In the case of no faults, according to the (2), the term $\min_z Q_{t-1}^y(d, z)$ equals to the minimum number of hops from y to d, so $1 + \min_z Q_{t-1}^y(d, z)$ means the minimum number of hops from y to d plus 1 hop (from x to y). It still equals to the minimum number of hops from x to d through y ($Q_0^x(d, y)$). So the routing table entry will not be updated in the case of no faults.
2) In the presence of faulty links: If there are faulty links on the shortest path from y to d, the routing table entry of the routers with faulty links will be updated immediately. It will take a limited time to propagate the updated routing table entry to reconfigure the routing table entry $Q^y(d, z)$ of y.

If the faulty link lies in the same row/column as the router y with the manhattan distance of 1 hop, the corresponding routing table entry of y will be updated immediately without packet transmission under the two-hop fault information transmission mechanism. If the router only knows the one-hop fault information, after sending one packet to the corresponding node, the routing table entry of y will be updated two cycles later. In general, if the faulty link lies in the same row/column as the router y with the manhattan distance m hops ($m \in [0, M-2]$, where M is the number of routers in a row/column), at least m and $m-1$ packets should be sent to the corresponding

node and it will take at least *2m* and *2m–2* cycles to transmit the fault state to router *y*, and update the routing table under one-hop and two-hop fault information, respectively.

After transmitting the fault state and updating the routing table entry $Q^y(d, z)$ of y in a limited time, $\min_z Q^y(d, z)$ is still the minimum number of hops from y to d. The corresponding table entry $Q^x(d, y)$ of x will be reconfigured to minimum number of hops to d one cycle after $Q^y(d, z)$ converges. ■

Similarly to the proof above, it can be proved that the local and region routing table of the FTDR-H algorithm will converge within a limited learning period.

E. Hardware Implementation

The FTDR and FTDR-H routers are developed in VHDL. For FTDR router, each router contains an $N \times 4$ routing table (N is the number of nodes in the network). Each entry of the routing table contains six bits, so the size of the whole table is $N \times 24$ bits. An entry of all "1" denotes "∞". The stress value, fault information, and Q-value are transmitted between two routers with dedicated signals. In addition to the encoded fault information, the six-bit Q-value for each direction is also encoded with Hamming (10, 6) code to enhance the reliability for Q-value transmission. The routing controller makes routing decision according to the above three kinds of information. For the FTDR-H router, the routing table includes local and region routing tables. In order to make a comparison, we also implement the other two fault-tolerant deflection routers (the FoN router [12] and the cost-based router [13]). We synthesize all routers in an 8×8 network with TSMC 65-nm technology. The results of the performance, area, and power consumption for the four routers are shown in Table III. The operating frequency shown in Table III is the maximum achievable frequency of the four routers. All routers are single-cycle ones, meaning that all processes are done in one-cycle without using pipeline. Each of the routers can send/receive four packets in one cycle. The routing process consists of the following steps: packet decoding, routing computation, switch allocation, header updating, and packet encoding. The routing computation together with the input and output priority sorting are performed in parallel, and then the output port allocator makes output allocation for each packet from the highest priority to the lowest priority. Because the FoN and the cost-based routers do not include the fault diagnosis mechanism and link-level error control scheme, to make a fair comparison, the results in Table III for the FTDR and FTDR-H routers also do not include the fault diagnosis mechanism and link-level error control scheme. The FoN router can achieve the highest maximum frequency and the smallest area and power consumption because it is not table-based. But it can only handle regular faulty regions. Although the cost-based router does not use a routing table, it has high hardware overhead because it has to find the best permutation among all permutations of input and output ports based on a cost function. Compared with the original FTDR router, the FTDR-H router can reduce the area by 27% in an 8×8 network. To evaluate the hardware cost of the fault diagnosis mechanism and link-level error control scheme, we also synthesize the FTDR router with two ECC strategies described in Section III-C. Table IV shows the area, power, and reliability comparison of a FTDR router with two ECC strategies. As discussed in Section III-C, although the ECC strategy 1) has a little bit smaller power consumption, the ECC strategy 2) has the smaller area and the higher reliability than the ECC strategy 1). Compared with the original FTDR router, the fault diagnosis mechanism and link-level error control scheme increase area overhead by 56%.

TABLE III
IMPLEMENTATION COMPARISON OF FOUR DEFLECTION ROUTERS

	Frequency (MHz)	Area (μm^2)	Power (mW/MHz)
FoN	500	39076	0.01
Cost-based	166	82277	0.042
FTDR	400	101754	0.028
FTDR-H	400	74323	0.022

TABLE IV
AREA, POWER, AND RELIABILITY COMPARISON OF A FTDR ROUTER WITH TWO ECC STRATEGIES

	Area (μm^2)	Power (mW)	Detect bits	Correct bits
ECC (1)	163712	15.7	2	1
ECC (2)	158950	16.2	14	7

VI. EXPERIMENTAL STUDIES

In order to evaluate the four fault-tolerant deflection routers, we construct an 8×8 2-D mesh NoC simulator in VHDL. The routers are compared in terms of throughput and average hop count in the presence of both permanent and transient link faults, under synthetic and real application workloads. The learning process and area cost of the FTDR and FTDR-H routers are also studied.

A. Experimental Setup

For synthetic workloads, a packet generator is attached to each router and uses a FIFO to buffer the packets, which cannot be injected into the network immediately due to the fact that there is no free output port to route the packet. Six synthetic traffic patterns (uniform random, transpose, bit complement, bit reverse, shuffle, and tornado) [37] are used.

For real application workloads, we utilize a distributed-shared-memory-based multicore NoC architecture integrated with the FoN, FTDR, and FTDR-H routers, in which a LEON3 processor and a local memory are attached to each router through a dual microcoded controller [32]. Three applications, as shown in Table V, are mapped manually on LEON3 processors. For matrix multiplication, three matrices are distributedly stored in the shared memory (that is two elements of A and B, and two rows of C are stored in the local shared memory of each node). The data of 1D radix-2 parallel fast Fourier transform (FFT) [38] are equally partitioned into 128 groups of eight complex numbers each and stored in the local shared memory of 64 nodes, respectively. In wavefront computations, given a 64×64 matrix (elements on the left and top edges are

TABLE V
APPLICATION WORKLOADS

Application	Data set
Matrix multiplication	$C_{128\times128} = A_{128\times1} \times B_{1\times128}$
1D radix-2 parallel FFT	1024 complex numbers
Wavefront	a 64×64 matrix

already known), the computation of each remaining element depends on its left, above and above-left neighboring elements. This parallel computation at any instant forms a propagating wavefront.

For permanent faults, the link failure rates vary from 10% to 30%. The link failure rate is defined as the ratio of the number of faulty links to the number of total links in the network. Ten fault patterns are chosen for each failure rate in the simulations. The throughput and hop count results are the average results of the ten simulation results. Each selected fault pattern does not cut the network into disconnected sub-networks.

B. Learning Process With Permanent Faults

In this section, we will explore the learning process of the FTDR algorithm using both one-hop and two-hop fault information, and also compare the learning process of the FTDR-H algorithm with the FTDR algorithm. Fig. 10(a) and (b) plots the average hop count versus simulation time under uniform random traffic with a fault pattern of 10% link faults and the packet injection rate of 0.1 and 0.2 packets/cycle/node, respectively, at which the network does not reach the saturation point. At the beginning of the simulation, the average hop count increases quickly and after reaching a peak value, the average hop count will slightly decrease and remain stable. Here, the learning period is defined as the time at which the average hot count reaches the peak value. It can be seen that the FTDR algorithm with two-hop fault information has less average hop count than with one-hop fault information especially at the high packet injection rate. The length of the learning period depends on the fault pattern. The more complex the fault pattern is, the longer the learning period is. For this fault pattern, the learning periods with one-hop and two-hop information are approximate 160 and 120 cycles, respectively, at the packet injection rate of 0.1 and 340 and 300 cycles at the packet injection rate of 0.2. With two-hop fault information, some unnecessary misroutings can be avoided, so the average hop count can be less than that with only one-hop fault information. Fig. 11 compares the learning process of the FTDR-H algorithm with the FTDR algorithm under uniform random traffic with a fault pattern of 10% link faults and the packet injection rate of 0.1 packets/cycle/node. The learning periods of the FTDR and FTDR-H algorithms are approximate 140 and 100 cycles in this fault pattern, respectively. The peak average hop count of the FTDR-H algorithm is 30% less than that of the FTDR algorithm. The reason lies in that the FTDR-H algorithm uses the local routing table for each region, which can converge more quickly than the global routing table of the FTDR algorithm.

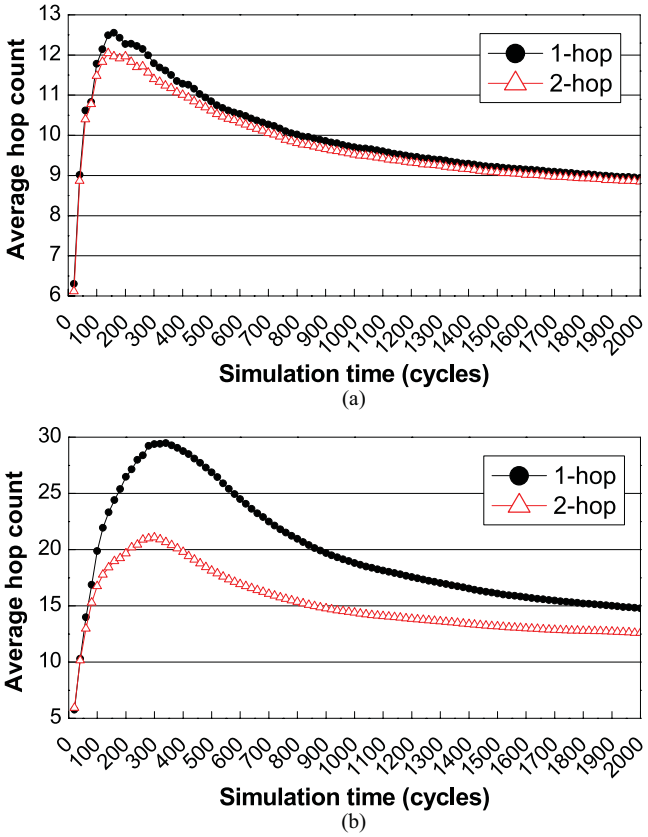


Fig. 10. Learning process of FTDR with one-hop and two-hop fault information. (a) Injection rate = 0.1 packets/cycle/node. (b) Injection rate = 0.2 packets/cycle/node.

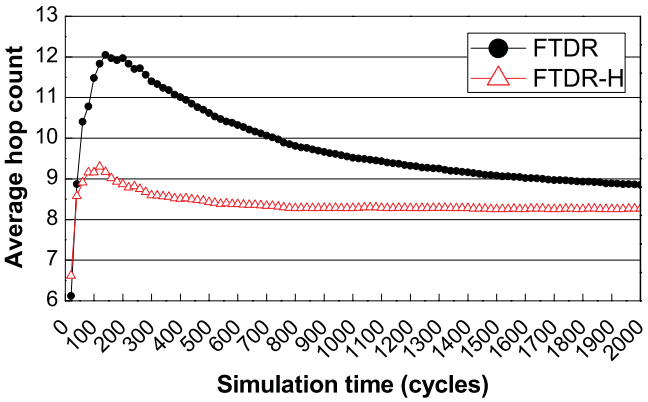


Fig. 11. Learning process comparison of FTDR and FTDR-H algorithms.

C. Performance With Permanent Faults

In this section, we evaluate the performance of the four fault-tolerant deflection routers in the presence of permanent link faults. Fig. 12(a)–(f) shows the throughput of the network achieved by the four algorithms in six synthetic traffic patterns, respectively, with link fault rates varying from 0 to 30%. Throughput, measured in packets/cycle/node, is defined as the saturation point of the network, which means the maximum accepted traffic. In the case of no faults, the throughput of the network with the cost-based router is slightly higher than that with the other three routers, because it can always find the best permutation for each packet even in a congested network

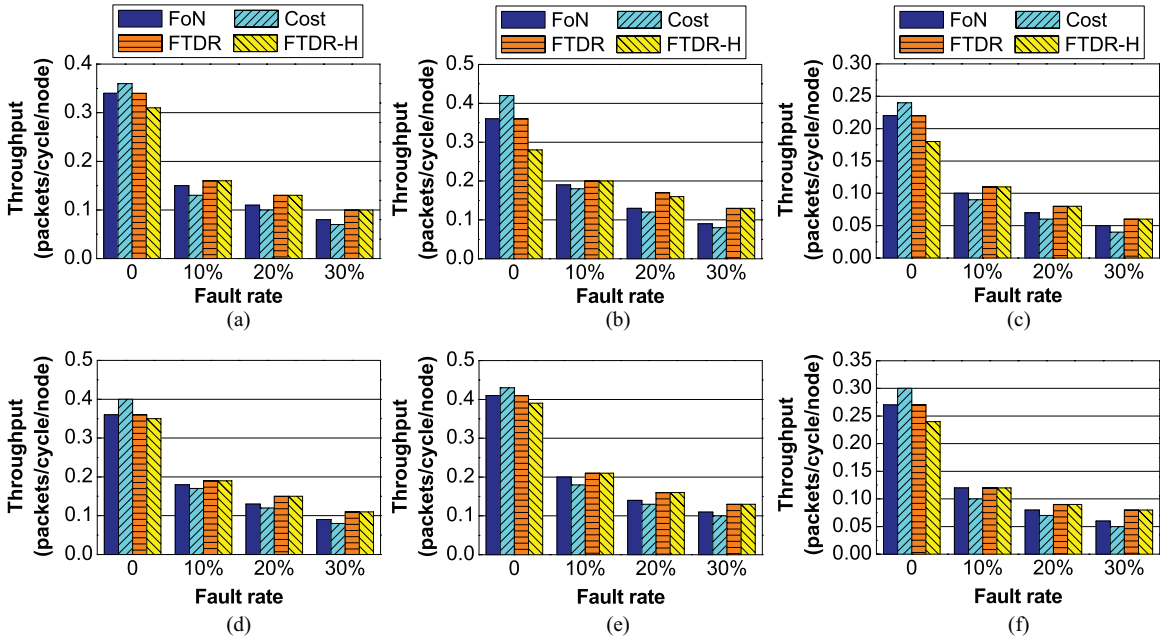


Fig. 12. Throughput at various link fault rates under synthetic workloads. (a) Uniform random. (b) Transpose. (c) Bit complement. (d) Bit reverse. (e) Shuffle. (f) Tornado.

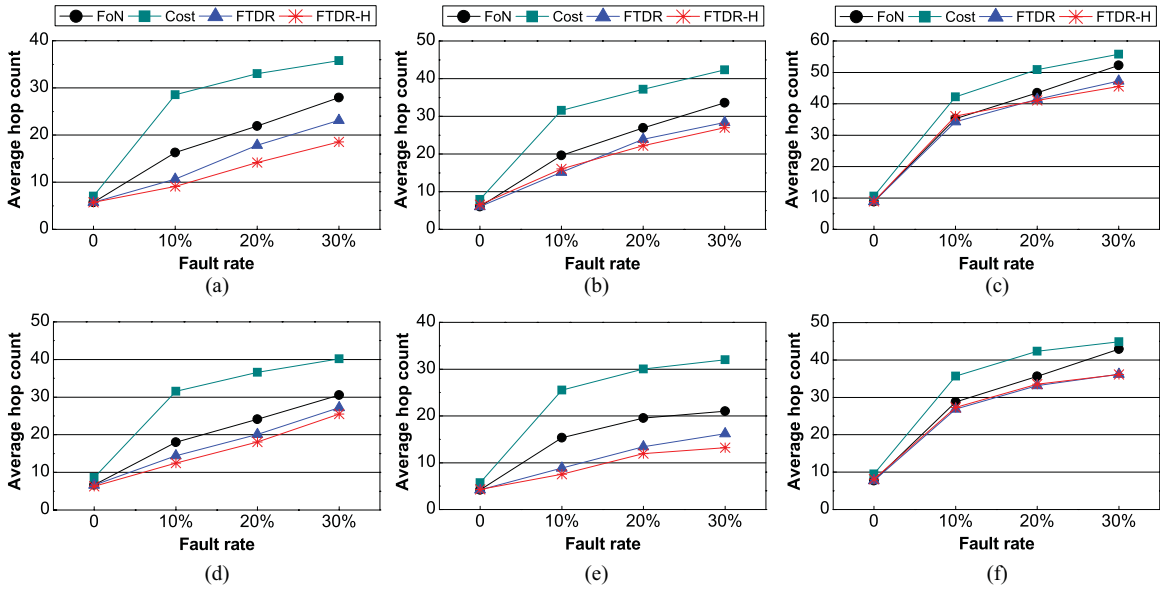


Fig. 13. Average hop count at various link fault rates under synthetic workloads. (a) Uniform random. (b) Transpose. (c) Bit complement. (d) Bit reverse. (e) Shuffle. (f) Tornado.

instead of unnecessary deflections. In the presence of link faults, the network with the cost-based router achieves the lowest throughput among the four routers, while the network with the FTDR and FTDR-H routers achieves almost the same throughput. The throughput of the network with the FTDR router is 14% and 23% higher on average than that with the FoN and cost-based routers, respectively.

Fig. 13(a)–(f) presents the average hop count of the four routing algorithms with various link fault rates in six synthetic traffic patterns, respectively. The packet injection rate is 0.1 packets/cycle/node, at which the network does not reach the saturation point. The hop count is defined as the number of hops that a packet travels from the source to the destination node. The FTDR-H algorithm can achieve 2.5×, 1.7×, 1.2×, 2×, 2.8×, and 1.3× less hop counts on average than the cost-based algorithm for each traffic pattern, respectively. The cost-based algorithm makes routing decision only according to the local fault status, so the hop count field of the packet can easily overflow even in some simple fault patterns, which may lead to livelock. The FTDR and FTDR-H algorithms perform better than the FoN algorithm because the routing table will converge after a limited learning period. For uniform random, bit reverse and shuffle traffic patterns, the average hop count of the FTDR-H algorithm is 18%, 10%, and 15% less than that of

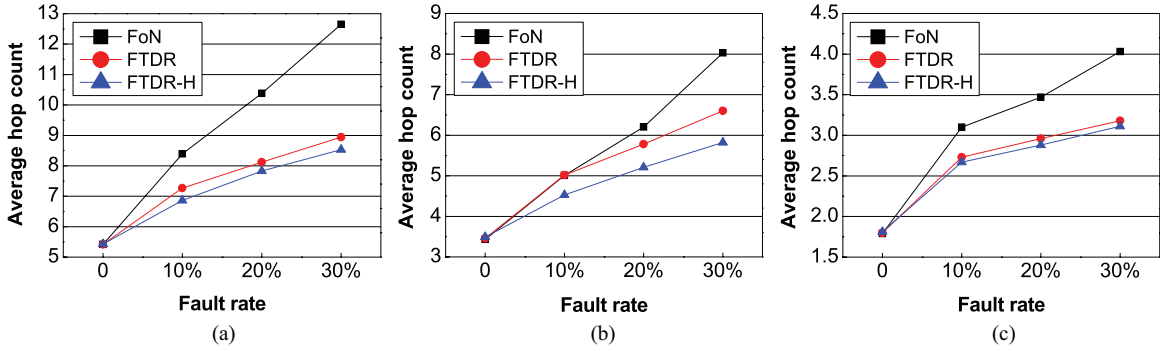


Fig. 14. Average hop count at various link fault rates under application workloads. (a) Matrix multiplication. (b) FFT. (c) Wavefront.

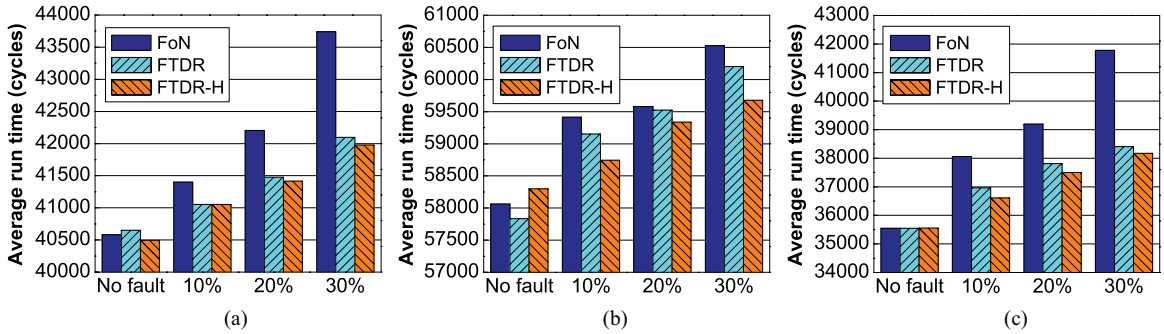


Fig. 15. Application run time at various link fault rates under application workloads. (a) Matrix multiplication. (b) FFT. (c) Wavefront.

the FTDR algorithm, respectively. For the rest of patterns, the FTDR-H algorithm performs similarly as the FTDR algorithm does.

Due to the fact that the cost-based deflection routing cannot guarantee being livelock-free in some fault patterns, we compare the FoN, FTDR, and FTDR-H algorithms on the application platform. Fig. 14(a)–(c) reveals the average hop count of the three algorithms with an increasing link fault rate varying from 0% to 30% for the three application workloads. As it can be observed, the FTDR and FTDR-H algorithms perform better than the FoN algorithm especially at the high fault rate. The average hop count of the FTDR-H algorithm is 25%, 18%, and 18% less than that of the FoN algorithm for the three application workloads, respectively. For matrix multiplication and wavefront computation, the FTDR-H algorithm performs slightly better than the FTDR algorithm. For FFT, the average hop count of the FTDR-H algorithm is 10% less than that of the FTDR algorithm in the presence of link faults. This is because the local routing table of the FTDR-H router will converge more quickly than the FTDR router as shown in the previous section. In order to evaluate the effect of faults on the performance of the real application, we also compare the average run time of the three real applications at different link fault rates. As shown in Fig. 15(a)–(c), for matrix multiplication, in the case of no faults, the shortest run time is achieved with the FTDR-H algorithm, while for FFT and wavefront the shortest run time is achieved with the FTDR algorithm. In the presence of link faults, the performance degradations with the FTDR and FTDR-H algorithms are much smaller than that with the FoN algorithm. Even at a high fault rate (30% link faults), the performance degradations with the FTDR and FTDR-H algorithms are equal or less than 4% for matrix multiplication and FFT and 8% for wavefront, while the performance degradation with the FoN algorithm is 8%, 5%, and 18% for the three applications, respectively.

D. Performance With Transient Faults

In order to evaluate the performance of the FTDR router with transient faults, we develop a transient fault injector with Verilog, which can generate a one-cycle two-bit transient error in each output link of the router randomly. Only one outgoing link of the router suffers the transient error at each time. The transient error rate is defined as the probability of a link suffering a transient fault per cycle. A transient error rate $e = 1$ means that each router suffers a transient fault on one of its output links in every cycle. In the experiment, the transient error rate varies from 0.01% to 0.2%. We measure the average hop count on different traffic loads (0.1 and 0.2 packets/cycle/node). As shown in Fig. 16, the performance of the router degrades gracefully with the increasing transient error rate. Even at a high transient error rate (e.g., 0.2%), the performance only degrades 2.4% and 3.8% at the two injection rates, respectively.

E. Area Cost Evaluation

Fig. 17 compares the area cost in different network sizes for different deflection routers (FoN, cost-based, FTDR, and FTDR-H). All synthesized results are optimized for area. For the FTDR-H router, the 8×8, 12×12 and 16×16 meshes are

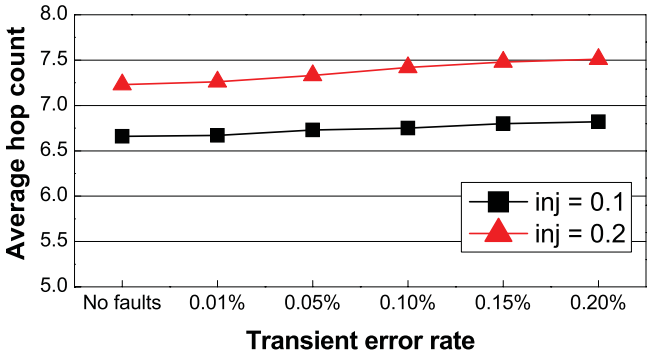


Fig. 16. Average hop count at various transient error rates with different packet injection rates.

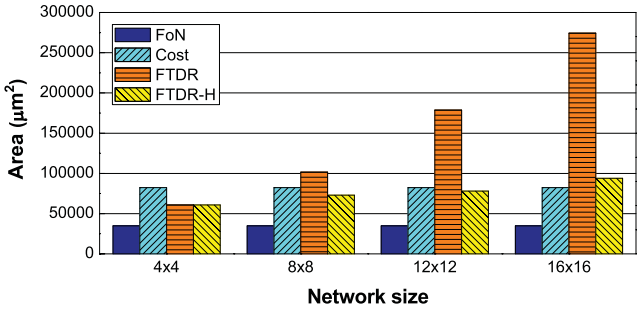


Fig. 17. Router area without ECC encoder/decoder in different network sizes.

divided into 4, 9, and 16 4×4 sub-meshes, respectively. The areas of the FoN and the cost-based routers do not increase with the network size. As the network size increases, the area of the FTDR router increases significantly. The area of the routing table is the main overhead of the FTDR router. For an $n \times n$ mesh, assuming each routing table entry has d bits, the routing table has a cost of $n^2 \times 4 \times d$ bits. In the FTDR-H router, an $n \times n$ mesh is divided into n^2/m^2 $m \times m$ sub-meshes, so the routing table cost can be reduced to $(m^2+n^2/m^2) \times 4 \times d$ bits. (m^2+n^2/m^2) has a minimum value $2n$ (when $m = \sqrt{n}$). The routing table cost of the FTDR-H router can be reduced up to $n/2$ times less than the FTDR router in theory.

VII. Conclusion

In this paper, we provided a fault-tolerant solution for a bufferless NoC to protect it from both transient and permanent faults on the links. Specific contributions of this paper can be described as follows.

1) An on-line fault diagnosis mechanism utilizes SECDED Hamming code to detect both transient and permanent faults, and the encoding scheme can silently correct between 1–7 faulty bits and detect between 2–14 faulty bits, depending on the distribution of the faults over the link.
2) A hybrid ARQ/FEC scheme, which can achieve graceful degradation even at a high fault rate, is proposed to tolerate transient errors during transmission.
3) The FTDR algorithm, which guarantee "0 lost packet" reconfigures the routing table through a reinforcement learning method to route packets avoiding permanent faults. A hierarchical-routing-table-based algorithm (FTDR-H) is also presented to reduce the area overhead of the router. The FTDR and FTDR-H routers outperform the other two fault-tolerant deflection routers.

The experimental results showed that FTDR and FTDR-H routers are high-reliability bufferless routers, which can protect against any fault distribution pattern, as long as the network is not cut into two or more disconnected sub-networks. The FTDR router is cost-efficient for small networks (e.g., less than 64 nodes), while the FTDR-H router has good scalability and is a feasible solution for at least several hundreds of nodes.

The main limitations of this paper lie on the following aspects: 1) a faulty link has to be shut down bidirectional and 2) transient faults last for only one cycle. In future work, we will address these limitations and explore more complex fault models, which will reflect real fault conditions in CMOS technologies of 20 nm and below.

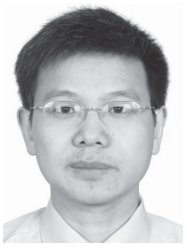
References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Annu. Design Autom. Conf.*, 2001, pp. 684–689.
[2] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul.–Aug. 2003.
[3] Y. H. Kang, T.-J. Kwon, and J. Draper, "Fault-tolerant flow control in on-chip networks," in *Proc. 4th ACM/IEEE Int. Netw.-Chip Symp.*, May 2010, pp. 79–86.
[4] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks on chips," *IEEE Design Test Comput.*, vol. 22, no. 5, pp. 434–442, Sep.–Oct. 2005.
[5] S. Pasricha, Y. Zou, D. Connors, and H. J. Siegel, "OE+IOE: A novel turn model based fault tolerant routing scheme for networks-on-chip," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2010, pp. 85–94.
[6] A. Patooghy and S. G. Miremadi, "XYX: A power & performance efficient fault-tolerant routing algorithm for network on chip," in *Proc. 17th Euromicro Int. Parallel, Distrib. Netw.-Based Process. Conf.*, 2009, pp. 245–251.
[7] Z. Lu, M. Zhong, and A. Jantsch, "Evaluation of on-chip networks using deflection routing," in *Proc. 16th ACM Great Lakes Symp. VLSI*, 2006, pp. 363–368.
[8] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proc. 36th Annu. Int. Symp. Comput. Arch.*, 2009, pp. 196–207.
[9] M. Hayenga, N. E. Jerger, and M. Lipasti, "SCARAB: A single cycle adaptive routing and bufferless network," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarch.*, Dec. 2009, pp. 244–254.
[10] C. Feng, Z. Lu, A. Jantsch, J. Li, and M. Zhang, "A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip," in *Proc. 3rd Int. Workshop Netw. Chip Arch.*, 2010, pp. 11–16.
[11] H. Zimmer and A. Jantsch, "A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip," in *Proc. 1st IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2003, pp. 188–193.
[12] C. Feng, Z. Lu, A. Jantsch, J. Li, and M. Zhang, "FoN: Fault-on-neighbor aware routing algorithm for networks-on-chip," in *Proc. 23rd IEEE Int. SoC Conf.*, Sep. 2010, pp. 441–446.
[13] A. Kohler, G. Schley, and M. Radetzki, "Fault tolerant network on chip switching with graceful performance degradation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 883–896, Jun. 2010.
[14] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: The energy-reliability tradeoff," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 818–831, Jun. 2005.
[15] C. Grecu, A. Ivanov, R. Saleh, E. S. Sogomonyan, and P. P. Pande, "On-line fault detection and location for NoC interconnects," in *Proc. 12th IEEE Int. On-Line Test. Symp.*, Jul. 2006, pp. 145–150.

[16] J. Raik, R. Ubar, and V. Govind, "Test configurations for diagnosing faulty links in NoC switches," in *Proc. IEEE Eur. Test Symp.*, May 2007, pp. 29–34.
[17] 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.
[18] Q. Yu and P. Ampadu, "Transient and permanent error co-management method for reliable networks-on-chip," in *Proc. 4th ACM/IEEE Int. Netw.-Chip Symp.*, May 2010, pp. 145–154.
[19] A. Ejlali, B. M. Al-Hashimi, P. Rosinger, and S. G. Miremadi, "Joint consideration of fault-tolerance, energy-efficiency and performance in on-chip networks," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2007, pp. 1–6.
[20] M. Ali, M. Welzl, and S. Hessler, "An end-to-end reliability protocol to address transient faults in network on chips," in *Proc. Workshop Diag. Serv. Netw.-Chips*, 2007, pp. 376–380.
[21] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant network-on-chip architectures," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2006, pp. 93–104.
[22] T. Dumitras, S. Kerner, and R. Marculescu, "Toward on-chip fault-tolerant communication," in *Proc. Asia South Pacific Design Autom. Conf.*, 2003, pp. 225–232.
[23] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Feb. 2004, pp. 46–51.
[24] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip," in *Proc. 45th ACM/IEEE Design Autom. Conf.*, Jun. 2008, pp. 441–446.
[25] R. Holsmark, M. Palesi, and S. Kumar, "Deadlock free routing algorithms for irregular mesh topology NoC systems with rectangular regions," *J. Syst. Arch.*, vol. 54, nos. 3–4, pp. 427–440, 2008.
[26] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A highly resilient routing algorithm for fault-tolerant NoCs," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2009, pp. 21–26.
[27] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker, "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, pp. 1318–1335, Oct. 1991.
[28] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: Dependable routing for interconnection networks with arbitrary topology," *IEEE Trans. Comput.*, vol. 57, no. 12, pp. 1676–1689, Dec. 2008.
[29] T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosenstiel, "Fully adaptive fault-tolerant routing algorithm for network-on-chip architectures," in *Proc. 10th Euromicro Conf. Digital Syst. Design Arch., Methods Tools*, 2007, pp. 527–534.
[30] A. Mejia, M. Palesi, J. Flich, S. Kumar, P. Lopez, R. Holsmark, and J. Duato, "Region-based routing: A mechanism to support efficient routing algorithms in NoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 356–369, Mar. 2009.
[31] E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch, "Load distribution with the proximity congestion awareness in a network on chip," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2003, pp. 1126–1127.
[32] X. Chen, Z. Lu, A. Jantsch, and S. Chen, "Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2010, pp. 39–44.
[33] B. Narasimham, V. Ramachandran, B. L. Bhuva, R. D. Schrimpf, A. F. Witulski, W. T. Holman, L. W. Massengill, J. D. Black, W. H. Robinson, and D. McMorrow, "On-chip characterization of single-event transient pulsewidths," *IEEE Trans. Device Mater. Rel.*, vol. 6, no. 4, pp. 542–549, Dec. 2006.
[34] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. New York: Wiley, 2005, pp. 235–237.
[35] T. Lehtonen, P. Liljeberg, and J. Plosila, "Analysis of forward error correction methods for nanoscale networks-on-chip," in *Proc. 2nd Int. Conf. Nano-Netw.*, 2007, pp. 1–5.
[36] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in Neural Information Processing Systems 6*, vol. 6. San Mateo, CA: Morgan Kaufmann, 1994, pp. 671–678.
[37] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA: Morgan Kaufmann, 2004, pp. 50–51.
[38] R. M. Piedra, "Parallel 1-D FFT implementation with TMS320C4× DSPs," Texas Instruments, Dallas, TX, Tech. Rep. SRPA 108, Feb. 1994.

Chaochao Feng received the B.Sc. and M.Sc. degrees in computer science and electronic science from the National University of Defense Technology, Changsha, China, in 2005 and 2007, respectively, where he is currently pursuing the Ph.D. degree in electronics science and technology.

He was invited as a Joint Ph.D. Student with the Department of Electronic Systems, Royal Institute of Technology, Stockholm, Sweden, from 2009 to 2010. His current research interests include networks-on-chip and high-performance microprocessor design.



Zhonghai Lu (M'05) received the B.Sc. degree from Beijing Normal University, Beijing, China, in 1989, and the M.Sc. and Ph.D. degrees from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2002 and 2007, respectively.

He is currently an Associate Professor with KTH. His current research interests include computer systems and VLSI architectures, interconnection networks, system-level design, HW/SW co-design, reconfigurable and parallel computing, system modeling, refinements and syntheses, and design automation.

Axel Jantsch (M'97) received the Dipl.Ing. and Dr.Tech. degrees from the Technical University of Vienna, Vienna, Austria, in 1988 and 1992, respectively.

He has been a Full Professor of electronic system design with the Royal Institute of Technology, Stockholm, Sweden, since December 2002. His current research interests include VLSI design and synthesis, system-level specification, modeling and validation, HW/SW co-design and co-syntheses, reconfigurable computing, and networks-on-chip.

Minxuan Zhang received the M.Sc. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 1987.

He is currently a Professor of computer science and technology and microelectronics with the School of Computer, National University of Defense Technology. His current research interests include high-performance computer architecture, microprocessor design, and VLSI design.

Zuocheng Xing is a Professor with the School of Computer, National University of Defense Technology, Changsha, China. His current research interests include power-efficient processor architecture design, low-power architecture, general-purpose computing on graphics processing units, and reliability.